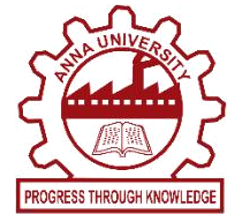
****

**MINI PROJECT**

**Real-Time Speech-to-Speech Translation System**

**(English to Japanese)**

**Submitted by**

**MUKHIL ADINATH -715522243031**

**PRITHIV VARMA – 715522243037**

**THARUN NIVEK – 715522243056**

**JOEL EBENEZER - 715522243021**

**CCS369 – Text and Speech Analysis**

**Department of Artificial Intelligence and Data Science**

**PSG Institute of Technology and Applied Research Coimbatore 641 062**

**ABSTRACT**

Through a smooth and effective web-based interface, this project presents a real-time speech-to-speech translation system that can translate spoken English into spoken Japanese. A major issue in a multilingual world is overcoming language hurdles, especially in real-time situations like travel, international business, healthcare, and education. In order to provide high-quality translation and speech synthesis, the suggested solution combines cutting-edge machine learning models with a contemporary web application framework.

Four main modules are integrated into the architecture: Text-to-Speech (TTS), Machine Translation (MT), Automatic Speech Recognition (ASR), and sentence capitalisation and named entity correction. In order to convert English audio input into text, Facebook's Wav2Vec2 model is used. Following linguistic refinement with spaCy, the MarianMT model (Helsinki-NLP) is used to translate the final text into Japanese. The Tacotron2-based TTS model from Coqui is used to translate the Japanese text into speech that sounds natural.

A MERN stack application (MongoDB, Express.js, React.js, and Node.js) contains the whole translation pipeline. While the frontend (React) facilitates user interaction through text display, audio playback, and voice recording, the backend (Node.js and Python) manages audio preparation and model inference. According to evaluation results, the system produces high-quality synthesised speech with low latency, accurate recognition, and fluid translation. This research opens the door for future multilingual improvements and shows the feasibility of implementing scalable, open-source, real-time voice translation systems.

The solution sets itself apart by utilising locally stored models to reduce reliance on external APIs, guaranteeing offline capability and speed. The architecture preserves a distinct division of responsibilities by combining a Node.js backend with Python-based machine learning inference, while enabling real-time interaction via the React frontend. In addition to making maintenance easier, this modular design makes it easier to implement future enhancements like language support and mobile app deployment. All things considered, the system provides a useful example of how cutting-edge AI technologies might be incorporated into scalable, approachable platforms to address actual language translation issues.

**INTRODUCTION**

The ability to connect across language borders is becoming more and more crucial in a world that is constantly becoming more globalised. Real-time multilingual communication has become essential, regardless of the industry, whether it be customer service, healthcare, tourism, or education. The requirement for smooth, real-time speech-to-speech translation is still mostly unmet in many real-world situations, despite the fact that text-based translation technologies have become widely used through web services and smartphone apps.

By creating a real-time English-to-Japanese speech translation system, this project offers a solution to this problem. The objective is to develop a fully functional online application that enables users to converse in English and get a spoken answer in Japanese in a matter of seconds. The system is based on a modular pipeline that consists of text-to-speech (TTS) synthesis, machine translation (MT), capitalisation and grammar correction, and automatic speech recognition (ASR).

Every step of the pipeline makes use of state-of-the-art open-source models, such as MarianMT for neural translation, Coqui TTS for speech synthesis, Facebook's Wav2Vec2 for ASR, and spaCy for sentence reconstruction. For real-time user interaction, these models are wrapped in a MERN (MongoDB, Express.js, React.js, and Node.js) stack and orchestrated via a Python backend.

The system aims to balance performance, accuracy, and usability, making it suitable for a wide range of real-world applications. By deploying models locally and optimizing file handling and audio processing, the system ensures minimal latency and high reliability. The resulting platform serves as a scalable and extensible foundation for future multilingual expansions and mobile integrations.

