# CLOUD-BASED

# TRANSCRIPTION OF

# TAMIL SPEECH

### A PROJECT REPORT

***Submitted by***

|  |  |
| --- | --- |
| **Sandeep S** | **(185002086)** |
| **Srividhya N** | **(185002103)** |

***in partial fulfillment for the award of the degree of***

**BACHELOR OF TECHNOLOGY**

**in**

**INFORMATION TECHNOLOGY ENGINEERING**



**Department of**

**INFORMATION TECHNOLOGY**

**Sri Sivasubramaniya Nadar College of Engineering**

###### (An Autonomous Institution, Affiliated to Anna University)

**Rajiv Gandhi Salai (OMR), Kalavakkam – 603 110**

**MAY 2022**

**Abstract:**

With the growing amount of online media in Indic languages, a growing language barrier separates the content creator from an otherwise global audience. This problem is also visible during real-time communication between people who speak different regional languages. Although pre-trained speech-to-text models are available for English and other foreign languages, they are seldom found for Indic languages.

We present a full-fledged application that transcribes Tamil speech to Tamil text with low latency. The Tamil Automatic Speech Recognition (ASR) model used for the same is trained on ~200 hours of CommonVoice Tamil speech corpus. Multiple deep learning architectures are investigated to build the Tamil ASR model. The developed ASR model will be deployed as an online cloud application with an intuitive user interface with the following services: transcription of Tamil speech, translation of Tamil text into other languages, transcribed/translated text to synthesized speech. Since the service does not perform any computation-intensive tasks on the client machine, it does not need expensive or powerful system requirements for the same. We also present a cost-effective approach to building a large vocabulary corpus using semi-supervised learning for low-resource languages using our trained ASR model. Our proposed application will enable the conversion of massive web resources in Tamil speech into other languages and can also be used by content creators who wish to reach a global audience through the Tamil language.

**Introduction:**

Automated speech transcription has been explored by many over the last few decades. There are many models available for high-resource languages such as English, but there is a lack of availability of robust ASR models for low-resource languages such as Tamil. This is primarily because of the unavailability of a sufficient amount of open-source validated datasets required for training these models. We preprocessed and aggregated around 200 hours of validated speech training data from different reliable online resources. The model was iteratively improved by selecting the best combination of hyperparameters from each version to train the next. We also used various dataset augmentation options like pitch, tempo and amplitude augmentation to improve the performance of the model on different types of speech distributions. We propose to deploy the final version of the ASR model onto the cloud for providing instant access to our transcription service irrespective of the client device platform.

Digital media content in Tamil presents a natural language barrier between content creators and the global audience. While this issue could be resolved with subtitles, it requires extra effort from the content creator to provide those subtitles in other languages. This is further accentuated when the people involved are monolingual, i.e. they not able to communicate in more than one language. In courts of law, it is essential to capture every word spoken in Courtroom and Interview Room scenarios. The ability to transcribe audio into text reduces the manual burden and instantly provides a written record. Traditionally, transcription and translation have been done manually by human experts which can be considerably laborious and expensive. There is a lack of availability of open-source speech-to-text models and validated datasets required to train these models. By automating the process of Tamil speech transcription, we minimize the cost and time taken to generate the transcript. Further, the generated text could be translated to other languages or fed into text-to-speech engines to obtain voice output.

Building a novel Tamil ASR model poses several challenges: (i) There is a lack of availability of validated speech datasets for Tamil (ii) Tamil language consists of multiple dialects (iii) The open-source speech corpus consists of multiple audio formats.

We used Mozilla DeepSpeech to build our ASR model, trained on an aggregated open-source Tamil speech corpus. By leveraging DeepSpeech’s RNN-based architecture, our model is able to achieve end-to-end speech recognition where the input is a sequence of spectrograms and the output is text transcriptions. We also built a 5-gram language model to improve transcription performance from around 25GB of Tamil text data scraped from Tamil news articles, Wikipedia articles, Kaggle datasets and other online resources. We have followed an iterative method for training and evaluation of the model to achieve the minimum word error rate (WER) possible. We propose to host and deploy our model as a cloud-based transcription service on the Internet to enable fast Tamil speech transcription irrespective of the client platform.

