SPEECH RECOGNITION AND TRANSLATION FOR EXTREMELY LOW RESOURCE LANGUAGES

Report by

BASIL K RAJU

In Partial Fulfillment of the Requirements

for the Degree Of

MSc in Computer Science with Specialization in Machine Intelligence

Supervisor: Dr Elizabeth Sherly

School of Computer Science and Engineering

KERALA UNIVERSITY OF DIGITAL SCIENCES, INNOVATION AND TECHNOLOGY

06 July 2023

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **Speech Recognition and Translation for Extremely Low Resource Languages** submitted by **Basil K Raju (Reg. No: 211109)** in partial fulfillment of the requirements for the award of **Master of Science in Computer Science with Specialization in Machine Intelligence** is a bonafide record of the work carried out at **Virtual Resource Center for Language Computing Lab of Kerala University of Digital Sciences, Innovation and Technology** under my supervision.

Supervisor

Dr Elizabath Sherly

Distinguished Professor

KUDSIT

DECLARATION

I, **Basil K Raju** , student of **MSc in Computer Science With Specialization in Machine Intelligence**, hereby declare that this report is substantially the result of my own work , except, where explicitly indicated in the text and has been carried out during the period **February 2023 - June 2023**.

Place: Trivandrum

Date: 06-07-2023

Student's signature

Acknowledgements

We would like to express our utmost delight in presenting our work on ”Speech Recognition and Translation for Extremely Low Resource Languages.” a project centered around Natural Language Processing. We extend our heartfelt gratitude to all those who played a significant role in the successful completion of this endeavor.

First and foremost, we would like to express our deepest appreciation to Dr. Elizabeth Sherly from the Digital University of Kerala for her unwavering support and invaluable guidance throughout the project. Her expertise and insights were instrumental in shaping our work.We are also immensely grateful to Kavya Manohar and Leena G for their dedicated guidance and assistance throughout the project. Their expertise, feedback, and encouragement were pivotal in our progress and overall success.

Furthermore, we would like to acknowledge Wycliff India for providing the essential data required for the development of our project. Their generous contribution and support enabled us to effectively tackle the challenges and achieve our goals.

We would also like to extend our heartfelt thanks to all members of our department for their valuable input and support. In addition, we would like to express our sincere gratitude to our parents for their unwavering encouragement, love, and understanding.

Lastly, we would like to acknowledge the Almighty for His blessings, without which this project would not have been possible. We are grateful for the opportunities and resources provided, allowing us to complete this project successfully.

ABSTRACT

In this project, we are developing a system to save endangered low-resource languages. Building translation tools and Automatic Speech Recognition for these languages will help to preserve them. Transcribing the audio into its written form in the same language is essential for safeguarding cultural heritage, language documentation, linguistic research, education and communication. Transfer learning and fine-tuning methodologies are adopted for low-resource languages and also for languages that do not have any written forms. The languages which do not have any written form are transcribed into most similar languages. This project also involved collecting the required data to develop a functioning Speech Recognition and Translation system. The Automatic Speech Recognition Model got a Word Error Rate of 28% and the Machine Translation model got a Bleu score of 30.

LIST OF CONTRIBUTIONS

The main contributions to the project are listed below.

1. Automatic Speech Recognition Corpus for Malasar

   Created a corpus for speech recognition which is the first of its kind.

2. Machine Translation Corpus for Malasar to English

   For developing a machine translation model a Malasar to English parallel corpus has been created.

3. Automatic Speech Recognition Model For Malasar

   This is a novel work that includes a speech-to-text model for the Malasar language.

4. Malasar-to-English Translation Model

   This is a novel work that will translate the Malasar language into English.

5. Website for Malasar Speech Recognition

   This project aims to deploy the developed automatic speech recognition model, which can be accessed by a website.

Contents

List of Figures **ix**

List of Tables **1**

1 Introduction **2**

1.1 Introduction 2

1.2 Main Areas of the Report 2

1.2.1 Corpus Creation 3

1.2.2 Automatic Speech Recognition(ASR) 3

1.2.3 Machine Translation (MT) 3

1.3 Motivation 4

1.4 Organization of the Report 4

2 Literature Review **5**

2.1 Low Resource Languages 5

2.2 Automatic Speech Recognition 6

2.3 Machine Translation 7

2.4 Summary 8

3 Methodology **9**

3.1 Introduction 9

3.2 Data and Its Source for Malasar Language. 10

3.3 Corpus Creation 11

3.3.1 Corpus Creation For ASR 12

3.3.1.1 Creating Synthetic Speech for ASR 12

3.3.2 Corpus Creation For MT 13

3.4 Why Transformer Based Models for Automatic Speech Recognition and Machine Translation for Low Resources Languages? . . . 14
3.5 Using Wav2vec Pretrained Model as First Approach to Build ASR. . . . 16
3.5.1 Creating JSON Files And Dataset Cleaning . . . 17
3.5.2 Extraction Of Unique Characters . . . 18
3.5.3 Tokenizer and Feature Extractor . . . 19
3.5.4 Create Wav2Vec2CTCTokenizer . . . 19
3.5.5 Create XLSR-Wav2Vec2 Feature Extractor . . . 20
3.5.6 Preprocess Data . . . 21
3.5.7 Training . . . 21
3.6 Proposed Architecture of Transformer Based Model for ASR . . . 22
3.6.1 Load Dataset . . . 27
3.6.2 Prepare Feature Extractor . . . 27
3.6.3 Preparing Tokenizer . . . 28
3.6.4 Prepare Data . . . 29
3.6.5 Define Data Collater . . . 29
3.6.6 Training . . . 30
3.6.7 Evaluation Metric For ASR . . . 30
3.7 Transformer Based Machine Translation. . . . 30
3.7.1 Architecture of Transformer Based Model for MT . . . 31
3.7.2 Load Dataset . . . 32
3.7.3 Preprocessing the Data . . . 32
3.7.4 Training . . . 33
3.7.5 Evaluation Metric For MT . . . 33
3.8 Summary . . . 33

4 Results and Discussions **34**
4.1 Results of ASR . . . 34
4.2 Results of MT . . . 36
4.3 ASR Model Deployment on WEB . . . 37

5 Conclusions and Future Work **39**
5.1 Conclusions . . . 39
5.2 Future Work . . . 39

References **40**

List of Figures

3.1 Proposed System. 9
3.2 Lexical Similarity Of Malasar. 10
3.3 Spectrogram, Pitch and Intensity, Pulse and Formats Analysis of the Audio. 11
3.4 Structure of Corpus Created for ASR. 13
3.5 Structure of Corpus Created for MT. 14
3.6 Random Elements After Removing Special Characters 18
3.7 Unique Integer Index to Each Character 19
3.8 Overview of Sequence-to-Sequence Transformer Model[1] 23
3.9 Left: sampled 1-dimensional audio signal. Right: corresponding log-Mel spectrogram.[2] . . . 28
3.10 Encoding and Decoding Using Tokenzier 29
3.11 Transformer Based MT Model 32

4.1 Maximum Step vs WER 35
4.2 Training and Validation Loss 35
4.3 Transcription Using Trained Model 35
4.4 Epoch vs BLEU vs Generated Length 36
4.5 Machine Translation Using Trained Model. 37
4.6 Web Application: Home Page 38
4.7 Web Application: ASR Page 38

List of Tables

3.1 ASR Dataset Using Original Audios Details 12
3.2 ASR Synthetic Dataset Details 13
3.3 Machine Translation Dataset Details 13

4.1 WER of Different Indic Languages Using Transformer Models 36
4.2 BLEU rates of Indic languages 37

Chapter 1

Introduction

1.1 Introduction

Automatic Speech Recognition (ASR) and Machine Translation (MT)technologies have been employed to document endangered languages by transcribing and translating spoken language data. These resources help linguists and communities preserve linguistic knowledge and create comprehensive language documentation. A project is underway to document the Malasar language and community under the Scheme for Protection and Preservation of Endangered Languages (SPPEL). The project will result in the creation of a grammar of the Malasar language, a trilingual electronic dictionary of Malasar, English, and Tamil, and an ethnolinguistic profile of the Malasar community.

Economic, Political, Social and Educational factors lead a language to be endangered. Thus leading to the loss of Cultural Heritage, Linguistic Diversity, Cognitive Diversity and many Opportunities. Since Malasar has extremely low resources and the number of speakers is less, the Malasar language needs to be preserved by all means. This project aims to build an Automatic Speech recognition and Machine translation system for the Malasar language.

1.2 Main Areas of the Report

This report gives detailed information about the three domains of Natural Language Processing in the context of saving endangered languages. Corpus Creation, Automatic Speech Recognition and Machine Translation are the three main areas of work done.

1.2.1 Corpus Creation

Corpora play a vital role in natural language processing (NLP) as they serve as extensive collections of text or speech data used to train and evaluate NLP systems. These comprehensive datasets provide valuable insights into the patterns, grammar, and contextual dependencies of a language. The corpus creation process involves carefully collecting and organizing diverse and representative data to meet the specific objectives of the NLP task at hand.

Corpora are typically collected through a variety of methods. Web scraping, accessing public datasets, utilizing APIs, and conducting surveys or interviews are some common approaches. The data is then subjected to a cleaning process to remove any irrelevant or noisy content, ensuring the corpus's quality and consistency. However, the significance of corpora goes beyond training and evaluating NLP systems. They serve as a rich resource for developing new NLP algorithms and advancing research in language processing. By studying corpora, researchers gain valuable insights into linguistic phenomena, discover new language patterns, and develop algorithms that can accurately analyze and interpret text or speech data.

Preserving corpora is of utmost importance for the NLP community. Corpora are not only valuable for the present research but also serve as a historical record of language usage and evolution. By preserving corpora, future researchers can build upon previous work and conduct comparative studies to observe language changes over time. Moreover, preserving corpora helps in reproducibility and transparency. By documenting the corpus creation process, including data sources, cleaning methods, and annotation guidelines, researchers ensure the ability to reproduce and validate results. This fosters scientific rigour and allows for the advancement of the field. Ethical considerations also come into play when preserving corpora. It is essential to handle data with respect to privacy regulations, intellectual property rights, and ethical guidelines. Care must be taken to avoid biases and ensure fairness and inclusivity in the collected data.

1.2.2 Automatic Speech Recognition(ASR)

Automatic Speech Recognition (ASR) is the technology that allows computers to understand and transcribe spoken language into text. ASR systems work by first converting the spoken language into a digital signal. This signal is then analyzed by the ASR system to identify the individual words that were spoken. The ASR system then uses a dictionary and grammar to determine the meaning of the words and generate a transcript of the spoken language. The accuracy of ASR systems has improved significantly in recent years. However, there are still some challenges that need to be addressed.

1.2.3 Machine Translation (MT)

Machine translation (MT) is the process of automatically translating text or speech from one language to another. It is a sub-field of computational linguistics that investigates the use of software to translate text or

speech from one language to another. MT systems work by first converting the source language text into a sequence of words and phrases. This sequence is then analyzed by the MT system to identify the meaning of the text. The MT system then uses a dictionary and grammar to generate a translation of the text in the target language.

1.3 Motivation

Languages are part of a culture's heritage and identity. There are many languages on the edge of existence due to several reasons. Developing an Automatic Speech Recognition and Machine Translation system for such low-resourced languages will help to document the language and preserve the knowledge it contains. And also, this will help the governing bodies to a great extent and help the community to engage with the society.

