University of Padova

Department of Information Engineering

Master Thesis in Computer Engineering

Embedded Speech Technology

Supervisor
Prof. Satta Giorgio
University of Padova

Co-supervisor
Domenico Crescenzo
Screevo

Master Candidate
Hafiz Muhammad Zakria

Student ID
2013045

Academic Year
2022-2023

“All truths are easy to understand once they are discovered; the point is to discover them.”
— Galileo Galilei

Abstract

End-to-End models in Automatic Speech Recognition simplify the speech recognition process. They convert audio data directly into text representation without exploiting multiple stages and systems. This direct approach is efficient and reduces potential points of error. On the contrary, Sequence-to-Sequence models adopt a more integrative approach where they use distinct models for retrieving the acoustic and language-specific features, which are respectively known as acoustic and language models. This integration allows for better coordination between different speech aspects, potentially leading to more accurate transcriptions.

In this thesis, we explore various Speech-to-Text (STT) models, mainly focusing on End-to-End and Sequence-to-Sequence techniques. We also look into using offline STT tools such as Wav2Vec2.0, Kaldi and Vosk. These tools face challenges when handling new voice data or various accents of the same language. To address this challenge, we fine-tune the models to make them better at handling new, unseen data. Through our comparison, Wav2Vec2.0 emerged as the top performer, though with a larger model size. Our approach also proves that using Kaldi and Vosk together creates a robust STT system that can identify new words using phonemes.

Contents

Listing of figures

Listing of acronyms

ASR	Automatic Speech Recognition
E2E	End-to-End
AM	Acoustic Model
LM	Language Model
STT	Speech-To-Text
RNN	Recurrent Neural Network
RNN-T	Recurrent Neural Network-Transducer
CTC	Connectionist Temporal Classification
RNA	Recurrent Neural Aligner
NLU	Natural Language Understanding
HMM	Hidden Markov Model
DTW	Dynamic Time Warping
WER	Word Error Rate
CER	Character Error Rate
GMMs	Gaussian Mixture Models
TTS	Text-To-Speech
RPA	Robotic Process Automation

1 Introduction

1.1 Research Goal and Scope

Speech recognition systems are engineered to interpret and transcribe human speech into text. However, the complex details of how people speak create several problems including variations in accents as there are multiple accents for the same language across the globe. Moreover, the intonation patterns including the rise and fall in pitch while speaking can make the speech recognition systems less robust.

These challenges necessitate that speech recognition systems be equipped with vast datasets and sophisticated algorithms to cater to the wide range of human speech variations. An emerging solution is to convert spoken sentences into a sequence of phonemes, the smallest units of sound in speech, before transcribing them to text [1]. This intermediate phonetic representation can then undergo further processing to derive the intended text.

The goal of this research is to analyze various methodologies of Automatic Speech Recognition (ASR) that can address the challenges of converting speech to phonemes including:

- **Hidden Markov Models (HMMs):** Traditionally, Hidden Markov Models (HMMs) have been the go-to for speech recognition tasks. Each phoneme is modeled as a state chain with its respective transition probabilities.

- **Deep Neural Networks (DNNs):** Here, a neural network undergoes training to associate acoustic features from speech to phoneme labels.

- **Transfer Learning and Pre-trained Models:** Models like wav2vec, which come pre-trained on massive datasets, can be fine-tuned for specific phoneme recognition tasks to achieve high accuracy.

- **End-to-End Models:** These models intend to directly link the input speech signal to the output, be it phonemes or voice, without requiring any intermediate representation.

It also explains Speech-to-Text tools designed for the methods mentioned above, mainly highlighting End-to-End Models and DNNs. Key tools in this domain are Wav2Vec2.0, Kaldi and Vosk. We'll explore how to generate phonemes, train the Acoustic Model (AM) and Language Model (LM). Additionally, the fine-tuning of Wav2Vec2.0 with and without LM, fine-tuning of Vosk models, when integrated with Kaldi training using voice datasets, aims to enhance the accuracy of speech recognition systems. Consequently, by boosting accuracy, the speech recognition systems can better handle the variations of human speech. So, the lower the Word-Error-Rate (WER), the more accurate and robust the speech-to-text model is.

1.2 Thesis Structure

The thesis is organized as follows:

- Chapter 2 gives the overview of ASR and technologies associated to it
- Chapter 3 introduces the Speech-to-Text Tools
- Chapter 4 presents the Screevo Model and it's implementation
- Chapter 5 demonstrates about research problem and it's solution
- Chapter 6 shows the results achieved
- Chapter 7 concludes the thesis and discusses potential future work

2 Automatic Speech Recognition (ASR)

This chapter begins with an overview of the Automatic Speech recognition (ASR) architecture. Then it explains in detail about different technologies of ASR, followed by a detailed overview of Speech-To-Text (STT) components.

2.1 Background

In recent years, recurrent neural networks (RNNs) have revolutionized fields like machine translation, speech recognition, time series prediction, and more through their ability to be trained End-to-End without prior knowledge. A RNN is any network in which neurons send feedback signals to each other and are usually applied in conjunction with other models, where one component of the system is replaced with a neural network.

As an example, in speech recognition, a neural network model is trained to output grapheme or word sequences for a given utterance without utilizing external languages. This advancement in neural network technology has paved the way for more efficient and accurate speech-to-text systems. By eliminating the need for external language resources, these models can adapt to a wider range of dialects and accents, making them increasingly valuable in a globalized world where linguistic diversity is the norm. Additionally, the ability to train end-to-end models reduces the complexity of the system, making it more accessible for various applications, from voice assistants to transcription services and beyond. This trend of self-contained, end-to-end neural network solutions holds great promise for the future of artificial intelligence and natural language processing.

2.2 End-to-End Models

End-to-end models refer to architectures that take raw data as input and directly produce the desired output, eliminating the need for traditional hand-crafted features or intermediate processing steps. This approach contrasts with traditional methods where tasks might be broken down into distinct sub-tasks, each requiring specialized processing. By eliminating pre-processing, these models can often uncover intricate patterns in the data that might be missed by conventional methods. Popular in fields like speech recognition, machine translation and autonomous driving, end-to-end systems have demonstrated superior performances.

E2E learning mentions the training of a potentially complex learning system by a single model that shows the complete target system. It is an interesting topic in the field deep learning field to exploit the structure of deep neural network's (DNNs), composed of multiple layers, to solve complex problems like automatic speech recognition [2].

2.2.1 Recurrent Neural Network - Transducers

End-to-End ASR models like the RNN-Transducer have become popular. The Recurrent Neural Network Transducer, denoted by "RNN-T," is a model proposed by Alex Graves [3] where he demonstrated that RNN-T is a reasonable model for application in speech recognition and he proved his claim by achieving good results on small datasets as demonstrated in the Table 2.1.

However, the RNN-T has been underutilized when compared to Connectionist Temporal Classification (CTC) models which is explained later in this chapter. After a while, the RNN-T gained serious attention when Google researchers demonstrated that it could enable fully

Table 2.1: Phoneme Recognition Results on the TIMIT Speech Corpus

Network	Epochs	Log-Loss	Error Rate
CTC	96	1.3	25.5%
Transducer	76	1.0	23.2%

on-device, low-latency speech recognition for Pixel phones. And recently, the RNN-T was utilized to achieve improved word error rates for the Libri Speech benchmark [4].

2.2.2 RNA - Recurrent Neural Aligner

Recurrent Neural Aligner, denoted by RNA, is a restricted version of RNN-Transducer. In fact, RNA is a neural network model in an encoder-decoder framework that can be taught end-to-end to map input sequences to target sequences. For $x = (x_1, ..., x_T)$, RNA model tries to predict $y = (y_1, ..., y_N)$ as an object sequence [5].

The model can be defined as a conditional distribution, $P(z|x)$, where $z = (z_1, ..., z_T)$ and demonstrates one of the possible alignments between the sequences x and y. In order to estimate the distribution over the sequences y, it is marginalized over all alignments as follows:

$$P(y|x) = \sum_z P(z|x) \tag{2.1}$$

RNA model contains an encoder network that may be a unidirectional or bidirectional RNN, or any other neural network. The decoder network has a soft-max output layer of size $L + 1$ units, where L stands for the number of labels in the output space and it has an extra unit for the blank label which helps RNA model to generate a label for every input vector. For a given z, the input to the decoder network at time t is the concatenation of input vector x_t and one-hot encoded label vector z_{t1}. Therefore,

$$P(z|x) = \prod_t P(z_t|x) \tag{2.2}$$

Advantages of End-to-End Models:

- **Better metrics:** Currently, the systems that perform best based on metrics such as precision and recall are end-to-end models.
  - Reason: Performance is often the primary criterion when evaluating systems. If end-to-end models outperform others based on standard metrics, it's a compelling reason to use them.

- **Simplicity:** End-to-end models pass the sometimes difficult problem of determining which components are required to perform a task.
  - Reason: Simplicity can accelerate development, reduce the potential for errors, and make the model easier to understand and modify.

- **Applicability to New Tasks:** End-to-end models are able to work by applying new data for a new task, but a significant re-engineering is needed using component-based systems.
  - Reason: Flexibility and adaptability to new tasks without significant re-engineering is crucial, especially in rapidly evolving fields like machine learning.

2.3 Sequence-to-Sequence Models

Sequence-to-sequence models are deep learning model that have been used in many fields such as machine translation, text summarization and image captioning. In fact, they are a special class of Recurrent Neural Network architectures that are used to solve complex Language problems. And some of them are explained as follows.

2.3.1 Connectionist Temporal Classification

CTC models are widely used in various sequence-to-sequence tasks, with one of their primary applications being speech recognition. They are designed to handle situations where the alignment between input and output sequences is not one-to-one, making them suitable for variable-length inputs and outputs.

- **Monotonic Input-Output Alignment:** One of the fundamental assumptions of CTC models is that they assume a monotonic alignment between the input and output sequences. This assumption simplifies the modeling process significantly [6].
- **Length of Output Sequence:** CTC models require that the length of the output sequence (denoted as U) is less than or equal to the length of the input sequence (denoted as T). In speech recognition, this is usually not a problem since the input audio signal (T) is typically much longer than the recognized words or phrases (U). However, this constraint may limit the applicability of CTC models in scenarios where extensive pooling or compression of data is required to speed up processing.
- **Independence of Outputs:** Another assumption of CTC models is that they treat the outputs as conditionally independent of each other. This means that the model predicts each element of the output sequence independently based on the input sequence. This can sometimes lead to incorrect outputs, such as misrecognizing ”You ate Pizza” as ”You eight Pizza.”

2.3.2 Sequence-to-Sequence with CTC

Connectionist Temporal Classification (CTC) has been used for training utterances without requiring a frame-level alignment of the target labels. It enhances the set of target labels with an extra “blank” symbol [7].

