A project report on

REAL-TIME SPEECH-TO-SPEECH TRANSLATION

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning

by

VINEETHA C – 21BAI1358

RUEBEN VARGHESE PHILIP – 21BAI1817

SRIHARI GOPAL KARTHIKAYINI– 21BAI1811

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April, 2025

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I hereby declare that the thesis entitled "REAL-TIME SPEECH-TO-SPEECH TRANSLATION" submitted by SRIHARI GOPAL KARTHIKAYINI (21BAI1811), for the award of the degree of Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning, Vellore Institute of Technology, Chennai is a record of bonafide work carried out by me under the supervision of Dr. Ashoka Rajan R.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: 2nd April, 2025

[illegible]

Signature of the Candidate

Srihari Gopal Karthikayini

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled **“Real-time Speech-to-Speech Translation”** is prepared and submitted by **Srihari Gopal Karthikayini (21BAI1811)** to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning** is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide: R. Ashokarajan

Name: Dr. Ashoka Rajan R

Date: 2nd April, 2025

[illegible]	A. Aravindh.
Signature of the Examiner	Signature of the Examiner
Name: Dr. SUGUNA M	Name: A. ARAVINDHAN
Date: 16.4.25	Date: 16.04.2025

Approved by the Head of Department, [illegible]

B.Tech. CSE with Specialization in Artificial Intelligence and Machine Learning

Name: **Dr. Sweetlin Hemalatha C**

Date: 2/01/2025

(Seal of SCOPE)

VELLORE INSTITUTE OF TECHNOLOGY SCHOOL OF COMPUTER SCIENCE & ENGINEERING (SCOPE) Chennai 600127

ABSTRACT

This project showcases the need and working of a real-time speech-to-speech translation (S2ST) system with three fundamental components: Automatic Speech Recognition (ASR), Machine Translation (MT), and Text-to-Speech (TTS) synthesis. Each of these modules is independently developed and pipelined to allow for free-flowing, real-time translation among Hindi, Tamil, and English languages.

The ASR module recognizes spoken speech as text via a pre-trained transformer-based model trained on the Common Voice corpus. The MT module converts the transcribed speech to the target language with a fine-tuned MT model that is end-to-end trained on Opus dataset. The TTS module, finally, converts the translated speech to natural speech with a neural synthesis model with the help of datasets: LJ Speech and IndicTTS.

The pipelined model is used to allow real-time applications to provide fast and uniform translation. WER (Word Error Rate) assesses ASR performance, BLEU evaluates MT performance, and normal human speech understanding by listener evaluates TTS performance. The solution shows the power of deep learning to facilitate real-time cross-lingual communication and can be applied in customer service, healthcare, tourism, and multilingual conferences.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Ashoka Rajan R, Assistant Professor Senior, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with he is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Image Processing and Machine Learning.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor, Dr. T. Thyagarajan Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Ganesan R, Dean, Dr. Parvathi R, Associate Dean Academics, Dr. Geetha S, Associate Dean Research, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant state, I express ingeniously my whole-hearted thanks to Dr. Sweetlin Hemalatha C, Head of the Department, B.Tech. CSE with Specialization in Artificial Intelligence and Machine Learning and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staffs at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: 2nd April, 2025 **Srihari Gopal Karthikayini**

CONTENTS

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

S2ST – Speech-to-Speech Translation

ASR – Automatic Speech Recognition

MT – Machine Translation

TTS – Text-to-Speech

HMM – Hidden Markov Model

GMM – Gaussian Mixture Model

RNN – Recurrent Neural Network

LSTM - Long Short-Term Memory

MFCC – Mel-Frequency Cepstral Coefficients

CTC – Connectionist Temporal Classification

WER – Word Error Rate

BLEU – Bilingual Evaluation Understudy

Chapter 1

Introduction

1.1. CRITICAL FACTOR

In the modern globalized world, the option of effective cross-lingual communication is an indispensable factor for the smooth interaction of individuals who speak different languages. Such language barriers have led to the fact that people have ceased to unite, to perform many tasks together and to establish effective communication. The problem of understanding a foreign language is relevant in the field of medicine, customer support, foreign economic activity, and education. After digital communication platforms became widespread, the need for instantaneous translation has become especially acute.

The most promising approach to such problems is speech-to-speech translation (S2ST). This technology allows converting spoken language into another spoken language. S2ST technology enables us to enhance accessibility and inclusiveness by overcoming the language barrier. Unlike normal text-based translation, speech-to-speech translation goes through three parts in the process: ASR (automatic speech recognition), MT (machine translation) and TTS (text-to-speech). Each stage requires high accuracy, speed, and fluency to guarantee effective real-time communication.

In the case of real-time conversations, a slightly delayed or wrong answer will lead to problems with understanding and the dialogue in general. That is why the development of a robust and efficient S2ST system implies the usage of high-quality neural machine translation models that have a low latency and inference pipeline optimization to ensure low latency and high quality.

1.2. PROBLEMS WITH SPEECH-TO-SPEECH TRANSLATION

Even with substantial progress in natural language processing (NLP) and speech synthesis, speech-to-speech translation is still a hard problem because of the following reasons:

a) Speech Pattern Variability:

Speech has natural variation, like accents, pronunciation types, and other interruptions. ASR systems must deal with these variations while keeping transcription accurate.

b) Noisy and Poor-Quality Audio:

Spontaneous audio from the real world tends to have background noise, speech overlap, and poor articulation, posing challenges for ASR models to produce good transcriptions.

c) Context and Semantics:

Converting spoken words is not just word-level mapping but maintaining context, tone, and semantics. Word-to-word translation usually leads to poor or unnatural phrasing. Idiomatic expressions or culturally relative phrases, for instance, are lost when literally translated.

d) Real-Time Constraints:

Real-time S2ST needs low latency and high-speed inference to not interfere with communication. Speed and accuracy are hard to balance since complicated translation models need more processing time.

e) Multilingual Support and Tokenization:

Handling multiple languages comes with the challenge of dealing with varying grammatical structures, word orders, and tokenization approaches. Hindi and Tamil, for example, are morphologically rich languages that demand specific tokenizers to maintain linguistic precision.

f) Speech Synthesis Challenges:

The TTS module needs to produce natural speech with correct wordings, and rhythm. Producing realistic speech in real-time with fluency is a technical challenge, especially for low-resource languages such as Tamil.

1.3. SOLUTION

To address these challenges, this project implements a three-stage real-time S2ST pipeline:

a) Automatic Speech Recognition (ASR):

The ASR component uses a Transformer-based model trained on the Common Voice dataset. It transcribes spoken language into text with high accuracy, even in noisy conditions.

b) Machine Translation (MT):

The MT model translates the transcribed text into the target language using a custom Transformer architecture, trained from scratch on datasets such as OpenSLR SLR64 (Hindi), OpenSLR SLR66 (Tamil), and LJSpeech (English).

It uses Indic tokenizers for Hindi and Tamil, ensuring efficient tokenization and language-specific handling.

c) Text-to-Speech (TTS) Synthesis:

The TTS component converts the translated text into natural speech using a neural TTS model. It generates expressive and fluent speech with proper prosody and rhythm, ensuring a natural listening experience.

Chapter 2

Literature Survey

2.1. INTEGRATING SPEECH RECOGNITION AND MACHINE TRANSLATION

This paper explores integrating Speech-to-Text (STT) and Machine Translation (MT) for Arabic/English translation in broadcast news and conversational speech. It addresses challenges like sentence boundary detection (SBD), punctuation restoration, STT accuracy, and spoken number conversion. The STT/MT system follows a pipeline approach using 1-best STT output and evaluates performance with Translation Edit Rate (TER).

Experiments show that improving MT has a greater impact on TER than enhancing STT accuracy. Optimizing segmentation via automatic audio parameters and using Hidden Event Language Models (HELM) for SBD improves translation quality. Spoken number conversion slightly reduces TER. Tuning MT decoding weights on broadcast news data enhances accuracy, with tuning on STT output yielding comparable results to reference transcript tuning.

2.2. SPEECH RECOGNITION AND MACHINE TRANSLATION USING NEURAL NETWORKS

This paper explores end-to-end English-Russian speech-to-text translation using deep recurrent neural networks (RNNs), specifically Long Short-Term Memory (LSTM) models, aiming to develop a direct translation system without intermediate speech recognition. Models were trained on English audiobooks with Russian translations using TensorFlow and NVIDIA GPUs. The system involves three key processes: modeling (Encoder-Decoder LSTM networks), learning (bilingual dataset training with parameter optimization), and searching (beam search for optimal translation). The Adam optimizer performed best. Text-to-text models used WMT16 datasets, while speech-to-text models were trained on *The Adventures of Huckleberry Finn* with MFCCs for audio features. Results showed that while text-to-text models performed well, speech-to-text models struggled with overfitting due to limited data.

2.3. A HAND-HELD SPEECH-TO-SPEECH TRANSLATION SYSTEM

This paper presents a handheld speech-to-speech translation system deployed on a PDA running embedded Linux, translating spoken English to Mandarin Chinese in near-real time. The system mirrors its desktop counterpart, MASTOR, with optimizations for resource-constrained hardware. It includes a Hidden Markov Model (HMM)-based speech recognizer, a decision-tree-based natural language understanding (NLU) module, a maximum entropy natural language generation (NLG) engine, and a formant-based text-to-speech (TTS) synthesizer, chosen for its small memory footprint and unlimited vocabulary support. To address the lack of floating-point hardware on the ARM architecture, the system uses extensive integerization, reducing resource consumption while maintaining accuracy. Evaluations in dictation and medical domains show an average word error rate (WER) of 12.98%, with translation speeds of 1-4 seconds per input.

2.4. THE ATR MULTILINGUAL SPEECH-TO-SPEECH TRANSLATION SYSTEM

This article introduces the ATR multilingual S2ST system for real-time English-Japanese-Chinese translation, comprising speech recognition, machine translation, and text-to-speech synthesis. The system uses a 600,000-sentence multilingual corpus for statistical learning. The speech recognition module employs Successive State Splitting (SSS) and advanced language modeling techniques for better accuracy. The machine translation module combines example-based and statistical approaches with a multi-engine framework for optimal output. The XIMERA TTS system uses large speech corpora and HMM-based prosody modeling for natural-sounding synthesis. Evaluations with the Basic Travel Expression Corpus (BTEC) and MT-assisted dialogs (MAD) show high translation quality (TOEIC score of 750), though conversational speech presents challenges. Future improvements include handling longer sentences, normalizing dialogue, and adding error filtering measures. The study highlights the feasibility of combining corpus-based statistical methods with multi-engine translation for multilingual S2ST.

2.5. USING VOICE-TO-VOICE MACHINE TRANSLATION TO OVERCOME LANGUAGE BARRIERS IN CLINICAL COMMUNICATION

This research explores the use of voice-to-voice machine translation (MT) in clinical settings to bridge language gaps, conducted across 60 consultations in 18 languages at Geneva University Hospitals. Microsoft Translator (MST) was primarily used after technical issues with Pocketalk W hindered full evaluation. The study found that 82.7% of consultations met their purpose, though only 53.8% of healthcare practitioners were satisfied with communication quality, compared to 86% of patients who found MT-facilitated communication easy. Both practitioners (73%) and patients (84%) were willing to reuse MT. Translation accuracy was higher for European languages, while disfluent speech and hybrid languages reduced performance. The study highlights limitations, such as technical issues, small sample size, and language bias, but suggests MT as a viable alternative when interpreters are unavailable for straightforward consultations.

2.6. REAL TIME VOICE TRANSLATOR

This work introduces the making a Real-Time Voice Translator (RTVT), a system that combines Automatic Speech Recognition (ASR), Machine Translation (MT), Text-to-Speech (TTS), and Natural Language Processing (NLP) into one pipeline. Developed with Python, TensorFlow, and cloud services, RTVT provides scalability and low-latency processing.

The RTVT pipeline works through three stages: ASR translates spoken words to text, MT translates the text with deep learning models, and TTS produces the translation into natural speech keeping intonation and rhythm intact. The system allows real-time multilingual communication to be used across voice assistants, remote communication, healthcare, business, and education.

Challenges are latency, accuracy degradation in low-resource languages, difficulty in maintaining emotional expressiveness, and diminished ASR performance in noisy conditions, all of which affect overall translation quality.

2.7. REGIONAL LANGUAGE TRANSLATOR USING NEURAL MACHINE TRANSLATION

This paper presents the development of a regional language translator using Neural Machine Translation (NMT), offering advantages over Statistical Machine Translation (SMT) by employing neural networks for end-to-end sentence translation and enabling automatic voice output. The methodology includes data preprocessing, tokenization, padding, one-hot encoding, and an encoder-decoder model with an attention mechanism for improved accuracy. Training uses the teacher forcing technique for faster convergence. A voice translation module using GTTS provides real-time speech output for applications in conversational aids and accessibility. Results show the NMT model outperforms Google Translate, achieving 98% accuracy and strong performance on regional language pairs, expanding its potential applications.

2.8. REAL TIME MACHINE TRANSLATION SYSTEM FOR ENGLISH TO INDIAN LANGUAGE

This paper proposes a Machine Translation (MT) system to overcome communication barriers in India due to linguistic diversity and limited English literacy. The system processes real-time English speech, translates it into an Indian language, and outputs the translation as speech, using three modules: speech-to-text (STT), text-to-text translation (MT), and text-to-speech (TTS). A 2-layer LSTM network with an encoder-decoder architecture handles sequential data. Trained on a parallel Hindi-English corpus, the system achieves accurate translations, outperforming rule-based and statistical MT systems. The results indicate faster and more accurate translations, making the system suitable for real-world applications in government services, healthcare, and education. The authors suggest expanding datasets to improve performance and support more Indian languages.

2.9. SPEECH TRANS: AND EXPERIMENTAL REAL-TIME SPEECH-TO-SPEECH TRANSLATION SYSTEM

The SPEECHTRANS project, developed at Carnegie Mellon University, is a real-time speech-to-speech translation (S2ST) system designed for noisy, speaker-independent speech. First demonstrated in 1988, it features a modular architecture with speech recognition hardware, a phoneme-based parser, a natural language generator (GENKIT), and a speech synthesis module (DECTALK). The system's key components include phoneme extraction, error-correcting parsing, and a confusion matrix for optimal hypothesis selection. GENKIT ensures consistency in translation, and DECTALK converts text to speech. Despite handling noisy speech well, the system faces challenges in recognition accuracy and pragmatic understanding, requiring further improvements for practical deployment. Public demonstrations showed the system’s feasibility, but further refinements are needed before it can be widely applied.