1.4 Organization of the Report

The main areas of the project are discussed in Chapter 1. The importance of corpus creation for NLP algorithms and the possibility of using ASR and MT models for saving endangered languages are discussed in this chapter. Chapter 2 discusses the previous works carried out by different researchers for preserving low-resource languages, and also the latest achievements in the ASR and MT are discussed in this chapter. The methodology and the experiments are explained in Chapter 3. This chapter also discusses the architecture used for developing ASR and MT. The results of the trained models and their interpretations are listed in Chapter 4. The future scope of building such systems for the low resources and conclusions of the entire work is discussed in Chapter 5.

Chapter 2

Literature Review

2.1 Low Resource Languages

Low-resource languages (LLRs) are languages with limited resources, such as data, tools, and expertise. Working with LLRs presents a number of challenges, including data scarcity, tool scarcity, and expertise scarcity. Past work has addressed these challenges by using techniques such as data augmentation, transfer learning, and domain adaptation[3]. Neural Machine Translation (NMT) has witnessed notable advancements, making it a viable option for low-resource languages (LRLs). However, selecting the most suitable NMT technique for a given data specification in this rapidly evolving field can be challenging, exacerbated by the lack of established guidelines for LRL-NMT selection. [4] endeavours to provide a comprehensive understanding of the LRL-NMT landscape by highlighting recent technological trends and offering guidance for selecting appropriate techniques based on specific data specifications. By examining current advancements, the paper offers valuable insights into state-of-the-art approaches and their relevance to LRL contexts. To further facilitate progress in LRL-NMT, the paper emphasizes the significance of creating more LRL resources, including datasets and tools. These resources are crucial for training and evaluating NMT models tailored to LRLs. Additionally, the paper encourages the open availability of computational resources and trained models, fostering collaboration and enabling researchers to build upon existing work. The paper also underscores the importance of engaging regional research communities to advance LRL-NMT solutions. By involving researchers from LRL contexts, it becomes possible to address language-specific challenges and develop techniques that are better aligned with the linguistic characteristics of these languages. Regional collaboration promotes the creation of contextually relevant solutions and supports the preservation of linguistic diversity. Furthermore, the paper suggests potential avenues for improving existing NMT techniques specifically for LRL pairs. These improvements may encompass architectural modifications, data augmentation techniques, or advancements in transfer learning approaches. By addressing the unique attributes of LRLs, these enhancements would contribute to the

development of more accurate and effective NMT systems for LRLs.

2.2 Automatic Speech Recognition

[5] discusses how to develop automatic speech recognition (ASR) systems for languages with very limited resources. These systems can be used for tasks such as categorizing and searching audio content. The paper proposes a universal phone modelling approach, which involves adapting universal phone models using very small amounts of transcribed speech. The DARPA LORELEI program provides a framework for studying such low-resource ASR systems. The paper presents Kaldi-based systems for the program and describes methods for rapidly adapting the universal ASR system. The results obtained by the authors significantly outperform results obtained by many competing approaches on the NIST LoReHLT 2017 Evaluation datasets.

[6] compared the performance of Transformer and LSTM encoder-decoder models for end-to-end speech recognition. They found that the Transformer model performed similarly to the LSTM model but required less training time. They also observed that the Transformer training was more stable but had more problems with generalization. The authors used data augmentation to improve the performance of both models. They also analyzed several pretraining and scheduling schemes, which were crucial for both models. They improved their LSTM model by adding additional convolutional layers. They achieved state-of-the-art performance on the TED-LIUM-v2 dataset for attention-based end-to-end models. The authors limited the training on the LibriSpeech dataset to 12.5 epochs to keep the results of practical interest. However, they showed that longer training time still improves the performance of both models. Overall, the paper provides insights into the performance of Transformer and LSTM models for end-to-end speech recognition and suggests ways to improve their performance.

[7] proposes a Transformer-based online CTC/attention end-to-end speech recognition architecture. It builds upon previous research in the field of automatic speech recognition (ASR) and addresses the challenge of deploying a Transformer-based end-to-end (E2E) model for online speech recognition. The authors also propose a state reuse chunk-SAE to reduce computational costs and improve performance. The proposed model is evaluated on the HKUST Mandarin ASR benchmark and achieves a 23.66% character error rate (CER) with a 320 ms latency. The paper does not provide a detailed literature survey of previous research in the field of ASR.

[8] proposes a new method for unsupervised pre-training of speech recognition models. The method, called wav2vec, is based on a convolutional neural network that is trained to predict future audio samples from past audio samples. The authors show that wav2vec can be used to improve the performance of supervised speech recognition models, even when the supervised data is limited. The paper builds on previous work on unsupervised pre-training for speech recognition, such as the Listen, Attend and Spell (LAS) model by Chan et al. (2016) and the Conv-TasNet model by Luo et al. (2018). However, wav2vec differs from these previous models in a number of ways. First, wav2vec uses a convolutional neural network instead of a recurrent neural

network. This makes it more efficient to train and allows it to learn longer-range dependencies in the audio signal. Second, wav2vec uses a masked prediction loss function that encourages the model to learn representations of the audio signal that are robust to noise and other distortions. The authors evaluated wav2vec on a number of speech recognition tasks, including the LibriSpeech dataset and the WSJ0 dataset. They showed that wav2vec can improve the performance of supervised speech recognition models by up to 5% absolute on the LibriSpeech dataset. They also showed that wav2vec can be used to learn representations of speech that are useful for a variety of tasks, such as speaker identification and emotion recognition.
[1] proposes a new approach to speech recognition that is based on large-scale weak supervision. The approach, called ”Robust Speech Recognition via Large-Scale Weak Supervision”, uses a transformer-based model that is trained on a massive dataset of audio recordings and transcripts. The transcripts are not fully aligned with the audio recordings, which means that the model must learn to infer the missing information. The authors evaluated the proposed approach on a number of speech recognition tasks, including the LibriSpeech dataset and the Switchboard dataset. They showed that the approach can achieve state-of-the-art results on these tasks, even when the amount of training data is limited. The paper makes a number of contributions to the field of speech recognition. First, it proposes a new approach to speech recognition that is more robust to noise and other distortions. Second, it shows that the proposed approach can achieve state-of-the-art results on a number of speech recognition tasks, even when the amount of training data is limited. Third, it provides a new way to use large-scale datasets for speech recognition.

2.3 Machine Translation

[9] proposes a new approach to machine translation (MT) that aims to address the problem of limited data availability for many languages. The approach, called Human-Centered Machine Translation (HCMT), combines the strengths of both human and machine translation.HCMT begins with a traditional MT system that is trained on a large dataset of parallel text. The output of the MT system is then reviewed by a human translator, who can correct errors and make improvements. The revised output is then used to retrain the MT system, which in turn produces better output for the next round of human review. The authors of the paper evaluated HCMT in a number of different languages, including African languages that have relatively little data available for MT. They showed that HCMT can produce high-quality translations for these languages, even when the amount of training data is limited. The paper makes a number of contributions to the field of MT. First, it proposes a new approach to MT that is more effective for languages with limited data. Second, it shows that HCMT can be used to produce high-quality translations for a wide range of languages. Third, it provides a framework for integrating human and machine translation in a way that can improve the quality of translations.

[10] introduces the Helsinki Neural Machine Translation (HNMT) system, which was developed by re-

searchers at the University of Helsinki. HNMT is a neural machine translation system that uses a convolutional neural network to translate text from one language to another. The authors of the paper evaluated HNMT on a number of different language pairs, including English-Finnish, English-German, and English-French. They showed that HNMT can produce high-quality translations that are comparable to those produced by human translators. The paper makes a number of contributions to the field of machine translation. First, it proposes a new neural machine translation system that is more effective than previous systems. Second, it shows that neural machine translation can be used to produce high-quality translations for a wide range of language pairs. Third, it provides a detailed analysis of the performance of HNMT on a number of different tasks.

2.4 Summary

The field of Automatic Speech Recognition and Machine Translation shows drastic changes. From a Rule-based models to Machine Learning models, especially Transformer based sequencetosequence models are the latest among them. There are no works done in the field of ASR and MT for the Malasar languages. There are many works done for the low resources languages using sequencetosequence models. This is the first work done for the Malasar ASR and Malasar MT.

Chapter 3

Methodology

3.1 Introduction

This project aims to build an Automatic Speech Recognition and Machine Translation system for the Malasar language. Along with that, the Corpus needed for each task is created. The entire project can be divided into three parts:

1. Corpus Creation for ASR and MT Task.
2. Automatic Speech Recognition of Malasar.
3. Translation of Malasar to English.

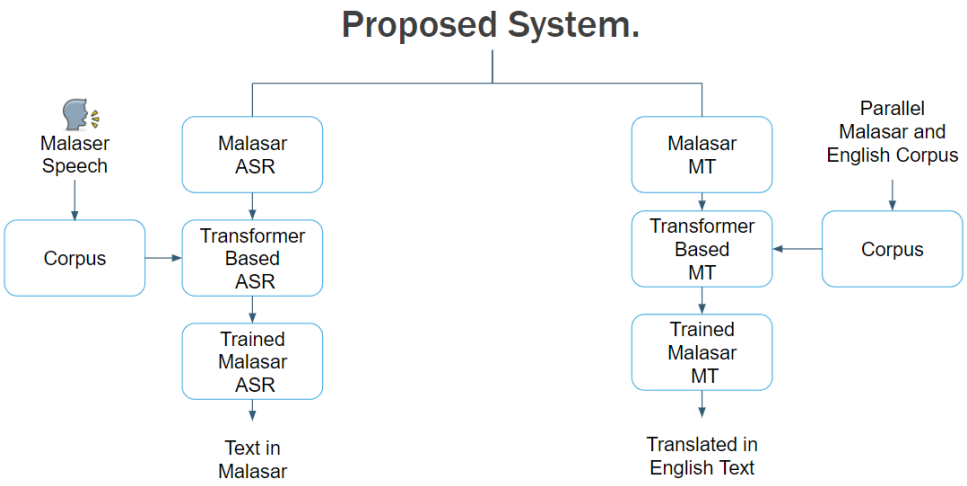

Figure 3.1: Proposed System.

Figure 3.1 shows the overview of the proposed system. Two different Corpus is created from the raw data for each separate task. Then the Transformer based encoder-decoder model is trained on the created Corpus. Finally, the two models are evaluated using different evaluation metrics.

3.2 Data and Its Source for Malasar Language.

The data which are needed for the creation of the Corpus are received from Wycliffe India. Wycliffe India is an interdenominational, non-sectarian and non-profit mission organization.

The data source for the Malasar language consists of 22 audio recordings of Malasar speech, along with their transcriptions. The audio recordings are a total of 5 hours in length. In addition to the audio recordings, the data source includes a word list and dictionary of the Malasar language. The speakers consist of males and females between the age of 30-35. The context of the audio is stories and histories. The audio quality is excellent.

