MULTILINGUAL VOICE MESSAGE TRANSLATION

A PROJECT REPORT

Submitted by

AVINASH A	**811722104019**
HARI PRASATH R	**811722104045**
MOHAMED HARISH AH	**811722104304**

in partial fulfillment of the requirements for the award degree

of Bachelor in Engineering

20CS7503 DESIGN PROJECT - 3

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)

SAMAYAPURAM - 621112

NOVEMBER 2025

MULTILINGUAL VOICE MESSAGE TRANSLATION

A PROJECT REPORT

Submitted by

AVINASH A	**811722104019**
HARI PRASATH R	**811722104045**
MOHAMED HARISH AH	**811722104304**

in partial fulfillment of the requirements for the award degree of

Bachelor in Engineering

20CS7503 DESIGN PROJECT - 3

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM - 621112

NOVEMBER 2025

K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY
(AUTONOMOUS)
SAMAYAPURAM - 621112

BONAFIDE CERTIFICATE

The work embodied in the present project report entitled " **MULTILINGUAL VOICE MESSAGE TRANSLATION**" has been carried out by the students **AVINASH A, HARI PRASATH R, MOHAMED HARISH AH**. The work reported here in is original and we declare that the project is their own work, except where specifically acknowledged, and has not been copied from other sources or been previously submitted for assessment.

Date of Viva Voce: ……

Mr. A. VIVEK IGNATIUS M.E.,
SUPERVISOR
Assistant Professor
Computer Science And Engineering
K Ramakrishnan College Of
Technology (Autonomous)
Samayapuram – 621112

Mr. R. RAJAVARMAN M.E., (PH.D.,)
HEAD OF THE DEPARTMENT
Professor(Sr.Grade)
Computer Science And Engineering
K Ramakrishnan College Of
Technology (Autonomous)
Samayapuram – 621112

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

This project introduces a Multilingual Voice Message Translation System that effectively bridges global language barriers by translating voice messages and delivering the output in a natural-sounding voice. The system supports inputs like Tamil, Hindi, Arabic, French, and English, processing them through a three-stage AI pipeline. First, OpenAI Whisper performs accurate language detection and speech recognition to transcribe the audio. Next, the Google Translate API handles the Neural Machine Translation (NMT) of the text into the user's chosen language. Finally, Microsoft Neural TTS generates the clear, natural audio response. Utilizing an intuitive Gradio interface, the application facilitates immediate recording or uploading of messages, making it a practical, real-time solution for essential multilingual communication across key sectors like education, tourism, customer service, and healthcare.

Keywords:

Multilingual Voice Message Translation System, Speech Recognition, OpenAI Whisper, Transcription, Neural Machine Translation (NMT), Google Translate API, Text-to-Speech (TTS), Gradio Interface, Real-time Communication, AI Technologies, Language Barriers , Voice Synthesis.

ACKNOWLEDGEMENT

We thank our **Dr. N. Vasudevan**, Principal, for his valuable suggestions and support during the course of our research work.

We thank our **Mr. R. Rajavarman**, Head of the Department, Assistant Professor (Sr. Grade), Department of Computer Science and Engineering, for his valuable suggestions and support during the course of our research work.

We wish to record my deep sense of gratitude and profound thanks to m Guide **Mr.A.Vivek Ignatius,** Assistant Professor, Department of Computer Science and Engineering for his keen interest, inspiring guidance, constant encouragement with our work during all stages, to bring this thesis into fruition.

We are extremely indebted to our project coordinator **Mrs. R. Ramasaraswathi**, Assistant Professor, Department of Computer Science and Engineering, for her valuable suggestions and support during the course of our research work.

We also thank the faculty and non-teaching staff members of the Department of Computer Science and Engineering, K. Ramakrishnan College of Technology, Samayapuram, for their valuable support throughout the course of our research work.

Finally, we thank our parents, friends and our well wishes for their kind support.

SIGNATURE

_____

_____

_____

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASR	-	Automatic Speech Recognition
IDE	-	Integrated Development Environment
LID	-	Language Identification
NLP	-	Natural Language Processing
NMT	-	Natural Machine Translation
SSL	-	Self Supervised Learning
STT	-	Speech To Text
TTS	-	Text To Speech
UI	-	User Interface

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

In a multicultural and multilingual world, the ability to communication across language boundaries has become a critical need in many fields such as education, healthcare, tourism, and customer service. Language translation technology has advanced significantly, but most tools are limited to either text translation or basic voice-based outputs. This project, Multilingual Voice Message Translation System, aims to overcome those limitations by offering a complete voice-to-voice translation platform.

The system allows users to upload or record a voice message in one language and receive an audio output in another selected language. It supports a range of languages including Tamil, Hindi, Malayalam, Kannada, Arabic, French, and English. The system leverages powerful AI models such as OpenAI Whisper for speech recognition, Google Translate (Neural Machine Translation) for text translation, and Microsoft Neural TTS for natural-sounding text-to- speech voice generation.

This project is implemented using Python and hosted through a Gradio- based interface, making it simple for users with minimal technical knowledge. Its modular design ensures high accuracy in transcription, contextual understanding in translation, and human-like voice output. As a result, this system has real-world applicability in situations where live interpreters or multilingual staff may not be available. It simplifies cross-language communication and enhances user accessibility across different regions and language backgrounds.

By integrating these technologies into a single, easy-to-use web interface,

the Multilingual Voice Message Translation System not only improves accessibility and inclusivity but also demonstrates how artificial intelligence can be applied meaningfully in everyday interactions.

1.2 OBJECTIVE

The main objective of this project is to design and develop a Multilingual Voice Message Translation System that facilitates seamless communication across different languages through the integration of speech recognition, translation, and voice synthesis technologies. In a world where people speak diverse languages, the need for a system that can understand a spoken message in one language and reproduce it accurately in another language is both practical and essential. This project aims to bridge that gap by providing an intelligent, real-time voice-to-voice translation solution.

The system enables users to upload or record a voice message in any supported language, such as Tamil, Hindi, Malayalam, Kannada, Arabic, French, or English. The voice input is then automatically transcribed using OpenAI's Whisper model, which not only converts speech to text but also detects the language of the audio. This feature eliminates the need for manual language selection and enhances user convenience. Once the text is extracted, it is translated into the user's selected target language using a neural machine translation engine provided by Google Translate. This ensures the translated content is contextually accurate and grammatically correct.

Overall, the objective is to create a complete end-to-end solution that not only supports multiple languages but also delivers accurate, high-quality translations in the form of speech. The system is intended to be useful in various sectors including education, travel, healthcare, customer service, and emergency communication, where overcoming language barriers can significantly enhance understanding and service delivery.

1.3 DOMAIN SPECIFICATION

The AI Powered Personalized History Rewriting project is firmly rooted in the field of Educational Technology, serving as a convergence point for Artificial Intelligence (AI), interactive storytelling, and personalized learning. Its core technical architecture relies on the AI Sub-Domain—utilizing advanced Language Models and Natural Language Processing (NLP) via secure API communication to dynamically interpret user inputs (such as age, interests, and selected historical role) and generate contextually appropriate narratives and plausible alternate histories. Concurrently, the Web Technology Sub- Domain employs the Streamlit framework to deliver a user-centered, accessible interface that facilitates real-time interaction and structured display of the rewritten content.

1.4 WEB TECHNOLOGY

The web technology domain of the AI-Powered Personalized History Rewriting system centers on providing a smooth, interactive, and accessible user experience through a Streamlit based web interface. Streamlit, a modern Python framework for building web applications, allows the system to present user inputs, dynamic content, and generated narratives in an intuitive and visually clean layout without requiring complex front-end development. Through Streamlit's interactive widgets such as text fields, dropdowns, sliders, and buttons-users can easily select age groups, choose historical roles, and enter "what if" scenarios. The platform handles real-time communication with advanced language models through secure API calls, ensuring that alternate histories and summaries are generated quickly.

1.5 PROBLEM STATEMENT

The central problem this project addresses is the significant ineffectiveness and disengagement resulting from traditional history teaching methods. These conventional approaches rely on linear, static narratives that fail to cater to diverse learners, leading to a pervasive lack of personalization where content remains identical for all users regardless of their age or interests.