2.10. POLYVOICE: LANGUAGE MODELS FOR SPEECH TO SPEECH TRANSLATION

The PolyVoice framework introduces a language model-based approach for speech-to-speech translation (S2ST), improving translation quality and audio naturalness while preserving speaker characteristics. Unlike traditional S2ST systems, PolyVoice translates both written and unwritten languages using discretized speech units from unsupervised learning. Its architecture includes a speech-to-unit translation (S2UT) model and a unit-to-speech (U2S) synthesis model, utilizing VALL-E X for speech synthesis to maintain speaker style. The system bypasses intermediate text representation, using a decoder-only model for direct translation. Experimental results show superior speech quality, with a 3.9 BLEU point improvement over encoder-decoder models, and strong performance in multiple speech processing tasks, including ASR, ST, MT, and TTS.

2.11. REAL TIME DIRECT SPEECH-TO-SPEECH TRANSLATION

This paper presents a real-time speech-to-speech translation (S2ST) system that translates voice input into text and speech in the target language, incorporating prosody and emotion for a more natural and expressive translation. Built with Python, Spyder, and Google Translate API, the system features four modules: Login, Input, Translation, and Output. The Input module captures speech, the Translation module uses Google Translate, and the Output module displays translations in text and voice. The system supports both administrators and end users via a GUI. Built on Windows 10+ with OpenCV and Pillow, it uses speech recognition for text conversion, speeding up the process by bypassing transcription. Results show accurate, real-time translations, with scalability for additional languages, though accuracy relies on Google Translate and the speech recognition module.

2.12. LEARNING TO TRANSLATE IN REAL-TIME WITH NEURAL MACHINE TRANSLATION

This paper presents a neural machine translation (NMT) framework for simultaneous translation, where an agent decides in real-time whether to READ the next source word or WRITE a translated word. This interleaved approach balances translation quality and delay. The framework uses reinforcement learning with a reward function considering both BLEU score and translation delay, measured by Average Proportion (AP) and Consecutive Wait (CW). The architecture includes an RNN-based agent interacting with a pre-trained NMT system, using an attention-based decoder. Experiments with English-Russian and English-German pairs show that the system outperforms segmentation-based methods and NMT baselines, achieving higher quality with lower latency, though it struggles with long-distance dependencies. The framework offers a solution for real-time translation.

2.13. EVALUATING AUTOMATIC SPEECH RECOGNITION IN TRANSLATION

This article evaluates the use of Automatic Speech Recognition (ASR) in the translation process to improve efficiency and assist human translators. IBM Bluemix ASR and Google Translate ASR are compared across French, Spanish, and Serbian audio files, measuring Word Error Rate (WER) and its impact on human translation productivity. The study shows that ASR-enhanced workflows reduce time and effort for transcription and translation, making the process faster. Three workflows are tested:

- Task 1: ASR + Human Translation (BASIC Correction) – ASR output is manually corrected and translated by humans.
- Task 2: ASR + Machine Translation (FULL Correction and PEMT) – ASR output is corrected by humans, then machine-translated and post-edited.
- Task 3: Speech-to-Text Translation (PEST) – Direct end-to-end speech-to-text translation, skipping manual transcription correction, making it the most efficient.

Results show Google ASR outperforms IBM ASR, achieving about 10% lower WER for French, reducing transcription errors and human correction effort. The PEST workflow is the fastest, while Task 1 is the slowest due to its labor-intensive nature. Task 2 is moderately efficient.

2.14. SPEECH TO SPEECH TRANSLATION: CHALLENGES AND FUTURE

This paper discusses the state of art, challenges, and outlook of Speech-to-Speech Translation (S2ST) technology with respect to its capability of bridging the language gap and facilitating global communication. It is pointing out development and challenges that come with language differences, inadequate data, and system limitations.

S2ST finds everyday applications in travel guiding, lecture interpretation, disaster response, and film dubbing. Nonetheless, difficulties such as dealing with noisy situations and making speech sound natural continue to plague the system. The architecture of the system includes three primary modules: ASR (speech to text conversion), MT (text translation to target language), and TTS (synthesizing speech from translated text with maintained prosody for natural speech).

Shortcomings of S2ST are sequential ASR and MT processing leading to latency, defects in ASR and MT due to noisy input and colloquial speech, language difference between languages, data insufficiency for low-resource languages, challenges with unwritten languages, inability to capture context, emotion, and intent, and copyright concerns due to large corpora.

Evaluation techniques for S2ST include BLEU, NIST, and WER scores for translation correctness, TOEIC scores for usability, and human evaluations for speech quality and comprehension.

Some directions for future S2ST research include large-scale field and social testing, enhancing conversational speech models for disfluencies and idioms, developing dynamic language models, standardizing interfaces for cross-platform support, extending support for additional languages (including low-resource ones), and solving privacy and copyright issues.

Chapter 3

Proposed Methodology

3.1. PROPOSED SPEECH-TO-SPEECH TRANSLATION SYSTEM

The real-time speech-to-speech translation framework is made of comprised fundamental models: Automatic Speech Recognition (ASR), Machine Translation (MT), and Text-to-Speech Synthesis (TTS). The translation methodology follows to a systematic pipeline to translate the input speech and hence linguistic conversion among English, Hindi, and Tamil feasible.

For a direct translational procedure, such as English to Tamil, the system initially employs Automatic Speech Recognition (ASR) to transcribe the verbal English input into textual format. The Machine Translation (MT) module subsequently coverts this English text into Tamil. Finally, the Text-to-Speech (TTS) module transforms the translated Tamil text into synthetic speech output.

For a more complex scenario like Hindi to Tamil translation, an intermediary translational phase is incorporated. Initially, ASR converts Hindi speech input into Hindi textual representation – transcribing the speech. Given that English is utilized as a mediating language, the MT module first interprets Hindi text into English. Subsequently, another MT process translates the English text into Tamil. The final Tamil text output is then processed by the TTS module to produce the Tamil speech output.

This multi-tiered methodology ensures feasible translations, mitigating potential loss of significance and enhancing fluency. The system is engineered to facilitate real-time communication across 3Indian languages, with minimal processing delays.

3.2. WORKING OF THE PROPOSED SYSTEM

3.2.1. ASR – AUTOMATIC SPEECH RECOGNITION SYSTEM

Automatic Speech Recognition (ASR) is a key component of speech-to-speech real-time translation systems, to be used initially in the process of translation. The main task of ASR is to transform spoken words into text for later processing by the Machine Translation (MT) component. Being aware of the richness of human speech, including variation in pronunciation, accents, dialects, and ambient noise, ASR should be designed with robust preprocessing, feature extraction, and modeling procedures to provide very high accuracy and reliability for transcribing.

The success of ASR is indicated by its success in accurately recognizing speech while capturing contextual meaning. The early use of statistical approach and Hidden Markov Models (HMMs) in classical ASR has been replaced by deep learning, like transformer-based models that have greatly increased the accuracy in speech recognition to identify long-range dependencies in speech. This section outlines the modules' architecture used in ASR, including for audio preprocessing, feature extraction, acoustic modeling, language modeling, and post-processing.

The structure of a basic and conventional ASR model comprises four primary blocks: (1) Audio Preprocessing, (2) Feature Extraction, (3) Acoustic and Language Modelling and (4) Post-processing.

Historical ASR systems employed Hidden Markov Models (HMMs) and Gaussian Mixture Models (GMMs) to identify patterns of speech. The models employed statistical probabilities to transcribe speech features into text.

However, there were limitations:

- Poor handling of background noise
- Difficulty recognizing different accents and pronunciations in the speech
- Limited vocabulary (data) and context awareness.

Deep Learning advancements led to the evolution of ASR models using neural networks, such as:

- Recurrent Neural Networks (RNNs): Improved sequential speech processing but had difficulty with long audio sequences.
- Long Short-Term Memory (LSTM) Networks: Improved information retention over longer speech durations compared to RNNs, but were computationally expensive.
- Transformer-based Models: These models include self-attention mechanisms that capture short and long-term speech dependencies, thereby improving accuracy and speed.

The current generation ASR models, such as Wav2Vec, Whisper, and DeepSpeech, use these advances to achieve cutting-edge speech recognition.

Wav2Vec (Facebook AI) employs self-supervised learning, which means it derives speech representations from unlabeled audio. Fine-tunes effectively with labeled datasets (such as Mozilla Common Voice). Performs well in noisy environments and with various accents.

Whisper (OpenAI) is a large-scale, multilingual automatic speech recognition model trained on 680,000 hours of diverse data. Ideal for real-time transcription, translation, and multilingual speech processing. Handles low-resource languages more effectively than many ASR models.

DeepSpeech (Mozilla) uses Recurrent Neural Networks (RNNs) and LSTMs. Although open-source and lightweight, models based on transformers are more efficient. Suitable for simple ASR applications with limited computational resources.

Out of these three models, we decided to choose Wav2Vec for this model, because it excels at fine-tuning with small labelled datasets, making it highly adaptable. It can perform self-supervised learning that enables it to recognize diverse speech patterns without large amounts of transcribed data.

It is also capable of integrating the Mozilla Common Voice seamlessly, ensuring the compatibility with multilingual and diverse datasets.

The fine-tuned Wav2Vec ASR model designed for the translation system, consists of four components:

- Audio Preprocessing:
    - Converts input audio to a standardized 16 kHz sample rate.
    - Applies noise reduction to remove unwanted background sounds.
    - Uses padding for consistency in short audio samples.
- Feature Extraction:
    - Extracts important speech characteristics using:
        - Mel-Frequency Cepstral Coefficients (MFCCs) for frequency-based analysis.
        - Mel-Spectrograms, which represent the audio as a time-frequency graph.
    - This helps the model understand phonetic and acoustic patterns efficiently.
- Language Modeling (Fine-tuning and Training):
    - The model uses Wav2Vec2CTCTokenizer, Wav2Vec2FeatureExtractor, and Wav2Vec2Processor to prepare audio for training.
    - Implementation of CTC loss (Connectionist Temporal Classification), which allows the model to map speech features to words without strict alignments.
    - The model is fine-tuned using Mozilla Common Voice dataset, adjusting weights for better accuracy.
    - The model is trained with AdamW optimizer and mixed precision (fp16) for faster convergence.

- Post-processing:
    - Removes filler words, background noise, and redundant characters.
    - Applies basic grammar corrections to enhance transcription readability.
    - Uses WER (Word Error Rate) metric to evaluate performance and refine the model.

3.2.2. MT – MACHINE TRANSLATION SYSTEM

Machine Translation (MT) is also a critical component in real-time speech-to-speech translation systems, with the role of translating text from one language into another maintaining semantic meaning and contextual accuracy. The system proposed here is oriented toward multilingual translation across English, Hindi, and Tamil, and supports bidirectional translation for fluent communication across languages.

Standard MT systems used rule-based and statistical models, which were unable to capture long and complicated sentence structures, idiomatic expressions, and contextual nuances.

Modern Machine Translation models:

- Sequence-to-sequence (Seq2Seq) models: They help in improved translation by encoding the source sentence into a context vector and decoding it into the target language. Better than traditional ones.
- Transformer-based models: They have self-attention mechanisms that captured short and long-term dependencies, enhancing translation accuracy.
- Multilingual models: Like mBART and M2M-100, capable of translating between multiple languages simultaneously.

The approach of deep learning and transformer-based architectures has significantly enhanced translation quality by capturing long-term dependencies and context effectively.

This system utilizes the mBART-large-50 model, a multilingual transformer trained on 50 languages, fine-tuned specifically for English-Hindi and English-Tamil translation and vice-versa tasks.

The system for MT consists of four key components:

- Data Preprocessing

- The very first step in the MT pipeline is data preparation and preprocessing, ensuring that the input text is properly formatted and tokenized for the pre-trained model.
- Preprocessing steps:
    - Dataset selection:
        - The system uses OPUS100, a publicly available multilingual parallel dataset, containing sentence pairs for English-Hindi and English-Tamil translation.
    - Tokenization:
        - The mBART tokenizer is applied to split the text into subword tokens.
        - Language-specific tokens (<en_XX>, <hi_IN>, <ta_IN>) are appended to indicate the source and target languages.
    - Padding and truncation:
        - Sentences are padded or truncated to 128 tokens for consistency in batch processing.

- Model Architecture

- The multilingual machine translation (MT) system is built on the mBART-large-50 model; a transformer-based architecture designed for sequence-to-sequence (Seq2Seq) language generation. It is fine-tuned on OPUS100, a multilingual dataset, to perform bidirectional translation between English, Hindi, and Tamil.

- Key features of the mBART model:
    - 50-language support: Pre-trained on a large multilingual dataset.
    - Self-attention mechanism: Captures both local and global dependencies.
    - Beam search decoding: Ensures fluency and coherence in translated sentences.

- Fine-tuning and training

- The pre-trained mBART-large-50 model is fine-tuned on the OPUS100 dataset to adapt it for English-Hindi and English-Tamil and vice-versa translation tasks.
- Training process:
    - Model initialization:
        - The mBART-large-50 model is loaded with pre-trained weights.
        - The tokenizer is configured with source and target language codes.
    - Fine-tuning parameters:
        - Learning rate: 3e-5 with AdamW optimizer.
        - Batch size: 8 sentences per batch with gradient accumulation to stabilize updates.
        - Mixed precision (fp16): Reduces memory usage and speeds up training.
    - Training strategy:
        - Gradient accumulation: Combines gradients from multiple batches before updating the weights.
        - Epoch-based evaluation: The model is evaluated on a validation set after each epoch.
    - Checkpoint saving:
        - Model checkpoints are saved periodically to prevent data loss.

3.2.3. TTS – TEXT-TO-SPEECH SYSTEM

Text To Speech Synthesis is a crucial part of real time speech to speech translation systems and is the last step in the translation. The chief responsibility of TTS is to translate text into the mechanical sound derived from plain, fluent, and understandable speech which a listener can easily hearing. Speech generation is quite complex and highly demanding advanced linguistic processing, acoustic modelling and waveform generation techniques for TTS to produce high quality synthetic speech.

The effectiveness of TTS is defined by how close it can get its speech to sounding human in terms of pronunciation, rhythm and expressiveness. Typically, concatenative and statistical parametric synthesis of speech was used in the conventional TTS systems, and more recently, with the development of deep learning especially neural vocoder and sequence to sequence models, we achieved a great improvement in the speech quality by capturing the complex linguistic and acoustic patterns. In this chapter, the architecture of the TTS module is described including preprocessing for text, the linguistic features at utterance, acoustic modeling, and generation of the waveform.