Let $B(y,x)$ be the set of all sequences stand for the labels in $\mathcal{Y} \bigcup (b)$, where (b) corresponds to an additional “blank” symbol. every sequence in $B(y,x)$ stands for a frame-level assignment of the label sequence in y. The probability of the output sequence conditioned on the acoustics is defined by CTC as follows:

$$P(y|x) = \sum_{\hat{y} \in B(y,x)} \prod_{t=1}^{T} P(\hat{y}_t|\mathbf{x}), \tag{2.3}$$

such that labels at any time step are independent. $P(\hat{y}_t|\mathbf{x})$ can be estimated applying a deep recurrent neural network, as an encoder. In Figure 2.1, a sequence of vectors $h^{enc} = (h_1^{enc}, ..., h_T^{enc})$ is calculated by the encoder. This sequence of vectors are treated as logits and passed to a single softmax. Training the model to maximize the equation 2.3 is possible using gradient descent,

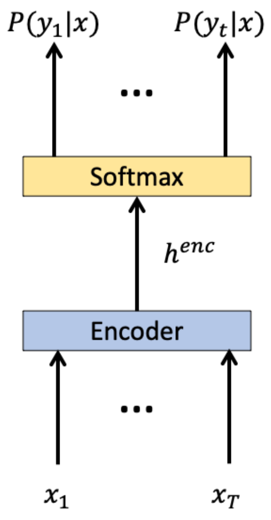


Figure 2.1: A schematic representation of the vector sequences in an encoder

where a forward-backward algorithm can be used for computing the required gradient.

2.3.3 Transformers

Transformers, a type of artificial neural network architecture, are applied to solve the problem of transformation of input sequences into output sequences in deep learning applications. Indeed, they take advantage of the concepts of attention and self-attention in a set of encoders and decoders [8].

By relying on the concepts of attention and self-attention, transformers can represent a wider range of context in a given sequence. They also can improve results for important natural language processing like as speech recognition. In addition, transformers can produce better results compared to other sequential models, since we can apply parallelization.

Encoder and Decoder Stacks

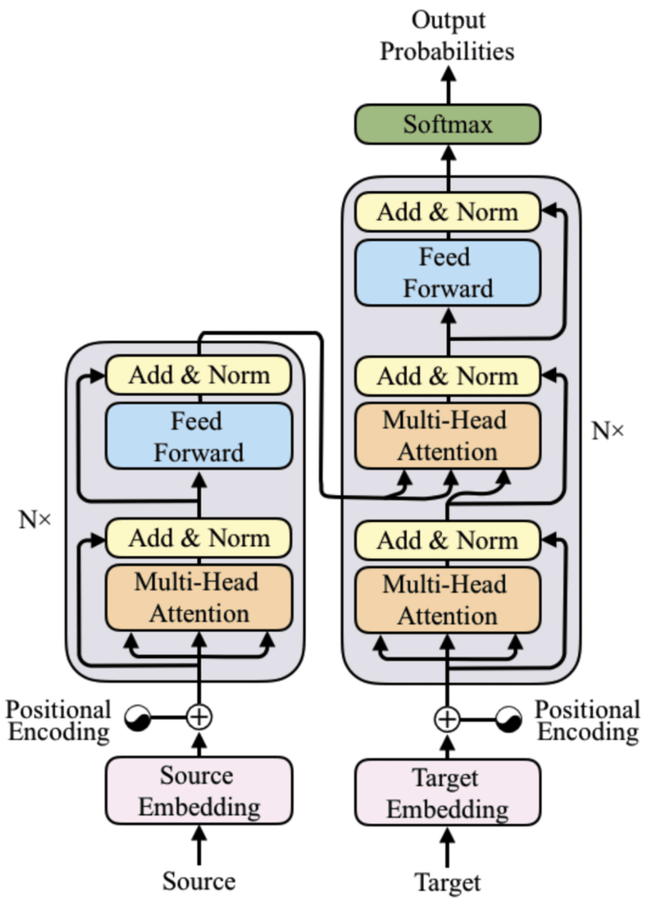


Figure 2.2: Transformer - Model architecture

Transformer has a structure that consists of self-attention and point-wise, fully-connected layers for both the encoder and decoder, as shown in Figure 2.2.

2.4 Speech-to-Text

Speech recognition, also known as automatic speech recognition (ASR), computer speech recognition or speech to text (STT) is a branch of natural language understanding (NLU). It focuses on methodologies and technologies that use computers to recognize and translate spoken language into text. Some speech recognition systems need training in which a speaker reads text or individual words into the system that analyzes the person's specific voice to increase the accuracy.

These developments are not only evident in academia but, more importantly, in the universal industry acceptance of a diverse range of deep learning methods for designing and deploying speech recognition systems. Following are some of the models and methods.

- **Hidden Markov Model (HMM):** It is a statistical model which is used in speech recognition because a speech signal is seen as a piece-wise stationary signal or a short-time stationary signal. Moreover, speech is approximated as a stationary process for a short time scales and in many stochastic purposes, it can consider as a Markov model.
- **Dynamic time warping (DTW)-based speech recognition:** Dynamic time warping is used as an algorithm to measure similarity between two sequences. More recently, it is used in video, audio and graphics processing.
- **Neural networks:** Neural networks have appeared as an attractive audio modeling approach in ASR. Compared to HMM, neural networks make less explicit assumptions about the statistical properties of features and they have several features that make them quite interesting recognition models for speech recognition. For example, sequence-to-sequence with CTC, Transformers and others.

2.5 Evaluation Metric

In machine learning, it is so important to measure the quality of models and one of the most important methods used, is evaluation metrics. There are different evaluation metrics in a different set of machine learning algorithms. We use classification metrics to evaluate classification models and regression metrics to evaluate regression models.

Classification: A category is predicted regarding to some inputs. These problems try to categorize a data point into a specific group. The target outcome will be a discrete/categorical values.

Regression: A certain number is predicted regarding to some inputs. These problems use input variables to predict continuous values by using only training data. The target outcome is a quantity/real value such as time-series data, weights, etc.

2.5.1 Word Error Rate - WER

Word error rate (WER) is a common metric of the performance of a speech recognition or machine translation systems. WER is derived from the Levenshtein distance, which operates at the word level rather than the phoneme level. The WER can be used to compare various systems and also to evaluate improvements in a system. However, this method does not provide details about the nature of translation errors and therefore more work is needed to identify the main source of error.

This problem is solved by aligning the detected word sequence with the reference word sequence using dynamic string alignment. To address this issue, [9] expresses the correlation between confusion and word error rate. Word error rate is given by:

$$WER = \frac{S+D+I}{N} = \frac{S+D+I}{S+D+C} \tag{2.4}$$

where S is the number of substitutions, D is the number of deletions, C is the number of correct words, I is the number of insertions and N is the number of words in the reference.

2.5.2 Character Error Rate - CER

Character error rate (CER) is a common metric of the performance of an automatic speech recognition system. CER is similar to WER, but works on character rather than word. CER is given by:

$$CER = (S+D+I)/N = (S+D+I)/(S+D+C), \tag{2.5}$$

where S is the number of substitutions, D is the number of deletions, I is the number of insertions, C is the number of correct characters, N is the number of characters in the reference (N=S+D+C). The output of CER is often represented by the percentage of incorrectly predicted characters. Consequently, the lower the value is, the better is the performance of the ASR system.

3 Speech-to-Text Tools

In this chapter, different Speech-to-Text tools have been explained with a brief overview about their architecture and applicability.

3.1 Kaldi

Kaldi is a tool designed for researchers to help with speech recognition. It's written in C++ and is free to use and modify under the Apache License v2.0. It can work well with Finite State Transducers (FSTs), has great math support and is flexible in design. An overview of kaldi operations is represented in Figure 3.1.

3.1.1 HMM Topology and Transition Modeling

The way Kaldi represents HMM topologies and how it uses them to train HMM transitions is outlined in the subsequent points:

- **HMM topologies:** Kaldi uses the HmmTopology class to specify the structure of HMMs for phones. Normally, a text form of the HmmTopology object is created and passed to command-line programs as shown in Table 3.1.

Table 3.1: HMM Topology Entry for Phones 0 1 2

State	PdfClass	Transitions
0	0	$0 \rightarrow 0.5$, $1 \rightarrow 0.5$
1	1	$1 \rightarrow 0.5$, $2 \rightarrow 0.5$
2	2	$2 \rightarrow 0.5$, $3 \rightarrow 0.5$

- **Pdf-classes:** These relate to the HmmTopology object, which sets a prototype HMM for each phone. Each HMM state has two main attributes: ”forward-pdf-class” and ”self-loop-pdf-class”. The ”self-loop-pdf-class” is associated with the self-loop transition and, by default, mirrors the ”forward-pdf-class”. If two states share the same pdf-class in the same phonetic context, they will have the same probability distribution function (p.d.f). This is due to the decision-tree code seeing only the pdf-class, not the HMM-state.

- **Decision Trees:** The algorithm Kaldi uses for decision-tree building is a top-down greedy splitting approach. It identifies the optimal way to split data by examining various factors like the left, central, and right phone, the state, and others [10]. In Kaldi, the tree-building process starts even with a mono-phone system, albeit producing a simple decision tree. Post monophone system training, the mono-phone alignments are used to accumulate statistics for tree building. This accumulation can come from mono-phone or context-dependent alignments, facilitating the construction of trees based on various alignments. Once statistics are compiled, the ”build-tree” program is used to establish the tree, utilizing statistics, questions configuration, and a roots file. Integral to this is the PDF (Probability Distribution Function) identifier, known as pdf-id. Each p.d.f. in the system is assigned a unique pdf-id, acting as an index.

- **Transition models:** The Transition Model object holds transition probabilities and HMM topologies information. Graph-building relies on this object to access topology and transition probabilities. Transition modeling usually considers: phone, source HMM-state, forward-pdf-id, self-loop-pdf-id, and transition index in the HmmTopology object.

3.1.2 Decoding Graph Construction:

- **Graph Creation:** Kaldi constructs the decoding graph as HCLG = H o C o L o G, where:

  - G represents the grammar or language model.
  - L stands for the lexicon, linking words to phones.
  - C encodes context-dependency, transforming context-dependent phones to windowed phone sequences.
  - H holds the HMM definitions, mapping transition-ids to context-dependent phones.

- **Disambiguation symbols:**

  Symbols like #1, #2, #3 and so on are appended to phoneme sequences in the lexicon for differentiation when sequences are prefixes of another or appear in multiple words. This ensures the determinizability of L o G.

- **Determinizability Considerations:**

  It's vital for the graph compilation stages to be determinizable to ensure the reliability of the Kaldi recipe. G must be determinizable, and for any such G, L o G should also be determinizable and the lexicon L must be functional in its inverse.

3.1.3 Feature Extraction

The tool compute-mfcc-feats is employed to calculate the Mel-Frequency Cepstral Coefficients (MFCC) features from audio data. This command-line program necessitates two arguments: an rspecifier for reading the .wav files (sorted by utterance) and a wspecifier to save the generated features (also sorted by utterance). Generally, the computed data is saved to a single "archive" file, but an accompanying "scp" file is also created to facilitate straightforward random access. The actual process of deriving the MFCC features is handled by an object from the MFCC class, utilizing its Compute() function to transform the audio waveform into features.

