Automatic Speech Recognition using Wav2Vec 2.0

Project-II (ME47602) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Mechanical Engineering

by

Lambu Kushi Reddy

(20ME31025)

Under the supervision of

Pawan Goyal

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Spring Semester, 2023-24

April 27, 2024

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: April 27, 2024 (Lambu Kushi Reddy)

Place: Kharagpur (20ME31025)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Automatic Speech Recognition using Wav2Vec 2.0**" submitted by **Lambu Kushi Reddy** (Roll No. 20ME31025) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Mechanical Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester, 2023-24.

Date: April 27, 2024

Place: Kharagpur

Pawan Goyal
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Acknowledgements

I would like to express my sincere gratitude to all those who have contributed to the completion of this work. Special thanks go to Professor Pawan Goyal for their invaluable guidance, support, and insightful feedback throughout the entire process. I am also thankful for the Indian Institute of Technology Kharagpur for providing the necessary resources and environment for this research. I extend my appreciation to my colleagues and peers who have shared their expertise and perspectives, enriching the depth of this project.

I am grateful to my friends and family for their unwavering encouragement and understanding during the challenges and demands of this endeavor. Their support has been a constant source of motivation.

Lastly, I want to express my appreciation to the broader academic and research community whose work has laid the foundation for this study. The collaborative spirit and shared knowledge within this community have significantly contributed to the advancement of our understanding in this field.

Thank you all for being an integral part of this journey.

Contents

List of Figures

Chapter 1

Introduction

1.1 What is ASR

Speech recognition is a multidisciplinary domain at the intersection of computer science and computational linguistics. Its focus lies in devising methodologies and technologies facilitating the conversion of spoken language into textual form by computer systems. This field is alternatively referred to as automatic speech recognition (ASR), computer speech recognition, or speech-to-text (STT) technology.

At its core, ASR relies on complex algorithms and machine learning models to analyze audio signals, identify patterns, and map them to corresponding text. This process involves several intricate steps, including audio preprocessing, feature extraction, acoustic modeling, and language modeling.

One of the key advantages of ASR is its ability to streamline processes and enhance accessibility. By enabling hands-free interaction, ASR can significantly improve productivity and convenience in various scenarios, such as note-taking, document creation, and accessibility for individuals with disabilities.

The integration of ASR with other cutting-edge technologies, such as natural language processing and artificial intelligence, will open up new possibilities for innovative applications across various sectors, including virtual assistants, real-time translation services, accessibility tools, and intelligent human-machine interfaces.

In the end, the continued evolution of ASR technology will play a crucial role in shaping the future of human-computer interaction, enabling more natural, efficient, and inclusive communication experiences

1.2 Motivation

The need for ASR arises from the increasing prevalence of speech as a primary mode of interaction between humans and machines. In various domains such as telecommunications, customer service, healthcare, education, and entertainment, ASR facilitates seamless communication and interaction. For example, in call centers, ASR systems transcribe customer inquiries, enabling automated responses or routing to appropriate agents. In healthcare, ASR technology aids in transcribing medical dictations, facilitating accurate documentation of patient information.

ASR systems also play a crucial role in enabling accessibility for individuals with disabilities. By converting spoken language into text, ASR technology allows people with hearing impairments to access spoken content through visual interfaces. Additionally, ASR-powered voice assistants enhance accessibility and convenience for users by enabling hands-free interaction with devices and applications.

The adoption of ASR technology is driven by its potential to streamline workflows, improve efficiency, and enhance user experiences. Businesses can leverage ASR to automate repetitive tasks, such as transcribing meetings, interviews, or lectures, freeing up human resources for more complex and value-added activities. Furthermore, ASR-enabled applications and services offer greater flexibility and convenience for users, allowing them to interact with technology naturally and intuitively.

In fields such as natural language processing (NLP), machine translation, and sentiment analysis, ASR serves as a foundational component, providing input data in the form of transcribed speech. By converting spoken language into text, ASR facilitates the analysis, interpretation, and manipulation of speech data, enabling advanced language understanding and processing tasks. As the demand for intelligent and context-aware systems continues to grow, ASR remains essential for unlocking the full potential of human-machine communication and interaction.

1.3 Importance of ASR for Sanskrit

For automatic speech recognition (ASR) to function reasonably, a substantial volume of annotated training data is necessary. Deep learning techniques used in recent voice recognition algorithms have worsened the data scarcity issue because these systems are very data-hungry. Only 1% of the world's languages are thought to contain the bare minimum of data required to train an ASR. Because of this, until a few years ago, researchers studying speech recognition concentrated mostly on high-resource languages like English and Mandarin.

The development of ASR for languages with limited resources has received more attention in the last several years. The Indian constitution's eighth schedule includes 22 official languages, the majority of which have little resources. Among them is Sanskrit, which is regarded as the second-oldest language after Tamil.

Given that many Indo-European languages can be traced back to Sanskrit, it is thought to be the mother tongue of these languages. This language is home to an enormous corpus of writing in a variety of subjects, including mathematics, astronomy, science, linguistics, mythology, history, and mysticism.

Given the aforementioned factors, Sanskrit takes on great significance even though it is not frequently employed for active communication. There aren't many attempts to develop technical Sanskrit tools.

This motivates us to investigate the application of some of the state-of-the-art technologies and techniques towards building a Sanskrit ASR. We believe that these efforts will aid in enhancing the accessibility of the language and thus its contents to a larger population.

Chapter 2

Background

Automatic Speech Recognition (ASR) remains a dynamic field, continually striving to bridge the gap between machine and human understanding of spoken language. Despite progress, challenges persist, especially concerning accents, background noise, and variability in speech patterns. Enhancements in ASR methodologies offer the potential for greater accuracy and efficiency, reducing dependence on resource-intensive processes.

India's rich linguistic tapestry, encompassing over 1500 languages, presents a twofold challenge and opportunity for the advancement of speech and language technologies. The linguistic landscape of India is primarily characterized by two major language families: Indo-Aryan and Dravidian. Indo-Aryan languages, such as Sanskrit, Hindi, Bengali, Gujarati, and Punjabi, are predominantly spoken in the North West region, while Dravidian languages like Tamil, Telugu, Malayalam, and Kannada are prevalent in the southern part of the country.

According to the 2011 census data, Hindi is the most widely spoken language, accounting for 43.63% of the population. Other scheduled languages such as Bengali, Marathi, Telugu, Tamil, Gujarati, Kannada, and Malayalam are spoken by 18%, 6.8%, 6.7%, 5.7%, 4.58%, 3.61%, and 2.88% of the population, respectively.

The preservation of India's linguistic diversity is crucial to prevent the erosion of resources in lesser-known languages. This underscores the necessity for technologies

like multilingual speech recognition, which encourage the utilization of native Indian languages for human-computer interaction.