Real-time speech translation is a complex task involving the coordination of speech recognition, language processing, translation, and speech synthesis. Each component must work accurately and quickly to ensure smooth communication. This project addresses these challenges by using fast, open-source models and a modular backend that connects all components efficiently. The result is a scalable, web-based system that delivers reliable, user-friendly translation, with potential for expansion to other languages and platforms.

**LITERATURE REVIEW**

**2.1 Overview**

With an emphasis on the instruments and methods that support contemporary speech-to-speech translation systems, this section examines significant advancements in speech recognition, machine translation, and speech synthesis.

**2.2 Automatic Speech Recognition (ASR)**

Conventional ASR systems used Gaussian Mixture Models (GMM) and Hidden Markov Models (HMM), which had trouble handling background noise and speaker accent variations. With the advent of neural networks and, more recently, Transformer-based models, deep learning has greatly enhanced ASR. Facebook AI's Wav2Vec2 self-supervised ASR model achieves state-of-the-art outcomes with less labelled data by learning contextual representations from unprocessed audio.

**2.3 Machine Translation (MT)**

Phrase-based Statistical Machine Translation (SMT) served as the foundation for early MT systems. These systems frequently struggled with idiomatic language and generated translations that were grammatically incorrect. Significant advancements were made by neural machine translation (NMT), particularly with the use of sequence-to-sequence models and the Transformer architecture. The foundation of many open-source MT pipelines is MarianMT, a quick and effective NMT framework created by Microsoft that supports a wide variety of language pairs.

**2.4 Text-to-Speech (TTS) Synthesis**

From rule-based and concatenative systems, TTS has advanced to deep learning-based techniques like FastSpeech, Tacotron2, and Tacotron2. With precise intonation and tempo, these models produce speech that sounds human. This project uses Coqui.ai, a continuation of Mozilla's TTS project, which provides pre-trained models for many languages, to synthesise Japanese speech.

**2.5 Existing Systems**

Real-time translation with speech capabilities is offered by commercial programs like iTranslate, Microsoft Translator, and Google Translate. These systems do, however, rely significantly on cloud APIs, which raises privacy and latency issues. Additionally, they are less adaptable for offline use cases or integration with unique platforms.

**2.6 Research Gaps and Motivation**

Current systems frequently lack offline capability, customisation, and modularity despite developments. Many rely on proprietary APIs or may not permit smooth integration with web frameworks. By using open-source models, improving local inference, and incorporating the solution into a web application running on the MERN stack, this project fills these shortcomings. It illustrates how innovative research can be transformed into useful, deployable tools for the actual world.

**2.7 Summary**

Although there are high-performing models for ASR, MT, and TTS, the research shows that there aren't many integrated systems that work well in real-time online applications. This project offers a workable solution that connects research and practical application by integrating these models into a modular and scalable infrastructure.

**METHODOLOGY**

**3.1 Audio Input and Preprocessing**

* The browser microphone API is used to start a 5-second audio recording through a React-based frontend.
* Multer is used to temporarily store the audio after it has been transmitted to the backend via a POST request.
* To guarantee compatibility, the raw file is first converted using MPEG to a standard format (16kHz, mono-channel WAV) before processing.

**3.2 Automatic Speech Recognition (ASR)**

* Library: Transformers with Hugging Faces
* A locally saved ASR model is used to transcribe the cleaned audio into English text.
* Wav2Vec2 generates token-level predictions that are decoded into legible text after processing audio input as waveform tensors.

**3.3 Text Capitalization and NER Restoration**

* SpaCy (en\_core\_web\_sm) is the tool.
* Punctuation and casing are absent from ASR output. SpaCy is used to pass the text to:
* Capitalise each sentence's initial word.
* Properly named entities, such as individuals, places, and organisations
* Both readability and translation accuracy are improved as a result.