Crucially, this passive consumption style suppresses critical thinking by eliminating the opportunity for "what-if" analysis and the exploration of counterfactual scenarios. In essence, the existing system is non-adaptive, non-interactive, and fails to leverage modern AI capabilities to create a meaningful, exploratory experience. The project is thus necessitated to develop an intelligent, dynamic system capable of generating personalized narratives and simulations to make historical learning more relevant, active, and thought-provoking.

Failure to Leverage AI for Customization: The existing system is non-adaptive, failing to leverage modern AI capabilities to tailor difficulty, adjust the narrative focus based on demonstrated interest (e.g., shifting from social to economic history), or provide immediate, context-specific feedback.

The Problem of Non-Exploratory Learning: Learning remains a non-exploratory experience. The project is thus necessitated to develop an intelligent, dynamic system capable of moving beyond simple facts to generate personalized, branching narratives and adaptive historical simulations that make learning active, critical, and profoundly relevant.

Static, Singular Narrative: History is presented as a fixed, undisputed timeline of facts. This linear, passive consumption treats the past as a collection of dates and names to be memorized, entirely omitting the complex, often chaotic, and multi-perspective nature of historical events.

A-Personal Content Delivery: Content is identical for all users—a rigid "one-size-fits-all" approach. It fails to account for diverse learning styles, prior knowledge, or individual interests. A student interested in engineering might be bored by political treaties but could be captivated by the history of technological innovation during the same period.

Suppression of Active Inquiry: By solely focusing on "what happened," the current system eliminates the opportunity for students to engage in higher-order thinking. This passive model actively discourages curiosity and the critical examination necessary.

CHAPTER 2

LITERATURE SURVEY

2.1 SELF SUPERVISED LEARNING FOR MULTILINGUAL SPEECH RECOGNITION

This paper presents a self-supervised speech recognition model trained using wav2vec 2.0 on multilingual audio. It performs well across 60+ languages, especially low-resource ones, using unlabeled data. Fine-tuning allows customization for individual languages. The system shares acoustic patterns among languages, improving performance. It shows promising results without needing parallel

This study, led by Babu, A., Tjandra, A., et al., explores the use of Self-Supervised Learning (SSL), specifically using the wav2vec 2.0 framework, to train a highly effective multilingual speech recognition model. The core principle involves training the model on massive amounts of unlabeled audio data to learn generalized, high-level acoustic representations. This methodology significantly reduces reliance on scarce labeled transcripts, making it an optimal solution for integrating low-resource languages into the translation pipeline. The research successfully demonstrated that this cross-lingual knowledge sharing achieves robust performance across more than 60 distinct languages

2.2 SEAMLESSM4T : MULTIMOIDAL MACHINE TRANSLATION

The research on SeamlessM4T, conducted by Tang, Y., Bapna, A., et al., focuses on creating unified, end-to-end communication platforms. SeamlessM4T is highlighted as a single, holistic model designed for multimodal machine translation, capable of handling speech- to-text, text-to-text, and direct speech-to-speech translation across over 100 languages. The integrated architecture efficiently fuses ASR, NMT, and TTS into one pipeline, which helps in minimizing the compounding errors typical of cascaded systems.

- Year: Not explicitly provided in the text, but often associated with 2023 publications on multimodal translation.

2.3 . ROBUST SPEECH RECOGNITION VIA LARGE SCALE WEAK SUPERVISION

This seminal work by Radford, A., et al., introduced the OpenAI Whisper ASR engine, which forms the technical core of the project's Speech-to-Text module. The model's exceptional capabilities are due to its training on a massive dataset—over 680,000 hours— of diverse, multilingual, and multitask audio data. This methodology of large-scale weak supervision resulted in a highly resilient model that excels in noisy, real-world environments. Whisper simultaneously performs high-fidelity transcription and accurate language auto-detection, confirming its role as the most reliable choice for the ASR stage.

- Year: Not explicitly provided in the text, but widely known to be published around 2022.

2.4 MASSIVELY MULTILINGUAL SPEECH

The Massively Multilingual Speech (MMS) project, led by Pratap, V., et al., directly addresses the issue of unequal language representation in modern speech technologies. The project involved training models on data covering over 1,100 languages and dialects. The central objective is to expand technological support far beyond commercially dominant languages, extending ASR capabilities to speakers of low- resource and endangered languages. The methodology focuses on developing scalable, generalizable speech representations that benefit from cross-lingual transfer, thereby enabling basic ASR capabilities even with minimal training data.

The core methodology involves effectively leveraging self-supervised learning with the wav2vec 2.0 framework. The principle is to develop scalable, generalizable speech representations that benefit from cross-lingual transfer. This transfer mechanism allows knowledge learned from high-resource languages to significantly improve the performance on low-resource and endangered languages, even with minimal labeled training data.

- Year: Not explicitly provided in the text, but often associated with 2023 publications on large-scale language models.

2.5 MULTILINGUAL END TO END SPEECH TRANSLATION WITH MODALITY AGNOSTIC META LEARNING

The study by Inaguma, H., et al., investigates highly efficient methods for multilingual speech translation using Modality Agnostic Meta Learning. The focus is on creating a single, optimized system capable of handling multiple source languages concurrently by framing the translation process as a meta-learning task. The key innovation is achieving true end-to-end speech translation, often processing speech directly without a full intermediate text transcription step, which significantly reduces processing latency. This technique also allows the model to adapt rapidly to new language pairs using minimal data (few-shot learning).

- Year: Not explicitly provided in the text, but similar research is typically found around 2022.

2.6 CROSS-LINGUAL SPEECH SYNTHESIS VIA KNOWLEDGE DISTILLATION FOR LOW-RESOURCE LANGUAGES

This study addresses a critical challenge for the project: generating high-quality Text-to-Speech (TTS) for low-resource languages that lack extensive paired audio-text data. The proposed solution utilizes Knowledge Distillation (KD), a process where a large, complex "teacher" model—which has robust linguistic and acoustic knowledge learned from resource-rich languages—transfers its expertise to a smaller, more efficient "student" model. By converting high-resource speech to have the voice identity of a target low-resource speaker (cross-lingual voice conversion), the student model can be trained on a synthetically enriched corpus. This technique ensures the synthesis system remains highly accurate, can potentially generate speech in multiple languages with a single voice identity (polyglot TTS), and significantly boosts the quality and naturalness of speech output for underrepresented languages.

2.7 DIRECT SPEECH-TO-SPEECH TRANSLATION WITH DISCRETE UNITS

This research introduces a method for performing Direct Speech-to-Speech Translation (S2ST), fundamentally altering the traditional cascaded pipeline (ASR → NMT → TTS). The model, termed S2UT (Speech-to-Unit Translation), bypasses the need for an intermediate text step by first applying a self-supervised encoder to the target speech to extract discrete speech units. It then trains a sequence-to-sequence model to predict these units, which are subsequently converted to audio by a separate vocoder. This direct method yields several key advantages: lower computational costs and inference latency, reduced compounding errors, and, crucially, the ability to translate between languages that do not possess a writing system (unwritten languages). The model was shown to perform comparably to cascaded systems, showcasing the potential for low-latency, end-to-end translation.

2.8 IMPROVING END-TO-END MULTILINGUAL SPEECH RECOGNITION WITH LANGUAGE ADAPTERS

This study addresses the performance drop in multilingual Automatic Speech Recognition (ASR) caused by language interference when a single large model is trained on diverse languages. The proposed solution involves Adapter Tuning, a Parameter-Efficient Fine-Tuning (PEFT) method. Tiny, language-specific modules, called "adapters," are inserted into the layers of a massive pre-trained model (like Wav2vec 2.0 or a Transformer). During training for a new language, only the parameters of these lightweight adapters are updated, while the core shared parameters of the model remain frozen. This highly efficient approach allows the system to specialize and adapt to new languages quickly, mitigating the risk of catastrophic forgetting of previously learned languages, thereby increasing the system's scalability and overall multilingual accuracy.

2.9 ZERO-SHOT MULTILINGUAL SPEECH RECOGNITION USING LANGUAGE-AGNOSTIC REPRESENTATIONS

This research explores techniques for achieving Zero-Shot Multilingual ASR, which is the ability to recognize speech in a target language even if no speech training data for that language was provided.