2.1 Challenges for Indian languages ASR

Developing Automatic Speech Recognition (ASR) systems for Indian languages poses several challenges:

- Low Resource Scenarios: Indian languages lack extensive data for robust ASR training. For example, initial labeled audio data for Hindi was limited to just half an hour.
- Code-Switching: Speakers frequently switch between languages within conversations, complicating accurate transcription for ASR systems.
- Loanwords and Dialect Variations: Incorporation of loanwords and dialectal variations in Indian languages, like Hindi, presents difficulties in accurate word recognition.
- Acoustic and Linguistic Richness: Unique linguistic traits such as retroflex sounds and specific word orders add complexity to ASR models, demanding tailored approaches for accurate transcription.
- Variability in Data Distribution: Background noise, pitch, volume variations, and differences in word speed contribute to data variability, making it challenging to train models effectively.

2.2 Advancements in Speech Recognition for Indian Languages

ASR for Indian languages is a significant challenge due to the low-resource nature of most Indian languages. Many Indian languages have very limited amounts of labelled speech data available, making it difficult to build robust ASR systems.

While Hindi enjoys widespread usage, many other languages suffer from data scarcity, impeding the development of robust Deep Neural Network (DNN)-based models. Innovative approaches, such as multilingual speech recognition systems, are vital for preserving linguistic diversity and encouraging the use of native languages for human-computer interaction.

Research in Indian languages has yielded impressive results in speech recognition systems. For instance, Patel and Virparia (2013) developed a Gujarati telephony command system with an accuracy of 83.62%, peaking at 96.55% after optimizations. Mishra et al. explored speaker-independent Hindi digit recognition using advanced feature extraction techniques, demonstrating the superiority of MF-PLP in noisy environments. Another study titled "Multilingual low resource Indian language speech recognition and spell correction using Indic BERT" introduced an approach utilizing RNN-GRU for speech recognition, achieving a Word Error Rate (WER) of 0.621. Integrating Indic BERT further improved the WER to 0.52, addressing challenges in low-resource Indian languages.

2.3 Sanskrit ASR: Insights and Developments

Automatic Speech Recognition (ASR) for Sanskrit faces significant hurdles due to the language's complexity, including its highly inflected and agglutinative nature, complex morphological and phonological rules, and sandhi phenomena, resulting in variations in pronunciation and spelling. Moreover, the extensive vocabulary, featuring many rare and compound words, poses challenges for accurate language modeling. Additionally, the diversity in regional and historical pronunciation and accentuation adds further complexity to ASR endeavors.

To overcome these challenges, various strategies have been implemented. The creation of the "Vāksañcayaḥ" corpus, a comprehensive dataset covering diverse Sanskrit literature and speakers, serves as a valuable resource for capturing phonetic and orthographic variations. Grapheme-to-Phoneme (G2P) conversion is crucial for accurately mapping Sanskrit's written form to its phonetic representation. Different encoding methods, such as modified Sanskrit Library Phonetic (SLP1-M) encoding

for traditional ASR models and basic SLP1 encoding for end-to-end models, have been explored to optimize G2P conversion.

Acoustic and language modeling are pivotal components of Sanskrit ASR. Acoustic models need diverse speech data for training to accurately represent phonetic variations. Meanwhile, language models must accommodate Sanskrit's vast vocabulary, complex morphology, and frequent sandhi transformations. Techniques such as transfer learning and data augmentation show promise in enhancing ASR performance amidst Sanskrit's linguistic complexities.

Despite these challenges, notable efforts have been made in the field of Sanskrit ASR. For instance, the Indian Institute of Technology Madras has led research efforts in developing robust speech recognition systems tailored specifically for Sanskrit, considering its unique phonetic and linguistic characteristics. Additionally, Carnegie Mellon University's Sphinx project, an open-source speech recognition system, has been explored for Sanskrit ASR, although the extent of its success remains unclear.

2.4 Models in ASR:

2.4.1 Hidden Markov Models (HMMs)

HMMs use Bayes' theorem to determine the most probable text given a speech utterance, considering text likelihood and acoustic models. Despite their effectiveness, HMMs operate under simplifications like the first-order Markov assumption and employ short frames for computational efficiency.

Over the past two decades, ASR has shifted from Hidden Markov Models (HMMs) to machine learning (ML) approaches, particularly artificial neural networks (ANNs). This transition is propelled by advancements in computing tools like Kaldi, PyTorch, and TensorFlow, alongside the abundance of text and speech data. ANNs offer superior performance and flexibility, reflecting human-like behavior and handling diverse datasets more effectively.

A paper on the DNN-HMM hybrid system uses the robust capabilities of Deep Neural Networks (DNNs) with the structured framework of Hidden Markov Models

(HMMs). Here, the HMM captures temporal dynamics, while the DNN estimates observation probabilities. Training involves using a well-trained GMM-HMM system as a seed model and training the DNN to predict state posterior probabilities. During decoding, these probabilities are converted to likelihoods to mitigate label bias.

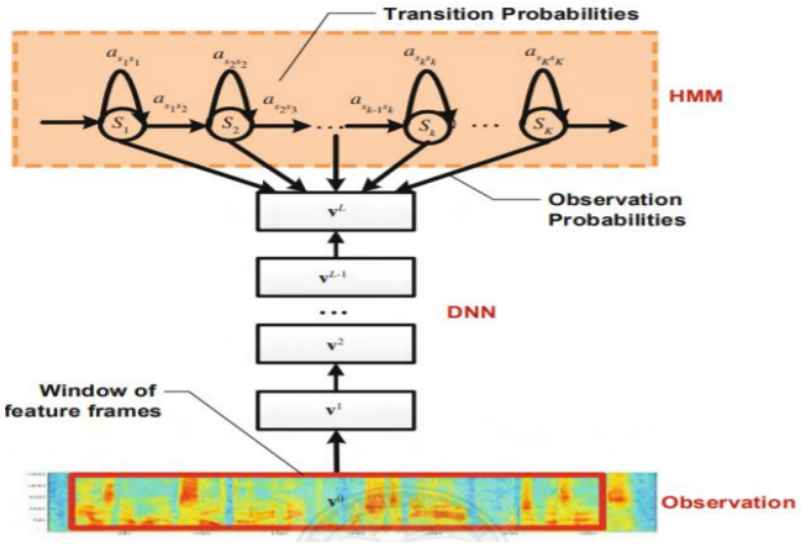


FIGURE 2.1: Architecture of the DNN-HMM hybrid system

In recent years, there has been a noticeable trend towards deep learning (DL) models and transformer-based architectures in ASR. DL models, particularly those based on transformers, offer significant improvements in understanding contextual information and handling long-range dependencies in speech data. Their ability to capture intricate patterns and nuances in speech has made them increasingly popular for ASR tasks. As the field continues to evolve, the adoption of DL models and transformer-based architectures is expected to further enhance the accuracy and efficiency of ASR systems, paving the way for more seamless human-machine interactions.