3.2 VOSK - Offline Speech Recognition

Vosk is a speech recognition toolkit. Some of the standout features of Vosk include:

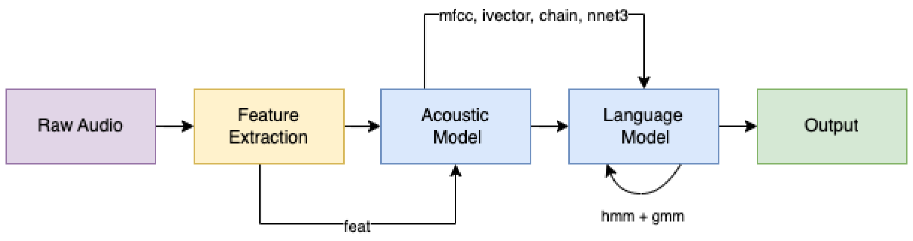


Figure 3.1: Flow of Kaldi Actions

- Supports 20+ languages and dialects - English, Indian English, German, French, Spanish, Portuguese, Chinese, Russian, Turkish, Vietnamese, Italian, Dutch, Catalan, Arabic, Greek, Farsi, Filipino, Ukrainian, Kazakh, Swedish, Japanese, Esperanto, Hindi, Czech, Polish, Uzbek, Korean. More to come.
- Works offline, even on lightweight devices - Raspberry Pi, Android, iOS.
- Portable per-language models are only 50Mb each, but there are much bigger server models available.
- Provides streaming API for the best user experience (unlike popular speech-recognition python packages)
- There are bindings for different programming languages.
- Allows quick reconfiguration of vocabulary for best accuracy.
- Supports speaker identification beside simple speech recognition.

3.2.1 Models

Small models are compact, about 50Mb in size, and use around 300Mb of run time memory. They're perfect for mobile apps, desktop applications, and devices like Raspberry Pi. Additionally, small models can adapt their vocabulary on-the-fly. Big models, on the other hand, are designed for high-precision transcription on servers. They need up to 16GB of memory due to their advanced AI features. These models are best run on high-end servers like the i7 or the latest AMD Ryzen, with cloud options like AWS's c5a machines.

3.2.2 Vosk Language Model Adaptation:

Information sources in speech recognition: Traditional Vosk models differentiate data sources into an acoustic model, language model, and phonetic dictionary, offering flexibility in updating and adapting knowledge sources. However, newer models, like Transformers and Conformers, integrate all these sources into a single neural network, increasing accuracy and decoding speed but complicating domain transfers and new word introductions. Vosk remains a preferred choice for domain-specific tasks, blending generic acoustic models with domain-specific language models for optimal recognition.

Accuracy issues: Speech recognition accuracy can vary due to multiple factors including poor audio quality, mismatched vocabularies, accents differing from training data, audio frame issues, and software glitches. Diagnosing issues requires in-depth analysis of components like audio quality checks, language model relevancy, and software comparisons.

3.3 Difference between Kaldi and Vosk:

Kaldi is a research speech recognition toolkit that implements many state-of-the-art algorithms. If you are conducting research, Kaldi is probably the best choice for you.

Vosk is a practical speech recognition library equipped with a set of accurate models, scripts, and best practices. It offers ready-to-use speech recognition for various platforms, including mobile applications and Raspberry Pi. If your aim is to build practical applications with a plug-and-play library, Vosk is the recommended option. Vosk draws from the best practices of many speech recognition tool-kits, not just Kaldi.

3.4 Wav2Vec

This new model, wav2vec 2.0, is designed to identify and predict basic speech units in audio. The model is trained to predict the correct speech unit for masked parts of the audio, while at the same time learning what the speech units should be. With just 10 minutes of transcribed speech and 53K hours of unlabeled speech, wav2vec 2.0 has enhanced speech recognition capabilities, achieving a word error rate (WER) of 4.8 on the clean/other test sets of the LibriSpeech benchmark [11]. This opens the door for speech recognition models in many more languages, dialects, and domains that previously required much more transcribed audio data to provide acceptable accuracy.

Similar to the Bidirectional Encoder Representations from Transformers (BERT), this model is trained by predicting speech units for masked parts of the audio. A major difference is that speech audio is a continuous signal that captures many aspects of the recording with no clear segmentation into words or other units.

3.4.1 Architecture

- **Learning discrete latent speech units:** Traditional speech recognition models heavily rely on annotated audio data for training. Meanwhile, some self-supervised methods for speech aim to recreate the entire audio signal. These methods have the challenge of accounting for all elements of speech, including the recording environment, any background noise, and unique speaker characteristics.

  In a newer development, a Deep Bidirectional Transformer is trained on discretized unlabeled speech data. These processed data points are then fed into a conventional acoustic model. Impressively, BERT representations outperform both log-mel filter bank inputs and the more dense wav2vec representations, as seen in the TIMIT and WSJ benchmarks[12]. By discretizing audio, it becomes possible to directly apply many algorithms, initially designed for NLP, to speech data. This transformation demonstrates, for instance, that a typical sequence-to-sequence model from NLP can effectively handle speech recognition over these discrete audio tokens.

- **Cross-lingual training:** To address the limited data issue for some languages, cross-lingual training has been explored. This technique involves training a single model on several languages simultaneously, yielding superior results compared to training on just one language. Such an approach has been notably effective in natural language processing, especially with the XLM-R model [13].

This methodology markedly enhances the performance for languages with scarce resources, as they leverage information from related languages. Wav2vec 2.0 identifies common speech units across different languages and interestingly, while some units are exclusive to certain languages, others appear in multiple languages, irrespective of their similarity. The process is shown in Figure 3.2 for training wav2vec2.0 cross-lingually.

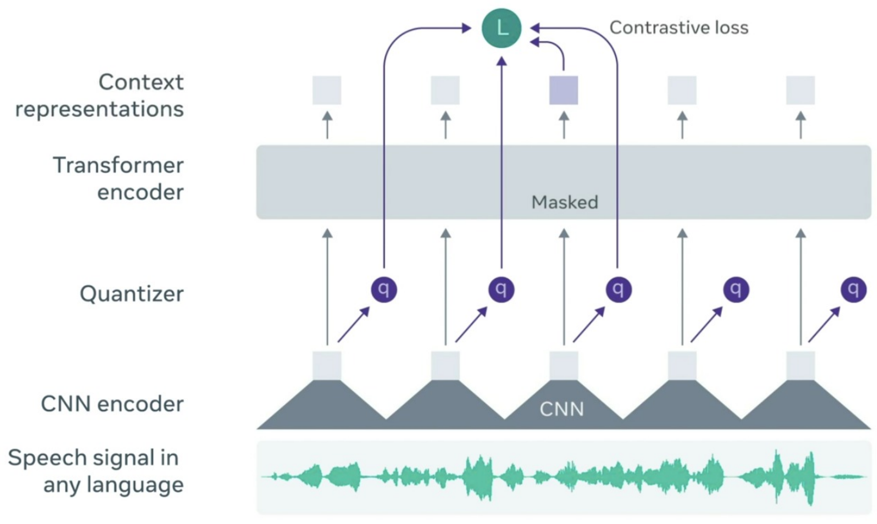


Figure 3.2: Cross-lingual training via wav2vec 2.0

4 Screevo Model

Screevo is the Voice Assistant for Desk-less Workers that allows the creation of voice controls on any software system, with no integration required. Technicians and operators spend 800 hours each year to enter data into forms, spreadsheets or software systems: about 40 percents of their time. The voice assistants created with Screevo are capable of guiding the user through business processes, automating data entry and allowing operators and technicians to focus on tasks with greater added value.

Screevo, through the use of natural language understanding (NLU) and robotic process automation (RPA) algorithms, allows the voice assistant to interact without any difficulty with any software system, eliminating complexity and friction.

4.1 Technological Components

Screevo boosts a technology capable of adapting to Android, iOS and Windows devices; it is able to control any type of software already present in the company and easily interfaces with ERP, MES, WMS systems and browsers. Speech recognition was designed and tested for the most complex pronunciations and highly noisy environments.

4.2 Architecture

Architecture of Screevo is demonstrated in Figure 4.1.

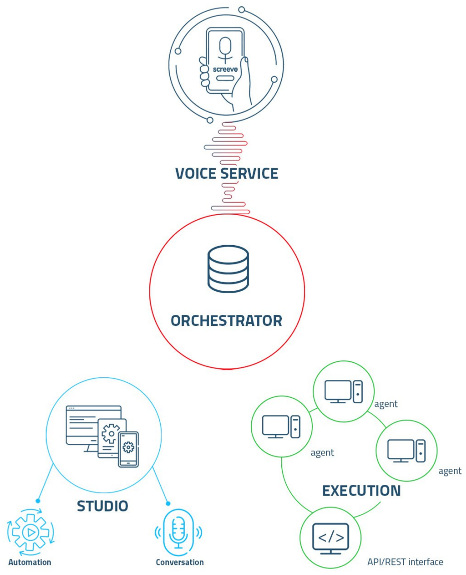


Figure 4.1: Screevo - Model Architecture

- **Orchestrator:** The Orchestrator is a server component that can be installed on premise or in the cloud and is responsible for orchestrating the data traffic between the voice service, on device and locally, and the Execution. It is the heart of the platform which can be accessed to consult the audit trail of the actions performed and monitor the performance status of the processes in progress.

- **Voice Service:** As for the Voice Service, Speech to Text (STT) via VOSK and Text to Speech (TTS) algorithms allow a natural interaction between the voice assistant and the user. Voice Assistant is multi-platform (Android, iOS and Web App) and the libraries developed in house do not require connections to external networks, allowing the use of Screevo in the company intranet.
  - Voice Service and Camera integration available to collect field pictures.
  - Voice Service and PDAs integration available to scan bar codes.
  - 20+ languages supported.

  - Extendable to domain-specific words
  - Resistant to noisy environment
  - Low to none latency

  The bio-metric data of the user entry is never saved, guaranteeing privacy and compliance with the regulations. Finally, Natural Language Understanding (NLU) algorithms allow voice assistant to interpret user requests and respond appropriately.

- **Execution:** It is the arm that performs the necessary automation to enable the voice control of third party software. This can be done through the installation of a Robotic Process Automation (RPA) agent on the local machine (physical or virtual) rather than through integration via REST API interfaces.

- **Studio:** Finally, the Studio part allows the process design for the customization of each single automation process. Within the studio it is possible to create conversations and / or automation and publish them on the Orchestrator to make them immediately available and ready for use.

5
Speech -to- Phonemes

In this chapter, the problems associated with Speech-to-Text (STT) including Wav2Vec2.0, Kaldi and Vosk are outlined and the implemented solutions to address them are discussed.

5.1 Problem

The Speech-to-Text (STT) models face challenges due to variations in how people speak. These variations are mainly caused by differences in the following factors:

- **Accent of the speaker:** People from different regions or countries have unique ways of pronouncing words. For example, the way someone from the United States says ”water” may sound different from how someone from the United Kingdom says it.
- **Pronunciations of the words:** Even within the same region, individuals may have their own way of pronouncing certain words. For instance, some people might say ”tomato” with a long ’a’ sound, while others might use a short ’a’ sound.
- **Position of Phonemes:** The specific arrangement of speech sounds, known as phonemes, can vary based on the individual’s speech habits. For instance, the placement of the ’l’ and ’n’ sounds in the word ”London” may differ slightly from person to person.

5.2 Practical Approaches

In this section, a hybrid solution is implemented to deal with the problems mentioned above. Following are the methods that can be addressed to improve the accuracy of STT models.

