Implementing a distributed approach for speech resource and system development

By

Nkadimeng Raymond Molapo

23380845

Dissertation submitted in partial fulfilment of the requirements for the degree

Master of Engineering (Electronic and Computer)

at the

Potchefstroom Campus

of the

North-West University

Supervisor: Professor E. Barnard

May 2014

ABSTRACT

The range of applications for high-quality automatic speech recognition (ASR) systems has grown dramatically with the advent of smart phones, in which speech recognition can greatly enhance the user experience. Currently, the languages with extensive ASR support on these devices are languages that have thousands of hours of transcribed speech corpora already collected. Developing a speech system for such a language is made simpler because extensive resources already exist. However for languages that are not as prominent, the process is more difficult. Many obstacles such as reliability and cost have hampered progress in this regard, and various separate tools for every stage of the development process have been developed to overcome these difficulties.

Developing a system that is able to combine these identified partial solutions, involves customising existing tools and developing new ones to interface the overall end-to-end process. This work documents the integration of several tools to enable the end-to-end development of an Automatic Speech Recognition system in a typical under-resourced language. Google App Engine is employed as the core environment for data verification, storage and distribution, and used in conjunction with existing tools for gathering text data and for speech data recording. We analyse the data acquired by each of the tools and develop an ASR system in Shona, an important under-resourced language of Southern Africa. Although unexpected logistical problems complicated the process, we were able to collect a useable Shona speech corpus, and develop the first Automatic Speech Recognition system in that language.

Keywords: automatic speech recognition, smart phones, transcribed speech corpora, under-resourced language, Google App Engine, data verification, Shona.

ACKNOWLEDGEMENTS

This research was performed in the Human Language Technologies (HLT) Research Group of the Meraka Institute in conjunction with North West University. It was inspired and guided by Etienne Barnard, for the past two years.

I would also like to thank:

- Febe de Wet, my colleague and mentor at the CSIR, Human Language Technologies Research Group for her tireless work and dedication to oversee the project to its conclusion.
- Nic de Vries, who guided and provided initial conception and massive technical input to the project.
- Neil Kleynhans, for transferring his ASR knowledge and skill to the advancement of the overall project.
- It is a pleasure to thank Pedro Moreno for planting the seeds that produced this research, and financial support through a Google Research Award is also gratefully acknowledged.

Finally, I would like to thank my family and friends for adapting their lives to suit my working hours.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

The high levels of illiteracy in the developing world, especially on the African continent, have proven to be a significant obstacles to economic development of many third-world countries. The use of speech technology as a significant tool to bridge this gap has been proposed by several authors [1–3] and initial work along these lines has been encouraging (see Chapter 2). The initial need to develop these systems for a new language was usually driven by scientific interest – often from expatriate scientists from developing countries who come into contact with advanced speech research in the first world. However, such tools require certain resources that are scarce or non existent in countries that are still in the process of development.

Among the many different languages spoken around the world, only a small number can be classified as well-resourced. For our purposes, the languages that do not have transcribed speech data are classified as under-resourced, despite the fact that some of these languages have millions of native speakers. The reasons for this can range from most native speakers having no interest in speech technology to accessibility problems because the native speakers live in remote areas; most commonly, however, economic issues determine the extent of resources available in a given language. Corpus development is typically quite expensive and these expenses generally prevent resource collection unless there are sufficient commercial reasons to justify the development of language technologies using the collected resources.

The range of applications for high-quality automatic speech recognition (ASR) systems have grown dramatically with the advent of smart phones, in which speech recognition can greatly enhance the user experience. Currently, the languages with extensive ASR support on these devices are languages that have thousands of hours of transcribed speech corpora already collected. Developing a speech

system for such a language is made simpler because extensive resources already exist. However for languages that are not as prominent or under-resourced, the process is more difficult. Many obstacles such as infrastructure, internet reliability and cost have hampered the progress in this regard, and various separate tools for every stage of the development process have been developed to overcome these difficulties.

The approach we explore in this study is to combine these partial solutions. This process includes creating new tools and incorporating existing ones to develop an end-to-end ASR system in typical under-resourced conditions. For the current work, we focus our attention on the Shona language, which is a typical widely-spoken but poorly-resourced language in Southern Africa.