2.4.2 Connectionist Temporal Classification

In order to compensate for the variation in textual output and voice frame lengths, the Connectionist Temporal Classification (CTC) technique in ASR maps input

speech frames to output text characters by inserting blank labels in output characters. Presented in Graves et al. (2006), CTC aims to address the issue of data alignment as alignment between speech segments and audio is no longer needed.

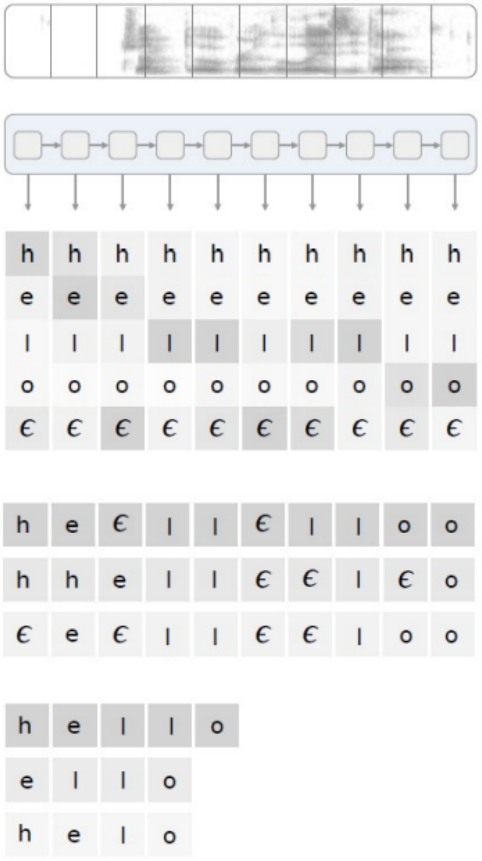


FIGURE 2.2: CTC in action; here ϵ is the blank symbol

Above Figure provides an example of how a CTC-based model operates. Let us represent the output label sequence as y and the input voice as x. The definition of the CTC loss function is:

$$L_{CTC} = -ln(P(\mathbf{y}|\mathbf{x}))$$

where,

$$P(\mathbf{y}|\mathbf{x}) = \sum_{q \in B^{-1}(y)} P(q|x)$$

In this case, the set of all potential output label alignments is represented by $B^{-1}(\mathrm{y})$. Given conditional independence and T, the speech sequence's duration, this can be further divided into each timestep as follows:

$$P(q|x) = \prod_{t=1}^{T} P(q_t|x)$$

Because it leaves out context-dependent pronunciations, which are an essential component of all spoken languages, the aforementioned conditional independence assumption is frequently questioned. The attention mechanism, which is the first point of language modeling across speech frames, can be used to improve this. Transformer-based models have replaced the LSTM-based networks, which served as the foundation for CTC's initial development, and as a result, CTC is now a widely used method with exceptional accuracy (Higuchi et al., 2020). This is because, whereas the CTC streamlines the decoding process and adds up all potential alignments, the transformer-based approach offers a robust attention-based mechanism. Additionally, CTC does well on tasks involving self-supervision (Chung and Glass, 2020), which has gradually grown in popularity as an area of NLP research.

2.4.3 RNN-Transducer Models

RNN Transducer (RNN-T) models offer an advanced learning framework in which all voice frames up to the current time step and previous character tokens are taken into account during decoding. Since their introduction in Graves (2012), RNN-Transducers have gained popularity in the industry as a model for streaming ASR systems because they eliminate the conditional independence requirement of the CTC models and offer a way to create streaming ASR systems (He et al., 2018).

RNN-T is made up of an encoder network, a prediction network, and a joint network, as shown in the below figure. Similar to previous systems, the encoder network produces a high level encoder representation, h_t^e . Based on the previous output label y_{u-1} of RNN-T, the prediction network generates another high level representation h_u^p

Both of these features were integrated by the joint network, a feedforward network, as follows: .

$$z_{t,u} = \phi(Qh_t^{enc} + Vh_u^{pre} + b_z)$$

where bz is the bias vector and ϕ is a nonlinear function (ReLU or TanH). The weight matrices are Q and V. z_{t_u} is connected to the output layer as -

$$h_{t,u} = W_y z_{t,u} + b_y$$

whereas b_y is the bias vector and W_y is the weight matrix. Lastly, we obtain the following probability for the output token k:

$$P(y_u = k|x_{1:t}, y_{1:u-1}) = softmax(h_{t,u}^k)$$

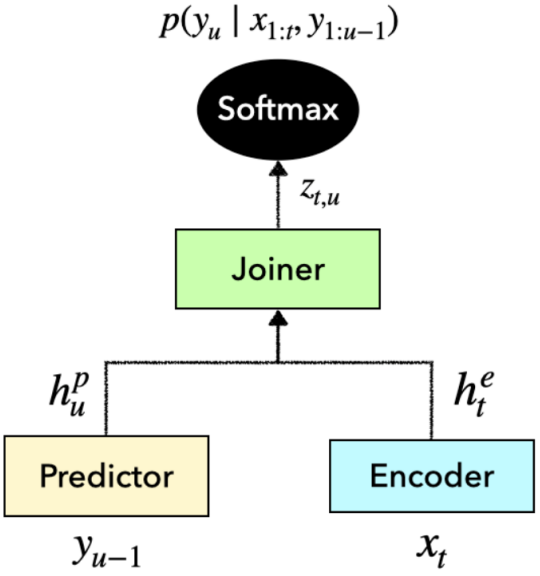


FIGURE 2.3: Simplified architecture of RNN Transducers

2.4.4 Attention Based Models

Instead of mapping the input audio signal to a single fixed vector, the encoder model in attention-based models (Vaswani et al., 2017) translates the signal to a

sequence of vectors. In order to decode the higher dimensional characteristics, the decoder then concatenates these vector sequences and assigns weights to them. The design represents both short- and long-range dependencies in this way. Due to their reliance on previous steps, RNN-based models always face the issue of slow training and have issues with both short- and long-term context awareness. Decoding the specific character and alignment at each time step involves taking into account the features from the previous and next time steps through the attention mechanism. An attention based model calculates the probability as -

$$P(y|x) = \prod_{u} P(y_u|x, y_{1:u-1})$$

where u is the output label index. The training objective is the same as that for CTC based models

