Automatic Speech Recognition Systems in Mathematical Domains

Ayan Gangopadhyay

Department of Computer Science
Ramakrishna Mission Vivekananda Educational and Research Institute
Belur Math, Howrah
Pin - 711 202, West Bengal

A thesis submitted to
Ramakrishna Mission Vivekananda Educational and Research Institute
in partial fulfillment of the requirements for the degree of
MSc in Big Data Analytics
2018

Dedicated to ...

My father and my guiding light Shri Tapas Gangopadhyay who was taken entirely too soon from us and my mother Shrimati Shobha Gangopadhyay who has been my greatest source of strength.

Acknowledgements

NVIDIA,
Santa Clara, California, United States

Purnendu Mukherjee

ConvAI

All of my co-workers without whom this work would not be possible

Jun 7, 2021

CERTIFICATE FROM THE SUPERVISOR

This is to certify that the thesis entitled *'Automatic Speech RecognitionSystems in Mathematical Domains'* submitted by *Mr. Ayan Gangopadhyay*, who has been registered for the award of MSc in Big Data Analytics degree of Ramakrishna Mission Vivekananda Educational and Research Institute, Belur Math, Howrah, West Bengal is absolutely based upon his/her own work under the supervision of *Mr. Purnendu Mukherjee* of NVIDIA, Santa Clara, California, United States and that neither his/her thesis nor any part of the thesis has been submitted for any degree/diploma or any other academic award anywhere before.

Mr. Purnendu Mukherjee
Senior Deep Learning Software Engineer
NVIDIA
Santa Clara, California, United States

Abstract

The field of Natural Language Processing has been moving forward at a breakneck pace as Deep Learning techniques have come into their own. Of particular interest to this work is the task of Automatic Speech Recognition with is simply converting audio to text automatically. Present approaches have shown massive improvements in this task but the dream of a completely automated Automatic Speech Recognition(ASR) system remains out of reach, and the need for human verification of transcriptions from such systems has not yet been completely done away with. Our objective was to build a solution for transcribing tutorial videos on basic high school Algebra as an integrated product. This system would require a module for actually transcribing the text and a text-editor like interactive tool for looking at the transcripts produced and allowing the human user to easily alter the transcript as and when required.

This presented work was done by our team at **ConvAI** (tentative name) for the popular tutorial platform Study Edge.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The end goal of this project is a consumer product and as such there are a few key differences between the philosophy that is being pursued here and the more open ended methodology of a pure research project. These differences as well as the background and the problem statement will be discussed here.

1.1 Background

Study Edge

Study Edge is an online learning platform which provides high quality study material in a wide variety of subjects including chemistry, economics, mathematics, accounting, and physics. They provide both video material as well as in-person coaching classes.

Since they put out a lot of videos, they are also in need of high quality transcription services. Much like what might be required for our university if we have to continue online classes.

Currently Study Edge hires a lot of interns to do the transcription. But this is expensive, slow and prone to human errors. As result this is an excellent avenue for automation and this why they opted for a system to help them generate transcripts.

1.2 Problem Statement

Our goal was to create a system for producing high quality transcripts with help from humans. This will consist of two major components.

- An state of the art ASR module to generate transcripts in the Mathematical domain

- An interactive tool to correct errors in the transcript

Building both of these components will be quite different from each other. And each has its own challenges. The first few months of my work were dedicated to building the first component, and the second part was dedicated to working on the second.

Although both of these components seem quite unrelated, as we will see there will be surprising ways in which one will help to build the other.

The problem statement at this level seems to be quite generic, so we should drill down into each of these components to see exactly what is required.

1.2.1 A State of the Art ASR model in the Mathematical Domain

There are quite a lot of speech to text systems that are available nowadays, on of the most widely prevalent ones is the Google Speech to Text Application Programming Interface (API) which almost all of us have come across. YouTube generates its closed captions using this API and for most general purpose video transcription this works reasonably well.

There are also some other vendors which provide speech recognition services. Here we list some popular ones below[1]:

- Microsoft Azure Speech to Text Microsoft speech to text is a paid service offering high quality transcriptions in more than 85 languages. It provides the option for choosing your own models and customizing them, flexible deployment and production level robustness.

- Amazon Transcribe Amazon transcribe is another widely available service which can be integrated into other workflows like automating service calls, generating subtitles, generating metadata for media assets and even create search-able archives. Amazon is also foraging into the medical industry with introduction of Amazon Transcribe Medical in order to handle transcriptions and ASR services in the medical field, where the accuracy of these services is

[1]in no particular order

highly critical. Amazon transcribe also gives us access to highly desirable features such as adding punctuation and making the generated transcripts very human friendly. One also has the option of filtering out Personally Identifiable Information (PII) and foul language while generating these transcriptions.

- IBM Watson Speech to Text IBM Watson also provides speech to text services which are very popular and are industry ready. The most important feature of Watson's suite is that it is protected by world-class data governance practices of IBM as well as being very versatile when it comes to deployment.

For our work we have used the Jarvis platform that has been recently released for public use by Nvidia. Jarvis is different from the above mentioned transcription tools in that it is not only geared towards building transcription services but for conversational AI services in general. It also boasts of real-time performance (inference) using GPUs. Since we plan on extending our suite of services to extend well beyond the discussed transcription tool, we found that it was a good option to build all our technology from the ground up using one unifying framework. This takes care of the first part of the transcription generation process since the transcription module was built using Jarvis.

1.2.2 An Interactive Tool for Transcript Correction

To create an interactive tool for creation for creation of transcripts we decided on using the popular `React.js` framework. Our back-end APIs (Application Programming Interface) are handled by `Python Flask`. Authentication systems have been build using `Firebase` which was built by Google to create Web applications. Our entire application is deployed on the `Google Cloud Platform`. We also heavily make use of containerization using `Docker`.

1.3 Break Up of This Thesis

Being a small start-up our current team is small and each member (currently five) has had to take up multiple roles while working on this product. For this reason it would be inefficient and difficult to follow the general outline that is recommended for this thesis. Instead, each of the subsequent chapters will enclose all the parts of the research pipeline i.e. Introductions, Literature reviews (where applicable), Methodology and Discussion of Results. I feel this is necessary to adequately represent the work that I have done over the course of this internship and hope this is acceptable.

The following chapters will discuss the three broad areas that I worked on in general. These are[2] :

- Creating an Automatic Speech Recognition training pipeline using the initial data provided by Study Edge.
- Gathering text pertaining to Mathematics to create language models which would be used in our inference pipeline.
- Creating a use able web application for quickly generating and editing domain specific transcripts.

As such, the following three chapters will cover the work that I did in some depth in each of these areas[3].

[2]In chronological order.

[3]Study Edge has not given express permission to share their data publicly and hence there will be no discussion of the data set that they had provided. Also most of the work presented here is an Intellectual Property of ConvAI and should not be released to the general public in any way shape or form even so, I am not allowed to discuss every detail of the work done and some parts must be left vague to protect our IP.