The architecture of a simple and traditional TTS model consists of four main components:

- Text Preprocessing
- Linguistic Feature Extraction
- Acoustic Modeling
- Waveform Generation

Traditional TTS systems generate speech through Concatenative Synthesis and Statistical Parametric Speech Synthesis (SPSS). These models had several limitations:

- Concatenative Synthesis: Pre-recorded speech segments were used, but flexibility and naturalness were not sufficiently met

- SPSS (HMM-based TTS): Used Hidden Markov Models (HMMs) to generate speech from statistical data, but frequently produced robotic-sounding speech.

Deep learning advancements led to the evolution of TTS models using neural networks, such as:

- Recurrent Neural Networks (RNNs): Used for sequence modeling, but had issues with long-term dependencies.
- Long Short-Term Memory (LSTM) Networks: Increased speech synthesis quality but were computationally expensive.
- Sequence-to-Sequence (Seq2Seq) Models: Added attention mechanisms to improve context retention, but required large datasets.
- Transformer-based TTS models used self-attention mechanisms to produce high-quality, natural speech while increasing efficiency.

Advances were utilized by modern TTS models such as Tacotron 2, FastSpeech and VITS to establish the state of the art in speech synthesis:

- Tacotron 2 (Google) is a sequence-to-sequence model with attention mechanisms that implements the WaveNet vocoder to produce high-quality speech. However, its inference is slow because of sequential processing.
- FastSpeech (Microsoft): A non-autoregressive model that significantly accelerates speech synthesis by predicting phoneme duration, thereby reducing training and inference time.
- VITS (Variational Inference TTS) is a generative model based on normalizing flows and variational autoencoder to yield highly expressive and natural sounding speech.
- ParlerTTS: A fast, efficient, and high-quality neural TTS model designed for real-time applications. It uses diffusion-based speech synthesis, which which increases voice naturalness, stability, and efficiency over traditional transformer-based voices.

ParlerTTS is chosen for this system because it has good quality, efficient speech synthesis capabilities. It is a neural vocoder-based diffusion TTS model that removes common artifacts from the neural vocoder generated speech, while keeping high inference speeds.

The fine-tuned ParlerTTS-based TTS model designed for the translation system consists of four components:

- Data Preparation:
    - Dataset that contains high quality .wav audio files and corresponding transcripts in their respective languages.
- Preprocessing:
    - Resampling the audio clips to match the ParlerTTS model requirements.
    - Normalizes text (e.g., expanding abbreviations, handling punctuation).
    - Tokenizes and cleans input to improve model accuracy.
    - Mel-spectrograms are computed as the intermediate speech representation used by the model during training.
- Fine-Tuning and Training
    - The pre-trained model is cloned and loaded using Hugging Face's transformers and datasets libraries.
    - A Custom python script is developed for configuration of the dataset path and for the training arguments.
    - The model uses a diffusion-based learning to refine audio synthesis over several iterations, reducing noise and improving clarity.
    - The model is trained through a GPU-accelerated environment by using an Adam optimizer with learning rate scheduling for stable training and mixed precision (fp16) for performance efficiency.
- Inference and Deployment:
    - The fine-tuned model can synthesize the speech from any input language text taken from the Machine Translated Model
    - Enhances speech clarity through post-processing techniques.

3.3. SYSTEM WORKFLOW (Appendix 1: Flowchart of the System)

The proposed Real-Time Speech-to-Speech Translation System works as a continuous flow of input speech into a pipeline that handles speech input and provides translated speech output in real time.

Workflow:

- Input Speech Acquisition:
  - The system captures live speech through a microphone or processes pre-recorded audio files.
  - The audio is converted into a consistent format for further processing.
- ASR Processing:
  - The ASR module transcribes the input speech into text.
  - It performs feature extraction, acoustic modeling, and language decoding.
  - The cleaned transcription is forwarded to the MT module.
- Text Translation:
  - The MT module receives the transcribed text and translates it into the target language.
  - The model applies tokenization and translation functions.
  - The translated text is sent to the TTS module.
- Speech Synthesis:
  - The TTS module synthesizes speech from the translated text.
  - The acoustic model generates mel-spectrograms.
  - The vocoder converts the mel-spectrogram into a waveform.
  - The system outputs the translated speech in real time.

Chapter 4

Experimental Setup

4.1. ASR – AUTOMATIC SPEECH RECOGNITION

4.1.1. DATA COLLECTION AND PREPROCESSING

4.1.1.1. DATA SOURCE

For the ASR model, we have used Mozilla Common Voices as the primary dataset, as it is a large-scale, open-source speech dataset platform driven by community-led data creation. It is specifically catered towards ASR models, particularly for underrepresented languages and accents. It supports over 100 languages, creating a diverse and high-quality dataset. The dataset we used for this model currently are English, Tamil and Hindi.

The dataset consists of distinct MP3 files along with matching text transcripts. Common Voice has garnered 26,119 hours of recorded content and 17,127 hours of quality-validated and accurate hours as of its most recent release. Each recording frequently comes with demographic metadata like age, sex, and accent, making the dataset more useful for building strong speech recognition models.

Common Voice has a broad range of languages supported, in keeping with its linguistic diversity focus. The dataset has 104 languages, with contributions still being added. Languages differ from universally spoken languages such as English, French, and Mandarin to less spoken languages such as Welsh and Kabyle.

Data is collected through Mozilla's Common Voice platform, where one can contribute either by recording sentences or verifying others' recordings. This dual approach ensures both the expansion and accuracy of the dataset. New datasets are released approximately every six months, incorporating the latest contributions.

4.1.1.2. DATA COMPOSITION

For every supported language, the corpus includes:

- Audio Recordings: Contributors read set sentences, leading to MP3 files that reflect different accents and speaking patterns.
- Text Transcripts: Every audio file has a corresponding text file with the verbatim transcript of the sentence being spoken.
- Demographic Metadata: Voluntary data contributed by participants, such as age, gender, and accent, which supports more diverse speech recognition systems.

The amount of data depends on the language, with varying degrees of community involvement and participation duration. For example, in December 2019, the English dataset consisted of 1,118 verified hours, while Catalan also had 246 hours. By July 2020, the hours tallied stood at 2,275 hours for English and 1,390 for Catalan, evidence of active community involvement.

Its open-source nature encourages innovation and transparency, and it is achievable for developers and researchers worldwide to create applications serving multilingual populations.

4.1.1.3. DATA LOADING AND PREPROCESSING

For the initial preprocessing setup, the English speech data contains over 100000 samples and more, which makes it sufficient to easily fine-tune and train the model with minimal amount of overfitting.

Meanwhile, the number of samples for Hindi and Tamil is very limited, so it requires more time to train with many epochs. We had to make time and calculation estimations to yield good results and save time in training the model.

The audio preprocessing is performed before the speech can be transcribed into text. The input audio data undergoes preprocessing to enhance the clarity and remove unwanted noise. This ensures that the audio is converted into a standardized format that is suitable for feature extraction and evaluation.

The preprocessing steps include: (Appendix 6: Code Snippets for ASR (i))

- Sample Rate Normalization: This process normalizes the audio to a fixed sample rate (16 KHz), ensuring the consistency across different input sources.
- Noise Reduction: Background noise and reverberations are filtered out using spectral subtraction or deep-learning-based noise suppression techniques.

The normalized clean audio data is now correctly mapped and then it is preprocessed through train test split.

For the transcription data that comes with the dataset, tokenization and feature extraction is performed by removing unused columns and retaining necessary metadata for the training, extracting characters to create a vocabulary for training, and Wav2Vec2CTCTokenizer, Wav2Vec2FeatureExtractor, and Wav2Vec2Processor are used to preprocess and prepare data for model training (Appendix 6: Code Snippets for ASR (ii)). The CTC-based data collator is used with padding to handle variable-length audio sequences effectively (Appendix 6: Code Snippets for ASR (iii)).

4.1.2. MODEL ARCHITECTURE (Appendix 3: ASR Model Architecture)

Wav2Vec is a self-supervised learning model developed by Facebook AI for automatic speech recognition (ASR). It processes raw audio waveforms directly, learning contextualized speech representations without requiring extensive labeled data. The model is trained in two steps: first, it is pre-trained on large amounts of unlabeled speech using contrastive learning, and then it is fine-tuned on smaller datasets with labels. Wav2Vec uses a convolutional feature extractor and a transformer-based encoder to capture local and global speech patterns. Its architecture ensures high transcription accuracy even with limited labeled data, making it ideal for a variety of speech-related applications.

In this project, we decided to use Wav2Vec2.0 model architecture, specifically the Wav2Vec2ForCTC variant. This model is a state-of-the-art framework developed by Facebook AI that utilizes self-supervised learning on large amounts of unlabeled speech data, followed by fine-tuning on smaller labeled datasets. The architecture consists of four main stages:

- Feature Extraction Module
- Feature Projection Layer
- Contextual Encoder with Transformer Blocks
- Classification Head (CTC layer)

The first part of the model is the *Feature Extraction Module*, which takes raw audio waveforms and maps them to low-level feature representations. The module is made up of seven convolutional layers. The first layer, Layer 0, is a 1D convolution with 512 channels, kernel size 10, stride 5, activation by GELU (Gaussian Error Linear Unit), and layer normalization. Layers 1-4 have the same pattern with 1D convolutional layers with 512 channels, kernel size of 3, stride of 2, GELU activation, and layer normalization. The last convolutional layers (5 and 6) also consist of 1D convolution with 512 channels, kernel size of 2, stride of 2, GELU activation, and layer normalization. These convolutional layers gradually decrease the temporal resolution of the audio signal, increasing the dimensionality of the features to prime the representations for subsequent processing.

After the convolutional layers, the *Feature Projection Layer* increases the dimensionality of the extracted features. It begins by normalizing 512-dimensional feature representations using LayerNorm. It then applies a linear projection to project the features into a 1024-dimensional space. There is a dropout layer included, but with a probability of 0.0, so no dropout occurs during inference. The primary function of this layer is to increase the dimensionality of the features and make them compatible with the next transformer-based Contextual Encoder.

The heart of the model is *the Contextual Encoder*, which employs a transformer structure. The encoder has 24 layers, with each layer comprising a self-attention mechanism and a feed-forward network (FFN). The self-attention mechanism employs 1024-dimensional Key, Query, and Value projections, and an output projection of the same dimension. The FFN consists of an intermediate dense layer that grows to 4096 dimensions, followed by a GELU activation function, and an output dense layer that brings the dimensionality back down to 1024. Dropout layers are used with 0.0 probability. Layer normalization is employed before and after both self-attention and FFN sublayers. Moreover, the encoder utilizes positional embeddings using a 1D convolution of 1024 channels, kernel size of 128, and padding of 64. The convolution is conducted with 16 groups for better feature separability, and weight normalization is performed using a parametrized convolution layer. This transformer encoder is essential to extract contextual information from the input audio.

Finally, the *Classification Head* is a fully connected linear layer that maps the 1024-dimensional encoder outputs into the vocabulary size of 55 for the current task. The output is passed through a CTC (Connectionist Temporal Classification) loss layer, which is responsible for managing the alignment between the output character sequences and ground truth transcriptions. This is a vital layer for training models that process sequential data, such as speech recognition, where the direct alignment between output and input is infeasible.

4.1.3. TRAINING SETUP

4.1.3.1. HYPERPARAMETERS

After completing the pre-processing set up, the model is imported from the transformers module. The pre-trained model that was used for fine-tuning is the Wav2Vec-XLS-R-300M. This model is a multilingual speech recognition model. It belongs to the XLS-R family model, which is a Cross-Lingual Speech Representation, an extension of Wav2Vec 2.0. This model is optimized for learning speech representations from vast amounts of unlabeled audio across multiple languages.

Table 1: Model Parameters for ASR

Hyperparameter	**Value**	**Description**
Model	facebook/wav2vec-xls-r-300m	The pre-trained Wav2Vec 2.0 variant model.
Attention_dropout	0.1	This controls the dropout rate applied to the attention weights in a transformer model.
Hidden_dropout	0.1	This is the dropout applied to the fully connected layers in the model.
Feat_proj_dropout	0.1	This is the dropout applied to the feature projection layer, which transforms input features before feeding them into the model.
Mask_time_prob	0.15	Used in time-masking augmentation for models like Wav2Vec2.
Layerdrop	0.1	A dropout technique applied to entire transformer layers.
CTC_loss_reduction	mean	It defines how the loss is reduced in a CTC loss function. “mean” indicates the loss is averaged over all elements in a batch.

Table 2: Hyperparameters for ASR model training arguments

Hyperparameter	**Value**	**Description**
Per_device_train_batch_size	16	Number of training samples per GPU in each batch.
Gradient_accumulation_steps	2	Gradients are accumulated over 2 steps before updating model weights, effectively doubling the batch size.
Eval_strategy	epoch	Model evaluation is performed at the end of each epoch.
num_train_epochs	10 (Varied)	The model will be trained for 10 complete passes over the dataset.
gradient_checkpointing	True	Saves memory by recomputing activations instead of storing them during backpropagation.
fp16	True	Uses mixed precision (16-bit floating point) to speed up training and reduce memory usage.
save_steps	1000	Saves a checkpoint of the model every 1000 steps.
eval_steps	1000	Runs evaluation every 1000 steps during training.
logging_steps	500	Logs training progress every 500 steps.

learning_rate	3e-4	Initial learning rate set to 0.0003 for optimizer updates.
warmup_ratio	0.05	Gradually increases the learning rate over the first 5% of training steps.
save_total_limit	2	Keeps only the last 2 model checkpoints to save disk space.
push_to_hub	False	Disables automatic model uploads to the Hugging Face Hub.
optim	adamw_bnb_8bit	Uses AdamW optimizer with an 8-bit optimizer variant for memory efficiency.

4.1.4. EVALUATION METRICS

4.1.4.1. LOSS FUNTION

The CTC Loss Reduction is a standard loss function mainly used in the sequence-to-sequence model to measure the training loss and validation loss of the model training. It can handle different lengths of inputs and outputs; it does not need to explicitly match the inputs with outputs during training and it is capable of handling variable-length sequences without relying on the pre-segmented data. By summing over all possible alignments, it reduces overfitting to specific alignments. The CTC Loss function is set to “Mean” which means, the loss is averaged over all sequences in the batch.