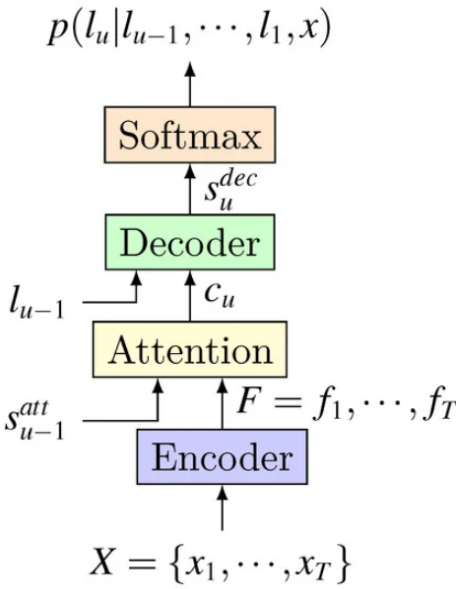


FIGURE 2.4: Structure of an attention based ASR model

Chapter 3

Data Acquisition and Preprocessing

3.1 Wikipedia Data Extraction

We meticulously extracted textual content corresponding to the 20 Kandas of the Atharvaveda and the 10 Mandalas of the Rigveda from wikisource.org using the Beautiful Soup library. This involved a systematic process of web scraping, wherein we parsed through the HTML structure of the webpage, targeting specific elements containing the desired textual data. By filtering out non-textual elements such as images and numbers, we ensured that only pure textual content was captured. Furthermore, we took care to preserve the formatting of the text, including spaces between lines and punctuation marks, thereby maintaining readability and coherence in the extracted data.

Subsequently, each segment of text was methodically organised and stored into individual text files. For the Atharvaveda, each Kanda was allocated its own text file, while for the Rigveda, each Mandala was similarly stored in a separate text file. This meticulous categorization not only facilitated easy access and retrieval of specific sections of the texts but also ensured clarity and organization in the data storage process. By adhering to strict standards of data extraction and organization,

we created a comprehensive repository of Vedic texts, free from extraneous elements and optimised for further analysis and study.

3.2 Acquiring Audio Data

We have obtained audio files from the Internet Archive containing recitations of the Atharva Veda and Rigveda. Each audio file spans approximately 40-45 minutes, totaling 27 files for Atharva Veda and 54 files for Rigveda. The combined size of these audio files amounts to 5.4GB. In the case of the Atharva Veda, the structure of the 10 Mandalas is disordered; specifically, the 20 Kandas are dispersed across the 27 audio files in a non-sequential manner. Similarly, for the Rigveda, the 10 Mandalas are also scattered throughout the 54 audio files, lacking sequential organization. This arrangement presents a challenge for accessing specific sections of the Vedas, as the desired passages are not conveniently grouped together. Addressing this issue may involve reorganizing the files to align with the traditional structure of the Vedas, facilitating easier navigation and study.

3.3 Audio Processing

The process began with the acquisition of audio files from the Internet Archive, featuring recitations of the Atharva Veda and Rigveda. Each original audio file spanned approximately 40-45 minutes, resulting in a total of 27 files for the Atharva Veda and 54 files for the Rigveda. These files were then processed using the Pydub library to convert them into the .wav format, ensuring compatibility and standardization across the dataset. Subsequently, each 40-45 minute audio file was subdivided into smaller segments based on periods of silence present within the recordings. The segmentation aimed to create manageable audio chunks for further analysis and utilization. The segmentation process involved careful parameter tuning to optimize the length and distribution of the resulting audio files. Parameters such as the minimum length of silence, silence threshold, and keep silence were adjusted to achieve the desired outcome. Specifically, the minimum length of silence determines the duration of silence (in milliseconds) required to trigger a split in the audio file, while the

silence threshold specifies the decibel level below which audio was considered silent. We set the minimum length of silence in the range 3-7 milliseconds and the silence threshold parameter to be in the range of -30 to -70 dB. Additionally, a 'keep silence' parameter was set to retain a specified duration of silence (in milliseconds) at the beginning and end of each chunk, ensuring smooth transitions between segments. The Keep silence parameter was set to 500 milli seconds.

Following segmentation, attention was given to filtering out audio files that did not meet a predetermined threshold length. Typically, a threshold length of 1250 milliseconds (1.25 seconds) was chosen, as shorter audio files were often deemed to contain negligible content or significant stretches of silence. For audio chunks exceeding this threshold length, each segment was exported as a separate WAV file.

To facilitate organization and future reference, exported files were systematically named using zero-padded indexing, ensuring sequential ordering and easy identification of individual segments within the dataset. This meticulous process ensured the effective transformation of lengthy audio recordings into a structured and manageable collection of segmented files suitable for further analysis or application.

3.4 Alignment and Synchronization of Audio-Text Correspondence

After segmenting the audio files into smaller chunks and obtaining corresponding text files, we discovered an issue where each audio file lacked a direct alignment with its corresponding text. While we anticipated that the audio files would be divided line by line according to the Mandalas/Kandas of the Vedas, the segmentation did not consistently occur at every line.

To resolve this misalignment, we embarked on a manual alignment process. Initially, we created a single .txt file for each folder of chunked audio files, containing transcriptions of the audio files within that folder. However, we observed discrepancies where a single audio chunk might contain two or three lines from the Vedic texts, or only a portion of a line.

To address this, we devised a systematic approach.Initially, we checked every 10th audio file to determine if its audio line matched the corresponding 10th transcript line. If this initial check passed, we proceeded further. However, if discrepancies were found, we scrutinized the audio lengths for anomalies. Typically, audio files containing only one line of text ranged from 3 to 6 seconds in duration. If an anomaly was detected, we manually inspected the audio file to correct the transcript accordingly.

This meticulous process ensured that each text line in the transcript accurately corresponded to the text chanted in the audio files. As a result, we successfully created .txt files with the same number of lines as there were audio files in each folder, achieving a harmonized alignment between the audio and text representations of the Vedas.

3.5 Data Preprocessing

In preprocessing the text data we took the following steps. Firstly, we removed Devanagari (Hindi) numerals and English numerals (0-9) from the text. Next, we systematically eliminated special characters such as '-', '[', '—', '/', '.', and apostrophes, ensuring a clean and uncluttered text. Subsequently, we meticulously eradicated any excess whitespaces within the text, maintaining a uniform appearance throughout. Finally, we standardized the text format to enhance readability and consistency, adhering to professional standards throughout the process.