5.2.1 Fine-Tuning Wav2Vec 2.0

Wav2vec 2.0 has achieved notable results, achieving a WER around 5 when trained on the English dataset i.e. LibriSpeech dataset. To enhance its performance, we fine-tuned the model in both languages English and Italian by utilizing Connectionist Temporal Classification (CTC). CTC is an algorithm employed for training neural networks to tackle sequence-to-sequence problems, primarily in Automatic Speech Recognition, as detailed in Chapter 2

This fine-tuning is implemented without using Language Model. The process of fine-tuning is demonstrated in Figure 5.1.

- **Prepare Data and Tokenizer:** ASR models are tasked with transcribing speech to text. This includes a feature extractor to convert the speech signal into a format the model can process such as a feature vector and a tokenizer to transform the model's output format into text. We need to adjust a Wav2Vec2 checkpoint to match a series of context representations with the right transcription. To do this, we add a linear layer on top of the transformer. This layer classifies each context representation into a token class. The size of the output from this layer is determined by the number of tokens in the vocabulary. This size doesn't depend on the original task Wav2Vec2 was trained for, but only on the dataset used for fine-tuning. So, we used CommonVoice dataset with Wav2Vec2CTCTokenizer to create a vocabulary according to dataset transcribed text.

- **Remove Special Characters and Add Padding:** The special characters are removed by using this code:

```
chars_to_ignore_regex = '[\,\?\.\!\-\;\:\"]'

def remove_special_characters(batch):
    batch["text"] = re.sub(chars_to_ignore_regex, '',
        batch["text"]).lower() + " "
    return batch
```

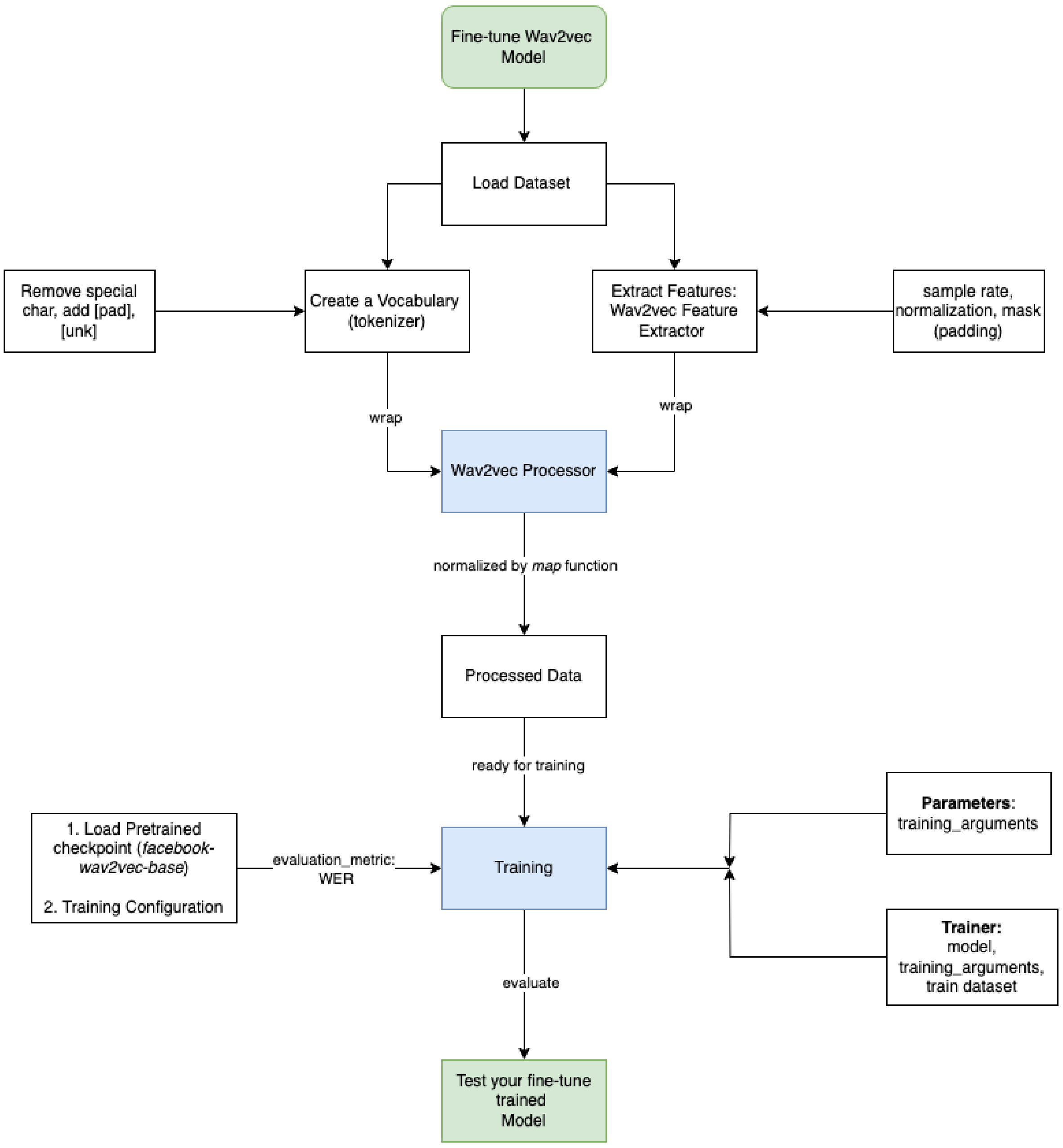

Fine-tune Wav2vec Model
Load Dataset
Remove special char, add [pad], [unk]
Create a Vocabulary (tokenizer)
Extract Features: Wav2vec Feature Extractor
sample rate, normalization, mask (padding)
wrap
wrap
Wav2vec Processor
normalized by *map* function
Processed Data
ready for training
Parameters: training_arguments
1. Load Pretrained checkpoint (*facebook-wav2vec-base*)
2. Training Configuration
evaluation_metric: WER
Training
Trainer: model, training_arguments, train dataset
evaluate
Test your fine-tune trained Model

Figure 5.1: FineTuning - Wav2Vec 2.0

Plus, we also include a padding token, which matches up with CTC's ”blank token.” This ”blank token” is a crucial part of the CTC algorithm. Consequently, the vocabulary is ready to be processed further for the Wav2Vec Processor.

```
vocab_dict["[UNK]"] = len(vocab_dict)
vocab_dict["[PAD]"] = len(vocab_dict)
```

- **Wav2Vec Feature Extractor:** To instantiate a Wav2Vec2 feature extractor object, several parameters are needed.
  - **feature size:** This defines the size of the feature vectors inputted into the speech models. For Wav2Vec2, it is set to 1 as the model is trained on raw speech signals.
  - **sampling rate:** This denotes the rate at which the model is trained.
  - **padding value:** This is used to pad shorter inputs for batched inference.
  - **do-normalize:** This determines whether the input should be normalized, which usually enhances the model's performance.
  - **return attention mask:** This decides whether the model should use an attention mask for batched inference. For Wav2Vec2's ”base” checkpoint, not using an attention mask yields better results due to a specific design choice.

    ```
    from transformers import Wav2Vec2FeatureExtractor
    feature_extractor = Wav2Vec2FeatureExtractor(feature_size=1,
        sampling_rate=16000, padding_value=0.0, do_normalize=True,
        return_attention_mask=False)
    ```

- **Training:** The data has been prepared and we are now ready to establish the training pipeline. We utilized HuggingFace Trainer, for which we essentially need to perform the following steps:
  - **Data Collator:** Define a data collator which is replicated from [14]
  - **Evaluation Metric:** During training, the model's performance should be assessed based on the word error rate. Consequently, we need to define a compute-metrics function to facilitate this evaluation.

    ```
    def compute_metrics(pred):
        pred_logits = pred.predictions
        pred_ids = np.argmax(pred_logits, axis=-1)
        pred.label_ids[pred.label_ids == -100] =
            processor.tokenizer.pad_token_id
    ```

```
pred_str = processor.batch_decode(pred_ids)
label_str = processor.batch_decode(pred.label_ids,
    group_tokens=False)

wer = wer_metric.compute(predictions=pred_str,
    references=label_str)
return {"wer": wer}
```

– **Pre-trained Checkpoint:** Load a pre-trained checkpoint. The model that is being used for fine-tuning is **facebook/wav2vec2-base**

```
model = Wav2Vec2ForCTC.from_pretrained(
 "facebook/wav2vec2-base",
 ctc_loss_reduction="mean",
 pad_token_id=processor.tokenizer.pad_token_id)
```

– **Training Configuration** Define a training configuration for the training environment.

The primary resources used for training are listed in the Table 5.1. And the training results are shown in the Table 5.2. In this context, the model was trained for 30 epochs, indicating that the entire dataset was passed forward and backward through the neural network thirty times. The Training Loss, a measure of how well the model is performing, was quite low at 0.021, suggesting that the model learned effectively from the training dataset. Finally, the Word Error Rate (WER), a common metric in speech recognition that measures the performance of the model in transcribing audio to text, was 9.40.

5.2.2 Language Model with Wav2Vec2.0 - Our Solution

In this section, we will explore the process of integrating a Language Model (LM) with Wav2vec 2.0 to achieve enhanced results.

Table 5.1: Specification - Wav2vec2.0 Training

Dataset	CommonVoice
Language	IT
Model	wav2vec2-large-xlsr-53-italian
Training Time (hr:min)	1:54
GPU	1x NVIDIA Tesla T4

Table 5.2: Training Parameter

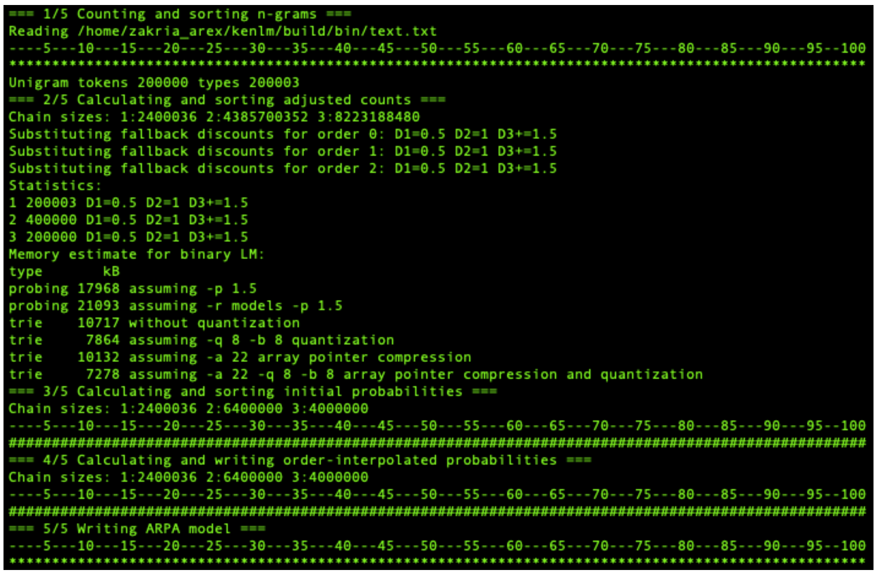
Epochs	Training Loss	Validation Loss	WER
30	0.021	0.220	9.40

Previously, models for classifying audio needed an extra language model (LM) and a dictionary to change a sequence of classified audio frames into understandable text. But Wav2Vec2 is built differently—it uses transformer layers, allowing each piece of processed audio to be related to all other pieces. Plus, Wav2Vec2.0 uses the CTC algorithm when fine-tuning, helping it handle differences in the length of input audio and output text. Because it can connect audio classifications and does not have issues with alignment, Wav2Vec2.0 does not need an outside language model or dictionary to create good audio transcriptions. Yet, as evident from Wav2vec2.0 self-supervised learning [11], pairing Wav2Vec2.0 with a language model can lead to notable enhancements.