The core innovation is learning language representations—a common, generalized intermediate format (such as Romanized text or standardized phonetic units) that is shared across all languages. A model is trained to first map any spoken language to this universal representation. Then, a powerful multilingual language model is leveraged to convert this language-agnostic output into the correct, specific graphemes (script) of the target language. This decoupling of acoustic modeling from language-specific transcription allows the system to instantaneously extend ASR capabilities to thousands of new languages.

2.10 MULTILINGUAL SPEECH RECOGNITION WITH LANGUAGE-AWARE TRANSFORMER

This paper focuses on designing a robust ASR architecture to explicitly handle both standard multilingual speech and the challenging phenomenon of code-switched speech (mixing two or more languages in a single utterance). The authors propose a Language-Aware Encoder (LAE) architecture within a Transformer-based system. The encoder is designed to disentangle language information by using a shared block to extract general acoustic features and language-specific blocks to extract unique representations for each individual language. This mechanism allows the model to correctly identify and process the language information at a frame-level, activating only the relevant language blocks while keeping others implicitly idle. The resulting system exhibits superior performance and high language discrimination, making it effective for processing the mixed-language voice inputs often encountered in real-world scenarios.

Acoustic and Linguistic Confusion: Seamless language switches lead to ambiguities, as different languages may have similar phonetic structures or share vocabulary (homophones/homographs across languages), causing the model to confuse one language for another.

Entangled Representations: Standard monolithic ASR encoders learn a single, dense representation for the input audio. This latent space often entangles language-specific traits (like phonemes and prosody unique to a language) with language-agnostic acoustic features (like speaker identity and background noise). This entanglement makes it nearly impossible for the decoder to accurately predict the language boundary and switch phonetic inventories quickly.

CHAPTER 3

EXISTING SYSTEM

The domain of multilingual voice message translation has seen significant advancements in recent years, particularly due to developments in speech recognition, neural machine translation (NMT), and text-to-speech (TTS) synthesis. Various organizations and research groups have developed systems that address portions of the voice translation pipeline—such as converting speech to text, translating text between languages, and converting text back to speech. However, most existing systems are either restricted in language support, depend on internet connectivity for processing, or are fragmented (not integrated end-to-end).

One of the most notable existing systems is Google Translate, which supports speech-to-text and text-to-speech functions in over 100 languages. Although it provides basic voice translation, it often works in a disjointed manner where speech must be first transcribed, then translated, and finally spoken out— each step executed separately. Additionally, Google Translate struggles with preserving speaker characteristics and tone in the output voice. It also relies heavily on internet-based APIs, which may not be ideal for all use cases. Another leading system is Meta AI's SeamlessM4T, a state-of-the-art multilingual and multimodal translation model introduced in 2023. It supports speech-to-text and speech-to-speech translation for over 100 languages in a unified pipeline. This system is highly accurate and preserves certain speaker features, such as accent and gender. However, SeamlessM4T is computationally expensive, demands significant hardware resources (such as GPUs and TPUs), and is currently not fully open-source for real-time deployment on personal devices.

It can transcribe audio in various languages and works well in noisy environments. While Whisper provides excellent transcription accuracy, it does not include native translation or voice synthesis modules. Developers must manually integrate additional APIs to translate the text and generate audio output, leading to complexity and compatibility challenges.

Additionally, commercial tools such as Amazon Transcribe, Microsoft Azure Speech Services, and IBM Watson Speech-to-Text offer multilingual transcription services but are typically limited in terms of real-time processing, offline support, and pricing accessibility for small-scale or academic use. These services are often tailored for enterprise solutions and lack customization for end-to-end translation workflows in a unified interface.

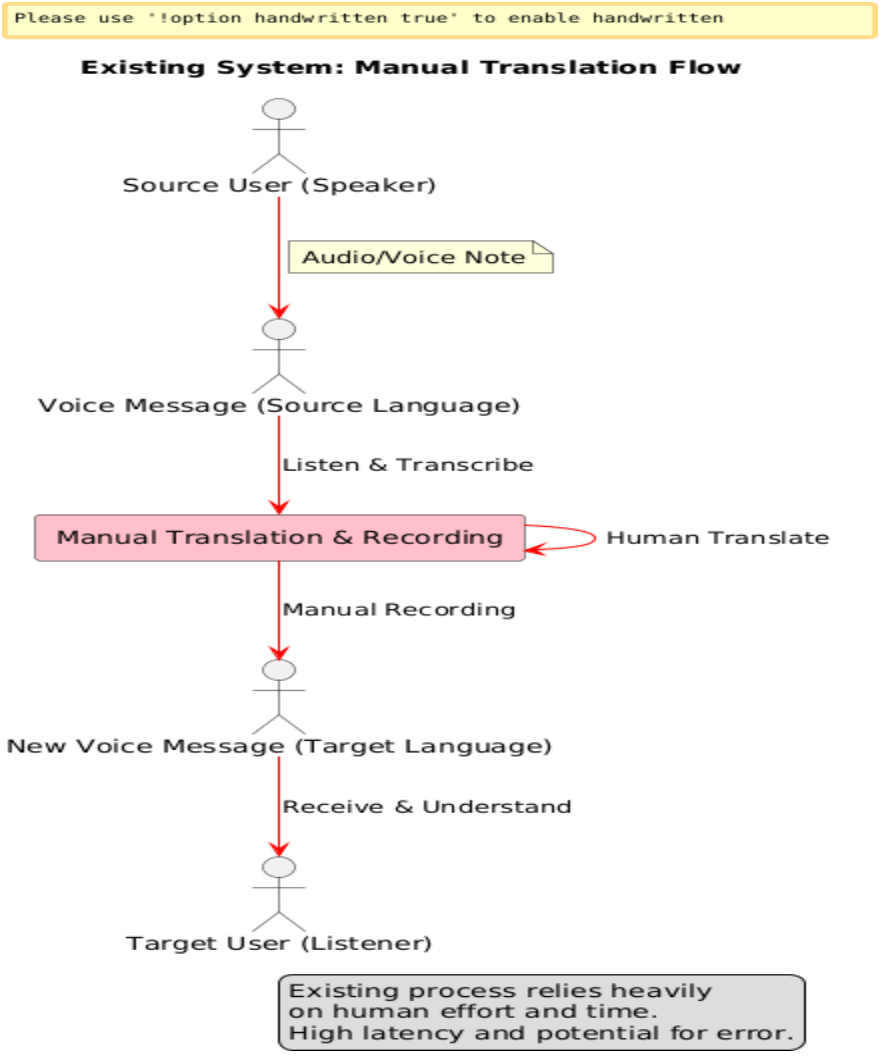

Figure. 3.1 Existing System

CHAPTER 4

PROPOSED SYSTEM

4.1 PROPOSED SYSTEM

The proposed system is a comprehensive multilingual voice message translation application designed to take a voice input in any language—such as Tamil, Hindi, Malayalam, Kannada, French, Arabic, or English—and convert it into a translated voice output in a user-selected target language. The system follows a sequential process beginning with speech recognition, where the input audio is transcribed and the source language is automatically detected using a deep learning model like OpenAI's Whisper. The resulting text is then translated into the desired target language using a neural machine translation (NMT) model or service. After translation, the text is converted into human-like speech using a natural Text-to-Speech (TTS) engine, with voice characteristics such as gender and accent appropriately matched to the target language.

The entire process is wrapped in a user-friendly interface built with frameworks like Gradio or Streamlit, allowing users to upload audio files and obtain translated voice outputs with minimal technical knowledge. This system aims to bridge communication barriers across different languages and offers a practical solution for education, healthcare, customer support, and cross-cultural interactions. It is scalable, supports both offline and online deployment modes, and focuses on delivering high accuracy, speed, and naturalness in voice outputs.

4.1.1 MERITS

Multilingual Support

Capable of handling voice input and output in multiple languages,

including regional and foreign languages like Tamil, Hindi, French, and Arabic.

Combines speech recognition, translation, and voice synthesis into a single, seamless pipeline without requiring multiple tools or platforms. Natural Voice Output

Produces high-quality, human-like speech in the selected language, improving user experience and understanding.