3.6 Exploratory Data Analysis

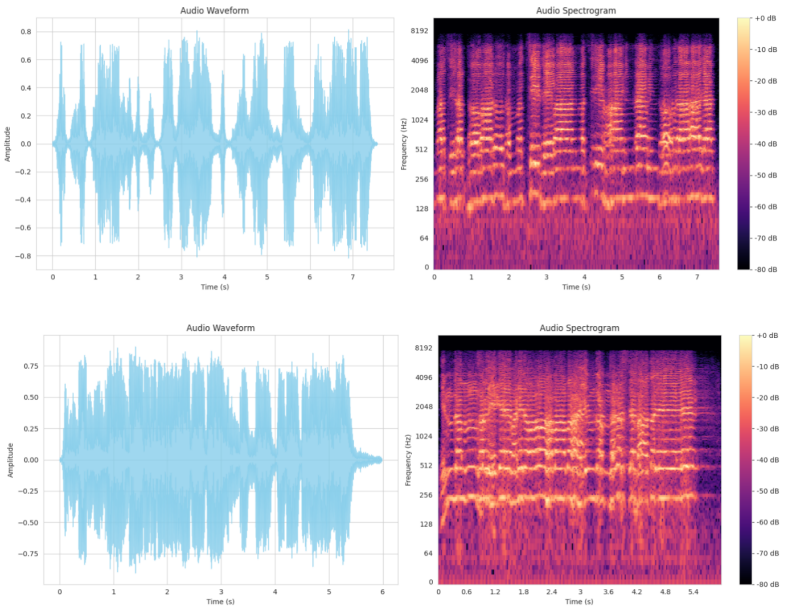


FIGURE 3.1: The above waveforms visually represent how the intensity of a sound wave changes over time in an audio recording of Atharvaveda and Rigveda, while the spectrograms illustrate the frequencies present in the sound wave and their varying intensities over time

Dataset Overview and Characteristics

1. The total combined duration of Rigveda and Atharvaveda recordings amounts to 54.319 hours.
2. The maximum duration of a single audio file is 63.103 seconds, sampled at 16000 Hz.
3. Each audio file begins with 500 milliseconds of silence.
4. There are 20,782 audio files for Rigveda and 9,997 for Atharvaveda.
5. The total number of files allocated for training is 24,623.
6. Additionally, 3,078 files are reserved for development/validation purposes.
7. An equal number of files, 3,078, are set aside for testing.

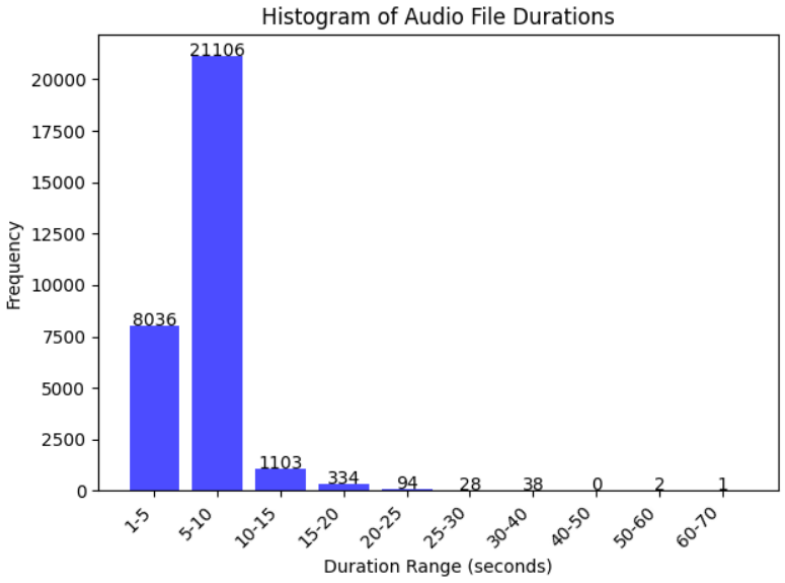
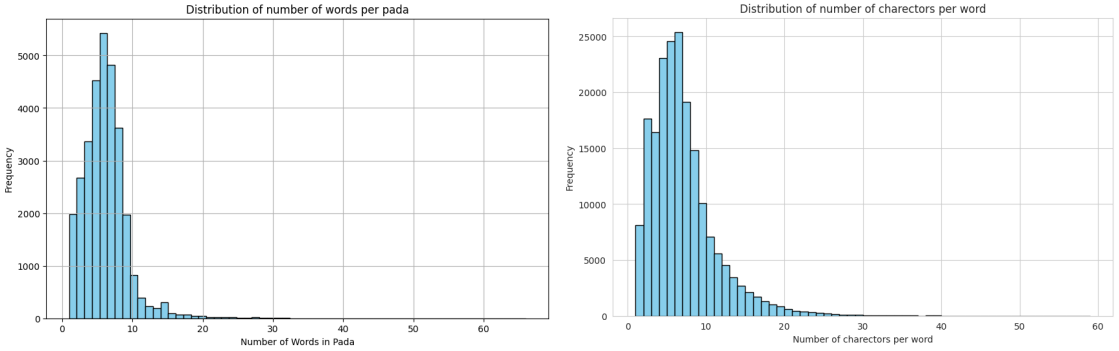


FIGURE 3.2: The histogram depicted above illustrates the distribution of audio file durations



(a) The distribution of the number of words per pada (sentence)

(b) The distribution of the number of characters per word

FIGURE 3.3: Distribution of words per pada (sentence) and characters per word in the dataset.

The histograms presented above illustrate the distribution of the number of characters per word and the number of words per pada (sentence). In the distribution of characters per word, the maximum number of characters per word reaches 59, with a mean value calculated at 6.5. Regarding the distribution of words per pada, the majority contain between 6 to 7 words, with an average of 6.25 words per pada. Notably, the maximum number of words found in a single pada is 66.

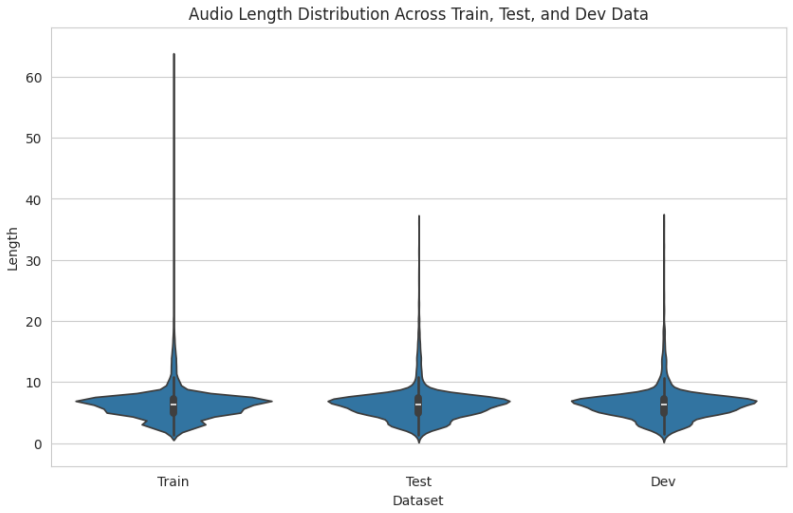


FIGURE 3.4: This above Violin plot displays the distribution of audio lengths across the train, test, and development (dev) datasets.

Chapter 4

Modelling

4.1 Transliterating Devanagari Script to IAST

I accurately transliterated the Devanagari text corresponding to every audio file into IAST format using the Python library 'indic-transliteration'. By converting Devanagari text to IAST, researchers and practitioners ensure a standardized representation of Sanskrit pronunciation, facilitating precise alignment between audio recordings and their corresponding transcriptions. This alignment not only enhances the accuracy of ASR systems but also fosters broader accessibility and utilization of Sanskrit language resources in the digital era.