- **Creating a Language Model:** We used KenLM [15] for creating an Italian based Language Model which will be later on used with the fine-tuned model of wav2vec2-large-xlsr-53-italian.

  Following are steps to create KenLM Language Model from scratch.

  - Download the repo and build the binaries: wget -o-https://kheafield.com/code/kenlm.tar.gz |tar xz
  - Make the directory for the environment: mkdir build and cd build
  - Build the environment: cmake .., make -j 4, sudo make install
  - Build the LM: After successfully creating the environment, run this: cd kenlm/build/bin and place the data.txt file which is dataset for training the LM.
  - Run this command: lmplz -o 3 –discount-fallback<"data.txt" > "3gram.arpa" The result is shown in Figure 5.2



```
=== 1/5 Counting and sorting n-grams ===
Reading /home/zakria_arex/kenlm/build/bin/text.txt
----5---10---15---20---25---30---35---40---45---50---55---60---65---70---75---80---85---90---95--100
****************************************************************************************************
Unigram tokens 200000 types 200003
=== 2/5 Calculating and sorting adjusted counts ===
Chain sizes: 1:2400036 2:4385700352 3:8223188480
Substituting fallback discounts for order 0: D1=0.5 D2=1 D3+=1.5
Substituting fallback discounts for order 1: D1=0.5 D2=1 D3+=1.5
Substituting fallback discounts for order 2: D1=0.5 D2=1 D3+=1.5
Statistics:
1 200003 D1=0.5 D2=1 D3+=1.5
2 400000 D1=0.5 D2=1 D3+=1.5
3 200000 D1=0.5 D2=1 D3+=1.5
Memory estimate for binary LM:
type       kB
probing 17968 assuming -p 1.5
probing 21093 assuming -r models -p 1.5
trie    10717 without quantization
trie     7864 assuming -q 8 -b 8 quantization
trie    10132 assuming -a 22 array pointer compression
trie     7278 assuming -a 22 -q 8 -b 8 array pointer compression and quantization
=== 3/5 Calculating and sorting initial probabilities ===
Chain sizes: 1:2400036 2:6400000 3:4000000
----5---10---15---20---25---30---35---40---45---50---55---60---65---70---75---80---85---90---95--100
####################################################################################################
=== 4/5 Calculating and writing order-interpolated probabilities ===
Chain sizes: 1:2400036 2:6400000 3:4000000
----5---10---15---20---25---30---35---40---45---50---55---60---65---70---75---80---85---90---95--100
####################################################################################################
=== 5/5 Writing ARPA model ===
----5---10---15---20---25---30---35---40---45---50---55---60---65---70---75---80---85---90---95--100
****************************************************************************************************
```

Figure 5.2: KenLM - LM Output

– The hierarchy of the folders is shown in Figure 5.3. In the bin folder, you can see the "3gram.arpa", this is the LM that we will use now with the fine-tuned wav2vec2-large-xlsr-53-italian.

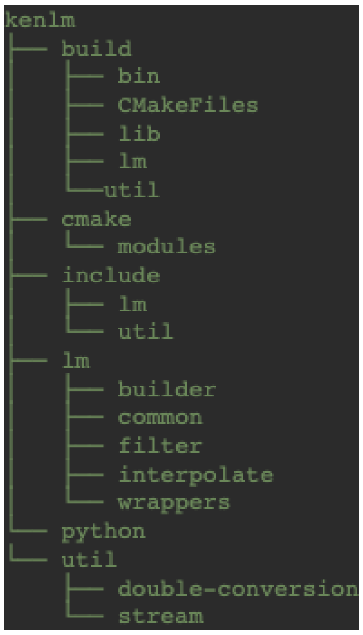
- **Merging with Wav2vec2.0 Base Model:** After merging the LM with the base model by using this code:

```
from transformers import Wav2Vec2ProcessorWithLM
processor_with_lm = Wav2Vec2ProcessorWithLM(
feature_extractor=processor.feature_extractor,
tokenizer=processor.tokenizer,
decoder=decoder)
```

And re-training the model again with specifications mentioned in Table 5.1, we were able to get the improved WER up-to 7.26 which is slightly better than the base wav2vec2-large-xlsr-53-italian model. The WER comparison is shown in Table 5.3

Table 5.3: WER Comparison

Without LM	With LM
9.40	7.26



```
kenlm
├── build
│   ├── bin
│   ├── CMakeFiles
│   ├── lib
│   ├── lm
│   └──util
├── cmake
│   └── modules
├── include
│   ├── lm
│   └── util
├── lm
│   ├── builder
│   ├── common
│   ├── filter
│   ├── interpolate
│   └── wrappers
├── python
└── util
    ├── double-conversion
    └── stream
```

Figure 5.3: KenLM - Directory Hierarchy

The performance demonstrated by wav2vec 2.0, when combined with a Language Model (LM), sets a high benchmark in terms of accuracy, standing out among several STT models, including Kaldi and Vosk which will be further elaborated on in the subsequent sections. Notably, wav2vec 2.0 not only showcases superior overall performance but also boasts a lower WER in comparison to the aforementioned STT tools.

However, despite its impressive accuracy and performance, wav2vec 2.0 poses a significant challenge due to its substantial model size of 1.26GB. This large footprint makes it particularly challenging to incorporate into mobile environments, where storage and computational resources are often limited. As a consequence of this limitation, it becomes necessary to explore other viable alternatives for mobile integration, leading us to consider other models such as Kaldi and Vosk. These alternative models, while potentially not matching wav2vec 2.0 in accuracy, may offer a better balance between size, efficiency and performance, making them more suitable for deployment in resource-constrained environments like mobile devices.

5.2.3 Kaldi Training

This section provides a brief overview of the directory structure of Kaldi and it's training stages. The main directories at the top level include egs, src, tools, misc and windows, out of which we will primarily be utilizing egs. The egs directory, short for "examples", contains training recipes for a majority of the significant speech corpora such as voxforge, yesno, mini-librispeech and several others. For this particular solution, mini-librispeech recipe with English dataset from OpenSLR was selected. It includes the scripts which encompasses the training of the acoustic model, aligning language model, MFCC feature extraction, GMMs and Time Delay Neural Network (TDNN) - chain training and finally the decoding process.

The training process in kaldi is executed through several stages, as shown in the Figure 5.4

- **Audio Dataset with transcription:** Firstly, direct to "kaldi/egs/mini-librispeech/s5" which includes all the required scripts. Download the dataset of English language from organizations such as OpenSLR and CommonVoice, which offer a substantial amount of audio data along with their transcriptions. To secure a transcript of the speech data, acquiring the start and end times for each utterance or sentence level can be helpful for achieving more accurate alignment, although it is not mandatory.

  ```
  local/download_and_untar.sh $data $data_url $part
  ```

- **Prepare the Dataset:** For acoustic model training, Kaldi necessitates transcripts in various formats. The required information includes the start and end times of each utterance, the speaker ID associated with each utterance and a list of all words and phonemes found in the transcript.

  ```
  local/data_prep.sh $data/LibriSpeech/$part data/$(echo $part | sed
      s/-/_/g)
  ```

- **Train Acoustic Model:** Training acoustic model depends on multiple files which hold information pertaining to the details of the audio files, transcripts and speakers. To obtain all required files, execute the following command:

  ```
  local/prepare_dict.sh --stage 3 --nj 30 --cmd "$train_cmd" \
  data/local/lm data/local/lm data/local/dict_nosp
  ```

  As a result, it will generate the following files in directory "data/train/":

  - **spk2gender** containing the gender tags 'm' and 'f'. The file format is like:

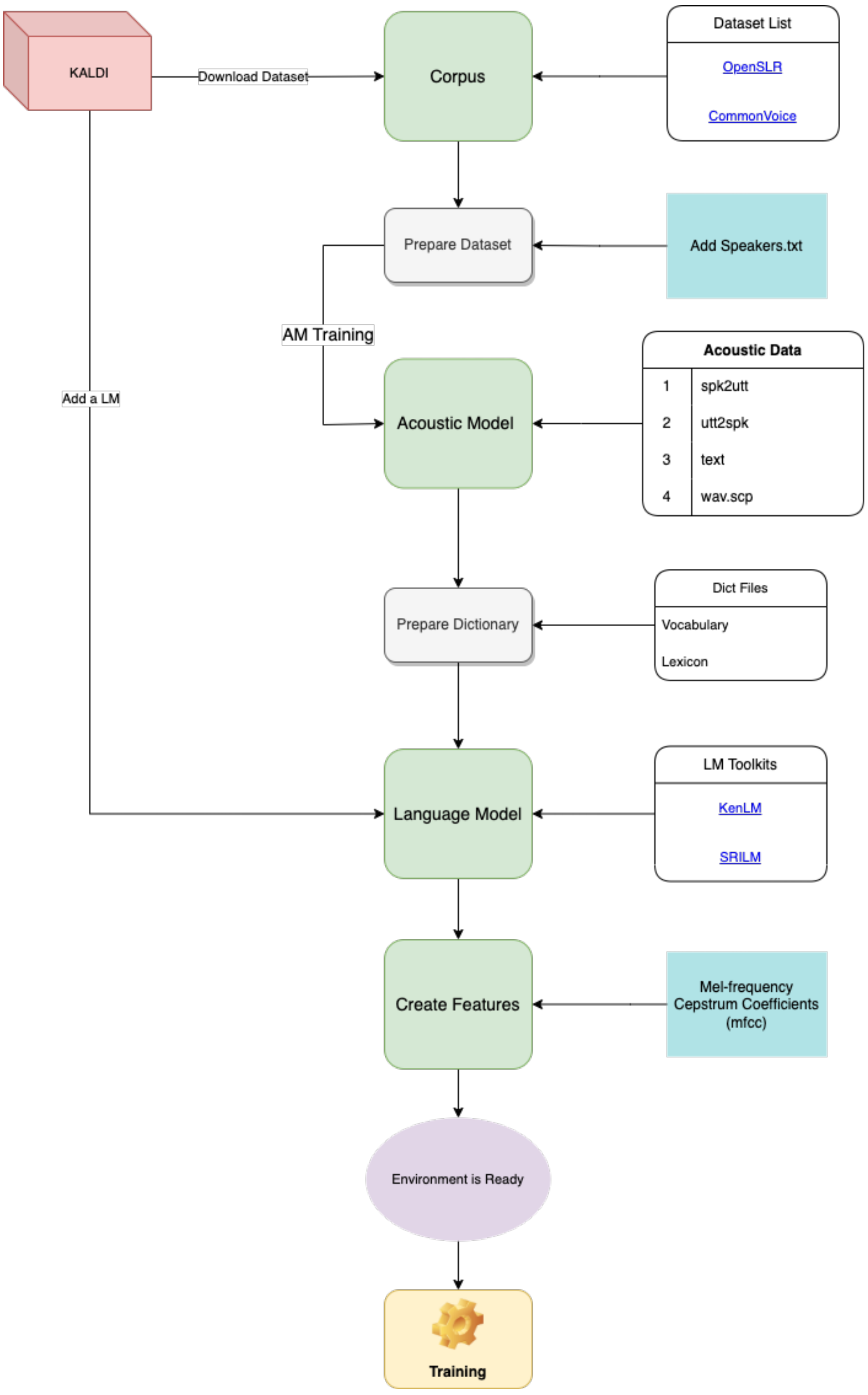


Figure 5.4: Flow Chart - Kaldi Training

```
1088-134315 f
```

- **utt2spk** file holds the information mapping each utterance to its respective speaker. The term ”speaker” doesn’t necessarily refer to an individual – it could represent an accent, gender or any factor that might affect the recording. The format is as follows:

  ```
  1088-134315-0000 1088-134315
  ```