**Literature Survey:**

We have done the literature survey based on the following parameters:

1. DeepSpeech Architecture
2. Datasets and the overall duration
3. Preprocessing and feature extraction
4. Training model
5. Test analysis
6. Deployment of the trained model
7. Language Model
8. Comparison with our methodology(to be included in all methods)

Baidu has proposed DeepSpeech[1], a speech recognition system developed using end-to-end deep learning that leverages a Recurrent Neural Network Architecture. Spectrogram features are extracted from the input audio which is mapped to phonemes. The RNN model consists of 5 hidden layers which convert an input sequence into a sequence of character probabilities. The above architecture has been implemented as a deep-learning model that has been trained on 5000 hours of read English speech from 9600 speakers. Deepspeech system has been integrated with N-gram language model which has been trained from huge unlabeled text corpora and is used to predict the words likely to follow each other. DeepSpeech has achieved a WER of 6.56% on clean utterances and 19.06% on noisy utterances. We propose to build our Tamil ASR model based on DeepSpeech architecture using open source Tamil datasets.

Dataset collection and preprocessing poses several challenges for low resource languages such as Tamil. The ASR model for Manipuri language developed by Tavina Patel et al.[2] has made use of a data collection portal to scrape together ~100 hours of data from telephonic conversations of native Manipuri language speakers. at 8000Hz) and recorded from 300+ native Manipuri speakers. The speech is read in nature and each speaker receives unseen 100-150 utterances corresponding to around 30 minutes. Next, the speech is transcribed to check for any mismatch between the text and audio. Speech preprocessing tasks like removal of poor recordings, tagging of non-speech parts and other fillers. It is then transcribed by trained Manipuri linguists post which it is annotated and the lexicon is obtained.

The DeepSpeech based German ASR model developed by Aashish et al. [3] has used the open source Voxforge, Tuda-de, Mozilla CommonVoice corpora. Voxforge corpus consists of 35 hours of German speech clips ranging from 5-7 seconds. Around 180 speakers have read aloud sentences from German Wikipedia, protocols from the European Parliament, and some individual commands. Tuda-de consists of 127 hours having recordings curated under more controlled conditions. The Mozilla German corpus contains 140 hours of clips with lengths varying from 3 to 5 seconds. However, the corpus is recorded outside controlled conditions as per the comfort of the speaker. The utterances have background noise, and users have varied accents. Speakers in this dataset are relatively young, and the male/female ratio is about 5:1, which might result in a severe bias when trying to transfer the model.

Aashish et al. have implemented dataset preprocessing as per DeepSpeech requirements. Since DeepSpeech expects audio and transcription data to be prepared in a specific format so that they can be read directly by the input pipeline (as per the figure below): 

The transcriptions were cleaned by removing commas as well as punctuation and converting all transcriptions to lower case. All audio clips were ensured to be in .wav format. The pruned results were split into training (70%), validation (15%), and test data (15%).

In the design and development of a large-vocabulary continuous speech recognition system by A. Madhavaraj et al.[4], the dataset contains 6.5 hours of transcribed speech recordings from the Central Institute of Indian Languages, Mysore. The recordings are single-channel, close-talk, PCM data sampled

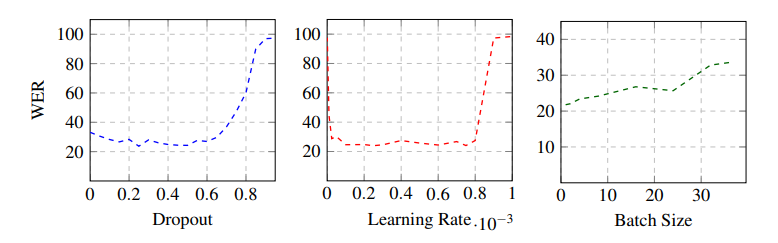
at 16 kHz with a resolution of 16 bits per sample. This corpus is a newspaper read-speech covering a vocabulary of 13,026 words recorded from 30 speakers (18 male and 12 female). The entire corpus is divided into three chunks: 4.5 hrs (training),

1 hour (development) and 1 hour (test). The training set is used to learn the AM parameters and the development set is used for the purpose of validation and the test set is used for reporting the recognition performance of the ASR system.