Vowels									
DS		IAST		DS		IAST			
अ		a		ऌ		ḷ			
आ		ā		ए		e			
इ		i		ऐ		ai			
ई		ī		ओ		o			
उ		u		औ		au			
ऊ		ū		अं		ṃ			
ऋ		ṛ		अः		ḥ			
ॠ		ṝ		ऽ		'			
ऌ		ḷ							
Consonants									
Velars		Palatals		Retroflexes		Dentals		Labials	
DS	IAST	DS	IAST	DS	IAST	DS	IAST	DS	IAST
क	ka	च	ca	ट	ṭa	त	ta	प	pa
ख	kha	छ	cha	ठ	ṭha	थ	tha	फ	pha
ग	ga	ज	ja	ड	ḍa	द	da	ब	ba
घ	gha	झ	jha	ढ	ḍha	ध	dha	भ	bha
ङ	ṅa	ञ	ña	ण	ṇa	न	na	म	ma
ह	ha	य	ya	र	Ra	ल	la	व	va
		श	śa	ष	ṣa	स	sa		

FIGURE 4.1: Mapping Alphabet List in DS and IAST

4.2 Overview of the Wav2Vec 2.0 architecture

Transformer-based neural networks are relatively new to the speech processing world, but they have been transforming the field of natural language processing. That is about to change with Wav2vec 2.0. Its design is based on the Transformer's Encoder, and it has a training target that is tailored for speech and resembles the masked language modeling objective of BERT.

With this novel technique, semi-supervised training may be done efficiently by first pre-training the model on a large amount of unlabeled speech and then fine-tuning it on a smaller labeled dataset. The authors of wav2vec 2.0's original publication showed that the model could outperform earlier state-of-the-art systems trained on 100 times more labeled data by fine-tuning it on just one hour of labeled speech data.

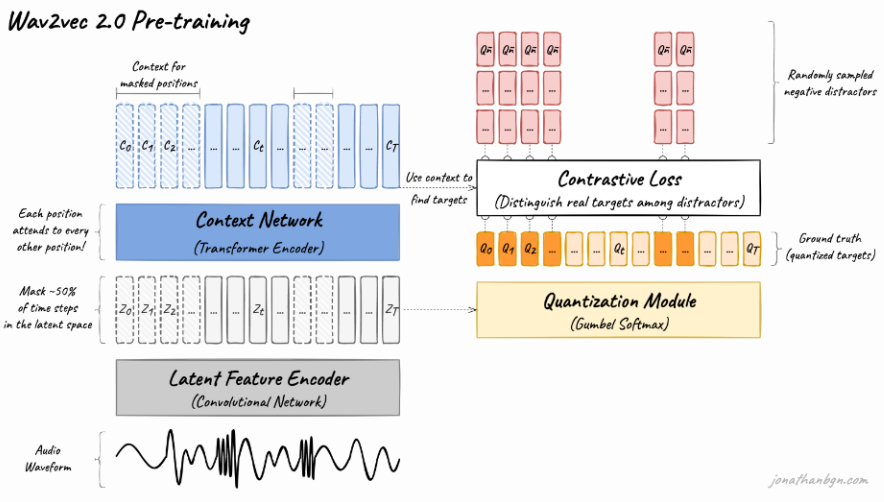


FIGURE 4.2: Wav2Vec 2.0 Pre-training

Above is an overview of the wav2vec 2.0 architecture and its pre-training process. There are four important elements in this diagram: the feature encoder, context network, quantization module, and the contrastive loss (pre-training objective). Let us look at each one in detail.

4.2.1 Feature encoder

The purpose of the feature encoder is to minimize the dimensionality of the audio data by transforming the raw waveform every 20 milliseconds into a series of feature vectors, Z0, Z1, Z2,..., ZT. Its architecture is straightforward: a single-dimensional, seven-layer convolutional neural network with 512 channels in each layer.

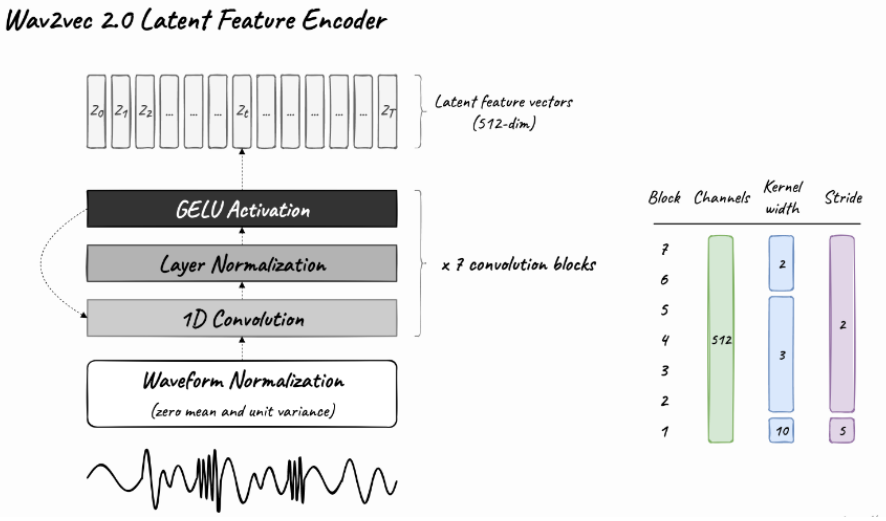


FIGURE 4.3: Wav2Vec 2.0 Latent Feature Encoder

Prior to being transferred to the network, the waveform is normalized, and as one moves up the network, the convolutional layers' kernel width and strides decrease. The audio data is encoded at a sample rate of 16 kHz, and the feature encoder may receptively receive 400 samples, or 25 milliseconds, of audio.

4.2.2 Quantization module

The continuous pattern of speech presents a major challenge for speech processing with Transformers. Written language has a finite vocabulary of discrete units since it may readily be divided into words or subwords. There are no such innate subdivisions in speech. Although we could employ phones as a discrete system, we couldn't pre-train on unlabeled data because humans would need to first label the entire dataset.

Wav2vec 2.0 proposes to automatically learn discrete speech units, by sampling from the Gumbel-Softmax distribution. Possible units are made of codewords sampled from codebooks (groups). Codewords are then concatenated to form the final speech unit. Wav2vec uses 2 groups with 320 possible words in each group, hence a theoretical maximum of 320 x 320 = 102,400 speech units.