Formula:

$$\mathcal{L}_{CTC} = -\sum_{(\mathbf{x},\mathbf{l})\in Z} \ln Pr(\mathbf{l}|\mathbf{x})$$

Figure 1: CTC Loss

where, P_r (l|x) is the total probability of the target sequence.

4.1.4.2. METRICS

The most used evaluation metrics for the ASR models are called Word Error Rate and Character Error Rate. For this project, we have finalized the use of Word Error Rate as the ideal evaluation metric.

Name of the metric: Word Error Rate

Description: It is a metric used to evaluate the accuracy of speech-to-text systems, measuring the proportion of errors (substitutions, insertions, and deletions) in a transcript compared to a reference transcript.

Formula:

$$\mathrm{WER} = \frac{S + D + I}{N}$$

Figure 2: WER formula

where,

- I – Insertion: It is the count of words that are incorrectly added in the OCR output.
- D – Deletion: It is the count of words that were not recognized and are thus missing in the OCR output.
- S – Substitution: It is the count of words that are incorrectly recognized and replaced with another word.
- N – Number of words in the reference

Interpretation: A lower Word Error Rate indicates better accuracy in recognizing the speech data.

4.1.5. POST TRAINING

The completion of training the model varied and was highly time-consuming. The next part is saving the model and this is a crucial thing to do as it will be used in the final pipelining of the system. After saving the model, the model is subjected to transcription testing by providing an audio recording as an input. This will be further mentioned in the results in a detail.

4.1.6. COMPUTATIONAL RESOURCES

4.1.6.1. HARDWARE SPECIFICATIONS

For ASR Model, it was trained on this hardware with the following specifications:

- GPU: NVIDIA RTX 3050 Laptop GPU
- VRAM (GPU Memory): 6GB VRAM
- CPU: Intel Core i5-12450HX (8 cores and 12 threads)
- RAM: 20GB (12GB + 8GB) (4800 MHz)
- Storage: 512GB NVMe SSD

4.1.6.2. RUNTIME CONSIDERATIONS

Training Time:

- English language - It took 26 hours (approx.) to train 10 epochs.
- Hindi language - It took 78 hours (approx.) to train 50 epochs.
- Tamil language - It took 80 hours (approx.) to train 15 epochs.

4.1.6.3. SOFTWARE VERSIONS

The following libraries and frameworks were used for building, training, and evaluating the model:

- PyTorch (2.6.0+cu118): An open-source deep learning framework optimized for GPU-accelerated computations.
- Torchaudio (2.6.0): A PyTorch library for audio processing, featuring I/O, transforms, and model support.
- Transformers (4.49.0.dev0): Hugging Face’s library for state-of-the-art NLP models like BERT, GPT, and Wav2Vec.
- HuggingFace-Hub (0.28.1): A client library to interact with Hugging Face's model and dataset hub.
- Datasets (2.21.0): A library for loading, processing, and sharing large-scale datasets efficiently.
- Tqdm (4.67.1): A fast, extensible progress bar for Python loops and iterables.

- Librosa (0.10.2.post1): A Python package for analyzing and processing audio and music signals.
- Soundfile (0.13.1): A library for reading and writing audio files in various formats.
- NoiseReduce (3.0.3): A simple noise reduction library for audio processing using spectral gating.
- Pandas (2.2.3): A powerful data analysis and manipulation library for structured data.
- Numpy (1.26.4): A fundamental package for numerical computing in Python.
- Regex (2024.11.6): A module for handling regular expressions efficiently.
- Jiwer (3.1.0): A library for computing Word Error Rate (WER) and other speech recognition metrics.
- Bitsandbytes (0.45.3): A lightweight library for 8-bit optimization in deep learning models.
- Evaluate (0.4.3): A library for computing evaluation metrics for NLP and ML models.
- Nltk (3.9.1): A leading Python library for natural language processing (NLP) tasks.
- Scipy (1.13.1): A scientific computing library for optimization, statistics, and signal processing.

4.2. MT – MACHINE TRANSLATION

4.2.1. DATA COLLECTION AND PREPROCESSING

4.2.1.1. DATA SOURCE

For the Machine Translation (MT) model, the OPUS-100 dataset was used as the primary and only dataset. OPUS-100 is a large-scale, open-source multilingual corpus specifically designed for neural machine translation (NMT) tasks. It offers parallel sentence pairs for 100 language combinations, making it an ideal choice for building robust translation models.

The dataset is derived from multiple sources, ensuring diverse linguistic representations and a wide range of sentence structures. The sources include:

- Wikipedia: Provides general-purpose sentences covering various topics, including science, history, and technology.
- GNOME and KDE: Contain sentences from open-source software interfaces, which are technical and structured in nature.
- OpenSubtitles: Includes dialogue-based sentences extracted from movie and TV show subtitles, featuring conversational and colloquial phrases.
- Tanzil: Comprises religious texts, mainly Quranic translations, which include formal and literary expressions.

These diverse sources provide balanced linguistic coverage, enabling the MT model to handle formal, informal, conversational, and technical language effectively.

4.2.1.2. DATA COMPOSITION

The OPUS-100 dataset used for training the Machine Translation (MT) model consists of parallel sentence pairs in English-Hindi and English-Tamil. Each sample contains a source sentence in English and a target sentence in Hindi or Tamil, making it a supervised dataset specifically designed for translation tasks. The dataset is divided into three subsets: training, validation, and test sets, which are used to train, fine-tune, and evaluate the model, respectively.

For the English-Hindi language pair, the complete dataset contains (all approximately):

- 534,000 sentence pairs in the training set
- 2,000 sentence pairs in the validation set
- 2,000 sentence pairs in the test set

For the English-Tamil language pair, the dataset consists of (all approximately):

- 227,000 sentence pairs in the training set
- 2,000 sentence pairs in the validation set
- 2,000 sentence pairs in the test set

In this project, focused on English to Tamil translation model, only 10% of the training dataset was used for fine-tuning the MBart-50 model, while the full validation and test sets were utilized for both language pairs. This configuration was chosen to reduce training time and computational costs while still achieving effective fine-tuning. The distribution ensures that the model is exposed to a diverse range of sentence pairs during training, while the full validation and test sets are used for comprehensive evaluation

The dataset contains a mixture of formal, informal, and technical sentences, making it suitable for general-purpose translation tasks. The sentence lengths vary depending on the language pair. For English-Hindi, the average sentence length in English is approximately 10 to 15 tokens, while the corresponding Hindi translations tend to be longer, averaging 12 to 18 tokens due to the syntactic and grammatical differences. Similarly, for English-Tamil, the English sentences contain an average of 9 to 13 tokens, whereas the Tamil translations are slightly longer, averaging 10 to 15 tokens due to the morphological complexity of Tamil.

This diverse and comprehensive dataset composition ensures that the model is exposed to various sentence structures, syntactic variations, and vocabulary usage patterns, allowing it to learn robust translation mappings across the English-Hindi and English-Tamil language pairs.

4.2.1.3. DATASET LOADING AND PREPROCESSING

Because the original dataset has many translation pairs, only a fraction of them is taken here to fine-tune and train the model.

- Dataset Splits

- Training Split:
    - English-Hindi: 5% subset of the training data.
    - English-Tamil: 10% subset of the training data.
- Validation Split:
    - Full validation set for both language pairs.
- Testing Split:
    - Full test set used for final evaluation.

- Sample Pairs

For English-Hindi:

```
{
        "en": "Where can I find this person?",
        "hi": " मैं इस व्यक्ति को कहाँ पा सकता हूँ?"
}
```

For English-Tamil:

```
{
        "en": "What time does the flight leave?",
        "hi": "விமானம் எந்த நேரத்தில் புறப்படுகிறது?"
}
```

- Once the dataset is loaded, it needs to be formatted and tokenized for model training. The steps involved include:

- Extracting Sentence Pairs: Extracting source and target sentences from the OPUS-100 dataset.
- Tokenization: Converting sentences into token IDs using the MBartTokenizer.
- Padding and Truncation: Ensuring all token sequences have a consistent length of max_length=128.

- Labels Assignment: Setting the tokenized target sentence as the labels for the model.

- Preprocessing Steps:

- Extracting Source and Target Sentences (and vice-versa):
    - For English-to-Hindi: src_lang = "en", tgt_lang = "hi"
    - For English-to-Tamil: src_lang = "en", tgt_lang = "ta"
- Tokenizing Sentences:
    - Source and target sentences are tokenized separately.
    - The resulting input IDs are padded or truncated to a maximum length of 128 tokens.
- Assigning Labels:
    - The target sentence tokens (labels) are assigned to the model’s labels field.

After preprocessing, the tokenized datasets are prepared for training. The model uses a data collator to handle padding and batching efficiently. Code mentioned in *Appendix 7: Code Snippets for MT (i).*

- Data Collator:

- The DataCollatorForSeq2Seq ensures:
    - Consistent padding of token sequences in batches.
    - Efficient memory usage during training.

  Code mentioned in *Appendix 7: Code Snippets for MT (ii).*

- Batching and Loading:

- The tokenized datasets are loaded into the model’s training pipeline.
- The model processes the data in batches, utilizing gradient accumulation for stability.

4.2.2. MODEL ARCHITECTURE (Appendix 4: MT Model Architecture)

The MBART-50 model, used in this project, was introduced in the paper "Multilingual Translation with Extensible Multilingual Pretraining and Fine-tuning" by Yuqing Tang, Chau Tran, Xian Li, Peng-Jen Chen, Naman Goyal, Vishrav Chaudhary, Jiatao Gu, and Angela Fan. It is an extension of the original MBART model, which was initially designed for cross-lingual denoising pretraining.

The MBART-50 version was specifically created to handle 50 languages, making it a highly effective and scalable model for multilingual translation. This model was built by extending the original mbart-large-cc25 checkpoint with additional embedding layers. These new layers were initialized with randomly generated vectors corresponding to 25 new language tokens, effectively doubling the language support.

The MBART-50 model was further pretrained on parallel text corpora covering all 50 languages, making it significantly more robust for multilingual translation tasks. The ability to train in multiple directions improved its efficiency and accuracy, allowing it to outperform bilingual models on several benchmarks.

The MBartForConditionalGeneration model is trained to perform sequence-to-sequence tasks, including machine translation, through a strong encoder-decoder architecture. The model is derived from MBartModel, with hyperparameter changes such as the number of epochs in fine-tuning. The model has a shared embedding layer, an encoder, a decoder, and a language modeling head.

The shared embedding layer embeds input tokens into a 1024-dimensional space with 250,054 possible tokens. The encoder, the central part of the model, makes use of MBartEncoder. It starts with the embed_tokens layer, projecting input tokens to a 1024-dimensional space. The embed_positions layer appends learned positional embeddings to the input sequence.

The encoder consists of 12 layers of MBartEncoderLayer, each of which has a self-attention mechanism, including key, query, and value projections, all of dimension 1024. The attention is then followed by a feed-forward network consisting of two fully connected layers: one that expands the features to 4096 dimensions and another that projects them back to 1024 dimensions. Layer normalization occurs at several places, such as before and after the self-attention mechanism and the feed-forward network.

The decoder has the same structure as the encoder with its own batch of 12 layers of MBartDecoderLayer. Every decoder layer consists of a self-attention mechanism and an encoder-decoder attention mechanism, enabling the decoder to attend to useful portions of the encoder output. The attention projections (key, value, query, and output) all have the same dimensionality of 1024. The feed-forward network in the decoder is also organized in the same way as the encoder, with a 4096-dimensional intermediate layer and a final projection down to 1024 dimensions. Layer normalization is applied throughout the decoder as well, ensuring stabilization during training and inference.

Toward the top of the model, the lm_head is a linear layer that projects the output of the decoder onto the vocabulary size (250,054). Projected by that is a CTC loss layer, which allows the model to process sequential predictions well. The robustness and flexibility of the architecture allow the model to be fine-tuned on several tasks, including translation, with the option of modifying hyperparameters such as the number of epochs during training to maximize performance.

4.2.3. TRAINING SETUP

4.2.3.1. HYPERPARAMETERS

The training setup for the machine translation model is based on fine-tuning the mBART-50 model using the Hugging Face Transformers framework. The training configuration includes various hyperparameters that control the optimization process, memory management, and model performance.

All the general details mentioned in fine-tuning and training the MT model here are for *English to Tamil translation*. Same process is followed for English to Hindi, Hindi to English and Tamil to English with just small changes in the amount of train data used and the number of epochs.

Table 3: Hyperparameters for MT fine-tuning

Hyperparameter	Value	Description
Model	facebook/mbart-large-50	The pre-trained multilingual mBART-50 model.
Learning Rate	3e-5	Learning rate for the Adam optimizer.
Batch Size (Train)	8	Batch size for training per device.
Batch Size (Eval)	8	Batch size for evaluation per device.
Gradient Accumulation	2	Gradually accumulates gradients over multiple batches.
Weight Decay	0.01	Regularization term to prevent overfitting.
Num Train Epochs	5	Epochs for training
FP16	True	Mixed-precision training for faster computation and lower memory.
Save Strategy	epoch	Save model checkpoints at the end of each epoch.
Save Total Limit	1	Maximum number of model checkpoints saved.

Predict with Generate	True	Uses beam search for better translation accuracy.
Num Beams	4	Number of beams for translation during inference.
Early Stopping	True	Stops translation once all beams have reached the end token.
Dataloader Workers	0	No parallel workers (for editor compatibility).
Logging Steps	1000	Logs training metrics every 1000 steps.

4.2.4. EVALUATION METRICS

BLEU (Bilingual Evaluation Understudy) Score

Description of the metric:

The BLEU score is a widely used evaluation metric in machine translation tasks. It measures how closely the machine-generated translation matches one or more human reference translations. The BLEU score calculates the precision of n-grams (continuous sequences of words) in the predicted translation by comparing them to the reference translation. It ranges from 0 to 1, where 1 indicates a perfect match with the reference translation, while 0 indicates no match. In practice, the BLEU score is typically represented as a percentage, with higher scores indicating better translation quality.

The BLEU metric uses a combination of:

- n-gram precision: The proportion of n-grams in the candidate translation that are present in the reference translation.
- Brevity Penalty (BP): A penalty applied when the model generates shorter sentences than the reference, preventing it from artificially boosting its score by generating overly concise translations.