Chapter 2

Setting up an ASR Fine-tuning Pipeline

2.1 Literature Review

Benchmarking the task of Automatic Speech Recognition is a difficult task due to its heavy dependence on data quality and the nuances of speech. Especially difficult is trying to evaluate and understand how to design a metric to measure the performance of a speech recognition system which takes into account the various nuances of the ASR task.

Currently the most popular metric is the Word Error Rate or WER which is what the papers with code community uses to benchmark the ASR task. Let us first discuss the WER metric.

2.1.1 The Word Error Rate Metric

The word error rate is based on the Levenshtein distance or the Edit distance. Since the predicted transcription is often not the same length as the ground truth it is first necessary to dynamically align the two pieces of text. The WER algorithm[1] works on the two pieces of text at the word level instead of the character or phoneme level which is more common when dealing with algorithms which measure Edit distance.

[1] [7] gives a thorough treatment of the WER algorithm

The word error rate is then computed as:

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

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 words
- N is the number of words in the ground truth (N=S+D+C)

Note: The WER can be > 1.0 since the length of the text in the ground truth can be more or less than the length of text that is predicted.

There are a number of problems with the WER metric. Some of which are detailed below[2]:

No record is taken of the results that various kinds of error may have on the probability of effective result, for example a few errors might be more problematic than others and some might be amended more effectively than others.

Another major theoretical problem in assessing the performance of a system is deciding whether a word has been "mis-pronounced," i.e. does the fault lie with the user or with the recogniser. This may be particularly relevant in a system which is designed to cope with non-native speakers of a given language or with strong regional accents.

The pace at which words should be spoken during the measurement process is also a source of variability between subjects, as is the need for subjects to rest or take a breath. All such factors may need to be controlled in some way.

Depending on the domain the acceptable WER may vary. For example, a system which transcribes a medical practitioners oral instructions needs to be highly accurate and hence will require a very low (comparatively) WER.

From these arguments it is clear that even though WER is the commonly accepted performance metric for ASR systems, it does not give us the whole picture about how well an ASR system is working and further analysis is often necessary to

[2]Most of these are given in the excellent article on WER on Wikipedia.

find out problem areas. Even so, this is currently the best metric that is available and we have to make do with it. Much of the results presented will be compared and analysed through the lens of the WER, further analysis, when necessary has been done.

2.1.2 Different Approaches to ASR

The main task of the ASR task is to convert some given audio into its corresponding text. As we have seen earlier the currently accepted measure of performance for this task is the WER error rate. Since a lower WER is usually taken to be better, we would want our system to produce transcripts which have a low WER when compared to some ground truth.

Traditional approaches to ASR take a generative approach, we try to understand the human process of actually generating speech and try to mimic the different steps using different modules. These modules are shown in Figure 2.1.

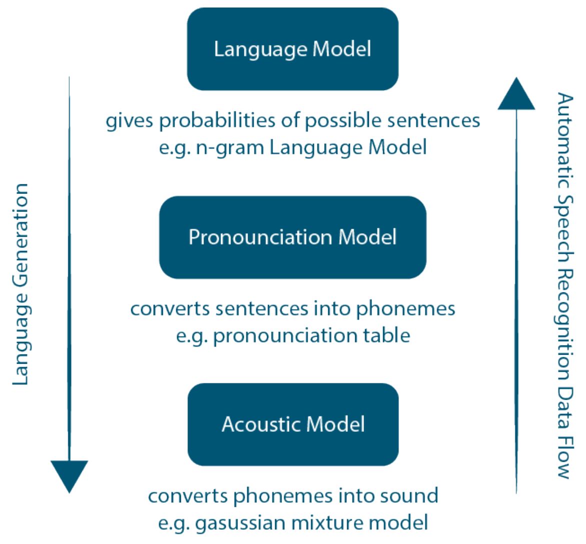

Language Model
gives probabilities of possible sentences
e.g. n-gram Language Model
Pronounciation Model
converts sentences into phonemes
e.g. pronounciation table
Acoustic Model
converts phonemes into sound
e.g. gasussian mixture model
Language Generation
Automatic Speech Recognition Data Flow

Figure 2.1: Traditional Generative Modelling of the Speech Synthesis Pipeline

Initially, we had statistical models for each of this modules but with the advent of GPUs and the increase in compute over the last decade or so, we can replace these old, error-prone and extremely specific statistical models with more general deep learning or neural network based models.

More recently though, there has been a shift away from the traditional generative approach and towards a more discriminative approach which simply takes audio as input and produce text as output. The best part of using this approach is that each module does not have to be trained differently on different tasks (which might lead to errors in one module throwing off the entire prediction process) thus decreasing the complexity in training the model while increasing the robustness of the whole training pipeline.

There are interesting parallels between the end-to-end discriminative approach and the traditional generative approach. The encoder used in the end to end approach conceptually corresponds to the acoustic model of yore and the decoder works as an amalgamation of the pronunciation model and language model. Modern end-to-end ASR models such as `Jasper`[6] and `QuartzNet`[5] even allow us to use a Language Model to improve the performance of these models. Let us delve a little deeper into these end-to-end/discriminative approaches[3].

It is a natural choice to pick RNNs in order to build our ASR systems because of the fact that both our input and outputs are sequences, onee audio and the other text. But this choice has a glaring problem, the length of the audio timesteps is obviously not going to be the same as the length of the transcription string. This matching of the audio timesteps to the correct output text was handled earlier by using temporarily aligned data which was difficult and expensive to attain as well as being error prone (the reader can easily think of the many ways this could go wrong). Instead, we now use two variants of the end-to-end approach called,

1. Connectionist Temporal Classification.

2. Sequence-to-Sequence with Attention.

Since the model that we have used, `QuartzNet` is a CTC based model, we will only discuss this in some more depth as further discussion is out of the scope of this work[4]. The notebook that is mentioned in the footnotes has a succinct description of what a CTC model is and is quoted here verbatim.

[3]These ideas are explored with much detail and with code in the `ASR-with-NeMo` python notebook which the reader can try out for themselves

[4]The interested reader can refer to

- An Intuitive Explanation of Connectionist Temporal Classification
- CTC Networks and Language Models: Prefix Beam Search Explained
- Sequence to Sequence with Attention[2]

In normal speech recognition prediction output, we would expect to have characters such as the letters from A through Z, numbers 0 through 9, spaces (”_”), and so on. CTC introduces a new intermediate output token called the blank token (”-”) that is useful for getting around the alignment issue.

With CTC, we still predict one token per time segment of speech, but we use the blank token to figure out where we can and can’t collapse the predictions. The appearance of a blank token helps separate repeating letters that should not be collapsed. For instance, with an audio snippet segmented into T=11 time steps, we could get predictions that look like BOO-OOO–KK, which would then collapse to ”BO-O-K”, and then we would remove the blank tokens to get our final output, BOOK.

Now, we can predict one output token per time step, then collapse and clean to get sensible output without any fear of ambiguity from repeating letters! A simple way of getting predictions like this would be to apply a bidirectional RNN to the audio input, apply softmax over each time step’s output, and then take the token with the highest probability. The method of always taking the best token at each time step is called greedy decoding, or max decoding.