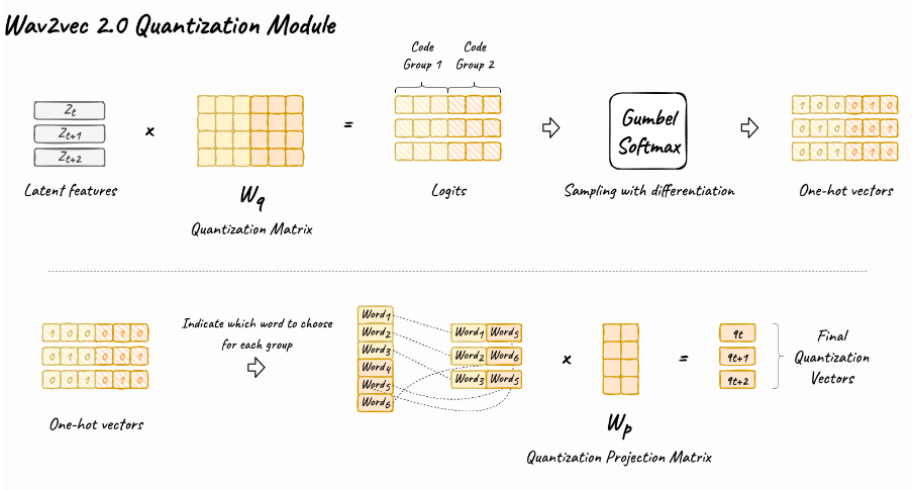


FIGURE 4.4: Wav2Vec 2.0 Quantization Module

The quantization matrix multiplied by the latent features yields the logits, which are one score for each conceivable codeword in each codebook. The Gumbel-Softmax method, which turns these logits into probabilities, permits sampling one codeword per codebook. The only difference between it and taking the argmax is that this operation is entirely differentiable. To aid in training and codeword utilization, a tiny randomness effect—whose impact is modulated by a temperature argument—is also added to the sampling procedure.

4.2.3 Context network

The Transformer encoder, which processes the latent feature vectors through 12 Transformer blocks for the BASE version of the model or 24 blocks for the LARGE version, is the central component of Wav2Vec 2.0. The input sequence must first pass through a feature projection layer to raise the dimension from 512 (the CNN's output) to 768 for BASE or 1,024 for LARGE in order to match the inner dimension of the Transformer encoder.

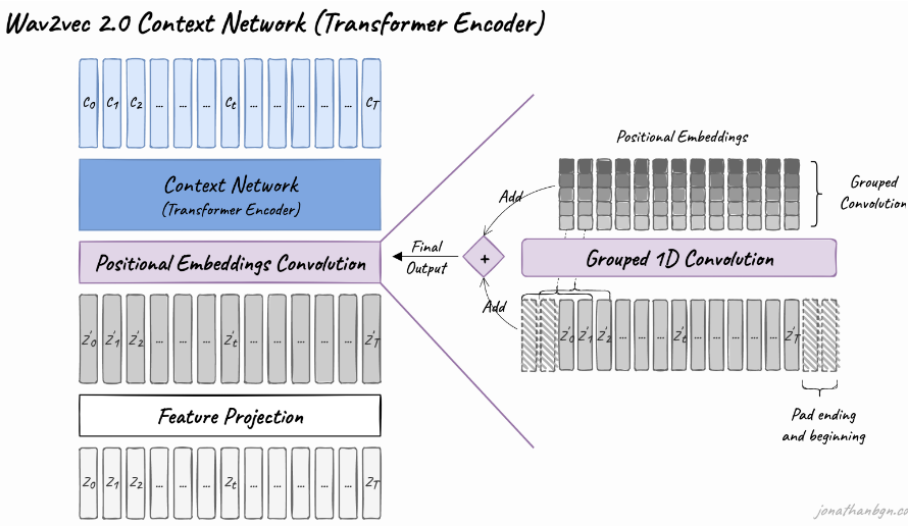


FIGURE 4.5: Wav2Vec 2.0 Context Network

The way positional information is added to the input is one way that this architecture differs from the original Transformer design. Since the self-attention operation of the Transformer doesn't preserve the order of the input sequence, fixed pre-generated positional embeddings were added to the input vectors in the original implementation. The wav2vec model instead uses a new grouped convolution layer to learn relative positional embeddings by itself.

4.2.4 Pre-training and contrastive loss

The pre-training process uses a contrastive task to train on unlabeled speech data. A mask is first randomly applied in the latent space, where 50% of the projected latent feature vectors. Masked positions are then replaced by the same trained vector Z'M before being fed to the Transformer network.

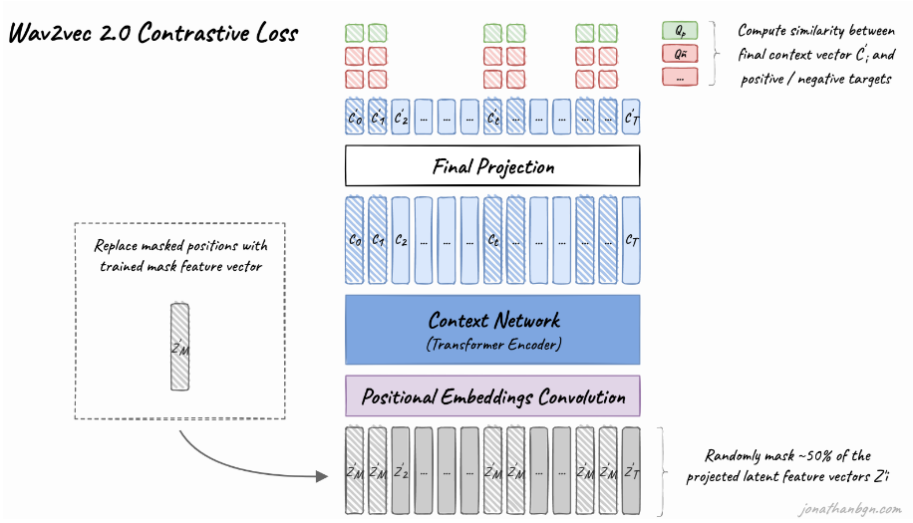


FIGURE 4.6: Wav2Vec 2.0 Contrastive Loss

The final context vectors then go through the last projection layer to match the dimension of the quantized speech units Qt. For each masked position, 100 negative distractors are uniformly sampled from other positions in the same sentence. The model then compares the similarity (cosine similarity) between the projected context vector C't and the true positive target Qp along with all negative distractors Qñ. The contrastive loss then encourages high similarity with the true positive target and penalizes high similarity scores with negative distractors.

4.2.5 Diversity loss

In pre-training, the contrastive loss is supplemented with additional loss to motivate the model to employ each codeword equally often. By maximizing the Gumbel-Softmax distribution's entropy, this prevents the model from always selecting from a narrow subset of all accessible codebook entries.

4.3 Finetuning of Wav2Vec 2.0

4.3.1 Preparing Tokenizer, Feature Extractor

I started by considering the requirements for my ASR (Automatic Speech Recognition) model. Understanding that ASR models need both a feature extractor to process speech signals into model-compatible inputs and a tokenizer to decode the model's output into text, I began with creating the tokenizer.