- **spk2utt** is a file encompassing the mapping from speaker to utterance. While this information is already present in utt2spk, it is not in the required format. Executing the following line of code will not only automatically generate the spk2utt file but also ensure that all data files are available and correctly formatted:

  ```
  utils/fix_data_dir.sh data/train
  ```

  The file format is as follows:

  ```
  1088-134315 1088-134315-0000
  ```

- **text** file essentially serves as a transcript of the corpus, broken down by each utterance, and follows a specific format. After the creation of this text file, it is also necessary to refine the lexicon to include only the words found in the corpus. This step is crucial to guarantee that no extra phones are being trained. It looks like this:

  ```
  1088-134315-0000 AS YOU KNOW AND AS I HAVE GIVEN YOU A PROOF
  ```

- **wav.scp** gives the location of all audio files. If the audio files format are different than wav (such as sphere, mp3, flac, ogg), a conversion to wav format will be necessary. Rather than manually converting and storing multiple versions of the data, you have the option to allow Kaldi to perform the conversion dynamically. The tool ”sox” can be particularly useful in many such instances. The format of this file is as follows;

  ```
  1088-134315-0000 flac -c -d -s corpus LibriSpeech
     train-clean-5/1088/134315/1088-134315-0000.flac |
  ```

• **Prepare Dictionary/Language Data:** It refers to the directory including language data that is specific to your individual corpus. For instance, the lexicon exclusively includes words and their corresponding pronunciations found in the corpus. Run the following command to prepare the language data:

```
utils/prepare_lang.sh data/local/dict_nosp \
"<UNK>" data/local/lang_tmp_nosp data/lang_nosp
```

As a result, it will generate the following files in directory “data/lang/phones/”:

- **lexicon.txt** presents each word capitalized and listed on a separate line, followed by its phonemic pronunciation. The format is like:

  ```
  ACTS AE_B K_I S_E
  ```

- **nonsilence-phones.txt** contains a list of all the phones, excluding those representing silence.

  ```
  EE_B
  SS_I
  ```

- **silence-phones.txt** includes only SIL (silence) phone
- **extra-questions.txt** file poses "questions" concerning the contextual information of a phone by segregating the phones into two distinct sets. Subsequently, an algorithm evaluates whether modeling that specific context is beneficial. The standard extra-questions.txt typically encompasses the most frequent questions.

• **Train Language Model (LM):** The language model is a crucial component of the configuration, informing the decoder about the sequences of words that are recognizable. It contains probabilities for individual words and combinations of words. These probabilities are estimated from sample data, inherently offering some flexibility. Every combination from the vocabulary is plausible, though the probability assigned to each combination may differ.

Numerous methods exist for constructing statistical language models. A language model can be created in three distinct formats: text ARPA format, binary BIN format and binary DMP format. The ARPA format consumes more space, but it is editable. So, in our case, we will be using an online tool "SRILM" for creating LM in ARPA format. The steps are demonstrated as follows:

- **preparing text** is the first step to gather a clean data in a text file. For data-processing, different methods can be adopted
- **train LM** via SRI Language Model Toolkit by executing following commands:

  ```
  ngram-count -kndiscount -interpolate -text data.txt -lm your.lm
  ```

  Then prune the model by executing:

  ```
  ngram -lm your.lm -prune 1e-8 -write-lm your-pruned.lm
  ```

Now, we can incorporate this LM into our Kaldi environment.

- **Extract features using Mel Frequency Cepstral Coefficients (MFCC):** The command provided below is aimed to extract MFCC acoustic features and calculate the cepstral mean and variance normalization (CMVN) statistics. After every operation, it also rectifies the data files to confirm their format remains correct.

  ```
  steps/make_mfcc.sh --cmd "$train_cmd" --nj 10 data/$part
      exp/make_mfcc/$part $mfccdir
  steps/compute_cmvn_stats.sh data/$part exp/make_mfcc/$part $mfccdir
  ```

  The option –nj specifies the number of jobs to be dispatched, currently set at 10, indicating the data will be segmented into 10 portions. It's worth noting that Kaldi maintains data from identical speakers together, hence the number of splits should not exceed the total number of speakers available. The mfcc files are stored in directory "mfcc" with two formats which are ".ark" and ".scp" respectively.

- **Prepare the Training Environment:** Training can demand substantial computational resources; however, it can be managed with multiple processors/cores or multiple machines. By dividing the dataset into smaller portions and processing them concurrently, both training and alignment can be optimized. The specification for the number of jobs or divisions in the dataset will be detailed in subsequent training and alignment phases. Kaldi offers a wrapper to facilitate this parallelization, enabling each computational step to leverage multiple processors. Kaldi wrappers are as follows:

  - **run.pl** enables the execution of tasks on a local machine.
  - **queue.pl** enables the distribution of jobs across machines utilizing the Sun Grid Engine.
  - **cmd.sh** defines the parallelization that adds training and decoding both separately.

- **Initiate the Training of Model:** The complete model incorporates several training algorithms, including:

  - **Training Monophones:** The training procedure begins with the monophone models. For efficiency, only a subset of the data will be utilized during this phase. Even with limited data, satisfactory monophone models can be developed, serving primarily to initiate the training for subsequent models.

    ```
    steps/train_mono.sh --boost-silence 1.25 --nj 2 --cmd
        "$train_cmd" \
    data/train_clean_5 data/lang_nosp exp/mono
    ```

The files are saved in directory “exp/mono/”

– **Training Delta-Triphone:** Training the triphone model necessitates additional parameters, specifically for determining the number of leaves or HMM states on the decision tree and the number of Gaussians. In this command, we designate 2000 HMM states and 10000 Gaussians.

```
steps/train_deltas.sh --boost-silence 1.25 --cmd "$train_cmd" \
2000 10000 data/train_clean_5 data/lang_nosp
    exp/mono_ali_train_clean_5 exp/tri1
```

  Determining the exact number of leaves and Gaussians often relies on heuristics. These numbers primarily depend on the volume of data, the count of phonetic questions and the objectives of the model. Additionally, it’s necessary that the number of Gaussians always surpasses the number of leaves. The output files are saved in the directory “exp/tri3b/”

– **Chain Training:** This approach uses Time Delay Neural Networks (TDNNs), which are a type of neural network adopted at modeling sequential data like speech by employing layers with varying temporal contexts. During chain training, a lattice-free Maximum Mutual Information (LF-MMI) criterion is applied to optimize the model, leveraging TDNNs to model the acoustic characteristics of the input speech signals. TDNNs, with their ability to handle different time scales and temporal contexts, capture the intricate temporal relationships within the speech signals, making them especially suited for this task.

```
local/chain/tuning/run_tdnn_1j.sh
```

  Chain training essentially leverages the strengths of TDNNs in handling sequential data and combines it with a specialized objective function to improve the accuracy and efficiency of ASR systems in Kaldi. The following trained files are stored in the directory “exp/chain/” which will be later on used to formulate VOSK model.

  * tdnn
  * tdnn-sp-online
  * tree-sp
  * tri3b-train-sp

5.2.4 Fine-Tuning Kaldi - Our Solution

In this section, we will discuss the process of fine-tuning Kaldi and describe how this leads to enhanced results across various accents and improved WER.

The steps implemented during the training process are outlined and explained below:

- **Download the Dataset:** Use mini-librispeech-en dataset by directing to "kaldi/egs/mini-librispeech/s5" and download the required dataset by executing "./run.sh" which is the compilation of all training algorithms.

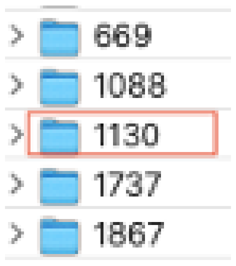
- **Prepare the Dataset:** To train new voices and words, we need to provide audio and add the transcribed text of those audios in the .txt file. The specification of each audio file should be:

  - 16KHz
  - flac/wav format
  - channel: mono
  - ideal time of recording of each audio file: 5-15 seconds

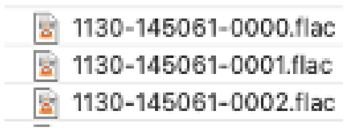
  Now, direct to train folder of dataset and add new folders for each new speaker. Be precise while adding a new speaker ID; it must be in sorted order and follow a specific precedence. Duplicate speaker IDs should be avoided in both the training and testing data. For verification, refer to the "SPEAKERS.TXT" file, which lists all the unique speaker IDs along with their specifications. To introduce a new user, simply append the new speaker ID to the "SPEAKERS.TXT" file. For example, in this use case, we added a speaker ID "1130" For reference, see the Figure 5.5a

  Open the folder "1130", there is another subfolder "145061". For picking the subfolder ID, again we should follow the precedence and order in the way we did for folder "1130."

- **Add Transcribed Text:** After the speaker ID folders and subfolders are created, then we need to add our new data with the transcribed text. For example: in our case of speaker ID "1130": go to folder -> 1130 -> 145061, place your-audio-file here with the specific format of "speakerID - subfolderID - audioid.flac." For reference, see the Figure 5.5b



(a) Speaker ID



(b) Precedence for Audio Files

Figure 5.5: Arrangements of Files

Then add the transcribed text in "1130 - 145061.trans.txt" file with the format of "speakerid - subfolderid - audioid TRANSCRIBED TEXT", demonstrated as follows:

```
1130-145061-0000 CHAPTER TWENTY FOUR
1130-145061-0001 IF HALSEY HAD ONLY TAKEN ME FULLY INTO HIS
    CONFIDENCE
```

- **Update SPEAKERS.TXT:** After these all files and folder are ready, go to corpus -> LibriSpeech: SPEAKERS.TXT and add a row of our speaker ID (details of that specific speaker) that we just created.

- **Update the Dictionary/Vocab:** Add new words with their phonemes in the files that are "librispeech-vocab.txt" and "librispeech-lexicon.txt" respectively.

- **Training Parameters:** The dataset and associated files are now prepared, however, there are some training configurations that require adjustments, as listed below:

  – In the directory "s5", open script "cmd.sh" and replace all the lines with following:

    ```
    export train_cmd="run.pl"
    export decode_cmd="run.pl"
    export mkgraph_cmd="run.pl"
    export cuda_cmd="run.pl"
    ```

  – In the directory "s5", go to this folder: "steps/nnet3/chain/" and open script "get-egs.sh." On line 60, change max-shuffle-jobs-run=50 to max-shuffle-jobs-run=10 and save it.