Formula:

$$\text{Unigram precision } \mathbf{P} = \frac{m}{w_t}$$

$$\text{Brevity penalty } p = \begin{cases} 1 & \text{if } c > r \\ e^{\left(1-\frac{r}{c}\right)} & \text{if } c \leq r \end{cases}$$

$$\text{BLEU} = p.\, e^{\sum_{n=1}^{N} \left(\frac{1}{N} * \log Pn\right)}$$

Figure 3: BLEU Score

Where:

- m is the number of matching n-grams between the model's translation and the reference translation.
- w_t is the total number of n-grams in the model's translation.
- N is the maximum n-gram size (usually set to 4).
- P_n is the precision for n-grams of size
- c is the length of the candidate translation.
- r is the length of the reference translation.
- BP (p) is the brevity penalty that penalizes overly short translations.

Interpretation:

A higher BLEU score indicates that the model’s translation is closer to the reference translation, demonstrating better translation quality.

4.2.5. COMPUTATIONAL RESOURCES

The model was trained using Kaggle's GPU environment, which provides NVIDIA Tesla P100 GPUs. The hardware configuration ensures efficient model training and evaluation by leveraging GPU acceleration.

The following details are mentioned for fine-tuning and training English-Tamil dataset. Similar setup, configurations and specifications were observed for the other MT models.

4.2.5.1. HARDWARE SPECIFICATIONS

- GPU Name: Tesla P100-PCIE-16GB
- CPU: Intel(R) Xeon(R) CPU
- Memory Usage: 15081MiB / 16384MiB
- CUDA Version: 12.6 (enabled for PyTorch operations)

4.2.5.2. RUNTIME CONSIDERATIONS

Training time:

- For English to Tamil: Around 3 hours for 5 epochs
- For Tamil to English: Around 3-4 hours for 5 epochs
- For English to Hindi: Around 6 hours for 10 epochs
- For Hindi to English: Around 6-7 hours for 10 epochs

4.2.5.3. SOFTWARE VERSIONS

Libraries and Versions:

- Python Version: 3.10
- PyTorch: 2.0.1+cu117
- CUDA Support: Enabled – Version: 12.1
- Transformers (4.37.2): Library used for loading the MBart model and performing tokenization and translation.
- Hugging Face Evaluate (0.4.1): Used for BLEU score evaluation with sacrebleu.

4.3. TTS – TEXT TO SPEECH

4.3.1. DATA COLLECTION AND PREPROCESSING

4.3.1.1. DATA SOURCE

For TTS, we utilized LJSpeech dataset for English Language and IndicTTS for Tamil and Hindi Language.

LJSpeech Dataset, which was created by Keith Ito, is a most widely employed single-speaker public domain dataset for academic and commercial applications as the standard to develop and compare TTS models.

The IndicTTS project is a joint effort by several research and academic institutions in India to develop open-source or freely accessible materials for text-to-speech systems in several Indian languages. Led by institutions like IIT Madras, IIT Guwahati, IIIT Hyderabad, and more, the project deals with a few languages like Hindi, Tamil, Telugu, Bengali, and others.

Among language-specific datasets, Hindi and Tamil are two of the most widely used. Hindi is one of the official languages of India and has hundreds of millions of native speakers. Tamil is a language spoken mainly in the Indian state of Tamil Nadu.

4.3.1.2. DATA COMPOSITION

a) English Dataset:
   The LJSpeech dataset consists of a folder of approximately 13100 audio recordings from a single female speaker spoken predominantly in American English, typically ranging between 1 and 10 seconds and a transcription CSV file that contains the audio file names and the texts which are derived from public domain sources, covering a wide range of English texts which includes novel, short stories, and essays.
   In total, the audio sums to around 24 hours of speech. This duration is sufficient for training many state-of-the-art TTS models while remaining manageable for academic and experimental use.

Each audio is recorded at 22050 Hz in Mono Channel with 16-bit PCM WAV bit depth storing it as a .wav file. Each audio file is named like an ID, indicating their order (Ex: LJ001-0001.wav). The Quality Characteristics of each audio file are its minimal noise, low amount of background noise or echo and its consistent volume, where each audio has consistent amplitude levels.

The Metadata CSV contains three columns:

- File ID: contains the audio filename without the .wav extension
- Text: The Original transcribed text, often directly taken from public domain sources
- Normalized Text: An Optional column that consists of texts that have been normalized (removing punctuations, special characters and standardizing abbreviations).

b) Hindi and Tamil datasets:

Both IndicTTS Hindi and Tamil datasets consist of a folder of audio files of male and female speakers each. The duration per language can vary based on the specific release. Commonly, each language dataset ranges between 5 to 12 hours of recorded speech. For example, one version of the Hindi dataset may contain around 10 hours, while a Tamil dataset might contain around 8 hours. (Exact durations depend on the specific version/release date.). The dataset also consists of a transcription text file each that provides the exact text (in the native script) spoken in each audio file.

Each audio at their respective languages is recorded in a controlled, studio like environment to minimize the background noise at 16000 Hz in Mono Channel with 16-bit PCM bit depth storing it as a .wav file. Each audio file names are stored and named in a systematic pattern (ex: hin_female_0001.wav, tam_male_0123.wav) indicating the language, speaker gender, and utterance index.

The Transcription text file consists of columns:

- File ID: Audio file names without the .wav extension
- Text: Extract text (in the native language) spoken in each audio file

Some versions of the dataset may include phoneme-level or syllable-level alignment annotations, though this is not always guaranteed.

4.3.1.3. DATA LOADING AND PREPROCESSING

For Both IndicTTS and LJSpeech datasets, the data preprocessing involves transforming raw audio and text data into a format suitable for the model's multimodal training pipeline.

Once the dataset has been loaded, it's been annotated with additional metadata such as SNR, C50 noise type reverberation time, and speaking rate which are used during prompt creation to aid. The annotations are then mapped to another dataset which will be created and used for the prompt creation script. Each audio file is resampled to match the requirements of the model. The TTS Model uses Prompt-Based generation so a script for creating the prompts is run using the dataset, speaker name, pre–trained model for the language embedding i.e Google's 2B Version of the Gemma Model and the output directory where it will save the prompt formatted dataset that will be used as the speaker's natural language descriptions and generate the prompt and original transcript text into the prompt description. *Codes can be referred at Appendix 8: Code Snippets for TTS (i) and (ii).*

The audio files and the text file are filtered on duration constraints and truncated respectively to ensure efficient training:

- Minimum duration = 2 seconds
- Maximum duration = 20 seconds
- Maximum Text length = 400

The audio files are encoded using a pre-trained feature extractor while the text and descriptions are tokenized to align the suitabilities for the model. Each processed data includes:

- Encoded representation of the audio files
- Original transcription of the audio
- Synthesized speaker prompt
- Tokenizer values (attention masks and input ids)

4.3.2. MODEL ARCHITECTURE (Appendix 5: TTS Model Architecture)

The ParlerTTSForConditionalGeneration model is an advanced text-to-speech (TTS) system integrating a text encoder, an audio encoder, and a decoder to transform text into high-quality audio. The text encoder, based on a T5EncoderModel, employs a transformer architecture with 12 blocks of self-attention and feed-forward layers, embedding a vocabulary of 32,128 tokens into 768-dimensional vectors. This component processes input text into a contextual representation, enhanced by layer normalization and dropout for stability, which conditions the subsequent audio generation process.

The audio encoder, a DAC (Descript Audio Codec) model, compresses raw audio into a latent space using a series of convolutional blocks with Snake1d activations and strided convolutions, increasing channel depth from 1 to 1024. A nine-layer residual vector quantizer then discretizes these features into codes with a 1024-entry codebook per layer. This quantized representation is efficient and serves as the bridge to the decoding stage, where audio reconstruction occurs.

The decoder comprises a DAC decoder and a ParlerTTSForCausalLM model. The DAC decoder upsamples the quantized codes through transposed convolutions, reducing channels back to 1 with a Tanh activation for the final waveform. Meanwhile, the ParlerTTSDecoder autoregressively generates audio tokens using 24 transformer layers with self- and cross-attention, conditioned on text via a projection layer and a prompt embedding.

4.3.3. TRAINING SETUP

4.3.3.1. HYPERPARAMETERS

The ParlerTTS model is a prompt-driven, multi-modal text-to-speech synthesis architecture. In this project, the model was fine-tuned on a customized version of the IndicTTS-Hindi dataset, which was enhanced with speaker-specific and environmental metadata.

Table 4: Model Parameters for TTS

Parameters	**Value**	**Description**
Model Name	parler-tts/parler_tts_mini_v0.1	Pretrained compact TTS model used as the base for fine-tuning
Audio Encoder	parler-tts/dac_44khZ_8kbp	Encoder for converting audio into compressed latent representations
Description Tokenizer	parler-tts/parler_tts_mini_v0.1	Tokenizer used for processing the descriptive prompt text
Prompt Tokenizer	parler-tts/parler_tts_mini_v0.1	Tokenizer used for processing the original spoken sentence
Freeze Text Encoder	True	Prevents updates to the text encoder during training
Audio Encoder Batch Size	2	Number of audio samples encoded per device per step
Max Audio Duration	20.0 seconds	Maximum allowed duration of audio clips used for training
Min Audio Duration	2.0 seconds	Minimum allowed duration of audio clips used for training
Max Text Length	400 characters	Maximum character length for input sentences

Table 5: Hyperparameters for TTS model training arguments

Hyperparameters	**Value**	**Description**
Batch Size (Per Device)	1	Number of samples processed per GPU per step
Gradient Accumulation Steps	36	Accumulate gradients across 36 steps to simulate larger batch size
Learning Rate	0.00008	Base learning rate for the optimizer
Optimizer	Adam	Optimization algorithm used during training
Adam Beta1 (β_1)	0.9	Exponential decay rate for first moment estimates
Adam Beta2 (β_2)	0.99	Exponential decay rate for second moment estimates
Weight Decay	0.01	Regularization to prevent overfitting
Scheduler	constant_with_warmup	Keeps learning rate constant after warmup steps
Warmup Steps	50	Gradual learning rate increase during the initial steps
Epochs	10	Total number of passes through the entire training dataset
Mixed Precision	float16	Enables faster training and reduced memory usage

Gradient Checkpointing	True	Saves memory by recomputing forward pass during backpropagation
Group by Length	True	Groups samples by sequence length to minimize padding
Evaluation Batch Size	4	Number of evaluation samples processed per device
Max Evaluation Samples	8	Maximum number of samples used during evaluation phase
Logging Tool	wandb	Tool used for visualizing training and evaluation metrics
Logging Steps	10	Frequency (in steps) of logging metrics
Random Seed	456	Seed value to ensure training reproducibility

4.3.4. EVALUATION

The evaluation strategy for the fine-tuned ParlerTTS model was designed to assess both the text-audio alignment and the naturalness of generated speech. Since ParlerTTS is a prompt-based TTS model, the evaluation also ensures that generated audio accurately reflects descriptive prompts provided during training.

The evaluation process was enabled using the following flags in the training script:

- Do_eval
- Predict with generate
- Include inputs for metrics

These settings activate automatic speech generation during validation and allow computation of relevant metrics based on input descriptions and generated outputs.

4.3.4.1. AUTOMATIC EVALUATION SETTINGS

During each step of the model training:

- A Subset of the training dataset (upto 8 maximum evaluation samples) was used for quick, iterative validation.
- Generated audio outputs were compared against the original waveforms and text descriptions.
- The model utilized the 'predict with generate' metric flag to synthesize new audio from prompts and 'include inputs for metrics' metric flag to compute metrics over both generated and reference samples.

The evaluation was performed using:

- Per-device batch size: 4
- Group-by-length: Enabled to reduce padding and improve GPU utilization

4.3.4.2. OTHER EVALUATION METRICS

Enabling 'predict_with_generate' and 'include_input_for_metrics' involves:

- Text-audio alignment Quality: Ensures the generated audio reflects the intended content described in the textual prompt and spoken sentence.
- Subjective Quality Metrics
    - WandB (Weights and Biases) integration in the training script (report_to "wandb") was used to log training and evaluation loss values, along with generated sample previews.
    - Human Evaluation aspects like:
        - Pronunciation Clarity
        - Emotional Tone
        - Overall realism of Synthetic Voice

4.3.5. COMPUTATIONAL RESOURCES

4.3.5.1. HARDWARE SPECIFICATIONS

The ParlerTTS model is trained on Google Colab Environment, utilizing NVIDIA T4 GPUs for faster and efficient Model training process.

4.3.5.2. RUNTIME CONSIDERATIONS

On average, all the 3 languages took around:

- 7-10 hours when 5 epochs were considered
- 15-20 hours when 10 epochs were considered

4.3.5.3. SOFTWARE VERSIONS

The following libraries and modules used for the Fine-tuned ParlerTTS model are:

- PyTorch (2.6.0+cu124): Used to define model, compute gradient and GPU accelerate and supports mixed precision training
- Hugging Face Transformers (4.46.1): It supplies pre trained tokenizers and model architectures for prompt embedding and text embedding.
- Hugging Face Accelerate (1.5.2): Simplified multi-GPU and mixed-precision training
- Weights and Biases (WandB) (0.19.8): Logging for training and evaluation metrics (loss values, and sample outputs) that is integrated into the model and allows remote tracking and visualization of model performance at an epoch level.
- NumPy (1.26.3): Used for basic numerical operations on audio arrays and tensor formatting.
- Torchaudio (2.6.0+cu124): used for sampling and simple waveform manipulation, especially for preparing the audio, prior to DAC encoding.
- Librosa (0.11.0): used to compute audio durations and features such as verifying clip lengths.

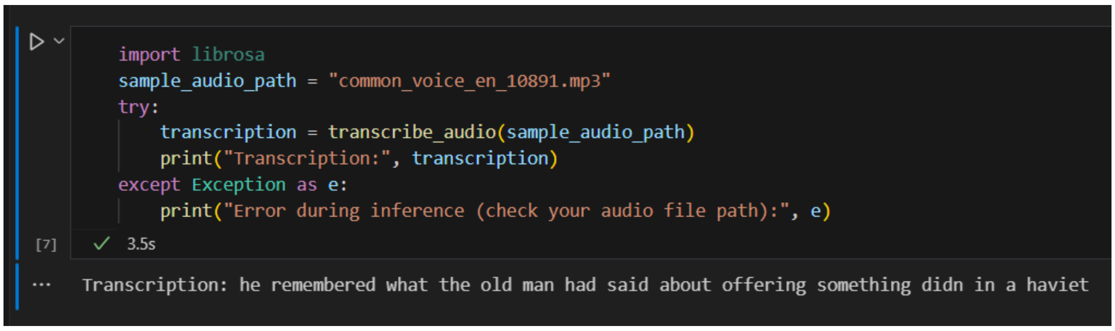
Chapter 5

Results and Analysis

5.1. RESULTS

a) ASR MODEL

English Audio Transcription:



```
import librosa
sample_audio_path = "common_voice_en_10891.mp3"
try:
    transcription = transcribe_audio(sample_audio_path)
    print("Transcription:", transcription)
except Exception as e:
    print("Error during inference (check your audio file path):", e)
```

```
[7]  ✓ 3.5s
```

```
Transcription: he remembered what the old man had said about offering something didn in a haviet
```

Figure 4: ASR English Sample Output

Using the saved model and processor, they are imported and using the transcribe_audio function, we can generate the transcription by providing the audio recording directly.