The Malasar have lexical similarity with up to 6 different languages. The lexical similarity between Irula and other languages suggests that it is closely related to these languages. The lexical similarity between Irula and Malasar is 68–74%, which is the highest of all the languages listed. This suggests that Irula and Malasar share a common ancestor. The lexical similarity between Irula and Tamil and Malayalam is lower, at 65% and 61% respectively. This suggests that these languages have been in contact for a shorter period of time. The Attapady dialect of Irula has a lexical similarity of 67–72% with Irula, while the Walayar dialect has a lexical similarity of 75%. This suggests that the Attapady dialect is more closely related to the Irula language than the Walayar dialect. The implications of these lexical similarities for the understanding of the history and relationships of the Irula language are complex and still being debated by linguists. However, the data suggests that Irula is a language with a rich history and that it is closely related to other Dravidian languages. compatibility, and technical characteristics of the audio file. The graphical representation can be seen in figure 3.2.

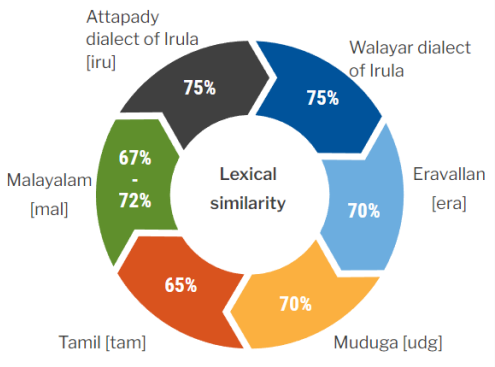

Figure 3.2: Lexical Similarity Of Malasar.

Analyzing audio can involve various aspects such as spectrogram analysis, pitch and intensity analysis,

pulse analysis, and format analysis. A spectrogram is a visual representation of the frequencies present in an audio signal over time. It displays how the intensity of different frequencies changes over the duration of the audio. Spectrogram analysis helps in identifying specific frequency components, patterns, and variations in the audio signal. Pitch refers to the perceived frequency of a sound and is commonly associated with the musical term ”high” or ”low” pitch. Intensity, on the other hand, refers to the loudness or amplitude of the sound. Pitch and intensity analysis involves examining changes in pitch and intensity over time, identifying pitch contours, detecting pitch variations, or analyzing the relative loudness of different sections of the audio. Pulse refers to the rhythmic aspects of sound, such as beats or pulses in music or speech. Pulse analysis involves identifying and analyzing rhythmic patterns, tempo, or beats in the audio. It can be useful for studying music, rhythm patterns in speech, or identifying specific characteristics of the audio signal related to timing and timing variations. Format analysis refers to the examination of the audio file format or the encoding scheme used to store the audio data. It involves understanding the technical specifications of the audio file, such as the sample rate, bit depth, compression method (if any), and file container format (e.g., WAV, MP3, FLAC). Format analysis helps in determining the quality, compatibility, and technical characteristics of the audio file. The graphical representation can be seen in the figure3.3

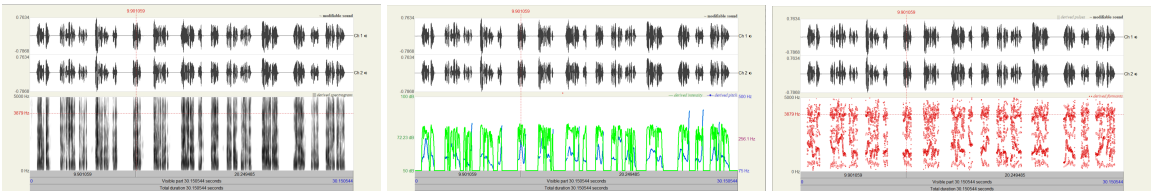

Figure 3.3: Spectrogram, Pitch and Intensity, Pulse and Formats Analysis of the Audio.

3.3 Corpus Creation

Corpus creation involves gathering and curating diverse and representative text data from various sources, such as books, articles, websites, and more. It is crucial for training transformer-based models as it provides the necessary training data, language understanding, and contextual knowledge. It enables the model to generate contextually relevant responses while adaptable to specific domains or tasks. In this project, we create a corpus for three different NLP tasks: Speech recognition, machine translation and speech translation. The three tasks need three separate corpora for training purposes. No corpus is available for the Malasar language for any of the functions. This is the first done for the corpus creation.

3.3.1 Corpus Creation For ASR

The Malasar text is similar to Tamil text, so we used Google Translate API to create the corresponding phonemes, thus enabling us to read the text along with the audio played. This step was crucial since the model we going to make only accepts audios of a maximum of 30 seconds. The 5 hours of audio are split into smaller chunks of less than 30 seconds using Audacity by manually reading the phonemes. The creation of phonemes made corpus creation much easier by enabling readability for those who are not familiar with Malasar Language.

Total Duration	5 Hours
Utterance	965
Speakers	1 Female,1 Male
Domain	Malasar Language and Community
Sample Rate	16KHz
Audio Length	Maximum 30 seconds
Structure	[Malasar_Synthetic _Audio : Malasar_Txt]

Table 3.1: ASR Dataset Using Original Audios Details

3.3.1.1 Creating Synthetic Speech for ASR

Automatic Speech Recognition for extremely low resources languages could benefit from creating a corpus using synthetic Speech. There was no audio for the Malasar text in the word list and Malasar Dictionary. The next step was to generate audio from the text. For that, the pyttsx3 python library was our first choice, the pyttsx3 was not able to generate synthetic Malasar audio. The audio generated by the Google Translate API was very similar to the Malasar pronunciation. The similarity is cross-validated by the linguistics from Wycliffe's end. After all, the generated audio will not be equal to the original high-quality audio, but it could help the model to generalise and predict the tokens from the audio. The latest text-to-speech technology can produce speech that is almost indistinguishable from human speech.[11].

From all the sources created three different TSV files for three separate tasks. Speech Recognition only needed two files in audio and its corresponding transcript in Malasar. For Speech Translation, one more field is needed, the target language to which Malasar needs to be translated. For Machine Translation, the source language and its corresponding text are in the target language. Once the TSV files are created, we use DatasetDict() constructor to create the corpus, which can be directly fed into the sequence-to-sequence models. The created corpus is uploaded to the Hugging Face hub, increasing the dataset's visibility and re-usability.

Total Duration	4.5 Hours
Utterance	17,700
Domain	Malasar Dictionary Words
Sample Rate	16KHz
Audio Length	1 second
Structure	[Malasar_Synthetic _Audio : Malasar_Txt]

Table 3.2: ASR Synthetic Dataset Details

```
DatasetDict({
    train: Dataset({
        features: ['audio_path', 'sentence'],
        num_rows: 16850
    })
    test: Dataset({
        features: ['audio_path', 'sentence'],
        num_rows: 1873
    })
})

{'audio_path': {'path': 'luke00இவருகனோடவு.wav',
  'array': array([ 4.26325641e-14, -6.57252031e-14, -5.86197757e-14, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00]),
  'sampling_rate': 16000},
 'sentence': 'இவருகனோடவு'}
```

Figure 3.4: Structure of Corpus Created for ASR.

Corpus uploaded on the Hugging Face Hub is stored in the Apache Arrow format. When the corpus is called using the load function corpus is loaded in exact same format, which allows the developer to use the corpus without any hassle.

3.3.2 Corpus Creation For MT

For the machine translation task, the Malasar text and its available corresponding English text are combined together for creating the parallel corpus. The details of the created corpus are listed below:

Utterance	905
Domain	Malasar Language and Communities
Structure	[Malasar_Txt : Eng_Txt]
v109	32

Table 3.3: Machine Translation Dataset Details

```
DatasetDict({
    train: Dataset({
        features: ['Transcript', 'Target'],
        num_rows: 814
    })
    validation: Dataset({
        features: ['Transcript', 'Target'],
        num_rows: 91
    })
})
```

Figure 3.5: Structure of Corpus Created for MT.

3.4 Why Transformer Based Models for Automatic Speech Recognition and Machine Translation for Low Resources Languages?

Transformers have gained significant popularity in Automatic Speech Recognition (ASR) and Machine Translation tasks also Transformers can be particularly beneficial for ultra-low resource languages and learning from small corpora due to several reasons:

- Modeling Long-Term Dependencies: ASR and Machine Translation involves processing and understanding long sequences of input data. Traditional recurrent neural networks (RNNs) struggle with capturing long-term dependencies efficiently, leading to difficulties in accurately transcribing or translating the input. Transformers, on the other hand, are designed to model long-term dependencies effectively. Their self-attention mechanism allows them to focus on relevant parts of the input at each time step, enabling better capturing of contextual information.

- Parallelization: Transformers can be parallelized more efficiently compared to RNNs. This is because each token's representation in a transformer can be computed independently, allowing for parallel computation across different positions in the input sequence. This parallelization capability makes transformers computationally efficient, enabling faster training and inference.

- Self-Attention Mechanism: Transformers employ a self-attention mechanism that enables the model to attend to different parts of the input sequence during the encoding process. This attention mechanism allows the model to identify and prioritize important features or words in the input sequence. In ASR and Machine Translation tasks, this attention mechanism is particularly useful as it helps the model to focus on relevant parts of the speech or text for accurate transcription or translation.

- Multilingual Capabilities: Transformers have been successfully applied in multilingual ASR and Machine Translation tasks. By leveraging the shared architecture of transformers, a single model can be used to handle multiple languages. This approach is more efficient and cost-effective than building separate models for each language. Transformers can learn to encode and decode different languages by

capturing the underlying patterns and representations in the input data, making them highly adaptable for multilingual tasks.

- Contextual Understanding: Transformers excel in capturing contextual information. They have the ability to consider the entire input sequence simultaneously, which helps in better understanding the context and relationships between different words or speech segments. This contextual understanding leads to improved accuracy in ASR and Machine Translation tasks by enabling the model to make more informed predictions based on the surrounding context.

- Transfer Learning: Transformers have shown remarkable success in transfer learning. Pre-training a transformer model on a large dataset from a high-resource language can capture general language patterns, phonetic features, and linguistic structures. This pre-trained model can then be fine-tuned on a smaller dataset or even on an ultra-low resource language. By leveraging the pre-trained knowledge, the transformer can effectively learn from limited data and achieve better performance in low-resource scenarios.

- Cross-lingual Knowledge Transfer: Transformers can facilitate cross-lingual knowledge transfer. Through pre-training on a high-resource language, the transformer learns to encode meaningful representations of language and can generalize well to different languages. This cross-lingual knowledge transfer allows the model to transfer its understanding of linguistic patterns and structures from resource-rich languages to resource-poor or low-resource languages, improving the model's performance even with limited training data.

- Data Augmentation: Transformers can benefit from data augmentation techniques. When the amount of training data is limited, data augmentation becomes crucial to artificially increase the diversity of the training examples. With the ability to model long-term dependencies and capture contextual information, transformers can generate augmented data effectively. Techniques such as speed perturbation, noise injection, or pitch shifting can be applied to the training data, creating additional samples that enhance the model's robustness and generalization capabilities.

- Multilingual Training: Transformers can be trained in a multilingual setting, combining data from multiple languages during training. By jointly training the model on data from different languages, the model can learn shared representations and improve its generalization across languages. This approach enables the model to leverage the information from other resource-rich languages to enhance the performance in low-resource languages.

- Adaptability to Small Corpus: Transformers can adapt well to small corpora due to their ability to capture intricate patterns and dependencies within the available data. With their self-attention mechanism, transformers can effectively focus on the important features and context within the limited training corpus.

Additionally, techniques such as regularization, parameter sharing, and model architecture modifications can be employed to mitigate overfitting and improve generalization on small datasets.