The feature extraction and training process is fairly common. Tanvina et al. use a 4-step process to implement the speech recognition: Language Identification (LID), Speech to Text, Keyword Search (KWS), Speaker Diarization (SD). The LID module was built on top of the KALDI toolkit and classifies speech utterances into 4 classes: English, Manipuri, Assamese and Unknown class. For training the LID module, 20 hours of speech data is used from each language, i.e., 100 hours

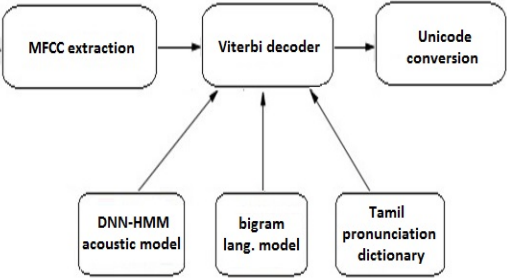
comprising of ∼52000 utterances in total. Both Gaussian Mixture Model-Hidden Markov Model (GMM-HMM) and Deep Neural Network-HMM (DNN-HMM) Acoustic Models (AMs) systems were built. The KALDI toolkit with the LibriSpeech recipe that uses speaker adaptive training is used. The CMU Language Model (LM) toolkit is used to build a 2-gram LM. The LM was built on ∼30k sentences, with an average of 10-15 words per sentence. Once the ASR decodes the speech, the KWS module, indexes the lattices and given a keyword/phrase, searches through the indexed lattices to get the occurrences of the desired KWs. The diarization module partitions an input audio according to the change in speakers. The LIUM diarization toolkit is used where for a given test speech, MFCC features are extracted and speaker segmentation is performed by first detecting instantaneous change points using Generalized Likelihood Ratio (GLR) distance.

Aashish et al. use an iteration-based method to train the DeepSpeech model with a combination of different hyperparameters in each iteration. Graphs are plotted between the various hyperparameters and the WER, and the optimum combination is arrived at (refer the figure below). The early stopping feature of DeepSpeech was used which stops the training of a neural network early before it overfits the training data.



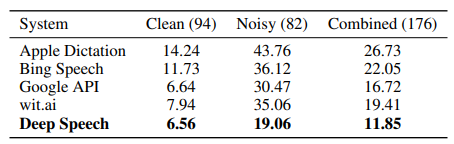
A probabilistic language model using KenLM toolkit to train a 3-gram model on a pre-processed corpus. It consists of eight million filtered sentences comprising 63.0% Wikipedia, 22.0% Europarl, and 14.6% crawled sentences. The corpus is normalized to a form that is close to how a reader would speak the sentence, especially changing numbers, abbreviations, and dates. Additionally, punctuations were discarded, as it is usually also not pronounced.

In the design and development of the end-to-end Tamil speech recognition system, A. Madhavaraj et al., have performed MFCC feature extraction of the input audio. The feature extraction block essentially extracts features, which serve as a good acoustic representation of the speech units (or phones), while suppressing other irrelevant variations in the signal due to the other factors such as the speaker, the channel, the speaking style of the speaker and the recording environment. The acoustic model was built using DNN-HMM architecture, as per the below diagram:

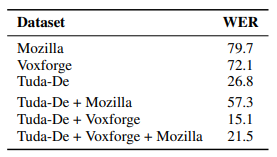


The language model used is a bigram statistical model that uses backoff and other estimation techniques to prune the LM for a desired language perplexity.

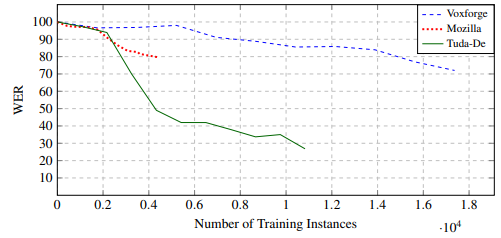
The test results of the publications are notable. The DeepSpeech model[1] yields a 6.56% WER for clean data, 19.06% WER for noisy data, 11.85% WER for clean and noisy data combined. The performance of the model is also compared to other popular speech recognition systems such as Google API, Apple Dictation and Bing Speech.

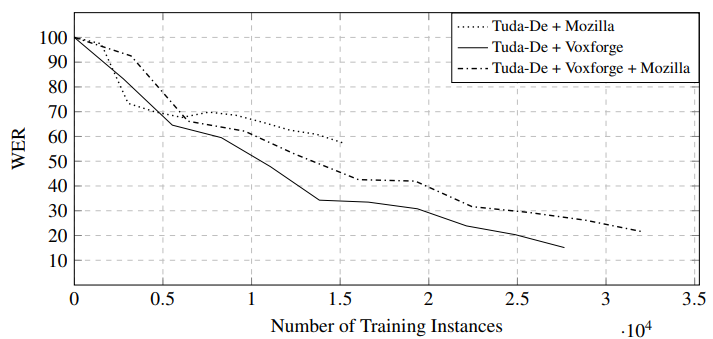


The German DeepSpeech model results are discussed below.