To calculate our loss for backprop, we would like to know the log probability of the model producing the correct transcript. We can get the log probability of a single intermediate output sequence (e.g. BOO-OOO–KK) by summing over the log probabilities we get from each token’s softmax value, but note that the resulting sum is different from the log probability of the transcript itself (BOOK). This is because there are multiple possible output sequences of the same length that can be collapsed to get the same transcript (e.g. BBO–OO-KKK also results in BOOK), and so we need to marginalize over every valid sequence of length T that collapses to the transcript.

Therefore, to get our transcript’s log probability given our audio input, we must sum the log probabilities of every sequence of length T that collapses to the transcript. In practice, we can use a dynamic programming approach to calculate this, accumulating our log probabilities over different ”paths” through the softmax outputs at each time step.

The model that I have used is `QuartzNet 15x5`, which is a CTC based model. This concludes the opening introduction to the literature review. Let us now look at some SOTA models and their performance in comparison to `QuartzNet`.

2.1.3 Benchmarking Datasets and current State of the Art

One of the most popular datasets which is currently used to create benchmarks for ASR systems is the LibriSpeech dataset. The following figure shows the popularity of the LibriSpeech dataset when compared to other ASR datasets. (Such as Common Voice and Libri Text To Speech)

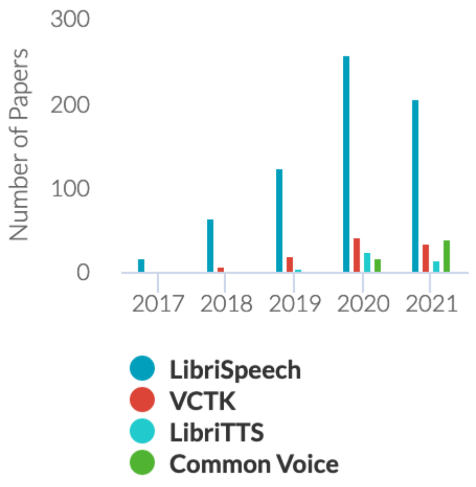


Figure 2.2: Popularity of LibriSpeech

It can be easily seen that the LibriSpeech Dataset is considerably more popular than other ASR benchmarking datasets. For this reason we will show a relative comparison of the WER achieved by different models based on this. But first let us see what the dataset itself is about and how the task for this dataset is defined. We can refer to the abstract given on the Papers with Code website to understand what the dataset is about.

> The LibriSpeech corpus is a collection of approximately 1,000 hours of audiobooks that are a part of the LibriVox project. Most of the audiobooks come from the Project Gutenberg. The training data is split into 3 partitions of 100hr, 360hr, and 500hr sets while the dev and test data are split into the 'clean' and 'other' categories, respectively, depending upon how well or challening Automatic Speech Recognition systems

would perform against. Each of the dev and test sets is around 5hr in audio length. This corpus also provides the n-gram language models and the corresponding texts excerpted from the Project Gutenberg books, which contain 803M tokens and 977K unique words.

The LibriSpeech dataset was introduced in [8] and has enjoyed a healthy legacy as a benchmarking dataset for ASR systems.

The current leading models for this data set are given in Table 2.1 along with their WER scores.

Paper	**WER**
Conformer + Wav2vec 2.0 + SpecAugment-based Noisy Student Training with Libri-Light	1.4
Conv + Transformer + wav2vec2.0 + pseudo labeling	1.5
ContextNet + SpecAugment-based Noisy Student Training with Libri-Light	1.7
. . .	
wav2letter++	3.26
QuartzNet-15×5 with 6-gram LM	**2.96**

Table 2.1: Current SOTA for the LibriSpeech Dataset*
* Taken from papers with code

We see that that the `QuartzNet` model has comparable performance to the leading ASR models. The attractiveness of this model comes from the fact that it has only 15 million parameters which is considerably less than the number of parameters in the closest `wav2letter++` model which has around 208 million parameters. Nvidia provides a pre-trained `QuartzNet` model which is ready to use for Transfer Learning through their `nemo-asr-app` Docker container. The better performing Conformer models are still not available for production use and therefore were not considered. A thorough reasoning for the usage a comparatively ”worse” model will be given in the next section.

2.2 Methodology and Technologies Used

Here the ASR training pipeline will be described in detail. I will also discuss the technologies that I have used and reasons for using them. Let us begin with the model selection.

2.2.1 Model Selection

The `QuartzNet` model which is available for use is described in [5] and a quick reference can be found on Figure 2.2[5]. There are many reasons why we have chosen this model, some of which include the fact that being relatively smaller; inference times using the `QuartzNet` model are much faster. Being available through Nvidia NGC means that this model is ready for acceleration using `Nvidia Apex`, which allows for faster processing using mixed precision arithmetic. This technique makes use of the fact that most floating point calculations in the Deep Learning setting do not need high accuracy and using a lower precision while dealing with floating point model parameters renders a regularizing effect. This is widely used in the industry while creating Deep Learning products and more information can be found on the documentation page for `Nvidia Apex`.

2.2.2 Implementation

Being available through Nvidia NGC, `QuartzNet` also has the capability of being trained in a multi-threaded way, making full utilization of Nvidia GPUs and drastically improving training as well as inference times by using `Nvidia TensorRT`. The considerable advantages of using these technologies can be seen from the fact that I was able to transcribe a 8-9 minute video while giving my presentation in real time. In the absence of the tremendous acceleration that is afforded by using these technologies, it would have been impossible to achieve such a feat.

The `nemo-asr-app` provides us with a docker container with can be deployed on any machine having an Nvidia GPU and CUDA installed. This gives us a powerful and reproducible way to train models especially when using Cloud instances such as `GCP` and `AWS`. This Docker + Cloud + Jupyter trio is extremely useful for creating ML research pipelines and the use of containerization means that this is extremely reproducible as well.