Overall, transformers offer a powerful architecture for ASR and Machine Translation tasks due to their ability to model long-term dependencies, efficient parallelization, self-attention mechanism, multilingual capabilities, and contextual understanding. Transformers also offer advantages for ultra-low resource languages and learning from small corpora. These advantages have made transformers a popular choice for various natural language processing tasks, including ASR and Machine Translation.

3.5 Using Wav2vec Pretrained Model as First Approach to Build ASR.

The methodology for developing an ASR system using the wav2vec model would involve collecting a dataset of audio recordings in Malasar and transcribing them manually to create a dataset of paired audio and text data. This dataset would then be split into training, validation, and testing sets, and the wav2vec model would be fine-tuned on the training set.

Various techniques could be used to optimize the performance of the model, such as adjusting the learning rate, using data augmentation to generate additional training data, and applying language-specific features such as phoneme embeddings or a language model to the model.

The results of the ASR system could be evaluated by comparing the output of the model to the ground truth transcription on the test set. Metrics such as word error rate (WER) or character error rate (CER) could be used to assess the accuracy of the model.

Developing an Automatic Speech Recognition (ASR) system using the wav2vec model involves several key steps as below:

1. Data Collection: The first step is to collect a dataset of audio recordings in the Malasar language. The dataset should be representative of the domain or application for which the ASR system is being developed. The dataset should include both clean and noisy audio recordings, covering a range of accents, speaking styles, and genders.

2. Data Preprocessing: The audio files in the dataset are preprocessed to convert them into a format that the model can process. This involves converting the audio files into a numerical representation, such as Mel Frequency Cepstral Coefficients (MFCCs).

3. Data Labeling: Once the audio files are preprocessed, the next step is to transcribe them into text. This involves manually labelling each audio file with its corresponding text transcription. This labelled dataset will be used to train and evaluate the ASR system.

4. Data Splitting: The labelled dataset is divided into three subsets: training, validation, and testing. The training set is used to train the model, the validation set is used to tune hyper-parameters and prevent overfitting, and the testing set is used to evaluate the final performance of the ASR system.

5. Model Selection: The wav2vec model is a pre-trained model, which can be fine-tuned on a specific language or domain. This model can be used as a starting point for training the ASR system. Alternatively, other models can be considered based on their performance on similar tasks.

6. Fine-tuning the Model: The next step is to fine-tune the model on the labelled dataset. This involves adjusting the pre-trained model's parameters to minimize the difference between the predicted transcriptions and the labelled transcriptions. This process may take several iterations, and various techniques can be used to optimize the performance of the model, such as adjusting the learning rate, using data augmentation to generate additional training data, and applying language-specific features such as phoneme embeddings or a language model.

7. Evaluation: The final step is to evaluate the performance of the ASR system on the testing dataset. This involves measuring the error rate of the system, typically using metrics such as Word Error Rate (WER) or Character Error Rate (CER). The results of the evaluation can be used to determine whether the system is suitable for the intended application and whether further improvements are needed.

3.5.1 Creating JSON Files And Dataset Cleaning

As the first step, we will be saving all the datasets as JSON files in separate output folders. For each sample in the dataset, we will create a dictionary with two keys: ”path” and ”sentence”, which respectively contain the path to the sample and the sample text. Then saves the sample dictionary as a JSON file with a filename that includes the sample number using the JSON.dump() method. One of the reasons to save a dataset as JSON files for Automatic Speech Recognition (ASR) is that JSON is a lightweight, widely used data format that can easily store structured data. JSON files can store the audio signal's metadata, such as speaker identity, transcription, and other relevant information, which is essential for ASR tasks. In ASR, JSON files can be used to store the audio signal's features such as spectrogram, Mel-frequency cepstral coefficients (MFCC), or filter banks. These features can be preprocessed and extracted from the audio signal and saved as JSON files, which can then be used as input to train an ASR model. By storing the dataset as JSON files, it becomes easier to read, manipulate, and preprocess the data, which is essential for ASR pipelines. The JSON file format is also language-independent, meaning it can be used across different programming languages and platforms, making it a flexible option for storing and exchanging data between different ASR systems. Datasets of male and female voices are separately created and stored in different folders. Later on, these datasets of JSON files are loaded and split and combined to make a dataset of both male and female voices. The transcriptions are

cleaned by removing special characters.

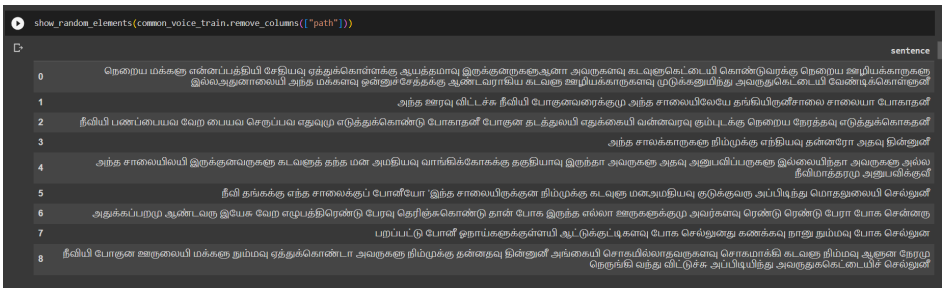

Figure 3.6: Random Elements After Removing Special Characters

3.5.2 Extraction Of Unique Characters

Once the dataset is loaded and generated the JSON files we need to extract the unique characters from the transcriptions and create a dictionary that maps each character to a unique integer index. A batch of examples from the datasets is taken and concatenates all the sentences into one long string. It then extracts a list of unique characters present in a long string and returns them as a dictionary with keys ”vocab” and ”long string”.This process is applied to both training and testing dataset and the entire dataset is processed in one batch. The resulting vocabulary lists are combined into a set and then back into a list, ”vocablist”, to ensure uniqueness. The ”vocabdict” dictionary is then created by iterating over ”vocablist” and assigning a unique integer index to each character. The resulting ”vocabdict” can be used to convert the text data to a sequence of integers, which can then be fed into a neural network or other machine learning model for training or inference. Our vocabulary consists of 39 tokens, which means that the linear layer that we will add on top of the pre-trained XLSR-Wav2Vec2 checkpoint will have an output dimension of 39.

In Automatic Speech Recognition (ASR), text data is often represented as a sequence of characters, which can be used to transcribe an audio signal into text. Before feeding the text data into an ASR model, it is often necessary to encode the characters as integers, using a process called character-level tokenization. This is because neural networks and other machine learning models can only process numerical data, not raw text.

In order to encode the characters as integers, each character in the text data is mapped to a unique integer index, which can then be used to represent the character in a numerical format. This requires a mapping between the characters and their integer indices, which can be stored as a dictionary. However, before creating the character-to-index mapping, we need to determine the set of unique characters that appear in the text data. This is because the size of the character-to-index mapping will depend on the number of unique characters in the text data. By extracting all unique characters from the dataset, we ensure that the size of the character-to-index mapping is as small as possible, which in turn reduces the number of parameters in the ASR model and can improve its performance. Additionally, encoding the characters as integers using a small set of indices,

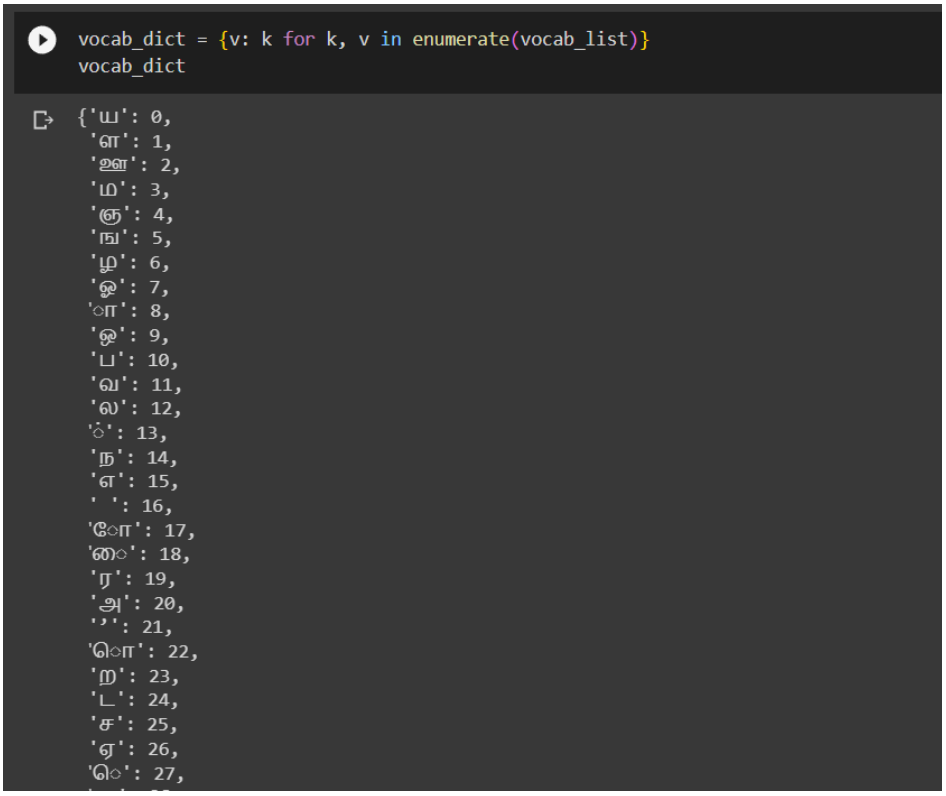

Figure 3.7: Unique Integer Index to Each Character

rather than a large number of indices, can also reduce the amount of memory required to store the text data during training and inference.

3.5.3 Tokenizer and Feature Extractor

ASR models transcribe speech to text, which means that we both need a feature extractor that processes the speech signal to the model's input format, e.g. a feature vector, and a tokenizer that processes the model's output format to text. In Transformers, the XLSR-Wav2Vec2 model is thus accompanied by a tokenizer, called Wav2Vec2CTCTokenizer, and a feature extractor, called Wav2Vec2FeatureExtractor.

3.5.4 Create Wav2Vec2CTCTokenizer

The pretrained Wav2Vec2 checkpoint maps the speech signal to a sequence of context representations. A fine-tuned XLSR-Wav2Vec2 checkpoint needs to map this sequence of context representations to its corresponding transcription so that a linear layer has to be added on top of the transformer block. This linear layer is used to classify each context representation to a token class analogous to how, e.g., after pretraining a linear layer is added on top of BERT's embedding for further classification. The output size of this layer corresponds

to the number of tokens in the vocabulary, which does not depend on XLSR-Wav2Vec2's pretraining task, but only on the labelled dataset used for fine-tuning.

In Connectionist Temporal Classification(CTC), it is common to classify speech chunks into letters, so we have done the same here. We have extracted all distinct letters of the training and test data and built our vocabulary from this set of letters.