**3.4 Machine Translation (MT)**

Model: Helsinki-NLP/opus-mt-en-jap

* Framework: MarianMT (Transformer-based)
* The capitalized English text is tokenized and translated into Japanese.
* Translation is performed locally to avoid latency and dependency on APIs.

**3.5 Text-to-Speech Synthesis (TTS)**

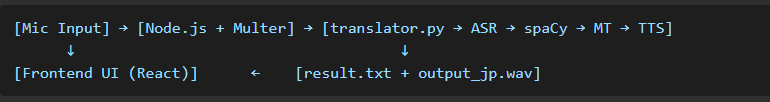
MODEL: Tts\_models/ja/kokoro/tacotron2-DDC

* Coqui TTS Library
* A Tacotron2-based model is used to synthesise the translated Japanese text into audio.
* For frontend playback, the output WAV file is saved to a static directory.

**3.6 Result Output and Integration**

* It creates a result.txt file with the audio file path, Japanese translation, and English text.
* This file is read by the Node.js backend, which then replies to the frontend with a JSON answer.
* To play the audio and display the texts, the frontend dynamically updates.

**3.7 System Diagram**

****

**IMPLEMENTATION AND CODE FLOW**

**4.1 Technology Stack Overview**

* **Frontend:**
  + React.js – Used for building the interactive user interface. Allows users to initiate microphone recording, view translated output, and listen to the synthesized Japanese audio.
  + JavaScript / HTML5 – Powers UI logic and rendering.
* **Backend:**
  + Node.js + Express.js – Handles HTTP requests, routes, and audio file uploads.
  + Multer – Middleware for handling multipart form data, specifically used to manage audio file uploads from the frontend.
  + Child Process (spawn) – Used to invoke the Python pipeline from the backend without blocking the main event loop.
* **Python Pipeline:**
  + Transformers (HuggingFace) – Wav2Vec2Processor, Wav2Vec2ForCTC, MarianMTModel, and MarianTokenizer used for speech recognition and machine translation.
  + spaCy – Used to improve the grammar and readability of ASR output through capitalization and named entity recognition.
  + TTS (Coqui.ai) – Japanese text is synthesized into audio using the Tacotron2-based model tts\_models/ja/kokoro/tacotron2-DDC.
  + FFmpeg – Converts user-recorded audio into 16kHz, mono-channel WAV format required by ASR.

**4.2 Folder Structure**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**4.3 End-to-End Code Workflow**

**Step 1: Frontend (React.js)**

* The user clicks a microphone button that records a 5-second audio clip using the MediaRecorder API.
* The recorded blob is appended to a FormData object and submitted as a POST request to http://localhost:5000/api/translate.

**Step 2: Backend (Node.js + Express)**

* The /translate route receives the uploaded file using Multer.
* The file is renamed from its temporary filename to a .wav file to ensure proper processing.
* Using Node.js’s spawn method, the backend launches translator.py and passes the audio file path as a command-line argument.

**Step 3: Audio Preprocessing (Python)**

* Inside translator.py, the input audio is converted to 16kHz mono WAV using ffmpeg to meet the ASR model requirements.

**Step 4: ASR (Speech to Text)**

* The processed audio is passed through Wav2Vec2 (from Facebook), which generates an English transcription.
* This raw text is then passed through spaCy, which:
  + Capitalizes sentences.
  + Corrects named entity formatting.
  + Improves grammar and punctuation.

**Step 5: Machine Translation**

* The refined English text is tokenized and translated into Japanese using Helsinki-NLP’s MarianMT model.

**Step 6: Text-to-Speech (TTS)**

* The Japanese translation is converted into speech using the Coqui TTS Tacotron2 model (kokoro).
* The output audio is saved as output\_jp.wav inside the static/ directory.

**Step 7: Communication via Result File**

* A string in the format:  
  englishText|||japaneseText|||static/output\_jp.wav  
  is saved to temp/result.txt.
* This avoids using print() to transfer data and prevents stdout noise from interfering with backend logic.