[5]Taken from the Nvidia Developer Blog

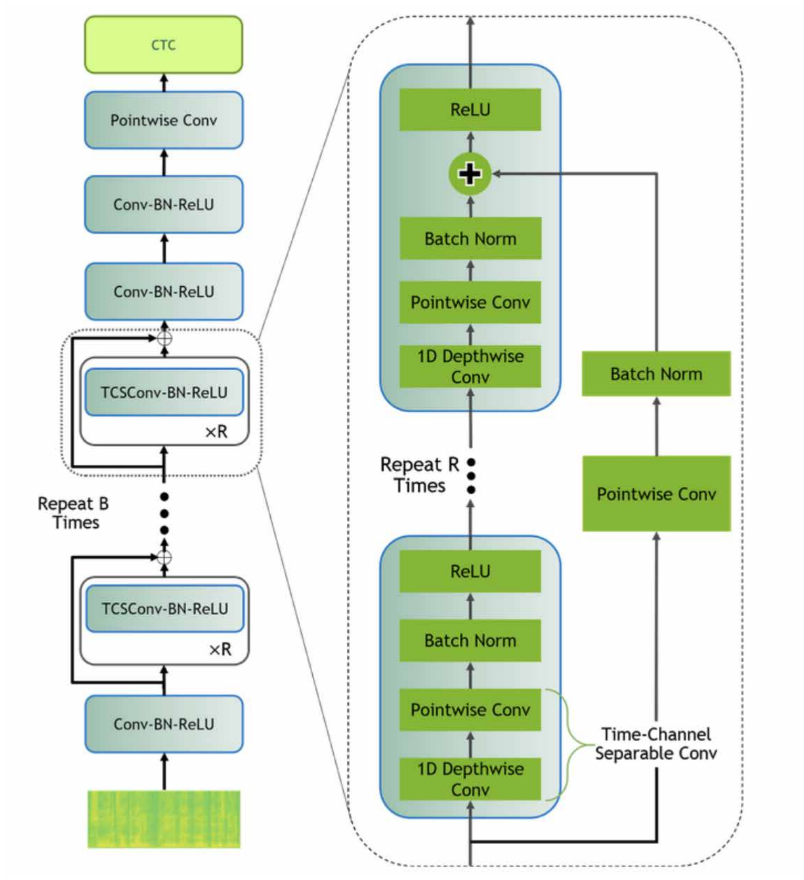


Figure 2.3: The QuartzNet architecture

Using the `nemo-asr-app` and serving it on a GCP instance means that we can run our models from Jupyter Lab using our browser as it it was running in our own machine. Any data cleaning and transformation can be done using the jupyter kernel and notebooks.

2.2.3 Dataset

Study Edge provided us with around 800 videos and their corresponding human generated transcripts. These were given the the form of `.srt` files and needed to be processed using python scripts to make them amenable for processing using the aforementioned `nemo-asr-app`. This code is present in our private GitHub repository and can be shown on request.

2.3 Discussion of Results

I split the data given to us into train, test and dev sets. We achieve a train accuracy of $\approx$ **94%** which was great but on evaluating the model at the test sets we saw that the accuracy dropped to around **54.65%**. This showed us that the model was heavily overfitting the data. I proposed two solutions to get around this.

The first was obviously to get more data, this has been the goal of our team for most of the second half of the work. Another was to switch up the training pipeline. This improved training pipeline has not been implemented yet, but the schematics for the original pipeline and the newer proposed pipeline are given here.

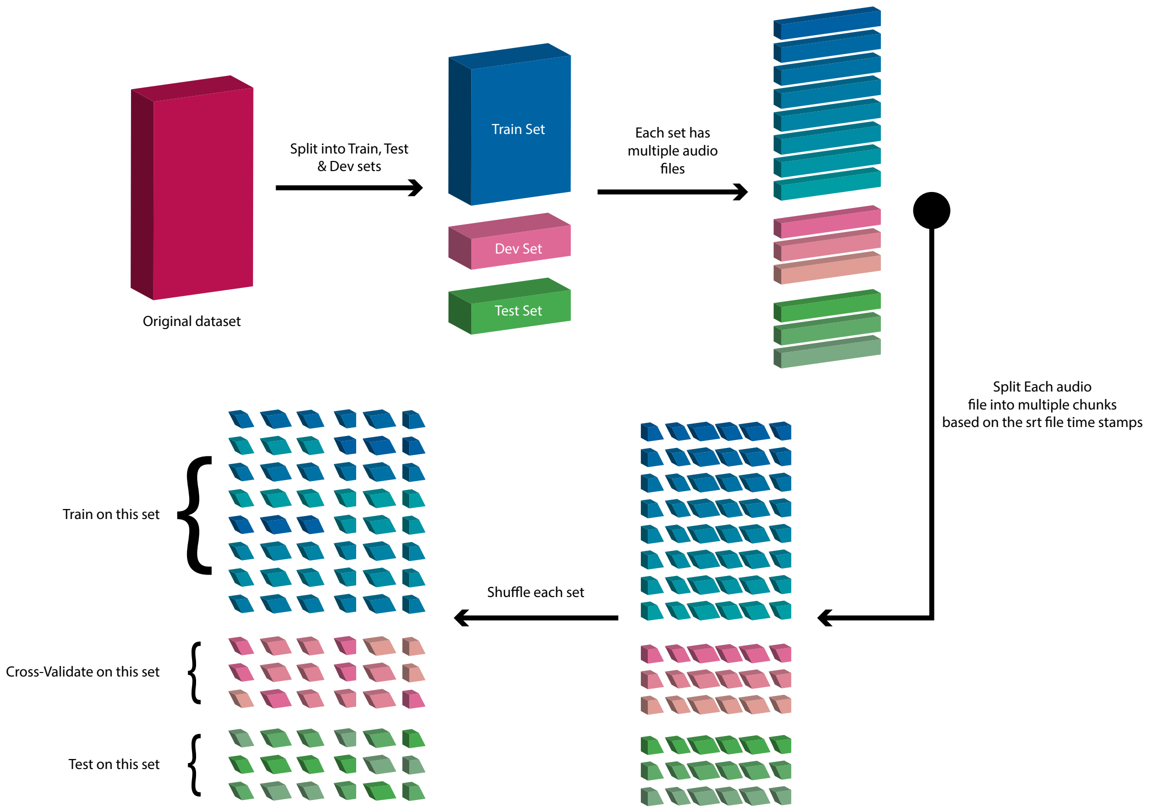
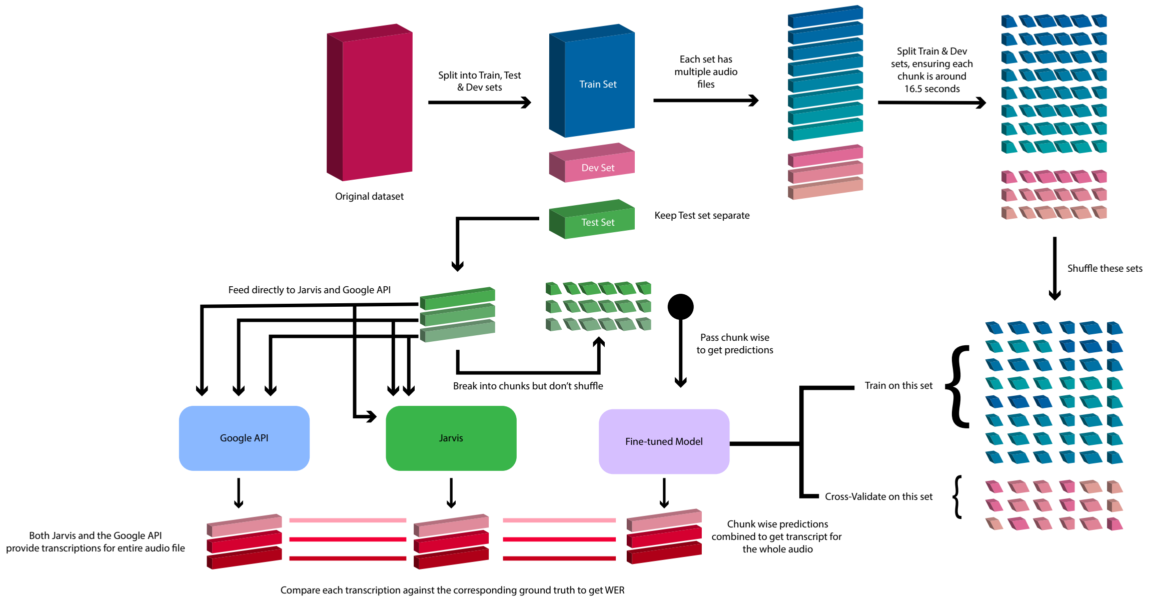