3.5.5 Create XLSR-Wav2Vec2 Feature Extractor

Speech is a continuous signal and to be treated by computers, it first has to be discretized, which is usually called sampling. The sampling rate hereby plays an important role in that it defines how many data points of the speech signal are measured per second. Therefore, sampling with a higher sampling rate results in a better approximation of the real speech signal but also necessitates more values per second. A pretrained checkpoint expects its input data to have been sampled more or less from the same distribution as the data it was trained on. The same speech signals sampled at two different rates have a very different distribution, e.g., doubling the sampling rate results in data points being twice as long. Thus, before fine-tuning a pretrained checkpoint of an ASR model, it is crucial to verify that the sampling rate of the data that was used to pretrain the model matches the sampling rate of the dataset used to fine-tune the model. XLSR-Wav2Vec2 was pretrained on the audio data of Babel, Multilingual LibriSpeech (MLS), and Common Voice. Most of those datasets were sampled at 16kHz, so Common Voice, sampled at 48kHz, has to be down-sampled to 16kHz for training. Therefore, we will have to down-sample our fine-tuning data to 16kHz in the following. An XLSR-Wav2Vec2 feature extractor object requires the following parameters to be instantiated:

1. feature_size: Speech models take a sequence of feature vectors as input. While the length of this sequence obviously varies, the feature size should not. In the case of Wav2Vec2, the feature size is 1 because the model was trained on the raw speech signal.

2. sampling_rate: The sampling rate at which the model is trained on.

3. padding_value: For batched inference, shorter inputs need to be padded with a specific value

4. do_normalize: Whether the input should be zero-mean-unit-variance normalized or not. Usually, speech models perform better when normalizing the input

5. return_attention_mask: Whether the model should make use of an attention_mask for batched inference. In general, XLSRWav2Vec2 models should always make use of the attention_mask.

To make the usage of XLSR-Wav2Vec2 as user-friendly as possible, the feature extractor and tokenizer are wrapped into a single Wav2Vec2Processor class so that one only needs a model and processor object.

3.5.6 Preprocess Data

The preprocessing steps are critical for ensuring the quality and reliability of the dataset used to train an ASR model, which can ultimately have a significant impact on the accuracy of the ASR system. XLSR-Wav2Vec2 expects the audio file in the format of a 1-dimensional array, so in the first step, let's load all audio files into the dataset object. The audio file is saved in the .mp3 format. The .mp3 format is usually not the easiest format to deal with. We found that the torchaudio library works best for reading in .mp3 data. An audio file usually stores both its values and the sampling rate with which the speech signal was digitalized. Both are to be stored in the dataset. Librosa library is used to downsample the audio files to 16KHz.Downsampled audio files are then reviewed to ensure all the audio files are correctly loaded. The speakers change along with their speaking rate, accent, and background environment, etc. Overall, the recordings sound acceptably clear though, which is to be expected from a crowd-sourced read speech corpus.

The data is a 1-dimensional array, the sampling rate always corresponds to 16kHz, and the target text is normalized. Hence, we can process the dataset to the format expected by the model for training. First, we check that the data samples have the same sampling rate of 16kHz. Second, we extract the input_values from the loaded audio file. In our case, this includes only normalization, but for other speech models, this step could correspond to extracting e.g. Log-Mel features. Third, we encode the transcriptions to label ids.

3.5.7 Training

The data is processed so that we are ready to start setting up the training pipeline. In addition to loading the pre-trained model, we also specify several hyperparameters that are used during training. These hyperparameters include:

1. attention_dropout: The probability of dropping out attention weights during training.
2. hidden_dropout: The probability of dropping out of hidden states during training.
3. feat_proj_dropout: The probability of dropping out feature projection weights during training.
4. mask_time_prob: The probability of masking a timestep in the input sequence during training.
5. layerdrop: The probability of dropping out entire layers during training.
6. gradient_checkpointing: Whether to use gradient checkpointing to save memory during training.
7. ctc_loss_reduction: The reduction method to use when computing the CTC loss (in this case, taking the mean).
8. pad_token_id: The token ID used to pad sequences to a fixed length during training.

9. vocab_size: The size of the vocabulary used by the model's tokenizer.

We will make use of hugging face's Trainer for which we essentially need to do the following:

- Define a data collator: In contrast to most NLP models, XLSR-Wav2Vec2 has a much larger input length than output length. E.g., a sample of input length 50000 has an output length of no more than 100. Given the large input sizes, it is much more efficient to pad the training batches dynamically meaning that all training samples should only be padded to the longest sample in their batch and not the overall longest sample. Therefore, fine-tuning XLSR-Wav2Vec2 requires a special padding data collator.
- Evaluation metric: During training, the model should be evaluated on the word error rate. We should define a function to compute metrics accordingly.
- Load a pretrained checkpoint: We need to load a pretrained checkpoint and configure it correctly for training.
- Define the training configuration.

After having fine-tuned the model, we will correctly evaluate it on the test data and verify that it has indeed learned to correctly transcribe speech. Data collator treats the input_values and labels differently and thus applies to separate padding functions on them (again making use of XLSRWav2Vec2's context manager). This is necessary because in speech input and output are of different modalities meaning that they should not be treated by the same padding function. Analogous to the common data collators, the padding tokens in the labels with -100 so that those tokens are not taken into account when computing the loss. To allow models to become independent of the speaker rate, in CTC, consecutive tokens that are identical are simply grouped as a single token. However, the encoded labels should not be grouped when decoding since they don't correspond to the predicted tokens of the model.The result of WER is not satisfactory. The model failed to predict the entire sentence from the audio. Only single words are recognised. The model also failed to learn from the small number of datasets.

3.6 Proposed Architecture of Transformer Based Model for ASR

Hidden state representations that capture key aspects of spoken speech are created from the audio input in a sequence-to-sequence model. The decoder then processes these hidden state representations to produce the matching text transcriptions. The language model is integrated into the system architecture itself in deep fusion. This contrasts with shallow fusion, in which the encoder and language model are integrated externally. The ability to train the whole system end-to-end using the same training data and loss function is one of the benefits of deep fusion. Greater flexibility and overall better performance result from this.
The resonant frequency of the audio is increased to 16,000 Hz. Using windows of 25 milliseconds and a stride

of 10 milliseconds, a log-magnitude Mel spectrogram representation with 80 channels is constructed. The input is scaled globally between -1 and 1, with the goal of achieving an essentially zero mean throughout the pre-training dataset to normalise the features. The encoder starts by employing a tiny stem to process this input representation. The stem uses the GELU activation function and has two convolution layers with a 3-filter width. The stride of the second convolution layer is two. The output of the stem is then further enhanced by sinusoidal position embeddings. Position embeddings are added, and then the encoder Transformer blocks are used. The transformer normalises the encoder output using the final layer and pre-activation residual blocks. The decoder uses coupled input-output token representations and learnt position embeddings. It matches the encoder's width and the number of transformer blocks that are there. For a representation of the model architecture, refer to Figure3.8

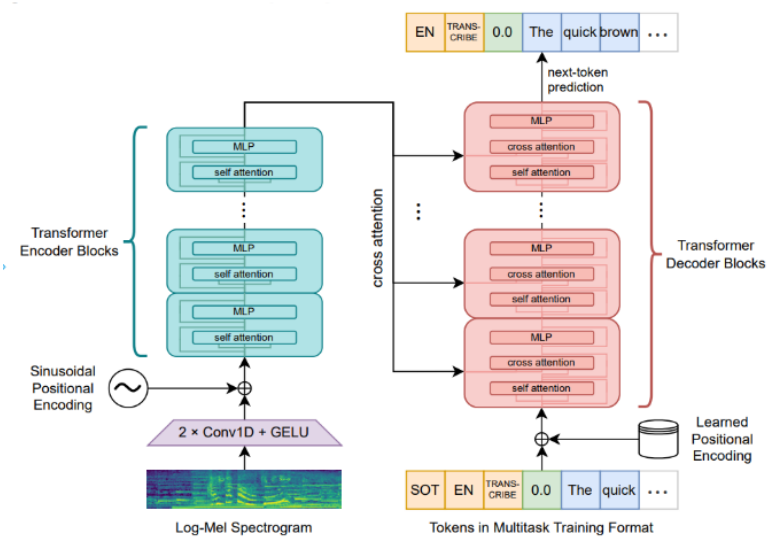

Figure 3.8: Overview of Sequence-to-Sequence Transformer Model[1]

The Automatic Speech Recognition Model pipeline can be divided into three components:

- The feature extractor converts the raw audio signal into a sequence of features that can be used by the model. This process typically involves extracting features such as the frequency spectrum, Mel-frequency cepstral coefficients (MFCCs), and preprocessing them.
- The model takes the sequence of features as input and outputs a sequence of probabilities that correspond to the possible words that were spoken. This model is typically trained on a large dataset of audio recordings and transcripts.
- The tokenizer takes the sequence of probabilities from the model and outputs a text transcript. This process typically involves decoding the probabilities into words and punctuation and correcting any

errors that were made by the model.

The Encoder block contains two components:

- **Self Attention**

  Self-attention is a mechanism in the transformer architecture that allows a model to attend to different parts of the same input sequence. This is done by computing a weighted sum of the input vectors, where the weights are determined by the similarity between the input vectors and a query vector. A self-attention mechanism is a powerful tool for modelling long-range dependencies in sequences. This is because it allows the model to attend to different parts of the sequence, even if they are far apart. This is in contrast to recurrent neural networks (RNNs), which are limited in their ability to model long-range dependencies. The self-attention mechanism is implemented using the following steps:

  1. The input sequence is first converted into a sequence of vectors.
  2. A query vector is then computed.
  3. The similarity between the input vectors and the query vector is computed.
  4. The weights for the weighted sum are then computed using the similarity scores.
  5. The weighted sum is then computed.

  The self-attention mechanism is a key component of the transformer architecture. It allows the transformer architecture to achieve state-of-the-art results on a variety of natural language processing tasks, such as machine translation, text summarization, and question answering. Here is an example of how self-attention can be used to model long-range dependencies in a sequence. Consider the sentence ”The cat sat on the mat.” The self-attention mechanism would allow the model to attend to the word ”cat” and the word ”mat”, even though they are far apart in the sequence. This is because the self-attention mechanism would compute the similarity between the word vectors for ”cat” and ”mat”, and then use these similarity scores to weight the contribution of the word vectors to the weighted sum. The self-attention mechanism is a powerful tool for modelling long-range dependencies in sequences. It is a key component of transformer architecture, and it has been shown to be effective in a variety of natural language processing tasks.

- **Multi Layer Perceptron**

  In the transformer encoder block, the MLP is used to learn a non-linear transformation of the output of the self-attention layer. This transformation allows the model to learn more complex relationships between the input tokens. The MLP in the transformer encoder block is typically composed of two or three hidden layers. The hidden layers are typically fully connected, meaning that each neuron in a layer is connected to every neuron in the next layer. The neurons in the hidden layers typically use a ReLU

activation function, which is a non-linear function that helps the model learn complex relationships. The output of the MLP is then fed into the next layer of the transformer encoder block, which is typically another self-attention layer. This process is repeated for a number of layers until the final representation of the input sequence is produced. The MLP in the transformer encoder block is a powerful tool for learning non-linear relationships between input tokens. This allows the model to learn more complex representations of the input sequence, which can improve the performance of the model on a variety of natural language processing tasks. Here is an example of how the MLP in the transformer encoder block can be used to learn a non-linear relationship between input tokens. Consider the sentence ”The cat sat on the mat.” The MLP could learn a non-linear relationship between the word vectors for ”cat” and ”mat”, such that the output of the MLP would be higher when the word vectors for ”cat” and ”mat” are close together in the sequence. This would allow the model to learn that the words ”cat” and ”mat” are related, even though they are far apart in the sequence.