  - Direct to this folder: “steps/nnet3/chain/” and open the script file “train.py” and replace the line 243 with: args.use-gpu = (”wait” if args.use-gpu == ”true” else ”no”)

  - Direct to this folder: “steps/chain/” and open the script file “train.py” and replace the line 243 with: args.use-gpu = (”wait” if args.use-gpu == ”true” else ”no”)

  - Go to this folder: “steps/online/nnet2/” and open script “train-ivector-extractor.sh” on line 43, change ivector-dim=100 to ivector-dim=30 and save it.

  - Also, direct to this folder: “steps/nnet/ivector/” and open script “train-ivector-extractor.sh” On line 38, change ivector-dim=100 to ivector-dim=30 and save it.

  - Direct to this folder: “local/chain/tuning/” and open script “run-tdnn-1j.sh” In line 161, change input dim=100 to input dim=30 and save it.

  - In “s5” folder, open “./run.sh” and in line 144, change it to: local/chain/tuning/run-tdnn-1j.sh.

  - Last but not the least, set the GPU to exclusive mode by running these commands:

    ```
    sudo su
    nvidia-smi -c 3
    ```

  Now the environment is ready to train as all the parameters are configured. Run the script “./run.sh” in the directory “s5.”

- **Train the Model:** In order to initiate the entire model training, refer to **Kaldi Training**, section 5.2.3 and follow from point **Train Acoustic Model** to **Chain Training** in sequence.

  Following the above steps, we obtained a newly trained model, including the incorporation of new words and their respective accents.

Training a model with Kaldi can be quite demanding in terms of resources; it often requires powerful GPUs and the training time can extend considerably, especially with larger datasets. This challenge can be addressed using Vosk, as we will explore in the following section.

Additionally, it's notable that the model size with Kaldi is significantly smaller compared to wav2vec 2.0, making it a more manageable option for mobile applications.

5.3 Integration with Vosk

In this section, we will outline the process of transferring the trained model files from Kaldi training to a specific location, enabling the creation of a Vosk model. This model can subsequently be utilized in a mobile environment. Plus, fine-tuning of Vosk model will be briefly explained.

There are some specific commands that are executed to retrieve the respective files from Chain Training folder. This folder contains following files:

- tdnn
- tdnn-sp-online
- tree-sp
- tri3b-train-sp

After executing the commands, a folder "model" is created in the home directory, where all the required files are available. Furthermore, it is necessary to create specific folders that are compatible with the Vosk environment. So, there are 4 folders in total when combined, creates a Vosk model as shown in Figure 5.6

5.3.1 Compress the Model

One file, "HCLG.fst," is quite large in size; executing the provided scripts will compress this file and divide it into two separate files, while maintaining consistent accuracy and performance. So, for this we need to direct to mini-librispeech recipe where the model is trained, go to -> s5 and run "local/lookahead/run-lookahead.sh" This script will create 2 files "Gr.fst" and "HCLr.fst" which are the required files to replace "HCLG.fst" in the "vosk-model/graph" and

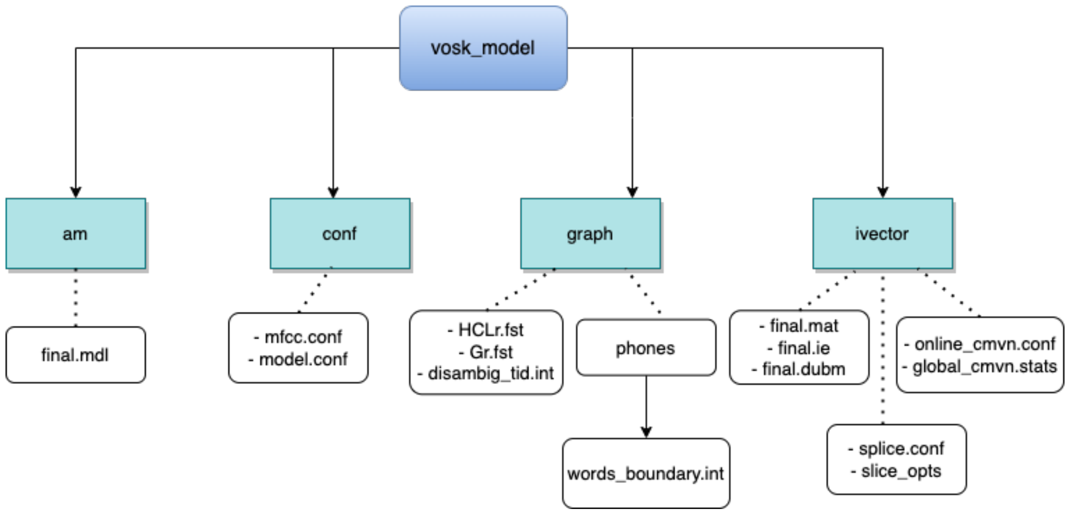


Figure 5.6: Vosk - Model Hierarchy

eventually the overall size has been compressed from 250MB to 60MB.

5.3.2 Fine-Tuning Vosk - Our Solution

Once the model has been prepared through Kaldi training and all files have been incorporated into the Vosk model, there is no need to retrain Kaldi from the beginning if we need to add new words along with their accents. Instead, we can fine-tune the Vosk model using the following approach:

- **Create a Vosk Environment:** Set up a Vosk environment and compile a directory that includes the following folders from Kaldi:
  - dict
  - lang
  - lang-local

- **Build the Environment:** Build a process by executing a script which compiles all folders mentioned above along with the Language Model (LM). We used “phonetisaurus” library from python and “bash scripting” for this process.

- **Build a Docker Image:** All the software needed for compilation (Kaldi + scripts) can be used by using a custom image provided by the Dockerfile. To build the image:

  ```
  docker build -t fine-tuning:1.0 .
  ```

- **Words and Pronunciations Files:** Now, we included two text files. In one file, we will incorporate the new words that need training and in the other, we will add the corresponding phonemes, which are provided by the Kaldi grapheme-to-phoneme library.

  – words.txt:

    ```
    lunesta
    medica
    ```

  – pronunciation.txt:

    ```
    lunesta L UW N EH S T AH
    medica M EH D IH K AA
    ```

- **Add the Vosk Model:** Incorporate the previously created Vosk model into the environment.

- **Training Script:** A bash script has been developed that initiates the training process by running the docker container and setting up the environment. Consequently, it trains the graph "Gr.fst" and substitutes the updated Gr.fst and corresponding files into the graph folder of the Vosk model.

  Thus, if there is a need to train new words, we simply add the new words along with their phonemes in the "words.txt" and "pronunciation.txt," respectively. Following this, executing the bash script will commence the training and returns an updated Vosk model, which is then ready for use in Screevo's mobile app.

Using Vosk for training takes considerably less time compared to Kaldi, making it a more efficient option. This advantage becomes particularly evident when there's a need to introduce new words along with their pronunciations to the system. In such instances, Vosk proves to be highly effective, as it only requires a few minutes to incorporate the new elements and update the model accordingly. This quick adaptability and time efficiency make Vosk an appealing choice for those looking to regularly update and expand their model's vocabulary.

However, Vosk presents a limitation when it comes to training new voices along with their distinct accents. In such scenarios, it does not offer a solution for incorporating these new elements into the existing model. The only workaround for this limitation is to initiate training with Kaldi which despite its resource-intensive nature, provides the flexibility needed to adapt the model to unique voice and accent characteristics.

6
Results

In this chapter, we will look at the outcomes we got from the activities carried out earlier and discuss about the improvements that we achieved.

Figure 6.1 illustrates the progression of the fine-tuning process for the Wav2vec2.0 model, specifically after the introduction of new data. This fine-tuning was conducted over 30 epochs, incorporating a range of training parameters. These parameters include a learning rate set at 1×10^{-4}, a CTC loss reduction method defined as "mean" and a batch size of 8. This model, sourced from MetaAI, is identified as "facebook/wav2vec2-base".

The entire fine-tuning procedure spanned a duration of 1 hour and 54 minutes, culminating in a training loss of 0.021 and a validation loss of 0.220. Notably, by the end of this training, WER of 9.40 was recorded. This performance surpasses nearly all other STT models discussed in the thesis, solidifying its position as a state-of-the-art STT model.

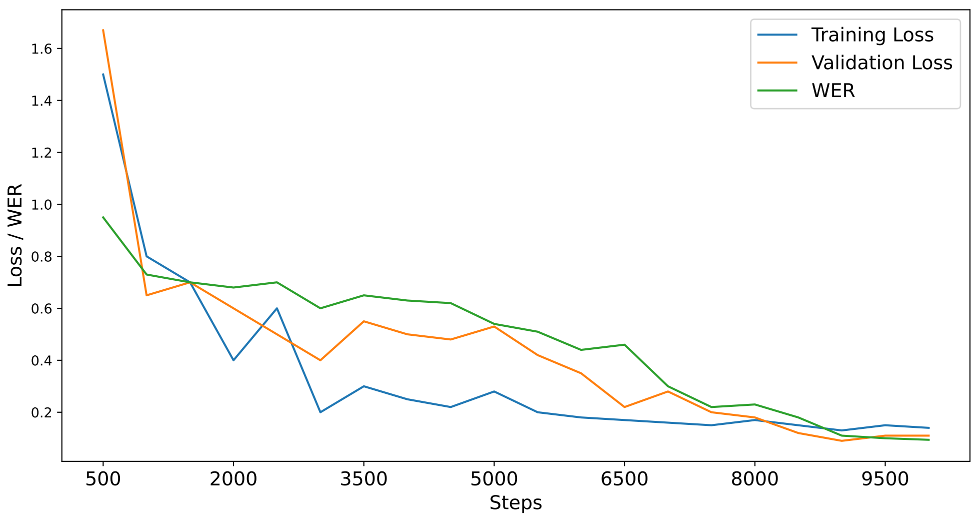


Figure 6.1: Trend of Training-Loss, Validation-Loss and WER

Furthermore, an evaluative comparison was undertaken to assess the relative performance of a fine-tuned Wav2vec2.0 model against a specialized Medical model as shown in Figure 6.2. This experimental setup involved the incorporation of a LM layered atop both of these models. To facilitate this comparison, an audio recording containing medical information was used as the test input, accompanied by its accurate transcribed text.

Upon feeding this audio sample and its corresponding transcription to both models, the fine-tuned Wav2Vec2.0 model consistently showcased superior performance, evident from its reduced WER values. This superiority held true in both scenarios: when the models operated autonomously and when they were enhanced with the LM. The findings underscore the robustness and adaptability of the fine-tuned Wav2Vec2.0, even in comparison to a domain-specific model like the Medical model.