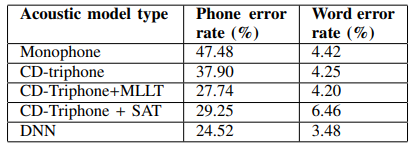
The model only trained on Tuda-De yields a comparable WER of 26.8%. Results for the other datasets are much lower, but apparently combining several datasets improves the results. While the combination of Tuda and Mozilla yields a WER of 57.3%, the combination of Tuda, Voxforge, and Mozilla gives a WER of 21.5%. Combining the very similar Tuda-De and Voxforge yields a WER of 15.1%, which is a remarkable improvement over using only a single dataset. Some tests performed comparing the various parameters like WER vs. no. of training instances for single and multiple datasets are depicted in the figures below:





The GMM-HMM system used in the Manipuri language ASR model gives 19.28% WER and the DNN-HMM system gives better performance of 13.57% WER.

The Tami speech recognition system gives the following results: the best performance is attained by DNN-based acoustic model in both the cases. DNN models show an absolute improvement of about 1% and 23% over monophone models for CSR and PR, respectively. The detailed results are tabulated as below:



**Datasets:**

**Dataset Collection:**

There are several datasets for the Tamil language available on the Internet. As our project would be completely open-source we have selected the following open-source corpora for building our ASR model:

1. Acoustic Model:
2. **Mozilla Common Voice Corpus v7.0:**

Common Voice is a multi-lingual crowdsourcing project started by Mozilla to create a free database for speech recognition software with diverse voice samples. The project is supported by volunteers who record sample sentences with a microphone and review recordings of other users. The transcribed sentences will be collected in a voice database available under the public domain license CC0. Each entry in the dataset consists of a unique MP3 and corresponding text file. Many of the 13,905 recorded hours in the dataset also include demographic metadata like age, sex, and accent that can help train the accuracy of speech recognition engines. The dataset currently consists of 11,192 validated hours in 76 languages. The audio clips are released as mono-channel, 16bit MPEG-3 files with a 48kHz sampling rate.

The Mozilla common voice v7.0 consists of 216 hours of Tamil speech out of which 196 hours of Tamil speech are validated. The following is the age distribution of the speakers who have contributed to the corpus:

|  |  |
| --- | --- |
| Age | % Distribution |
| 70-79 | 4% |
| 40-49 | 4% |
| 50-59 | 2% |
| 40-49 | 1% |
| 30-39 | 13% |
| 19-29 | 11% |
| < 19 | 5% |

1. **Open SLR:**

Open SLR (Open Speech and Language Resources) is a site devoted to hosting speech and multiple language resources, such as training corpora for speech recognition, and software related to speech recognition.

The Tamil language dataset contains transcribed high-quality audio of Tamil sentences recorded by volunteers. The dataset had been manually validated and packaged into a compressed and downloadable archive. The dataset consists of audio files in wav file format, and the transcription mappings for the same in a TSV file. Consisting of 7 hours of Tamil speech the dataset is 1.34 GB in size.

1. **MSR:**

Microsoft Speech Corpus (Indian languages) contains conversational and phrasal speech training and test data for Telugu, Tamil and Gujarati languages. It is aimed at helping researchers and academia build Indian language speech recognition for all applications where speech is used. This Indian language Speech Corpus content is provided by Microsoft Research Open Data initiative, a collection of free datasets from Microsoft Research to advance state-of-the-art research in areas such as natural language processing, computer vision, and domain specific sciences. The data package includes audio and corresponding transcripts. Licensed under C-UDA (Computational Use of Data Agreement), the MSR dataset contains around 1,24,600 files composed of TXT and WAV files amounting to almost 45 hours of validated speech data for the Tamil language.

1. **ULCA:**

Universal Language Contribution APIs (ULCA) is an open-sourced scalable data platform, supporting various types of datasets for Indic languages, along with a user interface for interacting with the datasets. ULCA aims to be the premier data and models repository for Indic language resources. ULCA is part of Vakyansh, an open-source project spanning all major Indic languages. Vakyansh aims to host the key essentials of Automatic Speech Recognition (ASR) technology, focusing on Indian languages. Consisting of more than 1200 hours of labelled Tamil audio with machine-generated transcripts, it is a resource that allows people to build applications that leverage speech recognition.