The Decoder block contains three components:

- **Cross Attention**

  Cross attention is a mechanism in the transformer architecture that allows the decoder to attend to the encoder's output. This is done by computing a weighted sum of the encoder's output vectors, where the weights are determined by the similarity between the decoder's input vector and the encoder's output vectors. Cross-attention is a powerful tool for modelling the relationships between the input and output sequences. This is because it allows the decoder to attend to the encoder's output, which contains information about the input sequence. This information can be used by the decoder to generate the output sequence. The cross-attention mechanism is implemented using the following steps:

  1. The decoder's input vector is first converted into a query vector.
  2. The similarity between the query vector and the encoder's output vectors is computed.
  3. The weights for the weighted sum are then computed using the similarity scores.
  4. The weighted sum is then computed.

  The cross-attention mechanism is a key component of the transformer architecture. It allows the transformer architecture to achieve state-of-the-art results on a variety of natural language processing tasks, such as machine translation, text summarization, and question answering. Here is an example of how cross-attention can be used to model the relationships between the input and output sequences. Consider the sentence ”The cat sat on the mat.” The cross-attention mechanism would allow the decoder to attend to the encoder's output vector for the word ”cat”, which would contain information about the meaning of the word ”cat”. This information could then be used by the decoder to generate the word ”mat”, which is the object that the cat is sitting on.

- **Self Attention**

  Self-attention is a crucial component within the transformer decoder block that enables the model to capture dependencies and relationships between different positions in the input sequence. It allows the model to attend to and weigh the importance of different parts of the input sequence while generating the output sequence. Self-attention operates on three sets of vectors derived from the input sequence: key vectors, query vectors, and value vectors. These vectors are obtained by applying linear transformations to the input sequence. Each position in the input sequence is associated with a key vector, a query vector, and a value vector. The self-attention mechanism calculates attention weights by measuring the similarity between a query vector and the key vectors of all positions in the input sequence. The similarity is computed using the dot product between the query vector and each key vector. The dot products are then scaled by the square root of the dimension of the key vectors to stabilize the gradients. Once the attention weights are calculated, they are used to weigh the corresponding value vectors. The weighted value vectors are then summed to obtain the context vector, which represents the attended representation of the input sequence. The self-attention mechanism allows the transformer model to consider the entire input sequence while generating the output sequence. It captures dependencies and contextual information by dynamically attending to different parts of the input sequence, enabling the model to focus on the most relevant positions and weigh their influence when generating the output.

- **Multi Layer Perceptron**

  The MLP is a feed-forward neural network component that operates independently on each position of the input sequence. It is typically positioned after the self-attention mechanism in the decoder block. The purpose of the MLP is to capture and model complex non-linear relationships within the context of each position in the input sequence.

  The MLP consists of two linear transformations followed by an activation function.

  1. Linear Transformation 1: The input to the MLP is a vector representation of the current position in the input sequence. This input is fed into a linear layer, which applies a matrix multiplication operation followed by a bias addition.
  2. Activation Function: The output of the first linear transformation is then passed through a non-linear activation function, usually a Rectified Linear Unit (ReLU) or a GELU (Gaussian Error Linear Unit). This introduces non-linearity into the model, allowing the MLP to learn complex patterns and relationships in the data.
  3. Linear Transformation 2: The output of the activation function is fed into another linear layer, which performs another matrix multiplication operation followed by a bias addition.

     The final output of the MLP is obtained by applying a final activation function (typically a ReLU or GELU) or by directly using the output of the second linear transformation. This output is then

passed on to the subsequent components of the decoder block. The purpose of the MLP in the transformer decoder is to introduce additional non-linearity and modelling capacity to capture more complex relationships and patterns within the context of each position in the input sequence. The MLP allows the model to learn and transform the representation of each position before generating the output sequence.

It's important to note that the MLP operates independently on each position, meaning the same weight matrices and biases are applied across the entire input sequence. This shared parameterization allows the MLP to capture and model patterns consistently across different positions.

Multi-Layer Perceptron (MLP) within the transformer decoder block applies linear transformations followed by activation functions to capture complex non-linear relationships within the input sequence. It enhances the modelling capacity of the transformer model, enabling it to generate accurate and contextually informed output sequences.

3.6.1 Load Dataset

Since we have uploaded the corpus to the Hugging Face hub, it made really easy to load the dataset for the Transformer based model. We use 16850 rows of data for training and 1873 rows of data for testing purposes. The model uses the test data to calculate the model's Word Error Rate. Our dataset contains approximately 5.30+ hours of Malasar Transcripted data. Our ASR datasets only provide input audio samples (audio) and the corresponding transcribed text (sentence). there is no additional metadata information, such as accent and locale.

3.6.2 Prepare Feature Extractor

Using a 1-dimensional array that changes over time, speech is represented. The signal's amplitude at every particular time step is represented by the value of the array at that instant. We can rebuild the audio's frequency spectrum and recover all of the acoustic properties just from the amplitude data.

Speech is a continuous signal with an infinite range of amplitude values. This can be a problem for computer hardware, which can only store a finite number of values. To overcome this, we can discretize the speech signal by taking samples of its values at predetermined time intervals. The sampling rate, which is measured in samples per second or Hertz (Hz), is the frequency at which we take these samples. A higher sampling rate produces a better approximation of the continuous speech input, but it also requires more storage space.

Audio signals with different sampling rates have different characteristics. Therefore, it is important to match the sampling rate of the audio input to the sampling rate that the model expects. Using the wrong sampling rate can lead to unexpected results. For example, an audio sample with a sampling rate of 16 kHz will sound like it is being played at half-speed when listened to at an 8 kHz sampling rate. Additionally, an

ASR model that expects a certain sampling rate may fail if it receives audio with a different sampling rate. In order to avoid these problems, we must match the sampling rate of our audio inputs to the 16 kHz sampling rate that the feature extractor requires.

The feature extractor first pads or truncates audio samples so that they all have a length of 30 seconds. Samples that are less than 30 seconds are extended by adding zeros to the end, while samples that are longer than 30 seconds are cut off. This ensures that all audio inputs to the Whisper model have the same length. In most audio models, an attention mask is used to indicate which parts of the input sequence are padding and should therefore be ignored. However, the Web Scale Model does not use an attention mask. Instead, it has been trained to learn to ignore padding based on the audio signals themselves. This makes the model more efficient and easier to train.

The feature extractor converts the padded audio arrays into log-Mel spectrograms. A log-Mel spectrogram is a visual representation of the frequencies of a signal, similar to a Fourier transform. The y-axis of a log-Mel spectrogram shows the Mel channels, which correspond to certain frequency bins. The x-axis shows time. The hue of each pixel represents the log intensity of that frequency bin at a particular instant in time. The model expects to receive input in the form of a log-Mel spectrogram. The Mel channels were selected to roughly correspond to the human auditory spectrum.

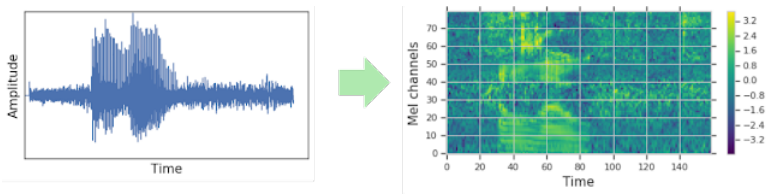

Figure 3.9: Left: sampled 1-dimensional audio signal. Right: corresponding log-Mel spectrogram.[2]

3.6.3 Preparing Tokenizer

Any ASR system needs a tokenizer, although transformer-based systems rely on tokenizers more than others. This is due to the fact that transformers are a sequence-to-sequence model and must be able to translate an audio feature sequence to a text token sequence. The tokenizer must be able to map the tokens back to the original text string in addition to breaking up the audio features into a series of tokens. The model's vocabulary size is determined by the tokenizer. The amount of distinct tokens that the model can identify is known as the vocabulary size. The model will be able to recognise more words if its vocabulary is greater, but it will also become more complicated and challenging to teach. A competent tokenizer should be able to separate the audio characteristics into a series of tokens that closely match the spoken words. This will make it easier for

the model to figure out how to correctly map audio characteristics to text tokens.
The model predicts the position of each word in the dictionary as a sequence of text tokens. The tokenizer converts these text tokens back into the original text string. For example, the sequence [1169, 3797, 3332] would be converted back to the string ”its raining here”.

When using encoder-only models for ASR, we decode using Connectionist Temporal Classification (CTC). This means that we need to train a CTC tokenizer for each dataset that we use. One advantage of using an encoder-decoder architecture is that we can use the tokenizer from the pre-trained model directly. The pre-trained tokenizer is trained on the transcriptions of 96 languages, so it has a large vocabulary that can be used for most multilingual ASR applications. We can verify that the tokenizer is working correctly by encoding and decoding the first sample of the dataset. When encoding the transcriptions, the tokenizer adds special tokens to the beginning and end of the sequence. These special tokens include the start/end of transcript tokens, the language token, and the job tokens (as specified in the previous step). We have the option to ”skip” these special tokens when decoding the label ids, which allows us to return a string in the original input format.

```
Input:                 இவருகளோடவு
Decoded w/ special:    <|startoftranscript|><|transcribe|><|notimestamps|>இவருகளோடவு<|endoftext|>
Decoded w/out special: இவருகளோடவு
Are equal:             True
```

Figure 3.10: Encoding and Decoding Using Tokenzier

3.6.4 Prepare Data

The audio should first be resampled at 16,000 Hz. Then, using our 1-dimensional audio array, we compute the log-Mel spectrogram input features using the feature extractor. The tokenizer is then used to encode the transcriptions to label ids. We now utilise torchaudio and librosa for loading and resampling audio. This is all in data preparation.

3.6.5 Define Data Collater

The data collator for a sequence-to-sequence speech model is different from other data collators because it treats the input features and labels separately. The input features are handled by the feature extractor, while the labels are handled by the tokenizer. The input features have already been padded to 30 seconds and transformed to a log-Mel spectrogram of fixed dimension, so the only thing that needs to be done is to convert them to batched PyTorch tensors. This is done using the feature extractor’s .pad function with the return_tensors=pt argument. Since the input features are of fixed dimensions, no additional padding is needed.

The labels, however, are not padded. We initially use the tokenizer’s.pad function to pad the sequences to

the batch's maximum length. The padding tokens are then changed to -100 in order to remove them from the calculation of the loss. The label sequence is then separated from the start of the transcript token so that it may be added later on during training.

3.6.6 Training

A sequence-to-sequence model's training is configured using the Seq2SeqTrainingArguments object. The batch size, learning rate, and evaluation approach are just a few of the important factors that may be utilised to set the training for the object. The object may be used to set up checkpointing and logging as well. A potent tool that allows for highly flexible sequence-to-sequence model training is the Seq2SeqTrainingArguments object.

To train a model using the Seq2SeqTrainingArguments object that we have defined, we utilise the trainer.train() function. The procedure will train the model for the provided number of steps and periodically evaluate it. At regular intervals, the technique will additionally store the model checkpoint. A TrainerState object, which is the result of the trainer.train() function, provides details about the training procedure, including the model checkpoint, the training loss, and the evaluation metrics.

3.6.7 Evaluation Metric For ASR

The performance of an automatic speech recognition system is often measured by the word error rate (WER). However, this metric can be misleading because the recognized word sequence may not have the same length as the reference word sequence. To address this issue, dynamic string alignment can be used to align the recognized word sequence with the reference word sequence. The power law theory, which posits a link between perplexity and word error rate, can then be used to analyze the results of the alignment.