Original dataset
Split into Train, Test & Dev sets
Train Set
Dev Set
Test Set
Each set has multiple audio files
Split Each audio file into multiple chunks based on the srt file time stamps
Shuffle each set
Train on this set
Cross-Validate on this set
Test on this set

Figure 2.4: Original Training Pipeline

2.4 Future Work

Future work will obviously include implementing the second training pipeline as well as incorporating more data. Jarvis allows continuous integration of models and as



Original dataset
Split into Train, Test & Dev sets
Train Set
Dev Set
Test Set
Each set has multiple audio files
Split Train & Dev sets, ensuring each chunk is around 16.5 seconds
Keep Test set separate
Shuffle these sets
Feed directly to Jarvis and Google API
Break into chunks but don't shuffle
Pass chunk wise to get predictions
Train on this set
Cross-Validate on this set
Google API
Jarvis
Fine-tuned Model
Both Jarvis and the Google API provide transcriptions for entire audio file
Chunk wise predictions combined to get transcript for the whole audio
Compare each transcription against the corresponding ground truth to get WER

Figure 2.5: Improved Training Pipeline

such we will improving our model in real time as more transcription data comes in. This feedback loop will also need to be implemented and provides its own set of implementation challenges.

Chapter 3

Language Model Fine-Tuning

We have seen that CTC based models like `QuartzNet` allow integration of Language Models for improving the beam search part of our pipeline to provide better transcriptions. This is a critical part of adapting a model to any specific domain because equipping a domain specific language model for an ASR system is analogous to a person completing their Bachelor's Degree in a specific discipline. Language models essentially give the system a "flavour" of a specific domain by assigning higher probabilities to sentences which seem plausible to have been generated in the domain that is being considered. This was also a part of our model which we could develop independently without any dependence on Study Edge and their private data.

3.1 KenLM Binaries

The KenLM binaries and arpas are explained and discussed in detail in [3]. These are basically query-able estimates of *Kneser-Ney n-gram* language models which will be described in the next section. The Jarvis Speech Skills documentation explicitly states that it supports KenLM binaries for all the available models that are offered in the Jarvis Speech Skills suite and that these provide an easy way to adapting a pre-trained model to a specific domain.

KenLM binaries can easily be be created using the excellent utility which can be downloaded from the GitHub Repo for free. These provide excellent estimates for *Kneser Ney n-gram models* is a fast and computationally stable manner. The only restriction is that only models up to 6-gram are supported which is acceptable since anything larger will be too computationally unwieldy to be of any practical use.

We have collected data for and trained a variety of KenLM binaries to improve our performance over the standard Google Speech to Text API. The data collection process for creation of binaries will be discussed in some detail in the following sections.

3.2 n-gram Language Models

An *n-gram* language model is a way to assign probability to a word given some previous words. For example, suppose we want to predict the next word in the sequence,

`Please turn your homework ...`

Based on our experience with language we might guess words like *over* or *in* but we would not expect words such as *helicopter* or *headphones.* This intuition allows us to rank words that we know as having some probability of occupying the next position. Of course without knowing how this sentence is being generate, all words are possible and the next word is essentially a random variable whose domain is the English dictionary (or maybe even all character strings).

When creating *n-gram* language models, we are doing something similar. We are trying to rank the words in our vocabulary based on the probability of being the next word given some previous words. The n in *n-gram* is basically an indication of how many words do we consider before trying to predict a next word. In reality we predict the next word using a *global context*, which means we consider the entire body of text that we have gone through before trying to predict the next word. The entire body of text has an effect on what the next word will be. This however is impractical in the case of an *n-gram* language model (the reason will become apparent when we see how *n-gram* models are constructed). Therefore we essentially make a Markovian assumption that any word depends on n previous words.

Let us try to compute $P(w|h)$ where w is a particular word, say *"the"* and h is the history, say *"its water is so transparent that"*. Now one way to estimate this probability is to find the count of the phrase *"its water is so transparent that* ***the****"* and divide it by the count of the the word *"the"* in a large corpus, like the entire text of Wikipedia. This is an application of the simple Bayes rule and can be written out as,

$$P(w|h) = \frac{Count(\text{"}its\ water\ is\ so\ transparent\ that\ the\text{"})}{Count(\text{"}the\text{"})}$$

The *n-gram* model assumes that to estimate the probability of the word *"the"* appearing, only the count of the previous n words are required. For instance in our example if $n = 3$, the *n-gram* model assumes the following,

$$\begin{aligned} P(w|h) &= \frac{Count(\text{"}its\ water\ is\ so\ transparent\ that\ the\text{"})}{Count(\text{"}the\text{"})} \\ &= \frac{Count(\text{"}so\ transparent\ that\ the\text{"})}{Count(\text{"}the\text{"})} \end{aligned} \tag{3.1}$$

Further details about *n-gram* models can be found in the excellent book on Natural Language Processing by Dan Jurafsky and James Martin[4]. To understand the role of *n-gram* models, this much background will suffice.

3.3 Webscraping versus APIs

The Python library BeautifulSoup4 provides and excellent interface for scraping of web pages, and even though we have a very capable webscraping library we often see that it is a very unreliable way to gather data from the web. For example, even a very carefully constructed webscraping script might break with only minor changes in the javascript that generates the page. As such after using both webscraping and collecting data through official APIs I found that though usually much slower because of the site throttling on API calls, they are by far the most reliable way of collecting data.

Therefore most of the data that was collected was done through API calls and not webscraping as the overhead was simply not worth the effort.

3.4 Corpus Selection

To create useful language models for our use case the following sources of data were considered.

- All Khan Academy mathematical video transcripts
- Math Stack Exchange questions, comments and answers
- Mathematical Subreddits

- arXiv (which has not yet been implemented)

I will now discuss the issues and findings regarding each of these data sources.

3.4.1 Khan Academy Transcripts

The first data source we tried was using Khan Academy transcripts, these were collected using a GitHub repo which provided scripts to download transcripts from a set of video urls. The video urls were collected using BeautifulSoup4 and the scripts form the GitHub repo were used download the actual transcripts. We created KenLM binaries from these transcripts which provided a marginal improvement over the Google Speech To Text average WER for our dataset of about 4%. It was concluded that the transcript data was too little to build a meaningful corpus.