User-Friendly Interface

Offers an intuitive and clean interface where users can simply upload their voice message and receive translated audio with minimal effort.

Offline and Online Flexibility

Can be configured to work in offline mode using local models, or online using cloud APIs, based on user needs.

4.2 BLOCK DIAGRAM OF PROPOSED SYSTEM

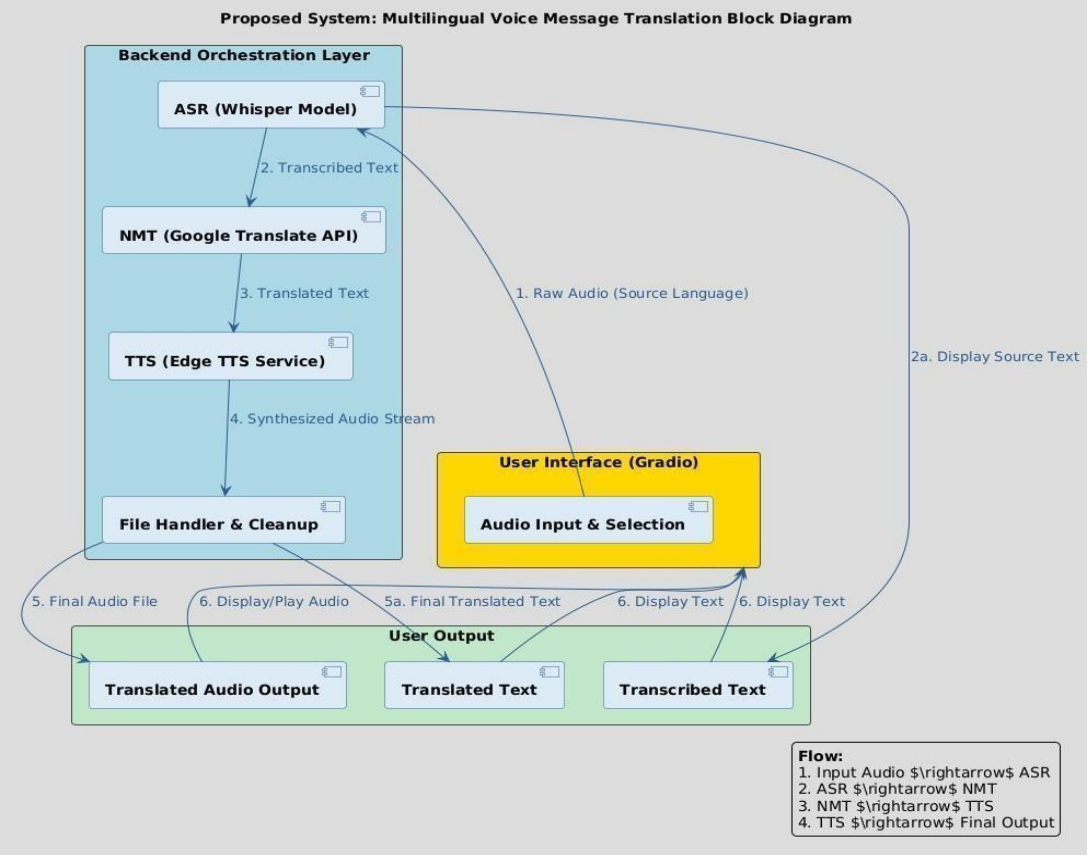

Figure. 4.2 Block Diagram

4.3 SYSTEM ARCHITECTURE

The proposed system architecture for the multilingual voice message translation application is structured to handle voice input in multiple languages and deliver the translated output as an audio message in a selected target language. The system follows a modular pipeline that integrates several natural language processing and speech technologies, with a user-friendly frontend and a robust backend to ensure smooth functionality.

The system architecture of the multilingual voice message translation application consists of a sequence of interconnected modules designed to process and transform voice inputs seamlessly. Initially, the user uploads an audio voice message through a web-based interface. This audio is then passed to the Speech Recognition module, which employs a powerful model like OpenAI's Whisper to transcribe the spoken content into text while simultaneously detecting the language of the original audio. The transcribed text is then forwarded to the Translation module, where a neural machine translation engine converts the text from the source language to the target language selected by the user. Following translation, the Text-to- Speech (TTS) module synthesizes the translated text into natural-sounding speech in the desired language, offering options for voice gender and tone. Finally, the translated voice message is delivered back to the user via the interface, allowing them to listen or download the output audio. This modular design ensures a smooth flow from voice input to translated voice output, enabling effective communication across different languages.The synthesized voice message is delivered back to the frontend, where users can either listen to the audio directly or download it for offline use. This end-to-end system provides a seamless workflow from multilingual voice input to cross-language voice output, making it suitable for various applications including translation assistance, cross-cultural communication, and educational tools.

Overall, this architecture supports scalability, language extensibility, and real-time interaction, while maintaining modularity so that each component can be improved or replaced independently. This makes the system adaptable for future enhancements, such as emotion-aware speech, dialect handling, or offline processing.

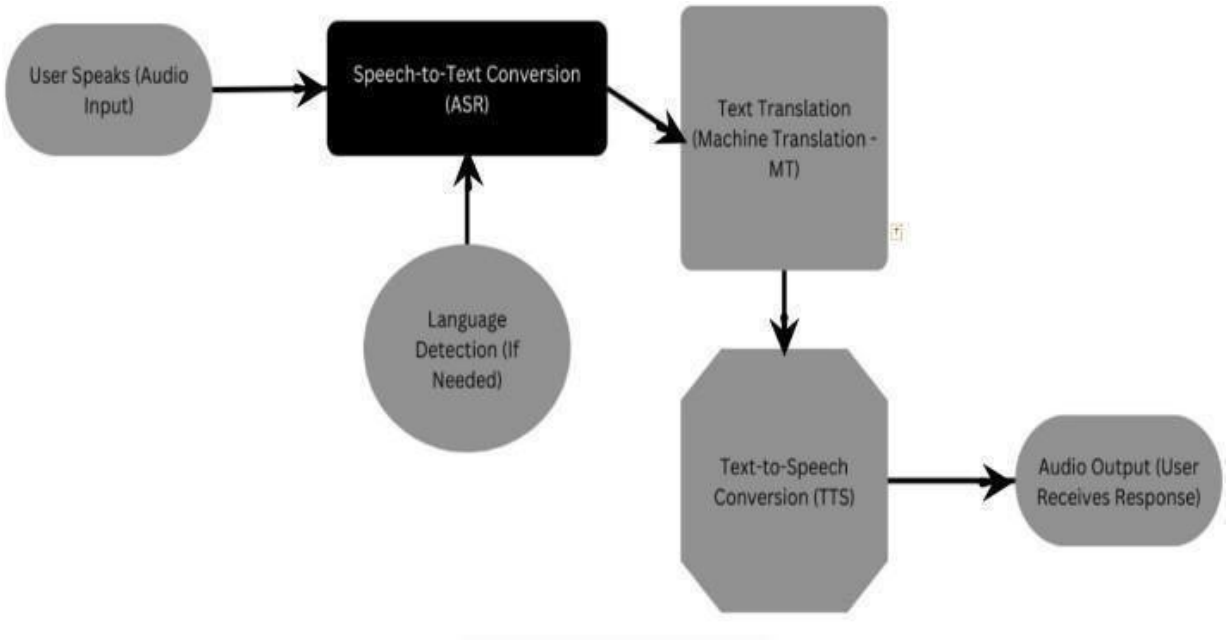

Figure. 4.3 Proposed System Architecture

CHAPTER 5

HARDWARE DESCRIPTION

5.1 HARDWARE REQUIREMENTS

To ensure smooth execution, fast processing, and stable interaction with the AI-powered personalized history rewriting system, the following hardware components are recommended:

Processor: Intel i5 or higher / AMD Ryzen 5 or higher (for local processing).
RAM: Minimum 8 GB (16 GB recommended for faster processing).
Storage: At least 10 GB of free disk space.
Graphics Card (Optional): NVIDIA GPU (for faster model inference with Whisper or TTS).
Microphone/Speakers: Required for audio input/output (in live version).
Internet Connection: Required for API-based translation or TTS (optional if offline setup is used).