1.1 MOTIVATION FOR RESEARCH

The foundation of most ASR and text-to-speech (TTS) systems is the availability of sufficient clean text and speech corpora. Most languages in developing and underdeveloped countries do not have the luxury of having such resources. Sixty percent of the world's population speak only about thirty of the 6900 living spoken languages, as native or second language; almost all the remaining languages are plagued by limited speech resources.

The emergence and use of hand-held devices for speech data collection promises to adjust this imbalance, for the case of speech data collection. Building on initial developments at Google [4], researchers at Meraka Institute have released an open-source tool, Woefzela [5], that significantly assists in the development of speech corpora using such devices. However, these devices have limited storage and processing capability; thus, the collected data needs to be combined and stored on a centralized location such as a server or cloud for further processing. This is a tedious and error-prone process with Woefzela, posing a challenge for both storage and distribution. Because the data collected during the recording sessions need to be manually transferred from the on board SD card and uploaded to a centralised location. It is this challenge that motivated us to develop a dynamic end-to-end system for speech resource collection. The major motivation for this project is to develop an efficient and cost effective way to collect text and speech corpora and combine them into a format that can be used for training high quality ASR systems. A distributed speech system of this nature enables speech-driven applications for under-resourced languages to be realized through information sharing, scalable data protection and combining different tools for rapid ASR development.

The work presented in the study also extends research done on the collection of text data from the World Wide Web. For the purposes of this project, a language has to have data on the internet. Fortunately, a substantial number of the under-resourced languages do have a significant presence on the internet. The internet sites can be crawled to retrieve the contents of the web pages, and the data can then be cleaned through suitable preprocessing stages to serve as general text corpora. Rapid

Language Adaptation Toolkit (RLAT) [6] is utilised for this purpose. RLAT permits speech system developers to rapidly collect text data from the internet using web crawlers and web robots. RLAT can also be used for speech data collection, for the development of ASR and TTS systems. However, that functionality requires that the audio data be recorded over the internet, which is often not feasible in developing countries, where sufficient internet connectivity is generally an issue.

1.2 SCOPE AND CONTRIBUTIONS

The main objective for this study is to develop and integrate several tools to enable the end-to-end development of an Automatic Speech Recognition system in typical under-resourced conditions. For the current work, the system focuses mainly on ASR resource collection and system development. Resource collection is subdivided into text and speech resource collection. For text resource collection, we explored an inexpensive and rapid way to harvest text data over the internet. RLAT removes the burden of text data preprocessing by performing language independent text normalization to serve as general corpora. This process include the removal of hypertext mark-up language (HTML) tags, foreign-language content and various forms of punctuation. For the specific purpose of ASR corpus development, suitable prompting material can be extracted from such general corpora. We perform various experiments to remove the high contents of English text in the crawled data.

Because of the distributed nature of the system, we incorporated Google App Engine (GAE) [7] as a second tool into the end-to-end system. GAE is used as a data repository and an environment for cloud computing. It also hosts the web site around which the project is centred. GAE provides all the benefits that come with the Google platform to the user. The data uploaded to the GAE is validated by comparing checksums of the file from the source location to that on the server to complete the upload.

The system also provides an on-line prompt verification through a web interface. It allows users to verify and generate a text file that is used for recording. For the verified prompts to be recorded, they need to be downloaded from the Google App Engine server. This prompted the development of an application that facilitates the interconnection between the server and the recording tool. The tool runs on the Android operating system. It is open source and for this reason it is low cost and will allow smooth system extension.

To complete our investigation, we evaluated the speech data collected using Woefzela. However, before experiments could be conducted, the data had to go through quality control measures both on the phone [8] and during off-line post processing [9]. We then tested and validated the grapheme based ASR end-to-end system using the Shona language. We further investigate and compare the accuracy of the system by removing English content from the overall data.

1.3 OVERVIEW OF THESIS

The framework of the thesis is as follows. In Chapter 2 we present the literature behind the collection of speech data. We distinguish between *established* and *emerging* strategies, with our attention being limited to text and speech data collection. The chapter also describes the literature behind the development of an ASR corpus. The end-to-end system is represented in Figure 1.1 below.