3.4.2 Math Stack Exchange

Next up we used the Math Stack Exchange API to collect question links and BeautifulSoup4 to extract question, answers and comments for the questions posted to those links. The api calls were throttled to 300 per day and as a result too much data could not be collected. The total corpus size was around 15Mb. This was also not enough to provide significant improvements for our model and we observed a further 1.5% in our average WER.

Most of the complication arose due to the extensive presence of MathJax and LaTeX present in the questions and there was no straightforward way to parse these into plaintext. This problem, we hypothesized posed a significant hindrance to collection of a Mathematical Corpus.

3.4.3 Mathematical Subreddits

Next up, all the text data that was present in 30 odd mathematical subreddits were collected and put together. Again the presence of MathJax and LaTeX was a hindrance, in addition to the problem of extensive use of colloquialisms in the language used on Reddit. The second problem was mitigated somewhat by using WordsAPI[1].

[1] WordsAPI provides an API for validating correctly formed English words and providing their meanings

We used a reddit downloader script which was developed by Nvidia and cannot be referenced here to collect around 3.4Gb of data which was by far our most comprehensive corpus. After validation this shrunk to around 2.4Gbs but still provided a lot of context for mathematical sentences.

The data itself was just plaintext from the questions, comments and discussions on these subreddits. Most of these discussions are mathematical in nature and gives more context math words.

The KenLM binary created gave us significant improvement over the last model and was found to have improved our average WER by about 8% over that provided by the Gooogle Speech to Text API.

3.4.4 arXiv

This seems to be the most promising of the avenues for creating a high quality mathematical/technical corpora. arXiv provides free download of all the papers that are hosted on their servers along with their metadata and plaintext versions (excluding the LaTeX). If we figure out a way to solve the issue of converting LaTeX to plaintext this might prove to be a literal goldmine for our ASR system since the amount of data $\approx$ 270Gb (and hence the context) present far outclasses anything that we had previously. This is yet to be integrated and might prove to be a lucrative avenue for future work.

This has not been currently implemented because of the considerable amount of compute required for building a KenLM binary using about 270Gb of data, once our start up has been approved external funding we will be able to afford the instances which can be used to build such large language models.

3.5 Discussion of Results

Using language model binaries has definitely been a significant approach through which we have improved our model. Some of the results of the model are shown in Figure 3.1.

But of course just having a better WER does not necessarily mean that an ASR system produces "good" transcriptions. To analyse the actual performance of our model we tried two different ways. One is to see how many occurrences of "math-words" our model actually identified. We expected that since our model is

	n-gram	Verified with WordsAPI	Average WER
Khan Academy Transcripts	**3**	✗	**26.6656**
Math Stack Exchange	**5**	✗	**25.9771**
Reddit	6	✗	**25.8450**
Reddit	6	✅	**20.6278**
Google STT	-	-	**28.5332**

Figure 3.1: Comparison of Performance of Various KenLM binaries

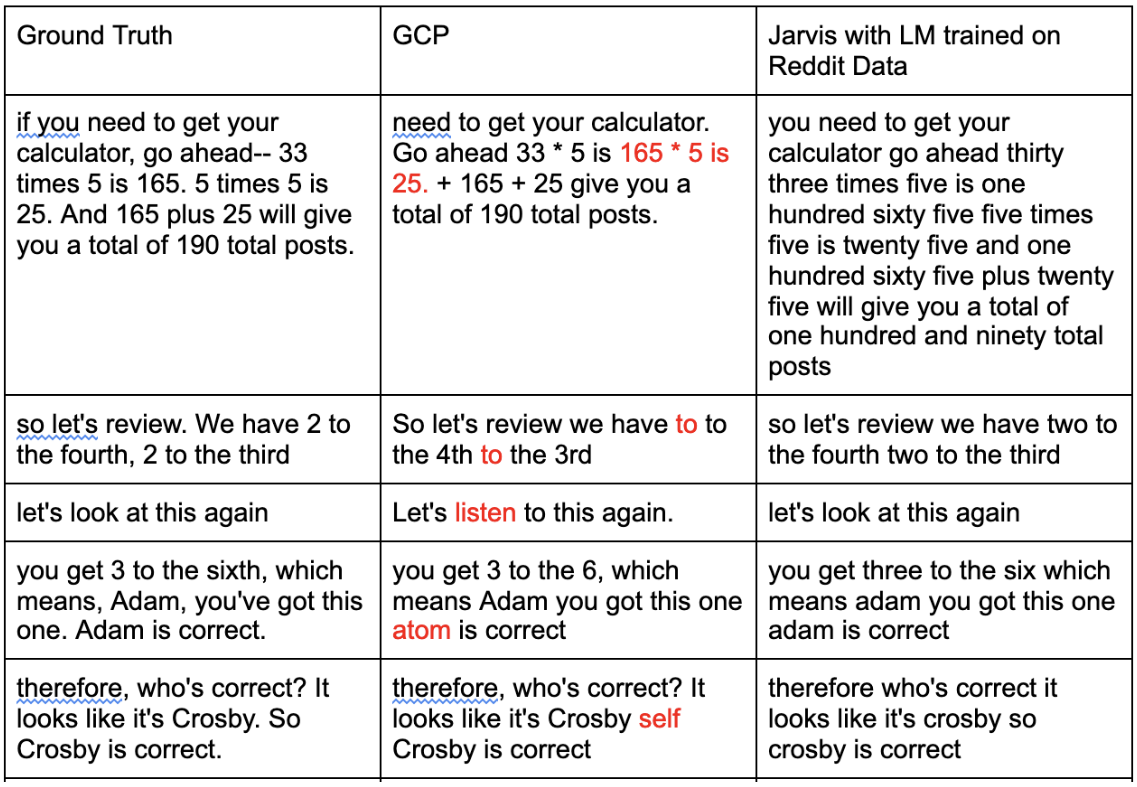
supposed to be suited to the mathematical domain, it would correctly identify such words which are commonly seen in the mathematical context. The results were inconclusive and are presented in Figure 3.2.

Word	Original	GCP	Jarvis
quadratic	1786	1718	1425
exponent	514	445	327
polynomial	741	705	533
algebra	138	104	155
factoring	265	218	174
subtraction	249	229	216
multiply	2223	1298	1784
divide	1351	599	1044

Figure 3.2: Comparison of Math-Word Counts

The other approach was to compare time stamps and align the transcripts generated by our model temporally to that generated by the Google Speech to Text API (which we consider our baseline) and manually see which transcripts ”look” to be better. Such a comparison (which was carried out with the help our transcript editor tool) proved to us that in many scenarios the Jarvis transcripts just made better sense than the Google transcript, but this ”sense” cannot currently be measured and

I hope that a few examples (shown in Figure 3.3) will convince you of my claim.