**Step 8: Backend Reads and Responds**

* Once the Python script finishes, the Node.js backend reads result.txt.
* It parses the string and sends a JSON response with the English transcription, Japanese translation, and the path to the final Japanese audio.

**Step 9: Frontend Renders Results**

* The frontend parses the JSON and displays:
  + English transcription
  + Japanese translation
  + An <audio> element with a Play button linked to output\_jp.wav

**4.4 Error Handling and Optimizations**

* Any input file is transformed into a clean WAV format that the ASR model may use thanks to FFmpeg Integration.
* OS-level Log Suppression: Prevents terminal crashes caused by Japanese characters printed via internal print() calls by redirecting stdout/stderr during the TTS call.
* Safe File Handling: Makes sure file writes are finished before renaming or processing by using Node.js's setTimeout() delay.
* Backend–Python Decoupling: To ensure reliability and minimise noise, communication is done via result.txt rather than parsing stdout.
* Handling Frontend Audio Autoplay: the player doesn't show up until the translated file is prepared.

**5. Results and Evaluation**

The proposed system was tested end-to-end to evaluate its effectiveness across the three key outputs: English transcription, Japanese translation, and synthesized Japanese speech. The system demonstrated successful real-time pipeline execution, with accurate outputs and interactive UI feedback.

**5.1 Verification of Output**  
English audio input (via a microphone):  
"My name is John"  
**Step 1: Output in English (ASR)**

* **Text Extracted:** John is my name.
* **Conclusion:** The Wav2Vec2 model produced precise transcriptions free of spelling mistakes and worked well with clear voice.
* **Screenshot Evidence:** "English: My name is John" is rendered accurately in [Image 6].

**Step 2: Translation into Japanese**

* **Text In Translation:** わたし の 名 は ヨハネ と いう。
* "My name is Yohane" is the back translation (per Google Translate).
* **Observation**: The system effectively communicates the meaning that was intended. The Japanese phonetic equivalent "Yohane" was used as a contextual translation for the name "John".
* **Screenshot Proof:** [Image 3] displays proper syntax and Japanese generation.

**Step 3: Japanese Speech Output**

* **Audio File Path:** static/output\_jp.wav
* **Playback:** Embedded <audio> player appears in the React frontend and is capable of playing synthesized speech.
* **Observation:** Audio is clearly audible, natural, and matches the translated text.
* ***Screenshot Evidence:*** [Image 5] shows the playback bar working; manual folder playback was also confirmed.