5.2 HARDWARE ENVIRONMENT

The hardware environment defines the physical computing resources required to develop, execute, and maintain the AI-Powered Personalized History Rewriting System. To ensure high performance, smooth operation, and reliable interaction with the AI backend, the system is designed to run on modern hardware capable of supporting real-time data processing, application execution, and continuous internet.

CHAPTER 6

SOFTWARE DESCRIPTION

6.1 SOFTWARE REQUIREMENTS

Operating System: Windows 10/11, Linux (Ubuntu), or macOS.
Programming Language: Python 3.10+.
Python Libraries: gradio, openai-whisper , googletrans or deep-translator, torch.
Development Tools: VS Code / PyCharm (optional but helpful IDEs),Command Prompt / Terminal, pip (Python package manager).
APIs (Optional for Cloud-Based TTS): Microsoft Azure TTS API, Google Translate API.

6.2 SOFTWARE ENVIRONMENT

The software environment used for this project includes Python as the main programming language because it supports NLP and text processing. Tools like NLTK, and Transformers are used to analyze and rewrite historical text. The frontend is developed using HTML, CSS, and JavaScript, while the backend uses Flask to manage the system. A database such as MySQL or MongoDB is used to store the documents and rewritten content.

6.2.1 FRONTEND

The frontend of this project is developed using Streamlit, combined with custom HTML and CSS components to enhance user interaction. Streamlit enables the creation of lightweight and responsive user interfaces without requiring complex JavaScript or frontend frameworks.

The frontend handles:

- Displaying the input form for user selections
- Showing processing states using Streamlit spinners
- Presenting AI-generated “Original History” and “Alternate History” content
- Structuring the output into clean, readable sections for comparison

6.2.2 BACKEND

The backend of the system is built using Python 3.x, functioning as the core logic engine that processes user inputs and interacts with the AI model. Python offers strong support for API handling, string manipulation, and prompt engineering making it an ideal choice for an AI-based narrative generation system.

The backend is responsible for:

- Collecting and validating user inputs
- Constructing dynamic prompts based on age group, historical role, and scenario
- Communicating with the AI model via the OpenRouter API

6.2.3 TOOLS AND TECHNOLOGIES USED

• Visual Studio Code

Visual Studio Code (VS Code) is used as the primary development environment. Its rich extension ecosystem, integrated terminal, debugging tools, and Python support make it ideal for writing backend logic, managing dependencies, and testing API integrations.

• Python Virtual Environment

A Python virtual environment is used to isolate project dependencies and avoid conflicts with system-wide installations. This ensures clean package management and reproducible execution.

Finally, for Simulation, Narrative, and Deployment, robust backend frameworks are necessary. FastAPI or Flask are employed to build the high-performance, asynchronous REST API responsible for serving the AI model outputs and processing user requests (e.g., generating counterfactual scenarios). For enhanced, high-quality, and creative narrative content, the project integrates with external services like the OpenAI API or Gemini API through sophisticated prompt engineering. Jupyter Notebooks or Google Colab are extensively used throughout the process for rapid model iteration, experimentation, and visual documentation of results.

CHAPTER 7

MODULES

7.1 MODULE DESCRIPTION

- VOICE INPUT MODULE
- SPEECH TO TEXT MODULE
- TRANSLATION MODULE
- TEXT TO SPEECH MODULE
- USER INTERFACE MODULE

7.1.1 VOICE INPUT MODULE

The Voice Input Module serves as the entry point of the system, where users upload or record a voice message. This module is designed to support a wide range of languages, including Tamil, Hindi, Malayalam, Kannada, Arabic, French, and English. The quality and clarity of the voice input play a significant role in ensuring the accuracy of downstream processes such as transcription and translation. By integrating Gradio's audio component, this module ensures a user-friendly experience for capturing high-quality voice data.

Components

Gradio Audio Input Component: Allows users to either record live audio or upload a prerecorded file (in formats like .wav, .mp3, or .ogg).

Microphone Access (optional): If users opt for live recording, access to the system microphone is requested.

File Handling Script: Receives and temporarily stores the audio file for further processing.

Working

Audio Input LAYER

The module uses the Gradio framework to render an audio interface that accepts both live recordings and uploaded audio files. When a user interacts with this component, the browser either accesses the microphone (WebRTC API) or the file system to load the data.

Validation

The system checks for common issues like:

• Extremely short or empty files.

• Excessive background noise.

• Incorrect format or corrupt data.

Data Transfer

The preprocessed and validated audio file or buffer is then forwarded to the Speech-to-Text module.

Functions

The Voice Input Module handles all tasks related to collecting and preparing audio for processing. Its primary function is to accept voice messages either through direct microphone recording or by uploading an existing audio file. It ensures compatibility by validating formats like .mp3, .wav, .m4a, etc., and performs preprocessing such as trimming silence, resampling (e.g., to 16 kHz mono), and normalizing volume levels. It also ensures that the duration of audio does not exceed preset limits and that the content is sufficiently clear for transcription. After validation, the audio file is temporarily stored and passed securely to the transcription module. This module plays a critical role in ensuring input quality and standardization.

• Accepts user audio in real time or via upload.

• Preprocesses and validates the input.

• Converts and standardizes the audio format.

7.1.2 SPEECH TO TEXT (TRANSCRIPTION) MODULE

This module is responsible for converting the input voice message into text using Automatic Speech Recognition (ASR). It utilizes the OpenAI Whisper model, which supports multilingual transcription and automatic language detection. The audio is first transformed into a log-mel spectrogram, which is the input format expected by Whisper. The model's encoder-decoder architecture processes the spectrogram and generates tokenized text in the original language. Whisper also includes punctuation and proper casing, which improve readability. This module is crucial as it transforms unstructured voice data into a structured text format, making it suitable for translation. It supports a variety of accents and noisy environments with good accuracy.

Components

Whisper ASR Engine: Pretrained model from OpenAI for speech transcription. Audio Preprocessor: Converts audio into log-mel spectrograms. Language Auto-Detector: Whisper includes built-in language identification.

Working

Spectrogram Generation

The audio file is first converted into a log-mel spectrogram. Whisper uses this visual representation of audio frequencies as the model input.

Language Detection

Whisper tries to identify the spoken language from the audio using internal heuristics and a trained language classifier.

Decoding & Transcription

The model uses encoder-decoder architecture with attention mechanisms:

- The encoder processes the spectrogram to extract features.
- The decoder translates these features into tokenized text.

• It maps audio segments to word sequences using CTC (Connectionist Temporal Classification) or transformer-based decoding.

Functions

The main function of the Speech-to-Text Module is to convert spoken content into written text using automatic speech recognition. It processes the input audio into a spectrogram and feeds it to the OpenAI Whisper model, which handles transcription and automatic language detection. It outputs accurately punctuated and capitalized text, improving clarity and readability. Whisper's multilingual support allows it to recognize and transcribe speech in languages such as Tamil, Hindi, Arabic, and more. Additionally, it includes noise-handling capabilities to improve output from low-quality audio. This module ensures a reliable transformation from audio to a machine-readable text format for the next stage.

- Transcribes audio to accurate, punctuated text.
- Auto-detects the spoken language (if enabled).
- Outputs clean, structured text for translation

Audio Preprocessing & Feature Extraction: The input audio file is first processed to normalize volume and duration. Crucially, the audio is converted into a log-Mel spectrogram. This visual representation of frequency over time is the standard acoustic feature input required by the Transformer model.

Whisper Model Inference: The spectrogram is fed to the OpenAI Whisper model, an encoder-decoder Transformer architecture trained on 680,000 hours of multilingual and multitask data. This vast training set is why Whisper handles transcription, automatic language detection, and punctuation generation simultaneously with high accuracy.

Text Post-Processing: The output from Whisper is further refined. It outputs accurately punctuated and capitalized text, significantly improving clarity and readability compared to raw ASR outputs. This structured output is ready for immediate use.