1. Language Model:
2. **AI4Bharat IndicCorp:**

IndicCorp is one of the largest publicly-available corpora for Indian languages. It has been developed by discovering and scraping thousands of web sources - primarily news, magazines and books, over a duration of several months. The corpus is a single large text file containing one sentence per line. The publicly released version is randomly shuffled, untokenized and deduplicated. Licensed under Creative Commons (CC) v4.0, the monolingual corpus covers 12 languages. The Tamil text corpus contains 4.41million news articles , 31.5million sentences and 582million tokens.

1. **Tamil- Language Corpus for NLP:**

Tamil language Corpus consists of articles from Wikipedia & Tamil daily news, Dataset split into train and test for ease of use in building machine learning models. Corpus size is 2GB and it is released under Creative Commons (CC) licence v4.0.

1. **UEDIN CC-100 Tamil Dataset:**

Created by Conneau & Wenzek et al. at 2020, the CC100-Tamil This dataset is one of the 100 corpora of monolingual data that was processed from the January-December 2018 Commoncrawl snapshots from the CC-Net repository. The size of this corpus is 1.3Gb in Tamil language. Containing n/a in Text file format.

1. **Kaggle Datasets:**

Kaggle is an abundant source of datasets widely used by many in the field of machine learning. We collected multiple text datasets based on articles from Tamil newspapers like Dinamalar, Tamil Murasu etc. It also includes Tamil Wikipedia articles.

**Implementation:**

The transcription system consists of two components: Acoustic Model(AM) and Language Model (LM). The acoustic model takes audio as input and converts it to a probability over characters in the alphabet. The language model helps to turn these probabilities into words of a coherent language. The language model (aka. the scorer), assigns probabilities to words and phrases based on statistics from training data. The language model knows that “I read a book” is much more probable than “I red a book”, even though they may sound identical to the acoustic model.

Our implementation to develop the model consists of the following steps:

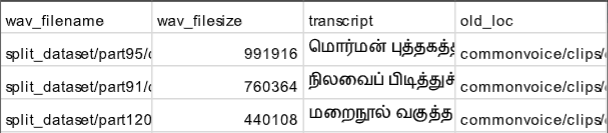
**Dataset Pre-processing:**

Since the datasets were collected from multiple sources we undertook pre-processing steps to standardize them:

1. Acoustic model:
   1. Downloading the dataset archives: The speech corpora archives were identified from the URLs mentioned on their websites and were downloaded to Google Drive through Google Colab.
   2. Standardization of speech datasets folder structure: We developed a folder structure to be used to store all the speech datasets efficiently, to allow for faster file access. This was needed as file access through Google Drive slows down dramatically when the number of files in a folder is large.
   3. Unzipping and extraction of the archives: The archives were decompressed into the folder structure developed earlier. Depending on the file format of the archive, various decompression utilities were used.
   4. Checking for invalid audio files and excluding them from the dataset: Performed preliminary checks on the audio files such as verification of file header, comparing the duration of the audio file with the length of the transcript, and checking for file corruption.
   5. Standardization of audio parameters: Converted all the audio files of the datasets to the following specification:
      * Sample rate: 48kHz
      * Bitrate: 16 bits
      * Channel: Mono
      * File format: WAV

DeepSpeech requires a 16-bit mono WAV file format, while the sample rate was fixed as 48kHz across all datasets to enable standardization

* 1. Parse the transcript files and generate DeepSpeech-compatible CSV: The transcript mapping file of the dataset was parsed and the data pertaining to DeepSpeech CSV format was extracted.



* 1. Aggregate all CSVs and split them into train, dev and test datasets:

Combined all the CSVs and split them into in the ratio of 75:15:10 for the training, validation and testing datasets.

1. Language model:
   1. Downloading the dataset archives: The text corpora were identified from the URLs mentioned on their websites and were downloaded to Google Drive through Google Colab.
   2. Standardization of text corpus folder structure: Since we used multiple text corpora each having a different structure, we logically separated them into different folders to later re-combine them into a single large corpus for further processing.
   3. Unzipping and extraction of the archives: The archives were decompressed into the folder structure developed earlier. Depending on the file format of the archive, various decompression utilities were used.
   4. Removal of symbols not present in Tamil vocabulary: Since most of the corpora contained raw scraped news articles, they had many foreign symbols such as special characters, punctuation marks, numerals etc. not in the vocabulary of our consideration. We scanned the text files for such occurrences and filtered them out, yielding pure Tamil sentences separated by a newline character.
   5. Tokenization of text corpus: Sentences in the corpora were split into collections of tokens and combined into one large corpus ready to be passed as input to the LM builder tool.