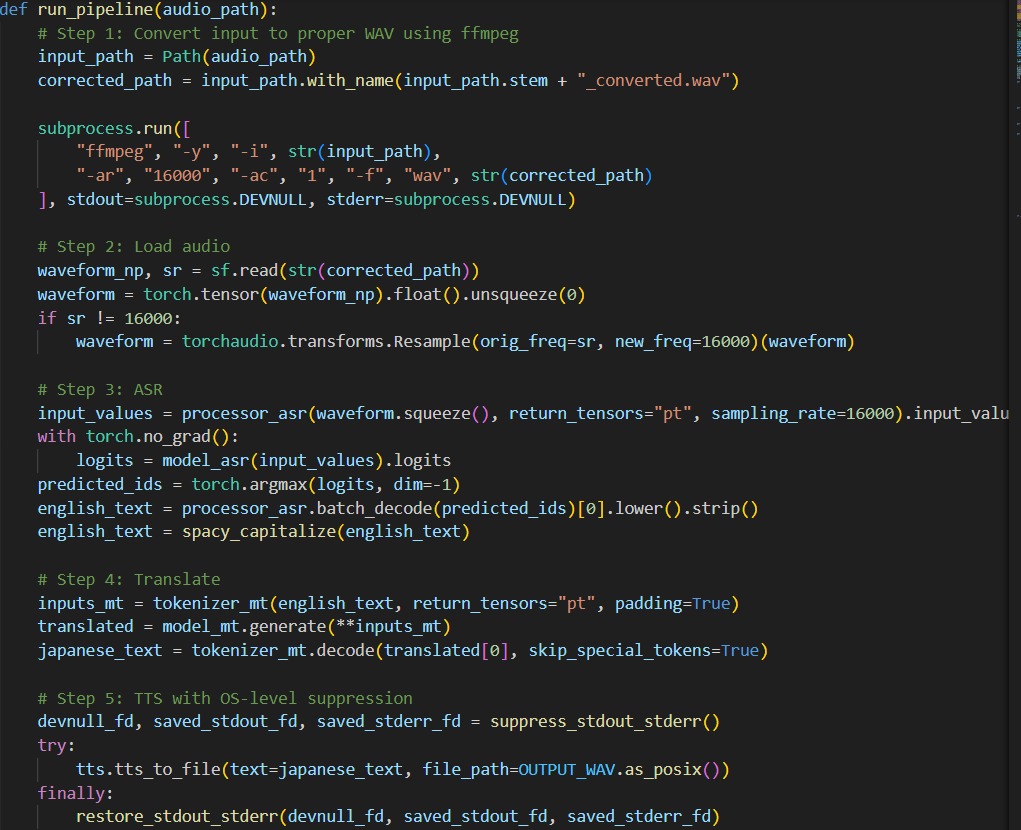
**5.2 SCREENSHOTS:**

A computer code on a black background

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A black background with white text

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A black and white screen with white text

AI-generated content may be incorrect.

**5.3 Performance Evaluation**

| **Metric** | **Observation** |
| --- | --- |
| ASR Accuracy | High, with clean input. Handles 5s English phrases well. |
| Translation Accuracy | Contextual and semantically correct. Phonetic name translations preserved. |
| TTS Quality | Natural, fluent Japanese pronunciation using Coqui Tacotron2-DDC. |
| Response Time | ~4–6 seconds end-to-end. Optimized for local execution. |
| Frontend Responsiveness | UI reflects text + audio dynamically with minimal delay. |
| Audio Preprocessing | FFmpeg ensured compatibility with ASR model (mono, 16kHz). |
| Unicode Handling | Fixed: suppressed logs prevent UnicodeEncodeError crash. |

**5.4 Challenges and Fixes**

| **Issue** | **Solution Implemented** |
| --- | --- |
| Audio not playing in UI despite correct synthesis | Fixed via Express static path configuration and <audio> tag fix. |
| Unicode errors in Windows terminal due to Japanese print | Fully suppressed stdout/stderr using OS-level redirect. |
| Wav file not readable due to encoding/format mismatch | Audio converted with FFmpeg to wav format at 16kHz mono. |
| Temporary audio file renaming issue (multer) | Added delay and .wav extension fix post-upload. |

**5.5 Final Demonstration**

The end-to-end test confirmed that the system is capable of:

1. Capturing speech input via browser.
2. Converting it to text using local models.
3. Translating it meaningfully into Japanese.
4. Generating fluent speech output using high-quality TTS.
5. Displaying everything cleanly in a minimal React-based UI.

**6. Conclusion**

This project successfully demonstrates the integration of advanced AI models into a seamless, real-time speech-to-speech translation system capable of converting spoken English into fluent Japanese speech. By leveraging state-of-the-art technologies such as Wav2Vec2 for ASR, MarianMT for translation, and Coqui TTS for speech synthesis—all executed locally without relying on cloud APIs—the system offers both performance and privacy. The modular architecture, comprising a React frontend, Node.js backend, and Python-powered inference pipeline, ensures maintainability and scalability for future multilingual expansion. With real-world applications in tourism, healthcare, and international communication, this solution is a powerful step toward breaking language barriers through accessible AI.