Ground Truth	GCP	Jarvis with LM trained on Reddit Data
if you need to get your calculator, go ahead-- 33 times 5 is 165. 5 times 5 is 25. And 165 plus 25 will give you a total of 190 total posts.	need to get your calculator. Go ahead 33 * 5 is 165 * 5 is 25. + 165 + 25 give you a total of 190 total posts.	you need to get your calculator go ahead thirty three times five is one hundred sixty five five times five is twenty five and one hundred sixty five plus twenty five will give you a total of one hundred and ninety total posts
so let's review. We have 2 to the fourth, 2 to the third	So let's review we have to to the 4th to the 3rd	so let's review we have two to the fourth two to the third
let's look at this again	Let's listen to this again.	let's look at this again
you get 3 to the sixth, which means, Adam, you've got this one. Adam is correct.	you get 3 to the 6, which means Adam you got this one atom is correct	you get three to the six which means adam you got this one adam is correct
therefore, who's correct? It looks like it's Crosby. So Crosby is correct.	therefore, who's correct? It looks like it's Crosby self Crosby is correct	therefore who's correct it looks like it's crosby so crosby is correct

Figure 3.3: Examples of Jarvis Performing Better

3.6 Future Work

The next step for this part of the project will of course be to create and integrate the KenLM binary from the arXiv corpus. Since the data is free to download from their Amazon S3 bucket, it would not me much of a hassle to actually get the data.

Key challenges include the fact that we don't know if the KenLM utility can actually create such large language model binaries. A very high RAM computer might be necessary to create these binaries since the KenLM utility does not provide GPU acceleration. Other than that the key issue of handling inline LaTeX remains a problem and will need to be dealt in some way. This paper [1] shows us some ways to handle such an issue and will need to be explored in more detail.

Chapter 4

A Transcript Editor Tool

At end of the last chapter we saw that it was difficult to evaluate the actual performance of our model using just the WER. We also saw that the data we had received from our client was not adequate for fine-tuning the Encoder of the `QuartzNet` that we had used. Therefore to improve our model and give our clients a useful utility as soon as possible, we came up with the idea of a Transcript Editor tool which would allow us to continuously retrieve data whenever it is used on any video and use this data as for improving our model.

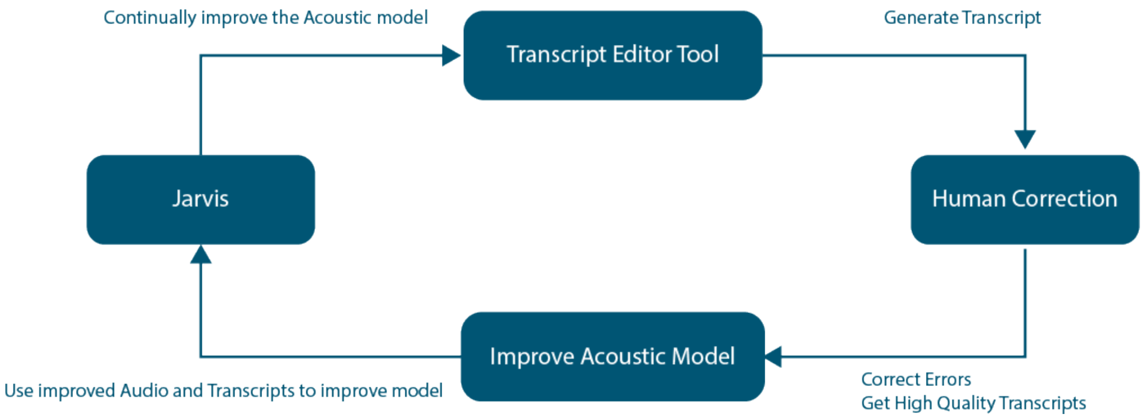

Continually improve the Acoustic model
Transcript Editor Tool
Generate Transcript
Jarvis
Human Correction
Improve Acoustic Model
Use improved Audio and Transcripts to improve model
Correct Errors
Get High Quality Transcripts

Figure 4.1: The Positive Feedback Loop

This will essentially work as a positive feedback loop where our model performs better the more the transcript editor tool is used and the better our model become the more people will actually want to use the editor tool. It should also be noted that the better our model gets, the less time the generation of an entire transcript will take. This constitutes a win win situation for us since we will keep improving our model at a faster rate the better it get. There will also come a time when an equilibrium is reached after which we will maybe starts getting diminishing return from updating our model. This will need to be monitored and we are hopeful to integrate the amazing library WandB to be able to continuously monitor our model.

Figure 4.1 gives us a schematic of the feedback loop that we are trying to achieve.

4.1 Planning our Web Application

We needed to design a web application which would give us some basic funtionalities, some of which are listed below.

- Ability to upload a video.
- Automatic Generation of Transcripts from two sources, one from our model and the other from the Google Speech to Text API.
- Compare these two transcripts time-step wise and show differences between them and visually colour them for better understanding, we wanted something similar to how GitHub showed differences while trying to merge branches.
- Ability to edit the transcripts inline and push the corrected transcript to a final one when the user is satisfied with the changes.
- We also wanted to have an Authentication System to segregate users and protect their privacy.
- Ability to download the corrected transcript.
- On the backend, the audio track corresponding to the uploaded video and the final corrected transcript would be pushed to our bucket for storage and future use.

It is easy to see how this application would increase the speed of transcript generation and give us our very own personalised version of a data collection system for this domain with the possibility of creating extremely high quality data for our ASR model.

4.2 Technologies Used

The following constitutes the full stack technology that we have decided on and are currently developing our application in.

Component	**Framework**
Front End	React.js
Authentication System	Firebase
Deployment	Google Cloud Platform and Node.js
Data Storage	Google Cloud Platform Buckets
Database	Firestore
API creation and management	Python Flask
Back-end Transcription Services	Nvidia Jarvis and Google Speech to Text

Table 4.1: Full Stack Components

Now we will give a little bit of an introduction to some of the components which the reader might not have information about. This is the part of our product that will be publicly available and even our university can use this for creation of their own transcripts.

4.2.1 React.js

The React Homepage simply states that React is a,

> ... is a JavaScript library for building user interfaces.

While being technically true, this statement does not do justice to the power of `React.js`. React gives us almost all the functionality of a full fledged framework like `Angular.js` while being much more lightweight. It is currently the most popular front end development framework and `React` developers are very much in demand due to the enormous number of webapps being built nowadays. `React` also supports mobile and embedded systems deployment using its `React Native` variant which allows us to make powerful applications running on lightweight systems.

4.2.2 Firebase

Firebase by Google is provides us with a multitude of services to add functionality to our Applications. These include but are not limited to providing authentication systems as well as secure hosting options. `Firebase` allows authentication through most social media providers as well as supporting simple email and password login. It also gives us advanced features such as email validation, user management routines and options to reset and update our user credentials.

We have used `Firebase` to great effect to develop our very own authentication system which looks and feels great!

`Firestore` is the database service which is inherently available with `Firebase` this gives us a great way to store the urls to the buckets where we store our data and user management is easy since it is integrated with `Firebase`.

4.2.3 Google Cloud Platform

Google Cloud Platform is one of the most well known cloud service providers and provides a plethora of services which are too many to list here. For our work we have made extensive use of GCP virtual machine instances (compute engines) and data storage buckets which are fast and easy to work with.