The model successfully transcribes the audio and generates the transcript and displays the output. The transcription success rate is mostly 90% with a minute number of errors present.

Hindi transcription sample output can be seen in *Appendix 9: Results of models (i)*. The transcription success rate is mostly 95% for Hindi with a minute number of errors present.

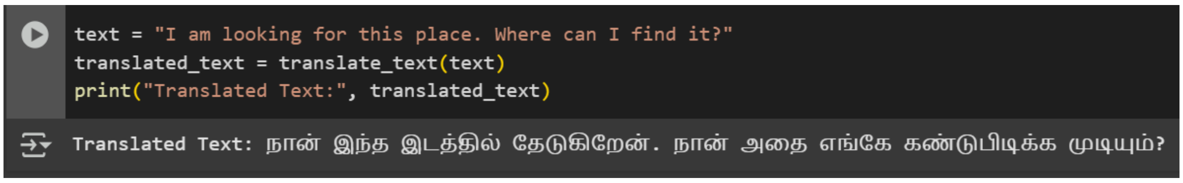
It is observed that the training loss is about 25% which means training accuracy is about 75%, the validation loss is about 45% which means validation accuracy is 55% and Word Error Rate is 30%, which means Word Accuracy is 70%.

b) MT MODEL

The training process of the mBART-large-50 model on the English to Tamil (en-ta) concluded after 5 epochs for Tamil (and 10 for Hindi), completing 7095 training steps in approximately 2 hours and 53 minutes. The recorded final training loss was 0.8055, indicating a stable convergence of the model. The training efficiency metrics reveal that the model processed approximately 10.9 samples per second and achieved a step processing speed of 0.681 steps per second. The total computational effort for the training process was approximately 3.07e+16 floating-point operations (FLOPs), showing the extensive computational requirements of transformer-based architectures.

During the evaluation phase, the model achieved a validation loss of 0.2609 at the final epoch, with a lowest recorded validation loss of 0.2507 at epoch 3. The evaluation process was completed in 57.49 seconds, where the model processed 34.79 samples per second and executed 4.35 steps per second, demonstrating an efficient inference speed. The final BLEU score obtained was 13.64, calculated based on n-gram precision matching between the predicted and reference translations. The brevity penalty (BP) of 0.9217 indicates that the model produced slightly shorter translations than the reference, contributing to a marginal reduction in the overall BLEU score.

A sample output for English to Tamil:



```
text = "I am looking for this place. Where can I find it?"
translated_text = translate_text(text)
print("Translated Text:", translated_text)
```

```
Translated Text: நான் இந்த இடத்தில் தேடுகிறேன். நான் அதை எங்கே கண்டுபிடிக்க முடியும்?
```

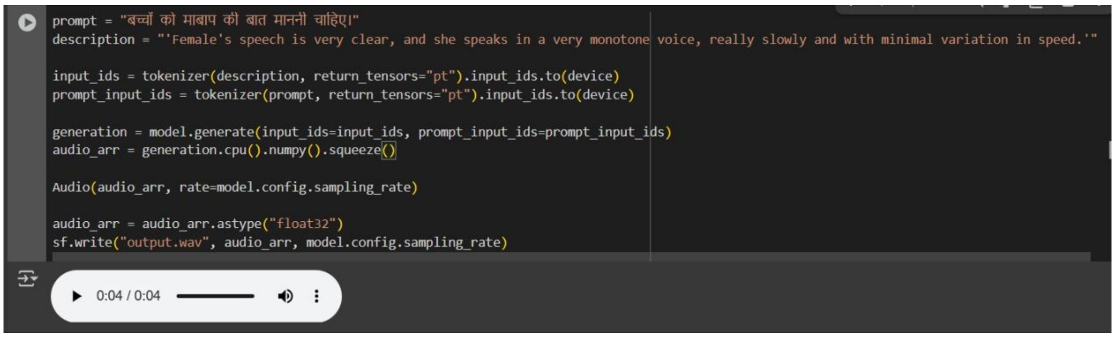
Figure 5: MT English to Tamil Sample output

The outputs for other models of MT can be seen at *Appendix 9: Results of models (ii), (iii) and (iv)*

c) TTS MODEL

The Fine-tuned ParlerTTS model was trained on a combination of both the LJSpeech English dataset and the IndicTTS Hindi and Tamil Dataset. The Model successfully generates the audio from the given input text that is transcribed and translated from the ASR and MT Model respectively. An Optional Speaker description can be given to set the condition for the audio to be generated in that speaker's tone or setting thereby making the audio sound more humane although, the audio is generated with some minute errors.

Below is a snapshot of an example Input and Description for the how the synthesized speaker is spoken from the generated audio output:



```
prompt = "बच्चों को माबाप की बात माननी चाहिए।"
description = "'Female's speech is very clear, and she speaks in a very monotone voice, really slowly and with minimal variation in speed.'"

input_ids = tokenizer(description, return_tensors="pt").input_ids.to(device)
prompt_input_ids = tokenizer(prompt, return_tensors="pt").input_ids.to(device)

generation = model.generate(input_ids=input_ids, prompt_input_ids=prompt_input_ids)
audio_arr = generation.cpu().numpy().squeeze()

Audio(audio_arr, rate=model.config.sampling_rate)

audio_arr = audio_arr.astype("float32")
sf.write("output.wav", audio_arr, model.config.sampling_rate)
```

Figure 6: TTS Hindi Speech output

Although, it is to be noted that the sentence has been taken independently from the previous two models and is used for testing and analysis only.

5.2. ANALYSIS

a) ASR MODEL

The Wav2Vec-tuned ASR optimized model scores ~80% accuracy and can thus be applied for real-time translation of speech. Its performance depends on hardware limitations, training time, data quantity, and voice variability. The model performs with a WER of ~20%, which is encouraging but cannot handle deep accent and noisy environments.

Implementing the model is challenging, especially in terms of hardware since fine-tuning requires high-end GPUs (16GB+ VRAM).

It takes a lot of time to train on large datasets such as Mozilla Common Voice, and though SSL minimizes the requirement for labeled data, high-quality transcripts are still needed. The model is also not good at non-standard accents and low-resource languages and needs more fine-tuning using diverse datasets.

Optimizations like quantization and distillation can compress model size to deploy on less powerful hardware. Denoising autoencoders and spectral subtraction can enhance noisy environment performance. A multi-stage fine-tuning approach—pre-training on general datasets followed by domain-specific training—can help improve adaptability.

b) MT MODEL

The model learned translation patterns successfully from English-Tamil to Hindi-English, with training loss decreasing from 3.54 (epoch 1) to 0.1867 (epoch 5). Validation loss leveled off after epoch 3 (0.2507), indicating increased training has a chance to overfit.

A BLEU score of 13.64 reflects moderate translation accuracy but is poor on longer n-grams (1-gram: 43.28%, 2-gram: 18.45%, 3-gram: 9.96%, 4-gram: 6.03%). The brevity penalty (0.9217) indicates shorter outputs, which may miss context and could be resolved by extending sequence length or modifying decoding with a broader beam search.

We can improve handling longer sequences and fluency by fine-tuning with larger data, data augmentation, and hyperparameter tuning on the sequence length and beam width.

c) TTS MODEL

The fine-tuned ParlerTTS model performs well but requires further training for more accurate and natural speech. Training loss decreased steadily to 8.79, while evaluation loss stabilized at 4.01, demonstrating effective learning despite the multilingual nature of the task. Codebook losses ranged from 2.95 to 4.21 (evaluation) and 6.92 to 9.27 (training), confirming strong latent representation learning for phonetic translation across languages. A stable gradient norm (0.21) and a learning rate of 8e-5 facilitated smooth optimization.

Codebooks, essential for encoding and decoding speech features, exhibited effective quantizer convergence across all nine codebooks. The eval/clap metric of 0.239, though modest, validates semantically aligned latent audio synthesis. Overall, the model effectively captures shared and distinct acoustic features in English, Hindi, and Tamil, but improvements in intelligibility and code-switching accuracy remain areas for enhancement.

5.3. FINAL PIPELINE (Appendix 2: Pipeline of the models)

The integrated ASR, MT, and TTS pipeline enables real-time speech-to-speech translation but faces challenges in fluency, latency, and quality. The ASR model transcribes accurately under ideal conditions but struggles with accents and noise, improvable via noise reduction and accent-specific fine-tuning. The MT model translates effectively between English, Hindi, and Tamil, yet grammatical inconsistencies and nuance loss persist, needing better sentence restructuring and idiom handling. The TTS model produces clear, natural speech but lacks emotional depth and smooth multi-lingual prosody, addressable through enhanced phoneme synthesis and diverse training data. Overall, the system bridges language gaps well, with future gains possible through domain-specific tuning, latency optimization, and context-aware improvements.

Chapter 6

Challenges in Real-World Scenario

To provide successful real-time S2ST, the system should have the following characteristics:

a) Low Latency:

Real-time translation requires little delay to ensure a natural conversation pace. Latency must be below 1-2 seconds from the speaking moment to the translated spoken response.

b) High Accuracy and Robustness:

Each of the components (ASR, MT, and TTS) needs to accurately read and process speech with fewer errors. ASR models need to address varying accents and noise, MT models need to generate natural-sounding translations that are contextually correct, and TTS models need to read naturally from text.

c) Multilingual Support:

The system needs to support multiple languages with exact tokenization and translation models with consistency and correctness between languages. It needs to be trained on large and diverse datasets.

d) Scalability and Efficiency:

The system should be scalable and efficient to manage real-time requests for translation. Model inference must be optimized by quantization, pruning, or low-bit precision methods to meet real-time performance.

e) Error Handling and Correction:

Real-time systems need mechanisms for error correction and detection like re-ranking the translated candidates or using language models to correct low-confidence outputs.

Challenges in Real-Time S2ST

a) Latency-Accuracy Trade-off: Low latency tends to decrease accuracy with limited processing time, and high-accuracy models incur delay.
b) Noisy Speech Input: Noisy backgrounds and muffled speech decrease ASR accuracy and affect the whole pipeline.
c) Maintaining Context and Fluency: Real-time applications must preserve contextual coherence and avoid discontinuous or mechanical translating.
d) Resource Efficiency: Real-time systems must utilize memory and computational resources to the fullest to operate optimally on edge devices or cloud platforms.

Chapter 7

Conclusion

The real-time speech-to-speech translation system developed in this project demonstrates the potential of integrating Automatic Speech Recognition (ASR), Machine Translation (MT), and Text-to-Speech (TTS) into a pipeline. The ASR model transcribes speech into text, the MT model translates the content transcribed into the target language, and the TTS model generates speech output. By using multilingual datasets such as Common Voice, Opus-100, LJSpeech, and IndicTTS, the system handles English, Tamil, and Hindi languages. The use of Transformer-based architectures for ASR and MT, along with ParlerTTS model for TTS, ensures good translation accuracy and speech synthesis.

This project demonstrates how real-time speech translation can break language barriers by facilitating cross-lingual communication. Despite the challenges of preprocessing datasets and building the models for long hours, the system achieves satisfactory translation quality and speech synthesis. The experimental results highlight the importance of high-quality datasets, fine-tuned models, and efficient pre-processing techniques in achieving accurate and fluent translation.

Furthermore, the system shows the capability to be extended for multilingual support, making it scalable and adaptable to various real-world applications, such as healthcare, education, customer support, and multilingual virtual assistants. The project also sets the foundation for exploring emotion-aware speech translation, where the translated speech preserves the sentiment and tone of the original speaker.

Chapter 8

Future Work

The current speech-to-speech translation system provides a solid foundation, but there are several areas where it can be further enhanced:

1. Multilingual Fine-Tuning and Large-Scale Models
    - The system can be extended to support more languages by fine-tuning on larger multilingual datasets such as OPUS, CCMatrix, or No Language Left Behind (NLLB), which cover hundreds of languages.
    - Using larger and more powerful models, such as M2M-100, NLLB-200, or SeamlessM4T, could drastically improve translation quality by capturing more linguistic differences.
    - Increasing the number of training epochs and using parallelized multi-GPU or TPU hardware will allow the models to handle larger datasets more efficiently, leading to better generalization and accuracy.
2. Emotion and Sentiment-Preserving Translation
    - An important enhancement would be enabling the system to capture and translate emotions and sentiment along with the text.
    - This can be achieved by incorporating emotion recognition models (e.g., Wav2Vec2 with emotion labels) into the ASR pipeline. The recognized sentiment can be appended as a tag during the translation process, ensuring that the translated speech carries the same emotional tone.
    - Fine-tuning the MT model on emotion-labeled parallel datasets can help the system maintain sentiment consistency across languages.
3. Enhanced TTS with Speaker-Specific Voice Cloning
    - To make the TTS output more natural and expressive, speaker voice cloning techniques can be incorporated.
    - Models such as YourTTS, VITS, or Meta’s Voicebox can be fine-tuned to replicate the original speaker's voice, maintaining the pitch, tone, and speaking style in the translated output.

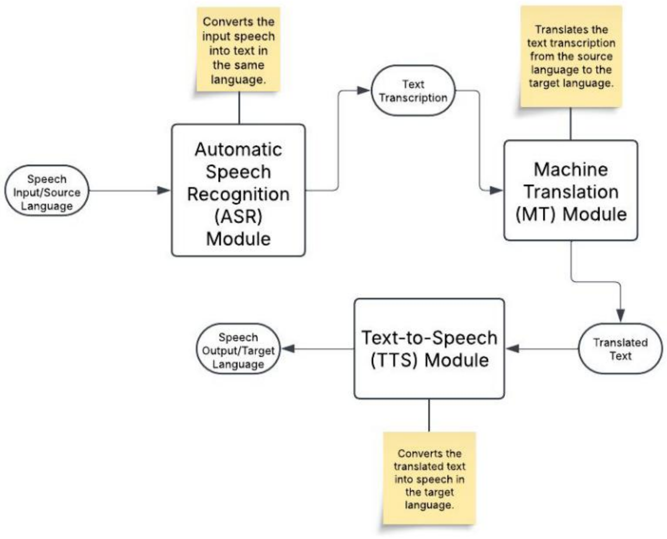
- This will significantly reduce the robotic nature of synthesized speech, making it sound more natural and preserving the speaker's vocal identity.