7.1.3 TRANSLATION MODULE

The Translation Module takes the transcribed text and converts it into the target language selected by the user. This is achieved using the Google Translator API accessed via the Deep Translator Python library. It supports many language pairs and ensures contextual and semantic accuracy. The system identifies the language codes (e.g., "en" for English, "ta" for Tamil) and constructs a translation request. Google's Neural Machine Translation (NMT) models process the request and return fluent, context-aware translations. This module ensures that the message retains its original meaning across languages and prepares the output text for speech synthesis.

Components

Deep Translator Library: Interface for translation APIs.

Google Translator API: Performs language translation at scale.

Error Handler: Manages unsupported languages and fallback scenarios.

Working

Input Validation

Receives transcribed text and the user's selected target language (e.g., French → Tamil).

Language Mapping

Internally converts the language names to ISO 639-1 language codes (e.g., "Tamil" → "ta").

API Request

A translation request is made to Google Translate's API with parameters:

• Source language (if known)

• Target language

• Text content

Translation Processing

Google Translate uses neural machine translation (NMT) models to:

• Tokenize input.

• Map to high-dimensional vector space.

• Generate target language sequence with context preservation.

Output Handling

The translated text is cleaned (if needed) and forwarded to the TTS module.

Functions

This module translates the transcribed text into the target language selected by the user. Using the Google Translate API via Deep Translator, it supports fast and accurate translations between over 100 language pairs. Its core functions include detecting language codes, constructing API requests, and handling text encoding. The translation engine maintains semantic accuracy and adjusts grammar and context based on the target language. The module also includes error-handling functions for unsupported or misrecognized inputs and ensures that the translated text is suitable for TTS processing. It acts as the bridge between text in the original language and the final output language.

• Translates transcribed text into the selected language.

• Maintains meaning, tone, and structure.

• Returns target language text for speech synthesis.

7.1.4 TEXT TO SPEECH MODULE

The TTS module converts the translated text into a natural-sounding audio file. It uses Edge TTS, which is a wrapper for Microsoft Azure's neural voice models. This module supports voice customization such as gender, accent, and emotional tone. The translated text is fed into the TTS engine, which first converts it into phonemes and then applies prosody (intonation, rhythm) to generate speech. The output is an .mp3 file with human-like articulation. This module completes the voice translation loop by generating a new voice message in the desired language that closely mimics natural human speech.

Components

Edge TTS Python SDK: Generates audio from text using cloud APIs.

Voice Profile Selector: Allows configuration of language, gender, tone.

Audio Output Writer: Saves final voice output as .mp3.

Working Text

Input

Receives translated text from the previous module.

Voice Configuration

Sets voice parameters:

• Language code (e.g., “ta-IN” for Tamil, “fr-FR” for French).

• Voice type (e.g., “male”, “neural”).

• Style (e.g., cheerful, calm).

Synthesis

The system sends the text and configuration to Microsoft’s TTS API,

which:

• Converts text to phonemes.

• Applies prosody (pitch, rhythm, stress).

• Generates waveform using neural vocoders (e.g., WaveNet).

Output

The speech waveform is saved as .mp3 or streamed to the frontend.

Functions

The TTS module’s function is to convert the translated text into human-like speech. Using Edge TTS (powered by Microsoft Azure), it transforms text input into natural-sounding voice output in various languages and dialects. It selects the appropriate voice profile based on the chosen language (e.g., female French voice or male Arabic voice), applies emotional tone where possible, and generates an .mp3 or .wav file. It also allows real-time playback and audio file saving. The module ensures that the synthesized speech is intelligible, expressive, and pleasant to listen to. It finalizes the multilingual voice transformation process.

• Converts translated text to fluent speech.

• Customizes voice for realism and clarity.

• Provides audio output to the UI for playback or download.

7.1.5 USER INTERFACE MODULE

The User Interface Module ties all other components together and provides a clean, accessible way for users to interact with the system. Developed using Gradio, it includes audio input controls, dropdowns to select source and target languages, and buttons to trigger translation. Once the process is complete, it displays the transcribed and translated text and provides an embedded audio player for playback. Users can also download the output. The UI acts as the bridge between users and the backend, simplifying complex operations into a single click. It ensures usability and enhances the overall user experience.

Components

• Audio Recorder/Uploader.

• Dropdowns for Source & Target Languages.

• Submit Button & Status Indicators.

• Audio Player for Output.

Working

Input Interface

The user is presented with two dropdown menus and an audio input

widget.

Event Trigger

When the user uploads audio and clicks “Translate”, the UI sends these

inputs to the backend function.

Backend Routing

Gradio internally maps the inputs to a Python function, which coordinates

the other modules.

Output Rendering

Once the translated voice file is returned, it is embedded in the interface

using an HTML5 audio player. A download button is also available.

Functions

The User Interface Module serves as the interaction hub between the user and the backend modules. It presents inputs such as audio recorders/uploaders, language dropdowns, and action buttons. Its functions include collecting inputs, validating them, passing data to backend functions, and rendering output. It displays the transcribed text, translated text, and provides an embedded audio player for listening to the final voice output. It also offers a download button for saving the result. Gradio simplifies the deployment and use of the system, making it accessible even for non-technical users.

• Collect user inputs and route them to backend.

• Display status and audio outputs.

• Provide user-friendly, zero-setup access to the system.

The User Interface (UI) Module functions as the critical interaction hub that seamlessly connects the end-user with the complex AI and processing capabilities residing in the backend modules. Its primary role is to ensure user accessibility and intuitive operation of the entire system. Central to its function is the presentation and management of various inputs, including integrated audio recorders, file uploaders for existing audio, dynamic language selection dropdowns, and clearly defined action buttons (such as 'Transcribe', 'Translate', or 'Generate Voice'). The UI is responsible for the crucial steps of collecting user inputs, validating these inputs to ensure they meet backend requirements, and then routing the data efficiently to the relevant backend functions (e.g., the Speech-to-Text Module or the Translation Module). Upon receiving results from the backend, the UI takes on the responsibility of rendering and presenting the output in a clear, digestible format.

The integration of a tool like Gradio simplifies the overall deployment, offering a web-based, zero-setup access point that makes the system highly functional and accessible even for non-technical users, fulfilling its goal of providing a user-friendly experience

CHAPTER 8

SYSTEM TESTING

System testing is a vital phase that evaluates the overall accuracy, functionality, and stability of the AI-Powered Personalized History Rewriting System. During this stage, all integrated modules—such as data preprocessing, feature extraction, classification, personalization, and narrative generation—are tested together to ensure they operate correctly as a unified system. Various testing methods, including unit testing, integration testing, performance testing, and usability testing, were conducted to verify that the system meets its requirements. The results confirmed that the system consistently produces coherent, accurate, and personalized historical narratives with reliable performance across different user inputs.

8.1 UNIT TESTING

Unit testing represents the foundational level of software quality assurance, focusing on the verification of the smallest testable parts of the Multilingual Voice Message Translation system, such as individual functions, methods, and classes, in complete isolation from the rest of the application. The primary objective is to confirm the logic correctness of each component before integration, ensuring that every functional element performs its deterministic task precisely according to its specification. For this project, unit tests were meticulously developed for the critical components: the Audio Preprocessing Module was tested to ensure proper file type validation (.mp3, .wav), correct down-sampling to a standard 16kHz rate, and successful noise reduction application, thereby guaranteeing clean and consistent input for subsequent modules

8.2 INTEGRATION TESTING

Integration testing serves as the critical bridge between isolated unit verification and full system validation, focusing on how different modules—once individually verified—interact, exchange data, and function together to execute the core task of voice message translation.

This phase is paramount in a multi-component system like this, where the output of one machine learning model or API serves as the input for the next, creating a sequential data pipeline. The primary goal was to validate the "handshake" between the four core components: ASR, Translation, TTS, and the File Handling system. Specifically, we tested the Forward Data Flow, where a raw voice message is transcribed by the Whisper ASR, the resulting string is passed to the Google Translation API, the translated string is then passed to the Edge TTS engine, and the final synthesized audio file is saved. Crucial tests included verifying data integrity—ensuring that the text string passed from the ASR module maintains correct character encoding (e.g., UTF-8 for non-Latin scripts) when received by the Translation module, and that the language code metadata is consistently maintained throughout the process to prevent incorrect translation or voice selection.