The WER can then be calculated as:

$$WER = (S + D + I)/N = (S + D + I)/(S + D + C) \tag{3.1}$$

where N = S + D + C and S = the number of words in the reference, S = the number of substitutions, D = the number of deletions, I = the number of insertions, C = the number of right words.

This quantity represents the average number of mistakes per reference word. The ASR system performs better the lower the value is.

3.7 Transformer Based Machine Translation.

In order to address the issue of machine translation, the Transformer architecture first surfaced in 2017 [12]. Multilingual neural machine translation is a sequence-to-sequence task. This means that it takes a sequence of words in one language (the source language) and produces a sequence of words in another language (the

target language). The model does this by first encoding the source language sequence into a fixed-length vector. This vector is then decoded into the target language sequence. In order to train a multilingual neural machine translation model, we need to have text data in both the source and target languages. We can use a variety of sources for this data, such as parallel corpora, monolingual corpora, and synthetic data. Once we have the data, we need to tokenize it. Tokenization is the process of breaking text into smaller units, such as words or subwords. We use a single SentencePiece (SPM) tokenizer for our text sequences. This tokenizer has been shown to work well for multilingual machine translation tasks. After we have tokenized the data, we need to downsample high-resource languages and upsample low-resource languages. This is done to ensure that all of the languages are represented equally in the model. We downsample high-resource languages by randomly removing some of the sentences. We upsample low-resource languages by randomly duplicating some of the sentences. The vocabulary size is a crucial hyperparameter in multilingual translation models incorporating low-resource languages. A large vocabulary size allows the model to represent a wider range of words and phrases. However, a large vocabulary size can also make the model more difficult to train. Our SPM model after training has a 256,000-word vocabulary. This vocabulary size is large enough to represent the words and phrases that are commonly used in our different languages. However, it is not so large that it makes the model difficult to train. The vocabulary size is just one of the many hyperparameters that need to be tuned when training a multilingual neural machine translation model. Other important hyperparameters include the learning rate, the batch size, and the number of layers in the model. The optimal values for these hyperparameters will vary depending on the specific dataset and model architecture. However, by carefully tuning these hyperparameters, we can train multilingual neural machine translation models that achieve high translation quality.

3.7.1 Architecture of Transformer Based Model for MT

The Transformer encoder-decoder architecture serves as the foundation for our sequencetosequence multilingual machine translation concept. Token embeddings are created by the encoder from the source token sequence. The decoder pays attention to the encoder output and constructs the target phrase in an autoregressive fashion, token by token. Transformer layer stacks make up the encoder and decoder. An input series of embeddings and an output sequence of embeddings are provided by each Transformer layer. Transformer levels in the encoder consist of a self-attention layer and a feedforward layer as their two sub-layers. These are applied consecutively, with a LayerNorm applied before them and a residual connection applied after them. As opposed to applying LayerNorm after the residual connection at the conclusion of each sub-layer (Post-LN), we do it before the start of each sub-layer (Pre-LN). This is as a result of Pre-LN being more stable in usage than Post-LN. While the feed-forward layer (FFN) sends each element of the sequence separately via a 2-layer MLP, the self-attention layer (SEL) is an attention layer that updates each element of the sequence by examin-

ing the other components. Between the self-attention and the feed-forward sub-layers in the decoder, there is an extra third sub-layer that computes attention over the encoder output.

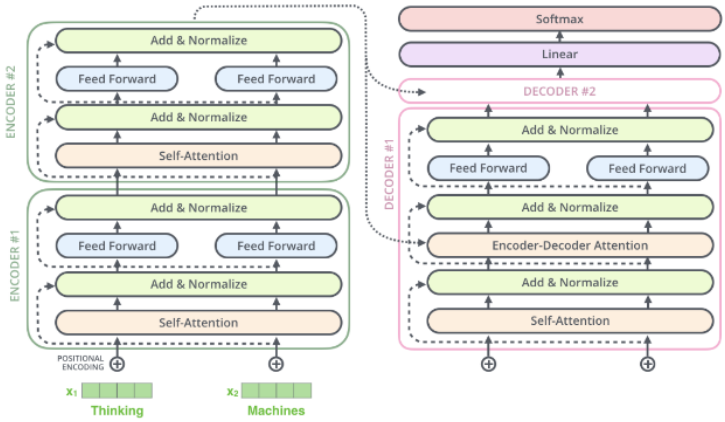

Figure 3.11: Transformer Based MT Model

3.7.2 Load Dataset

Since we have uploaded the corpus to the Hugging Face hub, it made really easy to load the dataset for the Transformer based model. We use 956 rows of data for training and testing purposes. The model uses the test data for calculating the Bleu score of the model.

3.7.3 Preprocessing the Data

We must preprocess the texts before we can input them to our model. A Linear Transformers Tokenizer does this task by tokenizing the inputs (including converting the tokens to their matching IDs in the pre-trained vocabulary), putting them in the format expected by the model, and also generating the other inputs needed by the model. The targets must be tokenized inside the as_target_tokenizer context manager to prepare them for our model. By doing this, you can be confident the tokenizer will utilise the unique tokens that match to the targets. This will guarantee that an input that is longer than the model chosen can manage will be shortened to the length the model will accept. We pad samples to the largest length possible in the batch since the padding will be taken care of later (in a data collator).

3.7.4 Training

A sequence-to-sequence model's training is configured using the Seq2SeqTrainingArguments object. The batch size, learning rate, and evaluation approach are just a few of the important factors that may be utilised to set the training for the object. The object may be used to set up checkpointing and logging as well. A potent tool that allows for highly flexible sequence-to-sequence model training is the Seq2SeqTrainingArguments object.

To train a model using the Seq2SeqTrainingArguments object that we have defined, we utilise the trainer.train() function. The procedure will train the model for the provided number of steps and periodically evaluate it. At regular intervals, the technique will additionally store the model checkpoint. A TrainerState object, which is the result of the trainer.train() function, provides details about the training procedure, including the model checkpoint, the training loss, and the evaluation metrics.

3.7.5 Evaluation Metric For MT

A statistic used to gauge the effectiveness of machine translation is called BLEU. It is determined by contrasting the text that has been machine translated with a collection of reference translations. The BLEU score increases in direct proportion to how well the machine translation matches the reference translations. One of the most often used measures for assessing machine translation is BLEU, which has been proven to be a reliable indicator of the quality as perceived by humans.

3.8 Summary

The proposed system creates a corpus for machine translation and Automatic speech recognition. Corpus size has been increased by creating synthetic Audio. The creation of synthetic audio helped to train the model with much more data. The Wav2Vec pretrained model is used as the first step, and the model failed to transcribe the audio into Malasar text; the Word Error Rate for this model was 100%. The proposed ASR model got a WER of 28% for the Machine Translation model a BLEU score of 30 is marked. This methodology analysis different methods for creating a corpus for different NLP algorithms and also training Transformer based models for ultra-low resource languages.

Chapter 4

Results and Discussions

4.1 Results of ASR

Fine-tuning the Web scale transformer architecture resulted in a satisfactory result. In total 18 hours of training resulted in a WER of 28%. The 28% of WEER is received while training only. When validated manually we got WER in between 15-20%. We used the Max_Step parameter for training and saved the model at every 50th step. The best model is finalised by manual validation. The manual validation is done by linguistic experts from Wycliffe India. The trained model is uploaded on the hugging face hub and it can be used for further fine-tuning. The result of ASR was satisfactory and can be used for transcribing Malasar languages with very less manual intervention. When compared to the output of the wav2vec model this fine-tuned web-scale model outperforms it. The WER rate at different evaluation steps is shown in Table 4.1.

The initial training loss was 0.676800, which gradually decreased over the course of training. At step 50, the validation loss was 0.159135, and the WER was 63.919886. As training progressed, the model improved its performance. By step 100, the training loss was significantly reduced to 0.171100, indicating the model's learning progress. The validation loss decreased to 0.115213, and the WER improved to 52.989986. At step 150, the training loss reached 0.139200, demonstrating continuous improvement. The validation loss was 0.106360, and the WER decreased to 42.002861, indicating the model's enhanced accuracy. Throughout the training process, the model continued to refine its performance. By the final step, step 1500, the training loss was 0.023500, the validation loss was 0.059908, and the WER was 30.901288. These results indicate that the model achieved a high level of accuracy and effectively reduced both the training and validation losses. In conclusion, the training process led to a well-performing model with a low loss and improved accuracy. The achieved results demonstrate the effectiveness of the training process and suggest the model's suitability for the intended application.

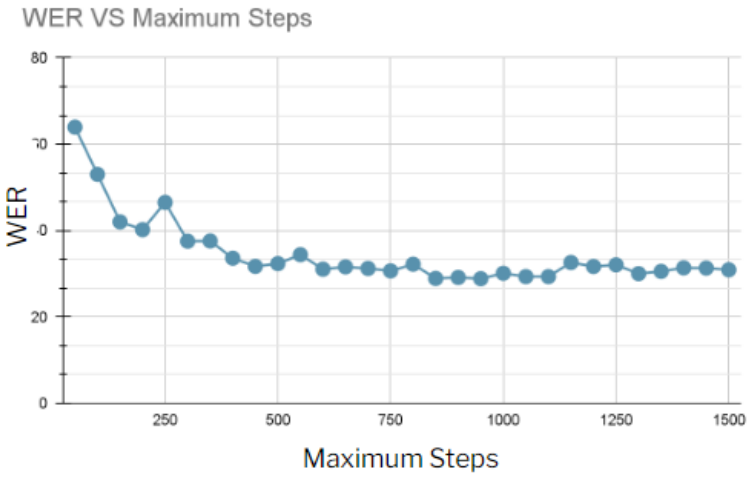

Figure 4.1: Maximum Step vs WER

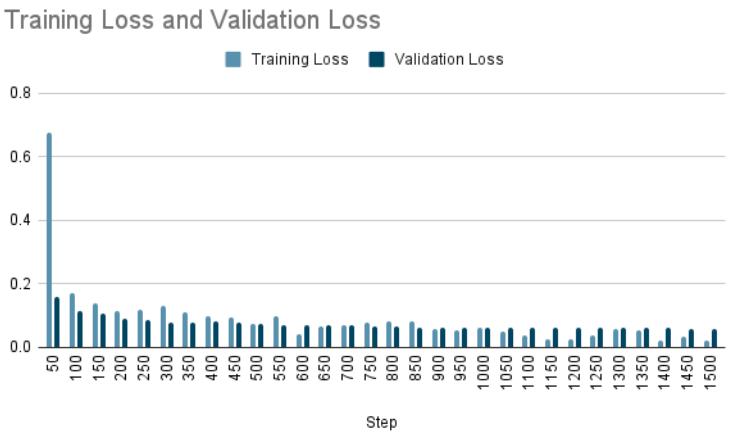

Figure 4.2: Training and Validation Loss

Transcript : அதற்குப் பிறகு அண்டாவர் ஏற்சு வேறை 72 பேராவு தெரிந்துகொண்டு தான் போகை இருந்தேன் எல்லா உர்களுக்குமும் அவர்களாவு இரண்டு இரண்டு பேராவு போகச்சென்று

Reference : அதுக்கப்பறமு ஆண்டவரு இயேசு வேற எழுபத்திரெண்டு பேரவு தெரிஞ்சுகொண்டு, தான் போக இருந்த எல்லா ஊருகளுக்குமு, அவர்களவு ரெண்டு ரெண்டு பேரா போக சென்னரு.