4. Context-Aware and Tone-Sensitive Translation
    - The system can be enhanced to understand contextual meaning and tone variations (e.g., sarcasm, formality) to improve translation accuracy.
    - By integrating contextual embeddings or using larger transformer models like GPT-4 or Gemini, the system can better interpret idiomatic expressions and contextual references.
    - The tone of speech can be recognized using prosody analysis models and mapped into the TTS synthesis process, making the translated speech more expressive.
5. Improved Real-Time Inference Efficiency
    - To ensure smoother real-time performance, the models can be optimized using quantization, pruning, and distillation techniques.
    - The ASR and MT models can be fine-tuned for faster inference speed without compromising accuracy, while the TTS model can utilize faster vocoders like HiFi-GAN or Multi-band MelGAN for near-instantaneous synthesis.
6. Integration with Real-World Applications
    - The system can be further developed into a cross-platform application for real-time translation in video calls, customer service, or live events.
    - It can also be adapted for accessibility solutions, providing real-time multilingual captions and voice translation for individuals with hearing or speech impairments.

By expanding the system to support more languages, preserving emotions, and replicating the speaker's voice, it can become a powerful multilingual speech translation solution. With larger models, more computational resources, and advanced speech synthesis techniques, the system can achieve highly natural and sentiment-aware translations, bridging linguistic and emotional gaps in cross-cultural communication.

Appendices

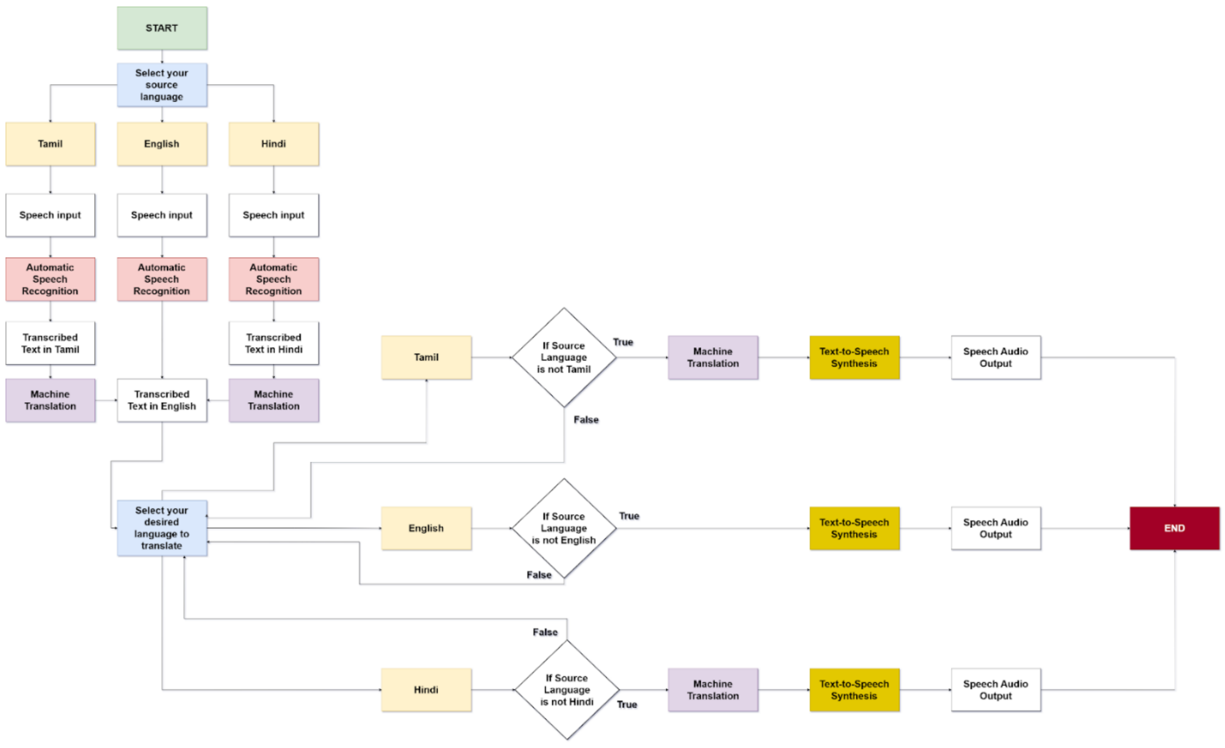
Appendix 1: Flowchart of the System



Converts the input speech into text in the same language.
Translates the text transcription from the source language to the target language.
Speech Input/Source Language
Automatic Speech Recognition (ASR) Module
Text Transcription
Machine Translation (MT) Module
Translated Text
Text-to-Speech (TTS) Module
Speech Output/Target Language
Converts the translated text into speech in the target language.

Figure 7

Appendix 2: Pipeline of the models



START
Select your source language
Tamil
English
Hindi
Speech input
Automatic Speech Recognition
Transcribed Text in Tamil
Transcribed Text in Hindi
Machine Translation
Transcribed Text in English
Select your desired language to translate
If Source Language is not Tamil
If Source Language is not English
If Source Language is not Hindi
True
False
Text-to-Speech Synthesis
Speech Audio Output
END

Figure 8

Appendix 3: ASR Model Architecture

```
Wav2Vec2ForCTC(
 (wav2vec2): Wav2Vec2Model(
  (feature_extractor): Wav2Vec2FeatureEncoder(
   (conv_layers): ModuleList(
    (0): Wav2Vec2LayerNormConvLayer(
     (conv): Conv1d(1, 512, kernel_size=(10,), stride=(5,))
     (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
     (activation): GELUActivation()
    )
    (1-4): 4 x Wav2Vec2LayerNormConvLayer(
     (conv): Conv1d(512, 512, kernel_size=(3,), stride=(2,))
     (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
     (activation): GELUActivation()
    )
    (5-6): 2 x Wav2Vec2LayerNormConvLayer(
     (conv): Conv1d(512, 512, kernel_size=(2,), stride=(2,))
     (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
     (activation): GELUActivation()
    )
   )
  )
  (feature_projection): Wav2Vec2FeatureProjection(
   (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
   (projection): Linear(in_features=512, out_features=1024, bias=True)
   (dropout): Dropout(p=0.0, inplace=False)
  )
  (encoder): Wav2Vec2EncoderStableLayerNorm(
   (pos_conv_embed): Wav2Vec2PositionalConvEmbedding(
    (conv): ParametrizedConv1d(
     1024, 1024, kernel_size=(128,), stride=(1,), padding=(64,), groups=16
```

```
          (parametrizations): ModuleDict(
            (weight): ParametrizationList(
              (0): _WeightNorm()
            )
          )
        )
        (padding): Wav2Vec2SamePadLayer()
        (activation): GELUActivation()
      )
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.0, inplace=False)
      (layers): ModuleList(
        (0-23): 24 x Wav2Vec2EncoderLayerStableLayerNorm(
          (attention): Wav2Vec2SdpaAttention(
            (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
          )
          (dropout): Dropout(p=0.0, inplace=False)
          (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (feed_forward): Wav2Vec2FeedForward(
            (intermediate_dropout): Dropout(p=0.0, inplace=False)
            (intermediate_dense): Linear(in_features=1024, out_features=4096,
bias=True)
            (intermediate_act_fn): GELUActivation()
            (output_dense): Linear(in_features=4096, out_features=1024, bias=True)
            (output_dropout): Dropout(p=0.0, inplace=False)
          )
          (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        )
      )
     )
    )
    (dropout): Dropout(p=0.0, inplace=False)
    (lm_head): Linear(in_features=1024, out_features=55, bias=True)
  )
```

Appendix 4: MT Model Architecture (This is for the English to Tamil translation)

The same model was used to fine-tune the remaining models with a slightly different hyperparameters like epoch count.

```
MBartForConditionalGeneration(
  (model): MBartModel(
    (shared): MBartScaledWordEmbedding(250054, 1024, padding_idx=1)
    (encoder): MBartEncoder(
      (embed_tokens): MBartScaledWordEmbedding(250054, 1024, padding_idx=1)
      (embed_positions): MBartLearnedPositionalEmbedding(1026, 1024)
      (layers): ModuleList(
        (0-11): 12 x MBartEncoderLayer(
          (self_attn): MBartSdpaAttention(
            (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
          )
          (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
          (activation_fn): GELUActivation()
          (fc1): Linear(in_features=1024, out_features=4096, bias=True)
          (fc2): Linear(in_features=4096, out_features=1024, bias=True)
          (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        ) )
      (layernorm_embedding): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
```

```
    (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
  )
  (decoder): MBartDecoder(
    (embed_tokens): MBartScaledWordEmbedding(250054, 1024, padding_idx=1)
    (embed_positions): MBartLearnedPositionalEmbedding(1026, 1024)
    (layers): ModuleList(
     (0-11): 12 x MBartDecoderLayer(
      (self_attn): MBartSdpaAttention(
        (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
        (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
        (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
        (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
      )
      (activation_fn): GELUActivation()
      (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
      (encoder_attn): MBartSdpaAttention(
        (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
        (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
        (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
        (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
      )
      (encoder_attn_layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
      (fc1): Linear(in_features=1024, out_features=4096, bias=True)
      (fc2): Linear(in_features=4096, out_features=1024, bias=True)
      (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
     ) )
    (layernorm_embedding): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
    (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
  ) )
  (lm_head): Linear(in_features=1024, out_features=250054, bias=False)
)
```

Appendix 5: TTS Model Architecture

```
ParlerTTSForConditionalGeneration(
  (text_encoder): T5EncoderModel(
    (shared): Embedding(32128, 768)
    (encoder): T5Stack(
      (embed_tokens): Embedding(32128, 768)
      (block): ModuleList(
        (0): T5Block(
          (layer): ModuleList(
            (0): T5LayerSelfAttention(
              (SelfAttention): T5Attention(
                (q): Linear(in_features=768, out_features=768, bias=False)
                (k): Linear(in_features=768, out_features=768, bias=False)
                (v): Linear(in_features=768, out_features=768, bias=False)
                (o): Linear(in_features=768, out_features=768, bias=False)
                (relative_attention_bias): Embedding(32, 12)
              )
              (layer_norm): T5LayerNorm()
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (1): T5LayerFF(
              (DenseReluDense): T5DenseGatedActDense(
                (wi_0): Linear(in_features=768, out_features=2048,
bias=False)
                (wi_1): Linear(in_features=768, out_features=2048,
bias=False)
                (wo): Linear(in_features=2048, out_features=768, bias=False)
                (dropout): Dropout(p=0.1, inplace=False)
                (act): NewGELUActivation()
```

```
                    )
                    (layer_norm): T5LayerNorm()
                    (dropout): Dropout(p=0.1, inplace=False)
                  ) ) )
                (1-11): 11 x T5Block(...)  # Repeated T5Block structure
              )
              (final_layer_norm): T5LayerNorm()
              (dropout): Dropout(p=0.1, inplace=False)
            ) )
          (audio_encoder): DACModel(
            (model): DAC(
              (encoder): Encoder(
                (block): Sequential(
                  (0): ParametrizedConv1d(1, 64, kernel_size=(7,), stride=(1,),
padding=(3,) ...)
                  (1): EncoderBlock(
                    (block): Sequential(
                      (0): ResidualUnit(
                        (block): Sequential(
                          (0): Snake1d()
                          (1): ParametrizedConv1d(64, 64, kernel_size=(7,),
stride=(1,), padding=(3,) ...)
                          (2): Snake1d()
                          (3): ParametrizedConv1d(64, 64, kernel_size=(1,),
stride=(1,) ...)
                        ) )
                      (1): ResidualUnit(...)  # Repeated ResidualUnit with dilation=(3,)
                      (2): ResidualUnit(...)  # Repeated ResidualUnit with dilation=(9,)
                      (3): Snake1d()
                      (4): ParametrizedConv1d(64, 128, kernel_size=(4,), stride=(2,),
padding=(1,) ...)
                    ) )
                  (2-4): 3 x EncoderBlock(...)  # Repeated EncoderBlock with
increasing channels and strides
                  (5): Snake1d()
                  (6): ParametrizedConv1d(1024, 1024, kernel_size=(3,), stride=(1,),
padding=(1,) ...)
               ) )
            (quantizer): ResidualVectorQuantize(
               (quantizers): ModuleList(
                  (0-8): 9 x VectorQuantize(
                     (in_proj): ParametrizedConv1d(1024, 8, kernel_size=(1,),
stride=(1,) ...)
                     (out_proj): ParametrizedConv1d(8, 1024, kernel_size=(1,),
stride=(1,) ...)
                     (codebook): Embedding(1024, 8)
                  ) ) )
            (decoder): Decoder(
               (model): Sequential(
                  (0): ParametrizedConv1d(1024, 1536, kernel_size=(7,), stride=(1,),
padding=(3,) ...)
                  (1): DecoderBlock(
                     (block): Sequential(
                        (0): Snake1d()
                        (1): ParametrizedConvTranspose1d(1536, 768, kernel_size=(16,),
stride=(8,), padding=(4,) ...)
                        (2): ResidualUnit(...)  # Repeated ResidualUnit structure
                        (3): ResidualUnit(...)  # Repeated with dilation=(3,)
                        (4): ResidualUnit(...)  # Repeated with dilation=(9,)
                     ) )
                  (2-4): 3 x DecoderBlock(...)  # Repeated DecoderBlock with
decreasing channels
                  (5): Snake1d()
                  (6): ParametrizedConv1d(96, 1, kernel_size=(7,), stride=(1,),
padding=(3,) ...)
                  (7): Tanh()
```

```
          ) ) ) )
      (decoder): ParlerTTSForCausalLM(
        (model): ParlerTTSModel(
          (decoder): ParlerTTSDecoder(
            (embed_tokens): ModuleList(
              (0-8): 9 x Embedding(1089, 1024)
            )
            (embed_positions): ParlerTTSSinusoidalPositionalEmbedding()
            (layers): ModuleList(
              (0-23): 24 x ParlerTTSDecoderLayer(
                (self_attn): ParlerTTSSdpaAttention(
                  (k_proj): Linear(in_features=1024, out_features=1024,
bias=False)
                  (v_proj): Linear(in_features=1024, out_features=1024,
bias=False)
                  (q_proj): Linear(in_features=1024, out_features=1024,
bias=False)
                  (out_proj): Linear(in_features=1024, out_features=1024,
bias=False)
                )
                (activation_fn): GELUActivation()
                (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
                (encoder_attn): ParlerTTSSdpaAttention(...)  # Repeated attention
structure
                (encoder_attn_layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
                (fc1): Linear(in_features=1024, out_features=4096, bias=False)
                (fc2): Linear(in_features=4096, out_features=1024, bias=False)
                (final_layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
              ) )
            (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          ) )
        (lm_heads): ModuleList(
          (0-8): 9 x Linear(in_features=1024, out_features=1088, bias=False)
        ) )
      (enc_to_dec_proj): Linear(in_features=768, out_features=1024, bias=True)
      (embed_prompts): Embedding(32128, 1024)
    )
```