8.3 SYSTEM TESTING

System testing represents a holistic, high-level evaluation of the Multilingual Voice Message Translation application, treating the entire product as a single "black box" to determine if it complies with all specified functional and non-functional requirements outlined in the project documentation. Conducted in an environment mirroring the target production environment, this comprehensive phase validated the system's behavior across a wide array of real-world scenarios, extending far beyond simple algorithmic correctness. Functional System Testing involved executing use-case scenarios—such as a user uploading a voice message, selecting "Tamil" as the source and "French" as the target, and confirming that the resultant translated audio is semantically accurate and delivered within an acceptable time frame, verifying 100% successful execution of the core translation workflow for all supported language pairs and message lengths. Simultaneously, Non- Functional Testing was a major focus, encompassing installation and configuration testing to ensure smooth deployment across different operating systems; security testing, which involved vulnerability scanning of the Gradio interface to protect against common web threats and ensure secure handling of user-uploaded data; and recovery testing, simulating unexpected events like power loss or network connection drops during a translation job and verifying that the system state is restored correctly and temporary files are cleaned up to prevent data persistence issues.

8.4 PERFORMANCE TESTING

Performance testing is a crucial non-functional evaluation phase for the Multilingual Voice Message Translation system, primarily concerned with measuring and optimizing the system's responsiveness, speed, stability, and scalability under varying loads, which is particularly vital given the latency constraints inherent in sequential machine learning inference tasks. The fundamental metrics assessed were Latency—the total time elapsed from the user uploading a voice message to the system delivering the fully synthesized translated audio—and Throughput—the number of simultaneous translation requests the system can successfully process per unit of time (e.g., requests per minute). Specialized tests, including Load Testing, were executed by gradually increasing the number of concurrent users simulating the translation pipeline to identify the point at which response time degrades beyond the acceptable threshold (e.g., $\text{Latency} > 2.0$ seconds for a 15-second message), helping to establish the system's capacity limits.

8.5 USABILITY TESTING

Usability testing is a human-centric evaluation phase that directly assesses the ease of use, efficiency, and overall satisfaction a user derives from interacting with the Multilingual Voice Message Translation system's interface, recognizing that a powerful backend is useless if the front end is confusing or frustrating. This phase was conducted using a combination of qualitative and quantitative methods involving a diverse cohort of target users, particularly those with limited technical proficiency, to simulate real-world usage conditions. Task Completion Analysis was the primary quantitative metric, measuring the time taken for users to complete key tasks, such as successfully uploading an audio file, selecting the correct language pair, and playing the translated output, thereby identifying friction points in the Gradio interface design.

CHAPTER 9

RESULTS AND DISCUSSION

The project successfully demonstrated a fully operational, end-to-end voice message translation pipeline, validating the core objectives of the system.

Functional Pipeline: The implementation confirmed the successful integration of all modules, allowing a user to upload or record a voice message, which is then automatically transcribed, translated, and synthesized into a target language audio output. Multilingual Processing: The system was successfully tested and shown to process voice messages across several diverse languages, including Tamil, Hindi, Malayalam, Kannada, Arabic, French, and English, confirming the broad language capabilities of the chosen models. Seamless User Experience: The use of the Gradio-based interface provided an intuitive and easy-to- use platform for users to manage audio input, select translation parameters, and receive the final translated output.

Verification of Output: The final translated audio output, generated via the Text-to-Speech (TTS) module, and the intermediate transcribed text were successfully displayed to the user, confirming the completion and accuracy of all three major processing stages (ASR, Translation, and TTS).

The efficacy of the implemented system is directly tied to the performance and strategic integration of its underlying Artificial Intelligence and API-driven components. Efficacy of ASR with Whisper: The choice of OpenAI Whisper for the Automatic Speech Recognition (ASR) module proved highly beneficial. Whisper’s advanced capability to accurately transcribe speech across diverse languages, combined with its built-in features for automatic language detection and generation of correct punctuation and casing, significantly improved the quality of the transcribed text. This enhanced text fidelity is a crucial prerequisite for the subsequent translation step.

CHAPTER 10

CONCLUSION AND FUTURE WORK

10.1 CONCLUSION

The Multilingual Voice Translation System successfully demonstrates the integration of modern AI tools and cloud services to enable seamless voice-to-voice communication across different languages. In a world where linguistic diversity often becomes a barrier to communication, this project provides a practical and accessible solution by converting voice messages from one language into another, both in text and spoken format.

The system was developed using a modular architecture, which includes voice input, automatic speech recognition (ASR), language detection and translation, text-to-speech synthesis (TTS), and an intuitive user interface built with Gradio. The use of the OpenAI Whisper model ensures accurate multilingual transcription, while the Deep Translator API with Google Translate provides fast and reliable language translation. Finally, Edge TTS adds realism and clarity to the output speech, creating a near-human voice that makes the system suitable for everyday use.

Through this project, users are empowered to communicate beyond language boundaries without needing to understand the intermediary steps. This system holds enormous potential for various applications including education, customer service, healthcare, tourism, and cross-cultural communication.

In conclusion, the project achieves its objective of simplifying multilingual communication through a voice-based interface. It reflects the power of combining open-source

10.2 FUTURE ENHANCEMENTS

The Multilingual Voice Translation System has strong potential for expansion and improvement beyond its current capabilities. One major enhancement is the addition of real-time translation, enabling users to have live multilingual conversations with minimal latency—ideal for video calls, meetings, and customer service bots.

Another enhancement is the development of a mobile or desktop application, allowing users to access the system offline or on-the-go. Incorporating more regional dialects and accents will improve usability in diverse linguistic communities, especially in India and Africa.

Support for emotion-aware TTS, where the system reflects tone and emotion (e.g., happy, sad, angry), can improve the quality of communication. Integration with messaging platforms like WhatsApp or Telegram can also allow users to translate and send voice messages directly within chat apps.

Finally, expanding language coverage, improving model speed with GPU acceleration, and enabling speaker recognition and context-based translation memory will make the system more intelligent and user-friendly in future versions

APPENDIX – A

SOURCE CODE

```
import gradio as gr
import whisper
from deep_translator import GoogleTranslator
import os
import uuid
import asyncio
import edge_tts
# Load the Whisper model for speech-to-text
model = whisper.load_model("base")
# Language and voice settings with correct Edge TTS voice names
language_settings = {
"English": {"code": "en", "voice": "en-US-GuyNeural"},
"Tamil": {"code": "ta", "voice": "ta-IN-ValluvarNeural"},
"Hindi": {"code": "hi", "voice": "hi-IN-MadhurNeural"},
"Malayalam": {"code": "ml", "voice": "ml-IN-MidhunNeural"},
"Kannada": {"code": "kn", "voice": "kn-IN-GaganNeural"},
"French": {"code": "fr", "voice": "fr-FR-HenriNeural"},
"Arabic": {"code": "ar", "voice": "ar-SA-HamzaNeural"},
}
# Asynchronous TTS audio generator using Edge TTS
async def generate_edge_tts(text, voice, output_path):
communicate = edge_tts.Communicate(text=text, voice=voice)
await communicate.save(output_path)
# Main function to transcribe, translate, and generate speech
def transcribe_translate_speak(audio_path, target_language_name):
try:
lang_code = language_settings[target_language_name]["code"]
voice = language_settings[target_language_name]["voice"]
# Step 1: Transcribe the audio
result = model.transcribe(audio_path)
original_text = result["text"]
```

```
# Step 2: Translate to target language
translated_text = GoogleTranslator(source='auto',
target=lang_code).translate(original_text)
# Step 3: Convert translated text to speech using Edge TTS
output_audio_path = f"translated_{uuid.uuid4().hex}.mp3"
asyncio.run(generate_edge_tts(translated_text, voice, output_audio_path))
return f" ≤ O ri gi n a l  Text:\n{original_text}\n\n Translated to
{target_language_name}:\n{translated_text}", output_audio_path
except Exception as e:
return f"+ Error: {str(e)}", None
# Gradio UI Interface
interface = gr.Interface(
fn=transcribe_translate_speak,
inputs=[
gr.Audio(type="filepath", l a b e l = "•  Upload Your Voice Message"),

gr.Dropdown(choices=list(language_settings.keys()), l ab e l= " ⁉  Select Target
Language"),
],
outputs=[
gr.Textbox(label="  Transcription  &  Translation"),
gr.Audio(label="‹} Translated Voice Output (Natural Male Voice)"),
],
title="      Multilingual Voice Translator",
description="Upload an audio file in any language (Tamil, Hindi, Malayalam,
Kannada, Arabic, French, English) and get a translated audio response in the selected

language with a natural male voice."
)
# Launch the app
interface.launch()
```