We have also deployed our own online services through the load balancer utility of GCP which provides us with reliable and secure SSL-verified hosting services.

4.2.4 Python Flask

Python Flask is a lightweight web development API for python and provides a much more *pythonic* interface for writing code to create web applications when compared to the much more heavyweight *Django* framework. For our work we have mainly used this to create API services for data flow and transcript generation.

4.2.5 Nvidia Jarvis[2]

Nvidia Jarvis Speech Skills is a toolkit for production-grade Conversational AI inference. We can build `gRPC APIs` or `Python APIs` (which we are currently using) and a variety of options for serving production grade software. The complete repertoire

[0]Much of the information presented here can be found on the Nvidia Jarvis Speech Skills documentation page

of services, models and customisation options that are available can viewed on their official documentation.

Jarvis provided us with the pre-trained `QuartzNet` model that we are trying to fine-tune. Currently the transcriptions generated by our backend only uses the base pre-trained `QuartzNet` model from NGC, with the additional KenLM binary that we created using the cleaned reddit data.

Jarvis provides a lot more customisation option to the extent that it can be almost considered a framework in itself, and working on this project has given me a new insight on how much the philosophy of creating production level AI products differs from that seen in academic settings.

4.3 Discussion of Results

Since this is a tool that was created we do not have results of this work as we would in a scientific experiment. But the outcome of the the work has been a success and we are currently hosting our tool online for usage. It is still in the early stages of development and any feedback will be valuable, we are going to provide our client this tool and improve both our model and front end in an iterative while receiving their feedback.

On the other hand developing this tool also gave us a way to directly compare the transcripts generated by our model and that generated by the Google Speech to Text API by breaking the entire transcript into time stamps. This analysis is not yet ready because of time constraints but will need to be done before further model tuning is performed.

4.4 Future Work

There is a lot of work yet to be done on this front as well, currently we are have a lot of issues integrating the database with our product. We need a thorough redesign of the database schema and dataflow process. We also need to set up a Continuous Integration / Continuous Development pipeline for training our model. We might even change the backend model used for our task (Citrinet seems very promising while being even faster). As for the front end `React` development, we also have a lot of things to do on the UI/UX front as well as adding more functionalities like adding advanced text editing options to our transcript editor field. As discussed in the meeting more people to help from our University would most definitely be welcome.

Chapter 5

Experimental Evaluation

5.1 Description of Data

We have used a lot of data from different sources, the primary data that we cannot disclose is the data that was provided to us by Study Edge, this constitutes about 792 video and their corresponding transcription files.

The data that was used for creation of language models are freely available and have been discussed in detail in Chapter 3. Some of the tools that were used to collect some of the data cannot be disclosed but other data generation sources are pretty standard using mostly packages like `BeautifulSoup4` or official APIs.

5.2 Evaluation Techniques

The primary evaluation technique that has been used is the Word Error Rate which has been described in some detail in Chapter 2.

Other than this WER metric we have developed a tool for showing us time-step separated deltas using the `diff` algorithm that is present in the `difflib` package in python. This is a much more manual way of finding which transcript is actually better and quite a lot of manual time and effort needs to be dedicated to actually analysing the results from the two models.

5.3 Experimental Setup

Since this entire work is novel and geared towards addressing a very particular problem, a proper state of the art or previous work does not exist in the literature. Nonetheless the parts that do have previous work done have been address and a relatively thorough literature review has been given in Chapter 2. More information about the current industry standards have been supplied via footnotes and links where necessary, I request the author to kindly go through these in case it sparks their interest.

5.4 Analysis of Results

Since my work was separated into three different but related streams, giving an analysis of the results in one place is infeasible. Instead each of the previous three chapters include a discussion of results area where the appropriate discussion is carried out. I would implore the reader to please refer to these links to reach the required areas.

1. Chapter 2
2. Chapter 3
3. Chapter 4

Chapter 6

Discussions

Since this was not a research project *per se*, there have not been any significant findings yet, other than the fact we have thoroughly researched this domain and are trying to build something that might be truly useful for a lot of educational institutions and individuals. We hope to keep working on this and improve each part of our product and achieve best-in-the-field results with our specifically tailored model.

There has been a lot of learning opportunities and I have been engaged in areas which was possible only because of working in such a small team. At various points of the internship I had to take on the role of a Deep Learning Specialist, Data Engineer and Full Stack developer over the course of these few short months.

I also came to realise how much work goes into making a piece of technology robust so that it holds up to stress testing. Working on these things were gruelling and boring and at times frustrating but nonetheless paramount to the designing of the product.

This internship help me to appreciate and explore in my own small way all the facets of the software development procedure in the field of Machine Learning and AI and as such has been a tremendous learning experience for me.

I have come to truly believe in the product that we are developing and my teammates who helped us achieve our vision for this product in such a short time.

Chapter 7

Conclusions and Scope of Further Work

Most of what needs to be said has been said, to bring it all together the scopes for future work include the following.

1. Create and include the data from the arXiv repository in order to boost our model further.

2. Carry out a thorough inspection of the transcripts generated by our model using the transcript editor tool to understand where exactly our model fails to perform.

3. Integrate a `Firestore` database to handle our data in an efficient and robust fashion.

4. Implement the CD/CI pipeline for continuous improvement and real time upgradation of our Jarvis models.

5. Carry out a comparative analysis to find out which model out of those providied by Nvidia Jarvis is the correct match for our use case.

6. Improve the front end UX to increase ease of use by implementing and integrating the feedback received from various sources.

7. Set up permanent hosting for our service so that potential clients can experience our product themselves and decide if its worth paying for.

8. Integrate various quality of life features such as email verification and more social logins using `Firebase`.

As you can see there is still much work to be carried out, and we would need more people to help us out. We will gladly accept people from our University to work on some of the pending issues if interested.

I would also like to thank our University and all the wonderful professors I have had the fortune of studying under who equipped me with the knowledge necessary to tackle this task and I hope that this report holds up to their scrutiny.

Bibliography

[1] Lara Alcock, Matthew Inglis, Kristen Lew, Juan P. Mejia-Ramos, Paolo Rago, and Christopher J. Sangwin. Comparing expert and learner mathematical language: A corpus linguistics approach. 12 2016.

[2] William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. Listen, attend and spell. *CoRR*, abs/1508.01211, 2015.

[3] Kenneth Heafield. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics.

[4] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., USA, 2009.

[5] Samuel Kriman, Stanislav Beliaev, Boris Ginsburg, Jocelyn Huang, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, and Yang Zhang. Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions, 2019.

[6] Jason Li, Vitaly Lavrukhin, Boris Ginsburg, Ryan Leary, Oleksii Kuchaiev, Jonathan M. Cohen, Huyen Nguyen, and Ravi Teja Gadde. Jasper: An end-to-end convolutional neural acoustic model, 2019.

[7] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):926–932, 1993.

[8] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE*

International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 5206–5210, 2015.