**ASR Model Training:**

Since our ASR model is composed of the AM and the LM, we trained them separately following the steps detailed below.

**AM Training:**

In line with the requirements of DeepSpeech library, we developed the following steps to define the model training lifecycle that was followed to develop the various iterations of models in Google Colab:

1. Setting up a GPU environment with necessary libraries: We defined a script which executed a set of commands for installing the following libraries to set up a GPU environment: Tensorflow-gpu 1.15.4, deepspeech-gpu 0.9.3, SoX, python-dev tools.

1. Cloning the DeepSpeech repository from GitHub: DeepSpeech v0.9.3 was cloned using git tool into local storage using the repository URL.
2. Installing the required dependencies of DeepSpeech: The train script installs required Python modules like absl-py, argparse, semver etc. These libraries form the basis for the DeepSpeech training module, which would handle the TensorFlow backend to implement the DeepSpeech model training CLI.
3. Finalizing model hyperparameters : Based on the performance of previous iterations, we tune the model hyperparameters with the expectation of achieving higher accuracy for the next iteration.

The following are the hyperparameters used for training Model A:

|  |  |
| --- | --- |
| **Hyperparameter** | **Value** |
| Learning rate | 0.0001 |
| Train batch size | 64 |
| Hidden units | 2048 |
| Dropout rate | 0.4 |
| Dev batch size | 128 |
| Epoch | 36 |
| No early stop | True |

The following are the hyperparameters used for training Model B:

|  |  |
| --- | --- |
| **Hyperparameter** | **Value** |
| Learning rate | 0.0001 |
| Train batch size | 64 |
| Hidden units | 2048 |
| Dropout rate | 0.4 |
| Dev batch size | 128 |
| Epoch | 36 |
| No early stop | True |
| Augment volume  Augment pitch | p=0.2,dbfs=-10:-40  p=0.2,pitch=1~0.2 |
| Augment Reverb | p=0.2,delay=50.0~30.0,decay=10.0:2.0~1.0 |
| Augment Tempo | p=0.2,factor=1~0.5 |

1. Running DeepSpeech training script with the hyperparameters: The DeepSpeech train script was started with the finalized hyperparameters passed as arguments. The current progress of model training was saved at regular intervals to Google Drive in the form of snapshots called checkpoints. Checkpoints allow interruption (also in the case of some unexpected failure) and later continuation of training without losing hours of training time. Since Google Colab resources are not guaranteed and not unlimited, the VMs can be pre-empted at any time. Using checkpoints helped us resume the model training from where it was last pre-empted.
2. Saving the final model to Google Drive: At the end of a logical phase in training, the final model was exported to Google Drive from the Colab VM. The model was then archived and classified to be tested later.

**Building Language Model:**

A language model is an n-gram model trained on a corpus of text. It predicts which words are more likely to follow each other. For example, the word *chicken* might be frequently followed by the words *nuggets*, *soup* or *rissoles*, but is lesser likely to be followed by the word *purple*. A scorer is a language model and is used by DeepSpeech to improve the accuracy of the transcription. The scorer identifies the probabilities of words occurring together. Certain words that sound the same but are pronounced different, like ‘red’ and ‘read’ are best distinguished by the language model.

1. Downloading and extraction of KenLM source files: The archive containing the C++ source files were downloaded from the official website and unzipped into the project directory in Google Drive.
2. Building the binaries from the sources: The executable binary files were generated by building the source files with the CMake build system.
3. Reading the text corpus and generating statistics: The aggregated text corpus was scanned token-wise, and the number of occurrences of each token was counted. This list was sorted in the descending order of occurrence, and the top 5 million tokens were selected and saved in a text file. This file would be the input for building the trie.
4. Building the LM trie binary: The trie file represents associations between words, so that during training, words that are more closely associated together are more likely to be transcribed by DeepSpeech. We decided to use a trie data structure rather than array-based probing as tries use the least memory, have the best memory locality, and are decently fast. The text file containing the tokens was passed as input to the binary. A on-disk sort of the tokens is performed, after which the probabilities of the tokens are derived from their frequency data. This probability is then quantized, and an ARPA model is built. Finally, this model is used to generate the LM binary file.
5. Packaging of the binary into DeepSpeech scorer format: The DeepSpeech library provides a utility (generate\_scorer\_package program) to convert the KenLM language model binary to a DeepSpeech-friendly scorer package which can be used during model testing and inference.