Figure 4.3: Transcription Using Trained Model

Language	WER Rate
Hindi	21.77%
Bengali	28.27%
Tamil	30.54%
Telugu	32.13%
Malayalam	33.32%
Gujarati	34.23%
Marathi	35.24%
Malasar	28%

Table 4.1: WER of Different Indic Languages Using Transformer Models

4.2 Results of MT

The trained Transformer based machine translation models translates the Malasar to English with a BLEU score of 30. When manually validated accuracy is low when compared to the system validation score. The BLEU rates were achieved using a multilingual machine translation (MT) system trained on the WMT 2021 dataset Table 4.2. The model was trained for 50 epochs using the transformer based architecture. The training

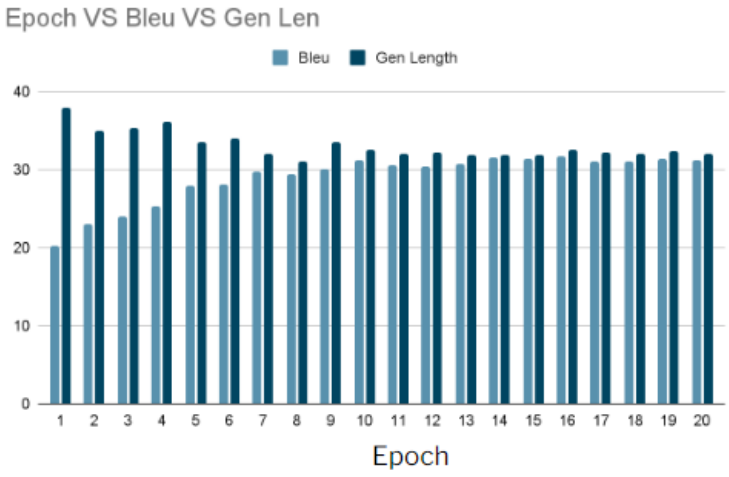

Figure 4.4: Epoch vs BLEU vs Generated Length

and validation losses were recorded for each epoch, along with the BLEU score and generated sentence length. The initial training loss was not logged. The training loss gradually decreased over the epochs, starting from 1.801988 in epoch 1 and reaching 0.265400 in epoch 50. The validation loss followed a similar trend, starting

at 20.520700 in epoch 1 and reaching 2.026935 in epoch 50. The BLEU score, which measures the quality of the generated sentences compared to reference sentences, started at 37.890100 in epoch 1 and fluctuated between 27.524600 and 35.549500 throughout the training process. The length of the generated sentences remained relatively stable, with a slight decrease from epoch 1 (37.890100) to epoch 50 (34.373600). Overall, the training process resulted in a decreasing trend in both training and validation losses. However, the BLEU score fluctuated within a certain range, indicating variations in the quality of the generated sentences. The sentence length remained relatively consistent throughout the training.

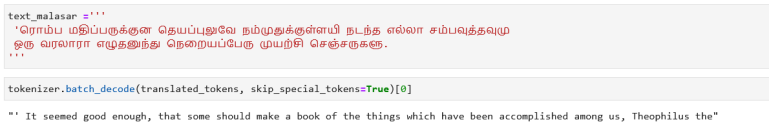

Figure 4.5: Machine Translation Using Trained Model.

Language	BLEU Rate
Hindi	30.11%
Bengali	25.23%
Tamil	23.84%
Telugu	22.82%
Malayalam	22.17%
Gujarati	21.79%
Marathi	21.58%
Malasar	30.432%

Table 4.2: BLEU rates of Indic languages

4.3 ASR Model Deployment on WEB

The trained model of the ASR is hosted on hugging face and created as a web application, and the model is integrated into the website using the Django Python framework. This enables public access, which will help to save the Malasar Language. This web application can be used to transcribe 30 seconds of Malasar audio. The transcribed example of audio can be seen in figure 4.7. The URL is as follows: `http://basilkr.pythonanywhere.com/`

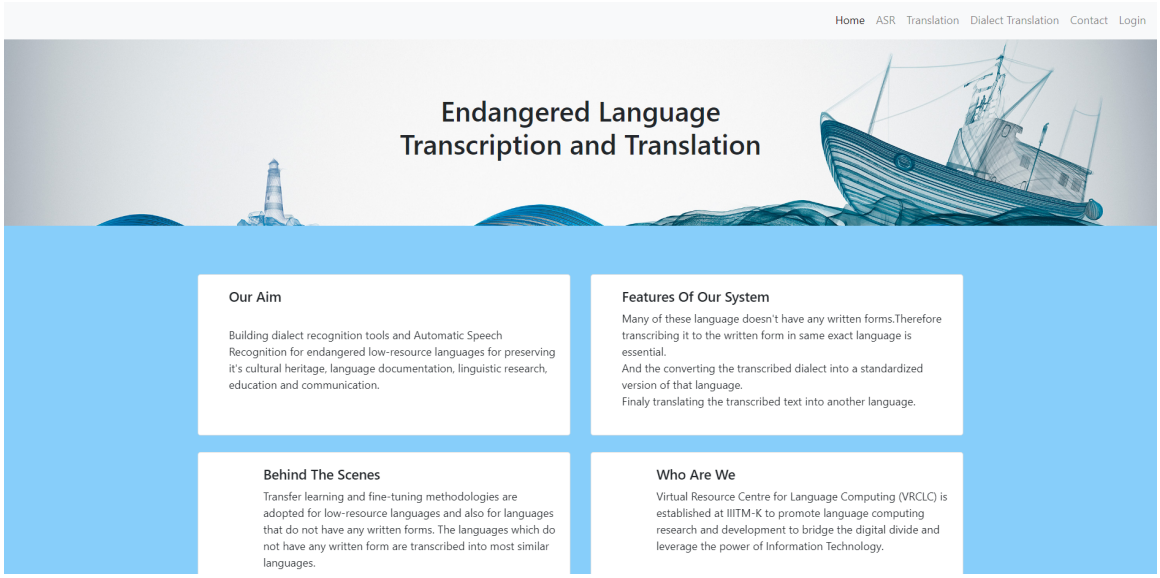

Figure 4.6: Web Application: Home Page

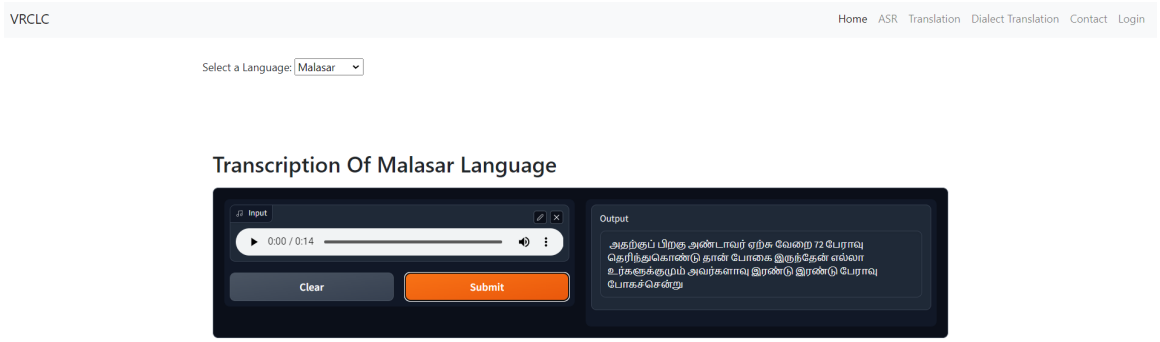

Figure 4.7: Web Application: ASR Page

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This project's aim was to build a Machine translation model for Malasar to English and speech recognition for Malasar. By creating a corpus for each separate task we trained transformer-based models. The corpus we have created is in reusable format and uploaded to the cloud for future use. The trained model of Speech Recognition can be accessed using the website `http://basilkr.pythonanywhere.com/`. The development of a tokenizer and feature extractor resulted in improved accuracy and efficiency in Malasar's speech recognition. This study demonstrates the potential of the transformer model for multilingual speech recognition and opens up new avenues for further research and development in the field of ASR. The results suggest that the web-scale transformer model has the potential to be applied to other low-resource languages for speech recognition, providing a scalable and flexible solution for the global ASR community.

5.2 Future Work

There are many ways to improve the proposed system. Developing a speech translation system for the Malasar language is one of them. This can be done by integrating the ASR and MT into a single model to perform both tasks. Such model input will be the audio and output can either be its translation to the target language or its transcription. Extending the translation capability of the model to Malayalam and also translating the Malasar language into the standard Tamil format is to be done for preserving the Malasar language. Creating a much larger corpus for MT is an essential need for improving accuracy. Finally deploying the translation model as same as the ASR will help the communities to use it with ease.

References

[1] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *International Conference on Machine Learning*. PMLR, 2023, pp. 28 492–28 518.

[2] "Specaugment: A new data augmentation method for automatic speech recognition," https://ai.googleblog.com/2019/04/specaugment-new-data-augmentation.html, accessed: 2010-09-30.

[3] A. Magueresse, V. Carles, and E. Heetderks, "Low-resource languages: A review of past work and future challenges," *arXiv preprint arXiv:2006.07264*, 2020.

[4] S. Ranathunga, E.-S. A. Lee, M. P. Skenduli, R. Shekhar, M. Alam, and R. Kaur, "Neural machine translation for low-resource languages: A survey," *ACM Computing Surveys*, vol. 55, no. 11, pp. 1–37, 2023.

[5] M. Wiesner, C. Liu, L. Ondel, C. Harman, V. Manohar, J. Trmal, Z. Huang, N. Dehak, and S. Khudanpur, "Automatic speech recognition and topic identification for almost-zero-resource languages," *arXiv preprint arXiv:1802.08731*, 2018.

[6] A. Zeyer, P. Bahar, K. Irie, R. Schlüter, and H. Ney, "A comparison of transformer and lstm encoder decoder models for asr," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 8–15.

[7] H. Miao, G. Cheng, C. Gao, P. Zhang, and Y. Yan, "Transformer-based online ctc/attention end-to-end speech recognition architecture," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6084–6088.

[8] S. Schneider, A. Baevski, R. Collobert, and M. Auli, "wav2vec: Unsupervised pre-training for speech recognition," *CoRR*, vol. abs/1904.05862, 2019. [Online]. Available: http://arxiv.org/abs/1904.05862

[9] M. R. Costa-jussà, J. Cross, O. Çelebi, M. Elbayad, K. Heafield, K. Heffernan, E. Kalbassi, J. Lam, D. Licht, J. Maillard *et al.*, "No language left behind: Scaling human-centered machine translation," *arXiv preprint arXiv:2207.04672*, 2022.

[10] R. Östling, Y. Scherrer, J. Tiedemann, G. Tang, and T. Nieminen, "The helsinki neural machine translation system," *CoRR*, vol. abs/1708.05942, 2017. [Online]. Available: http://arxiv.org/abs/1708.05942

[11] A. Fazel, W. Yang, Y. Liu, R. Barra-Chicote, Y. Meng, R. Maas, and J. Droppo, "Synthasr: Unlocking synthetic data for speech recognition," *CoRR*, vol. abs/2106.07803, 2021. [Online]. Available: https://arxiv.org/abs/2106.07803

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.