Appendix 6: Code Snippets for ASR

i. Data Cleaning and Denoising

```
def reduce_noise(audio_file):
    y, sr = librosa.load(audio_file, sr=None)
    reduced_audio = nr.reduce_noise(y=y, sr=sr)
    return reduced_audio, sr

def spectral_noise_reduction(y, sr):
    D = librosa.stft(y)
    D_magnitude, D_phase = librosa.magphase(D)
    D_denoised = np.maximum(D_magnitude - 0.02, 0)
    y_denoised = librosa.istft(D_denoised * D_phase)
    return y_denoised

def clean_audio(sample):
    y, sr = librosa.load(sample["audio"]["path"], sr=None)
    y_clean = nr.reduce_noise(y=y, sr=sr)
    output_path = sample["audio"]["path"].replace(".wav", "_clean.wav")
    sf.write(output_path, y_clean, sr)
    sample["clean_path"] = output_path
    return sample
```

ii. Feature Extraction and Tokenization Preprocessing

```
def remove_special_characters(batch):
    batch["sentence"] = re.sub(chars_to_remove_regex, '', batch["sentence"]).lower()
    return batch
```

```
def extract_all_chars(batch):
    all_text = " ".join(batch["sentence"])
    vocab = list(set(all_text))
    return {"vocab": [vocab], "all_text": [all_text]}


tokenizer = Wav2Vec2CTCTokenizer.from_pretrained("facebook/wav2vec2-base-960h",
unk_token="<unk>", pad_token="<pad>", word_delimiter_token="|")

feature_extractor = Wav2Vec2FeatureExtractor(feature_size=1, sampling_rate=16000,
padding_value=0.0, do_normalize=True, return_attention_mask=True)

processor = Wav2Vec2Processor(feature_extractor=feature_extractor, tokenizer=tokenizer)
```

iii. Dataset Preparation and Data Collator

```
def prepare_dataset(batch):
    audio = batch["audio"]
    inputs = processor(audio["array"], sampling_rate=audio["sampling_rate"],
return_attention_mask=True)  # Ensure mask is returned
    batch["input_values"] = inputs.input_values[0]
    batch["input_length"] = len(inputs.input_values[0])
    batch["attention_mask"] = inputs.attention_mask[0]  # Add attention mask to batch
    batch["labels"] = tokenizer(batch["sentence"]).input_ids
    return batch


@dataclass
class DataCollatorCTCWithPadding:
    processor: Wav2Vec2Processor
    padding: Union[bool, str] = True
    def __call__(self, features: List[Dict[str, Union[List[int], torch.Tensor]]]) -> Dict[str,
torch.Tensor]:
        input_features = [{"input_values": feature["input_values"]} for feature in features]
        label_features = [{"input_ids": feature["labels"]} for feature in features]
        batch = self.processor.pad(
            input_features,
            padding=self.padding,
            return_tensors="pt",
        )
        with self.processor.as_target_processor():
            labels_batch = self.processor.pad(
            label_features,
            padding=self.padding,
            return_tensors="pt",
          )
        labels = labels_batch["input_ids"].masked_fill(labels_batch.attention_mask.ne(1), -100)
        batch["labels"] = labels
        return batch
```

Appendix 7: Code Snippets for MT

i. Preprocessing

```
from transformers import MBart50Tokenizer
# Initialize the tokenizer – for English as src lang – MbartTokenizer is used instead.
tokenizer = MBart50Tokenizer.from_pretrained("facebook/mbart-large-50")
# Preprocessing function
def preprocess_function(examples, src_lang, tgt_lang):
    inputs = [ex[src_lang] for ex in examples["translation"]]
    targets = [ex[tgt_lang] for ex in examples["translation"]]
    # Tokenize inputs and targets
    model_inputs = tokenizer(
        inputs, max_length=128, truncation=True, padding='max_length'
    )
    labels = tokenizer(
        targets, max_length=128, truncation=True, padding='max_length'
    )
    # Assign labels
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs


# Preprocess English-Tamil dataset
tokenized_train_ta_dataset = train_ta_dataset.map(lambda x: preprocess_function(x, "en",
"ta"), batched=True)
tokenized_val_ta_dataset = val_ta_dataset.map(lambda x: preprocess_function(x, "en", "ta"),
batched=True)
```

ii. Data Collator

```
from transformers import DataCollatorForSeq2Seq
# Initialize data collator
data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=model)
```

Appendix 8: Code Snippets for TTS

i. Dataset Preparation – Annotating the dataset

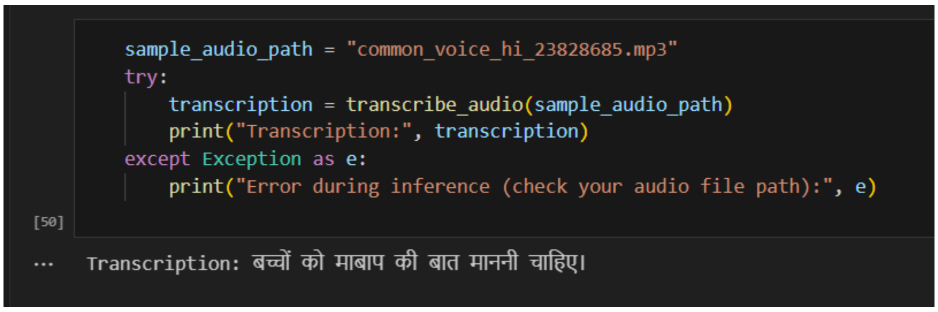
```
# Speaking Rate
# Signal to Noise Ratio (SNR)
# Reverbation
# Speech Monotony
!python main.py "SPRINGLab/IndicTTS-Hindi" \
  --configuration "default" \
  --text_column_name "text" \
  --audio_column_name "audio" \
  --cpu_num_workers 1 \
  --num_workers_per_gpu_for_pitch 1 \
  --rename_column \
  --repo_id "IndicTTS-Hindi-tags"
```

ii. Creating Natural Language Descriptions

```
#Helpful for the Speaker Prompts from the created features from our dataset
!python ./scripts/run_prompt_creation.py \
  --speaker_name "Priyanka" \
  --is_single_speaker \
  --dataset_name "SrihariGKS/IndicTTS-Hindi-tags" \
  --output_dir "./tmp_Priyanka" \
  --dataset_config_name "default" \
  --model_name_or_path "google/gemma-2-2b-it" \
  --per_device_eval_batch_size 12 \
  --attn_implementation "sdpa" \
  --dataloader_num_workers 2 \
  --push_to_hub \
  --hub_dataset_id "IndicTTS_Tamil-tagged" \
  --preprocessing_num_workers 2
```

Appendix 9: Results of models

i. ASR Hindi Sample output

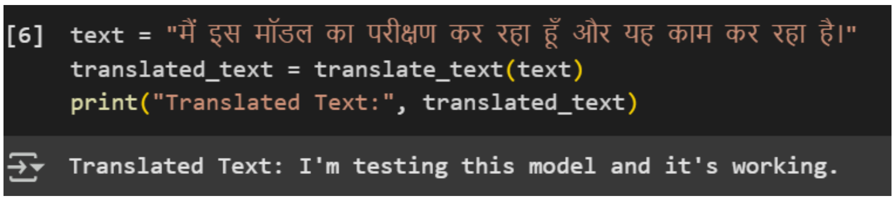


```
[50]
sample_audio_path = "common_voice_hi_23828685.mp3"
try:
    transcription = transcribe_audio(sample_audio_path)
    print("Transcription:", transcription)
except Exception as e:
    print("Error during inference (check your audio file path):", e)

... Transcription: बच्चों को माबाप की बात माननी चाहिए।
```

Figure 9

ii. MT Hindi to English Sample output

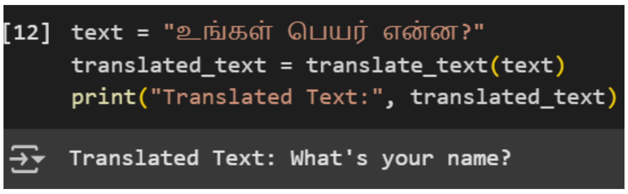


```
[6] text = "मैं इस मॉडल का परीक्षण कर रहा हूँ और यह काम कर रहा है।"
    translated_text = translate_text(text)
    print("Translated Text:", translated_text)

    Translated Text: I'm testing this model and it's working.
```

Figure 10

iii. MT Tamil to English Sample output

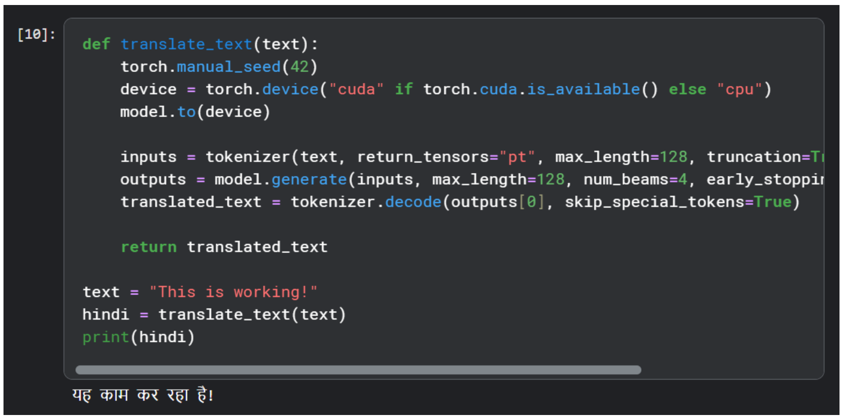


```
[12] text = "உங்கள் பெயர் என்ன?"
     translated_text = translate_text(text)
     print("Translated Text:", translated_text)

     Translated Text: What's your name?
```

Figure 11

iv. MT English to Hindi Sample output



```
[10]: def translate_text(text):
          torch.manual_seed(42)
          device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
          model.to(device)

          inputs = tokenizer(text, return_tensors="pt", max_length=128, truncation=Tı
          outputs = model.generate(inputs, max_length=128, num_beams=4, early_stoppiı
          translated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)

          return translated_text

      text = "This is working!"
      hindi = translate_text(text)
      print(hindi)

      यह काम कर रहा है!
```

Figure 12

REFERENCES

[1] Matsoukas, S., Bulyko, I., Xiang, B., Nguyen, K., Schwartz, R., & Makhoul, J. (2007, April). Integrating speech recognition and machine translation. In 2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07 (Vol. 4, pp. IV-1281). IEEE.

[2] Gibadullin, R. F., Perukhin, M. Y., & Ilin, A. V. (2021, May). Speech recognition and machine translation using neural networks. In 2021 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM) (pp. 398-403). IEEE.

[3] Zhou, B., Gao, Y., Sorensen, J., Déchelotte, D., & Picheny, M. (2003, November). A hand-held speech-to-speech translation system. In 2003 IEEE Workshop on Automatic Speech Recognition and Understanding (IEEE Cat. No. 03EX721) (pp. 664-669). IEEE.

[4] Nakamura, S., Markov, K., Nakaiwa, H., Kikui, G. I., Kawai, H., Jitsuhiro, T., ... & Yamamoto, S. (2006). The ATR multilingual speech-to-speech translation system. IEEE Transactions on Audio, Speech, and Language Processing, 14(2), 365-376.

[5] Tzoukermann, E., & Miller, C. (2018, March). Evaluating automatic speech recognition in translation. In Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 2: User Track) (pp. 294-302).

[6] Hudelson, P., & Chappuis, F. (2024). Using Voice-to-Voice Machine Translation to Overcome Language Barriers in Clinical Communication: An Exploratory Study. Journal of General Internal Medicine, 1-8.

[7] D. Shilvant, E. Sayyed, S. Jagdale, Prof. R. Waghmare (2024), Real Time Voice Translator, 42, Volume 4 Issue 1, April 2024, International Journal of Advanced Research in Science, Communication and Technology. (n.d.). Naksh Solutions. https://doi.org/10.48175/568

[8] V. Kharat, U. Chaudhari, K. Kesarwani (2022), Regional Language Translator Using Neural Machine Translation, 2, Volume 9 Issue 5 2022, International Journal Of Current Engineering And Scientific Research. https://doi.org/10.21276/ijcesr

[9] Vyas, R., Joshi, K., Sutar, H., & Nagarhalli, T. P. (2020, March). Real time machine translation system for english to indian language. In 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS) (pp. 838-842). IEEE.

[10] Dhawan, S. (2022). Speech to speech translation: Challenges and future. International Journal of Computer Applications Technology and Research, 11(03), 36-55.

[11] Tomita, M., Tomabechi, H., & Saito, H. (1990). Speech Trans: An Experimental Real-Time Speech-To-Speech Translation System. 어학연구.

[12] S. Chaudhari, A. Shukla, T. Gaware (2022), Real Time Direct Speech-to-Speech Translation, 104, Volume 9 Issue 1. January 2022, International Research Journal of Engineering and Technology. e-ISSN: 2395-0056

[13] Dong, Q., Huang, Z., Tian, Q., Xu, C., Ko, T., Zhao, Y., ... & Wang, Y. (2023). Polyvoice: Language models for speech to speech translation. arXiv preprint arXiv:2306.02982.

[14] Gu, J., Neubig, G., Cho, K., & Li, V. O. K. (2017). Learning to Translate in Real-time with Neural Machine Translation. Conference of the European Chapter of the Association for Computational Linguistics, 1, 1053–1062. https://doi.org/10.18653/V1/E17-1099.

[15] Limbu, S. H. (2020). Direct Speech to Speech Translation Using Machine Learning.