In the provided table 6.1, we have detailed the parameters and configurations utilized through-

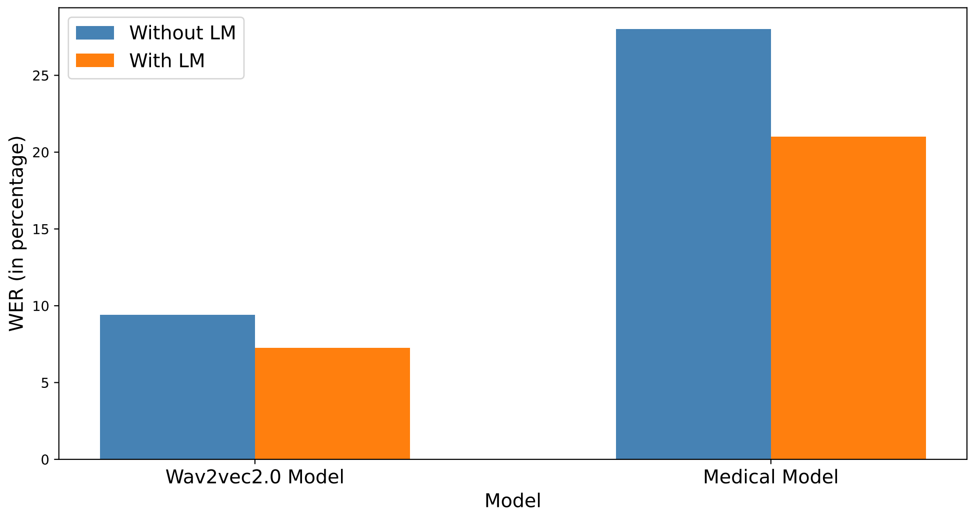


Figure 6.2: Comparison of WER between Wav2Vec2.0 Models

out the Kaldi training procedure. The table indicates two distinct versions for our experiment. The first, referred to as Version-1, represents a subset of the primary corpus extracted from the Mini-LibriSpeech-en dataset. Conversely, Version-2 stands as another subset derived from the same dataset. Their respective sizes are 458MB for Version-1 and 678MB for Version-2.

The details of the training method were explained earlier in the section 5.2.3. After this training phase, which spanned a duration of approximately 1 hour and 15 minutes for Version-1 and slightly less at 1 hour and 6 minutes for Version-2, we were able to obtain the final results. The WER achieved were 12.57% for Version-1 and a marginally better 10.06% for Version-2, highlighting the subtle performance variations between the two models. The computational resources used for this training were sourced from Amazon Web Services (AWS). Specifically, the setup included a machine equipped with 1 GPU and 4 CPUs, ensuring adequate processing power for the training tasks.

Table 6.1: Specifications - Kaldi Training

Properties	Version-1	Version-2
Dataset: Mini-LibriSpeech	train-clean-5 (train) dev-clean-2 (test)	dev-clean (train) test-clean (test)
Dataset Size	458 MB	678 MB
Language	EN	EN
Training Time (hr:min)	01:15	01:06
WER %	12.57	10.06
Machine Specification	AWS - 4 CPUs (16GB), 1 GPU - Tesla T4 (15GB)	AWS - 4 CPUs (16GB), 1 GPU - Tesla T4 (15GB)

So far, we have looked at how well these STT models perform. Now, we will quickly compare them side by side. The table 6.2 shows that Wav2Vec2.0 performs the best, having the lowest error rate at 7.26%. Kaldi and Vosk follow closely with error rates of 10.06% and 9.85% respectively. However, when we look at the training speed, Vosk beats the others. It trains in just 5 to 10 minutes, while Wav2Vec2.0 and Kaldi need hours.

Every STT system brings its own set of advantages and drawbacks, as highlighted in previous chapter. For instance, Wav2Vec2.0, despite boasting the best accuracy with the lowest WER, has a significant limitation due to its large size of 1.26GB. This size renders it unsuitable for the majority of smartphones, given their storage constraints.

In contrast, Kaldi and Vosk are much more smartphone-friendly, weighing in at approximately 250MB and 70.90MB respectively. These sizes are much more manageable for mobile platforms, making both systems viable options for smartphone applications. After evaluating the performances and results, we chose Vosk because it integrates the Kaldi model within its framework and the training duration is notably shorter than that of Kaldi. The following figure 6.3 visually displays the results, specifically focusing on training time and WER of these STT models.

Table 6.2: Comparison - STT Models

Models	**Wav2Vec2.0**	**Kaldi**	**Vosk**
Model Size	1.26 GB	250 MB	70.90 MB
WER %	7.26	10.06	9.85
Training Time (hr:min)	01:54	01:06	5-10 min
Smartphone-Adaptive	No	Yes	Yes

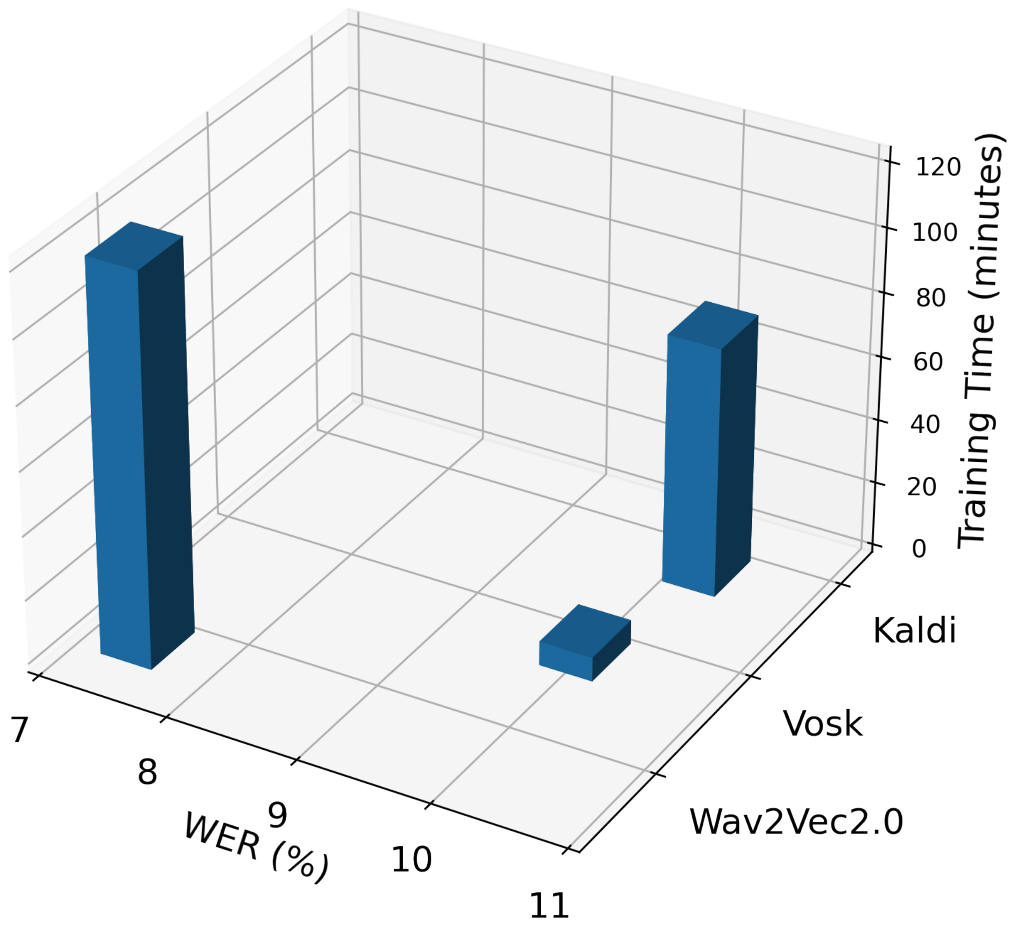


Figure 6.3: Comparing STT Models Based on Word Error Rate (WER) and Training Time

7 Conclusion and Future Work

In this dissertation, after a thorough review of pioneer STT models, we further examined Kaldi and Vosk models that were fine-tuned with diverse voice accents and pronunciations. Alongside, we incorporated text data with corresponding phonemes to improve training process. By training these models on new data, we enhanced the STT system's adaptability to a broader range of voice inputs. Interestingly, when we fine-tuned the Wav2Vec2.0 using a voice dataset, it outperformed both Kaldi and Vosk. It achieved the lowest WER, but its larger model size made it incompatible for the smartphones ecosystem.

The pace of progress in this domain is astonishing, with several groundbreaking products launched just this year alone. Among them, OpenAI's STT model "Whisper", in particular represents a significant paradigm shift in the Automatic Speech Recognition (ASR) and STT fields. However, as promising as Whisper appears, it's not without its challenges. There remains a considerable amount of research to be undertaken, especially in the realm of fine-tuning. Strategies such as prompt tuning are just the tip of the iceberg. Ensuring compatibility of these models with smartphones presents another hurdle. Additionally, enhancing the efficiency of Whisper, especially in terms of inference time and latency, remains a vital area of focus.

Numerous research institutions and emerging startups are directing their efforts towards these challenges which also leads to the new advancements in Natural Language Processing (NLP) domain. Now, with open-source Large Language Models like Llama2, Falcon and others available, we are entering a new phase of research in this field.

References

[1] Y. M. Cheng, C. Ma, and L. Melnar, "Voice-to-phoneme conversion algorithms for speaker-independent voice-tag applications in embedded platforms," in *IEEE Workshop on Automatic Speech Recognition and Understanding, 2005.* IEEE, 2005, pp. 403–408.

[2] J. Li *et al.*, "Recent advances in end-to-end automatic speech recognition," *APSIPA Transactions on Signal and Information Processing*, vol. 11, no. 1, 2022.

[3] A. Graves, "Sequence transduction with recurrent neural networks," *arXiv preprint arXiv:1211.3711*, 2012.

[4] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang *et al.*, "Streaming end-to-end speech recognition for mobile devices," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2019, pp. 6381–6385.

[5] H. Sak, M. Shannon, K. Rao, and F. Beaufays, "Recurrent neural aligner: An encoder-decoder neural network model for sequence to sequence mapping." in *Interspeech*, vol. 8, 2017, pp. 1298–1302.

[6] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 369–376.

[7] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, "A comparison of sequence-to-sequence models for speech recognition." in *Interspeech*, 2017, pp. 939–943.

[8] 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.

[9] D. Klakow and J. Peters, "Testing the correlation of word error rate and perplexity," *Speech Communication*, vol. 38, no. 1-2, pp. 19–28, 2002.

[10] S. J. Young, J. J. Odell, and P. C. Woodland, "Tree-based state tying for high accuracy modelling," in *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.

[11] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 449–12 460, 2020.

[12] A. Baevski, S. Schneider, and M. Auli, "vq-wav2vec: Self-supervised learning of discrete speech representations," *arXiv preprint arXiv:1910.05453*, 2019.

[13] A. C. Alexei Baevski and M. Auli, "Wav2vec 2.0: Learning the structure of speech from raw audio," 2020.

[14] "Data collator code from huggingface transformers," 2021.

[15] "Kenlm language model toolkit," 2011.

Acknowledgments

We have concluded this dissertation and I could not have done it without some great people by my side.

Thanks a lot to my supervisor, Prof. Satta Giorgio. You have guided and supported me throughout. I could not have done it without you.

My mentor and co-supervisor, Domenico Crescenzo, Thanks for expanding my views and strengthening my work. You constantly pushed me to think more critically and your support throughout the time was priceless.

Also, I would like to thank the University of Padova. Its atmosphere, resources and opportunities to learn from top experts have been essential for my studies.

Last but not the least, my family and friends that have been my pillars of support throughout this journey. During the challenging times, when stress took its toll, you all displayed immense patience and understanding. Your unwavering presence and encouragement kept me going, making the tough times easier.