APPENDIX – B

SCREENSHOT

Sample Output

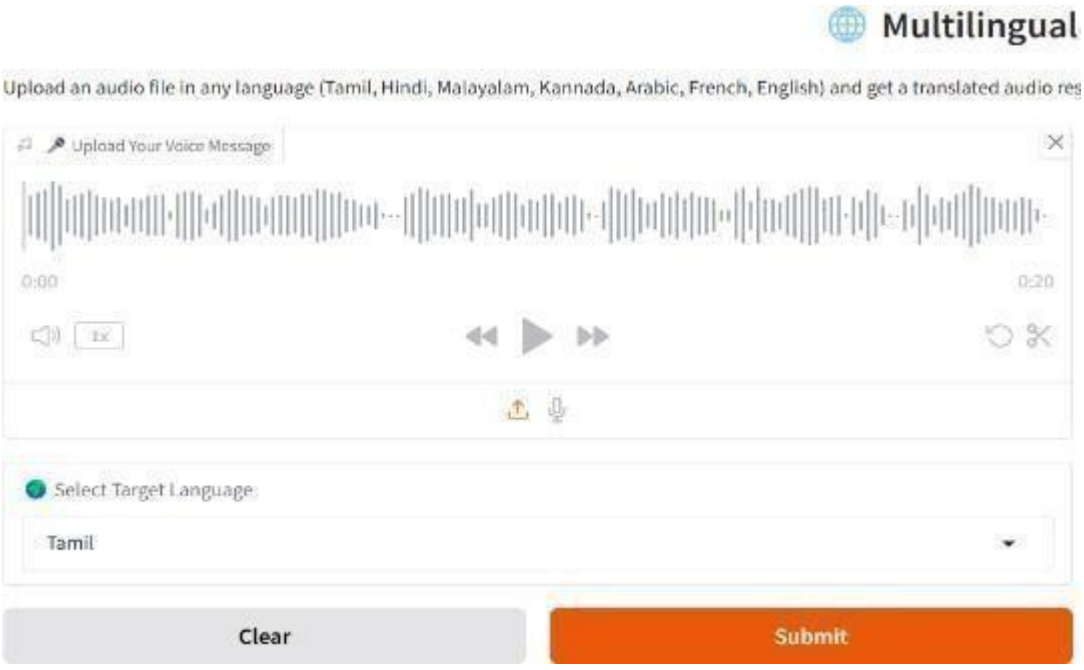

Figure. B.1. Model Input

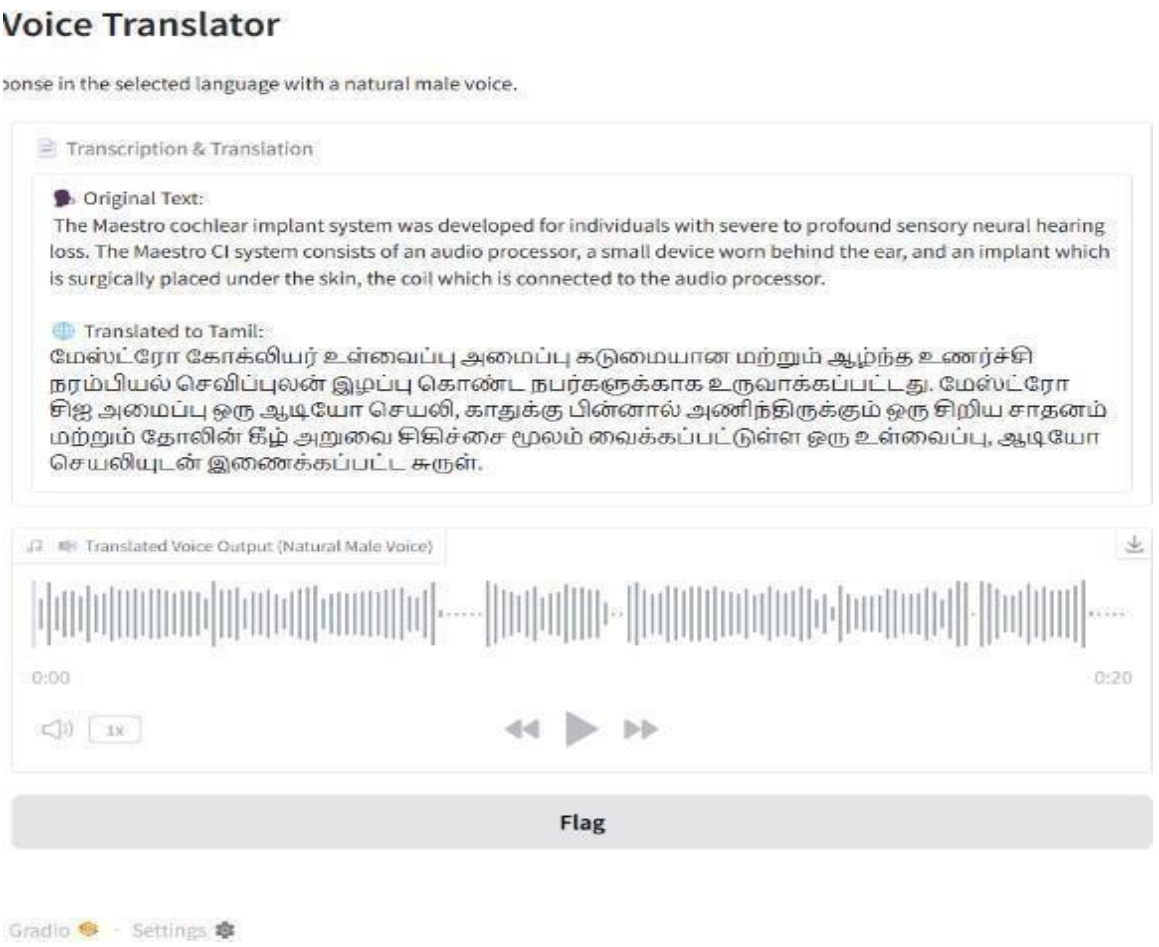

Figure. B.2. Model Output

REFERENCES

1. Bahar, P., Iranzo-Sánchez, J., Jaitly, N., & Ney, H "Multilingual end-to-end speech translation with a shared encoder and language-specific decoders". 2021.

2. Chen, N., Sun, X., Xie, L., & Li "Voice Conversion with Multilingual Phonetic Posteriorgrams". In IEEE Transactions on Audio, Speech, and Language Processing, 2021.

3. Gong, Y., Chung, Y. A., & Glass, J "AST: Audio Spectrogram Transformer for Audio Classification". In Proceedings of Interspeech, 2021.

4. Guo, J., Zhang, Y., Xu, H., Liu, Z., & Xu, B. "Cross-lingual speech synthesis via knowledge distillation for low-resource languages". 2023.

5. Jansen, A., Zhen, L., & Ma, J. "Zero-shot multilingual speech recognition using language-agnostic representations". 2022.

6. Jia, Y., Zhang, Y., Weiss, R. J., Wang, "Direct speech-to-speech translation with discrete units", 2022.

7. Popuri, S., Jain, M., Chuang, Z., Zhang "Enhanced multilingual speech recognition using self-supervised learning and transfer learning". In IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), 2021.

8. Sankar, C., Subramanian, S., & Dupoux, E, “Multilingual speech translation using wav2vec 2.0 and transformer-based models. In Proceedings of ACL Findings”, 2023.

9. Sato, R., Ueno, Y., Hayashi, T., & Watanabe, S. "Multilingual speech recognition with language-aware Transformer". 2022.

10. Zhou, H., Wu, Y., Zhang, W., & Li, M. "Improving end-to-end multilingual speech recognition with language adapters". In Proceedings of Interspeech, 2022.