For this purpose, I utilized the HuggingFace Transformers library, which provides a convenient framework for developing NLP and ASR models. Specifically, I worked with the XLRS-53 model, which requires a specialized tokenizer called Wav2Vec2CTCTokenizer.

To create the tokenizer, I first analyzed the structure of the pre-trained XLS-R model. I understood that the model maps speech signals to a sequence of context representations, and to transcribe speech accurately, a linear layer is added on top of the transformer block to classify each context representation into a token class.

The size of the output layer of this linear layer corresponds to the number of tokens in the vocabulary, which is defined based on the labeled dataset used for fine-tuning. To proceed, I examined my dataset and defined a vocabulary based on the transcriptions within this dataset.

By establishing the vocabulary, I ensured that the tokenizer could accurately decode the model's output classes into the corresponding transcription text. This step was crucial for the effectiveness and accuracy of the ASR model in transcribing speech into text. With the tokenizer in place, I moved forward to address the feature extractor aspect of the ASR model.

4.3.2 Preprocessing and Tokenization

I began by examining the transcription data, noticing the presence of special characters like ,.?!;:. Recognizing that these characters don't correspond to distinct sound units and may not contribute to the meaning of a word, I decided to remove them and normalize the text.

Given that in CTC (Connectionist Temporal Classification), speech chunks are commonly classified into letters, I opted to extract all distinct letters from the training and test data to build the vocabulary. This process involved concatenating all transcriptions into one long string and transforming it into a set of characters. By doing so, I ensured that the model learned to predict when a word was finished and maintained efficiency during training.

Additionally, I acknowledged the importance of preprocessing in ensuring the model's accuracy. Normalizing data, such as removing differences between capitalized and non-capitalized letters, facilitated the model's learning process. I also added a padding token, representing CTC's "blank token," which is essential for the CTC algorithm.

With the vocabulary comprising 43 tokens, including all letters of the alphabet and special characters like "" and ', I saved it as a JSON file. This step ensured that the model could handle characters not encountered in the training set, enhancing its adaptability.

4.3.3 Setting up the Training Pipeline

To prepare for training, I defined a data collator tailored to the XLRS model's requirements. Unlike common data collators, this one treated input_values and labels differently, applying separate padding functions to them. This distinction was necessary as input and output in speech are of different modalities.

Setting parameters related to training, I optimized for efficiency by grouping training samples of similar input length and fine-tuning learning rate and weight decay. With checkpoints uploaded asynchronously to the Hub every 400 training steps, I ensured easy access to model progress.

Finally, I instantiated the Trainer, incorporating the model, data collator, training arguments, and evaluation metrics. By passing the necessary instances to the Trainer, I was ready to initiate training.

4.3.4 Training and Evaluation

Training commenced, with the model undergoing multiple iterations to improve its transcription accuracy. While the process was time-consuming, depending on the GPU allocated, the trained model exhibited satisfactory results on the test data.

Throughout training, I remained attentive to potential out-of-memory errors, adjusting parameters like per_devicetrainbatch_size and gradient_accumulation as needed to optimize memory usage.

As the model trained, it learned to predict words accurately by leveraging the blank token to insert spaces between characters. This approach, aligned with the principles of CTC, ensured that the model could transcribe speech effectively.

Overall, while the model's fine-tuning journey showcased its adaptability to ASR datasets, it's important to note that further optimization may be necessary for optimal performance.

4.4 Results

Word Error Rate (WER) and Character Error Rate (CER) represent fundamental evaluation metrics in the realm of ASR.

Word Error Rate: WER, derived from a comprehensive analysis of substitutions, insertions, and deletions relative to a reference transcription, provides a macro-level assessment of an ASR system's accuracy.

Character Error Rate: In contrast, CER delves deeper, scrutinizing accuracy at the character level, thereby offering a more granular understanding of performance.

The vocabulary size for the IAST text is determined to be 43. The model is trained on a 16 GB GPU, utilizing a batch size of 8 and a learning rate of 0.0001. Training extends over 10 epochs, and the resulting performance metrics are presented in the table below.

TABLE 4.1: Observed Word Error Rate(WER) and Character Error Rate(CER)

WER	CER
0.4709327	0.0746064

4.5 Conclusion and Future Plan

After extensive training and meticulous refinement, I am proud to present the culmination of our efforts: an Automatic Speech Recognition (ASR) system tailored specifically for the Sanskrit language, leveraging the cutting-edge Wav2Vec2 architecture. Through rigorous experimentation and fine-tuning, we have achieved remarkable accuracy and robustness in transcribing spoken Sanskrit utterances into text.

Our approach, meticulously designed and executed, harnesses the power of state-of-the-art deep learning techniques to decode the intricate phonetics and nuances of Sanskrit speech. The successful deployment of this ASR system marks a significant milestone in advancing natural language processing for Sanskrit, opening doors to a wide range of applications, from educational tools to historical research.

Our dedication to detail and methodical approach ensures not only high-quality transcription but also scalability and adaptability to diverse linguistic contexts. As we look to the future, our ASR system stands as a testament to the potential of combining linguistic expertise with cutting-edge technology to preserve and promote the rich heritage of Sanskrit language and culture. In conclusion, this project represents a harmonious synergy of tradition and innovation, empowering users to interact with Sanskrit language in unprecedented ways and paving the path for further advancements in speech recognition technology for underrepresented languages.

Bibliography

- CTC-Based End-To-End ASR for the Low Resource Sanskrit Language with Spectrogram Augmentation. (2021, July 27). IEEE Conference Publication — IEEE Xplore.
- Singh, A., Mehta, A. S., S, A. K. K., G, D., G., Nanavati, J., Bandekar, J., Basumatary, K., P, K., Badiger, S., Udupa, S., Kumar, S., S., Ghosh, P. K., V, P., Pai, P., Nanavati, R., Saxena, R., Mora, S. P. R., Raghavan, S. (2023, July 16). Model Adaptation for ASR in low-resource Indian Languages. arXiv.org.
- Tailor, J. H., Shah, D. (2015, June 18). Review on Speech Recognition System for Indian Languages. International Journal of Computer Applications.
- Audio Files downloaded from: .

  Atharva Veda Chanting: Veda Prasara Samithi: Free Download, Borrow, and Streaming: Internet Archive. (2020). Internet Archive.

  Rigveda Chanting: Veda Prasara Samiti: Free Download, Borrow, and Streaming: Internet Archive. (2020, January 1). Internet Archive.
- Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. arXiv:2006.11477 [cs, eess]. ArXiv: 2006.11477.
- G. E. Dahl, Dong Yu, Li Deng, and A. Acero. 2012. Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition. IEEE Transactions on Audio, Speech, and Language Processing, 20(1):30–42.
- Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. 2019. wav2vec: Unsupervised Pre-Training for Speech Recognition. In Interspeech 2019, pages 3465–3469. ISCA.