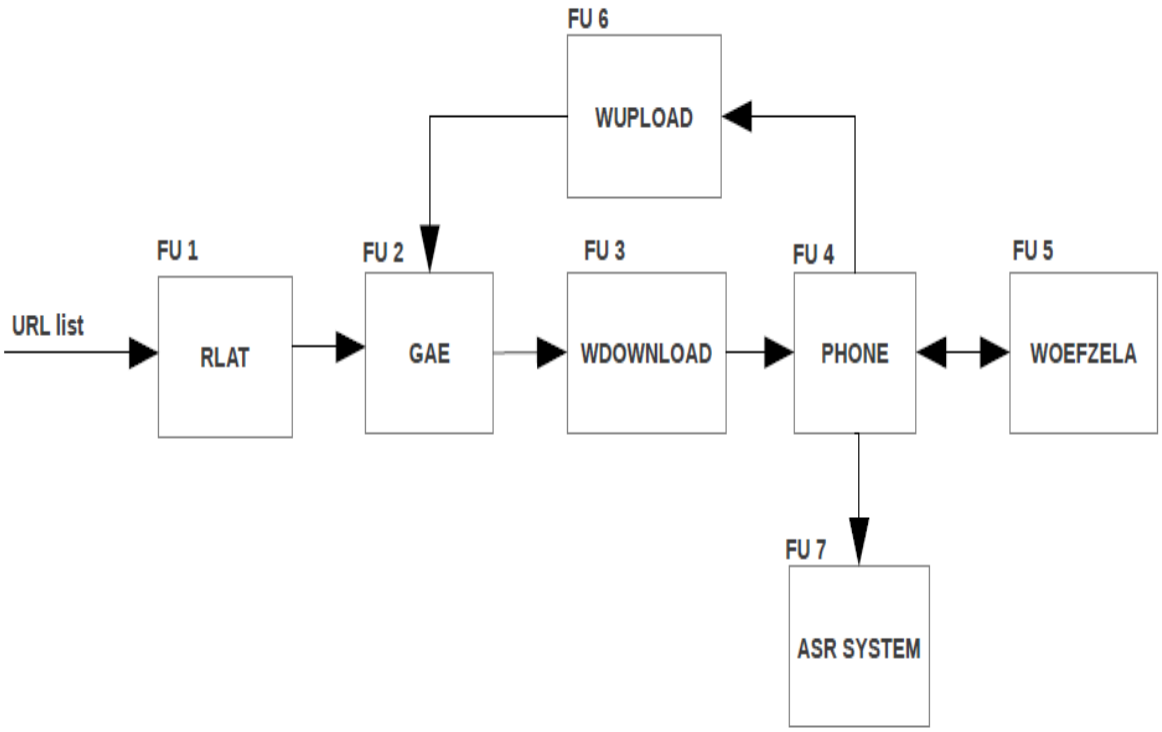

Figure 1.1: *A schematic diagram of the functional unit (FU) decomposition of the end-to-end system.*

For our purposes, FU1 (RLAT), discussed in Chapter 3, is used to search the World Wide Web to collect text data for a particular language. The text in the file is segmented into tri-gram prompts and the file is uploaded through a web interface to Google App Engine, represented by FU2 (GAE) which is discussed in more detain in Chapter 5. Prompts can be downloaded for recording from GAE through an Android application called WDownload. FU3 is described in more detail in Chapter 4. FU4 can be any smart phone that runs Android Operating System (OS). For prompt recording, FU5 (Woefzela) is incorporated into the system and also runs Android OS. FU5 and its functionality is briefly explained in Chapter 6. After the recording process is complete, FU6 (WUpload) fetched audio files and meta data from the phone and uploads them to FU2.

CHAPTER TWO

BACKGROUND

2.1 INTRODUCTION

This chapter provides background information with regard to the main topics discussed in subsequent chapters. Section 2.2 provides an overview of various approaches used to collect speech system resources. Section 2.2.1 focuses on the different strategies used to collect text data, and Section 2.2.2 describes in more detail work done in collecting speech data. Thereafter, Section 2.3 describes the development of the resources that make up the ASR system, and Section 2.4 provides a summary and concludes the topics discussed in the chapter.

2.2 LANGUAGE RESOURCE COLLECTION

2.2.1 TEXT DATA COLLECTION

The development of speech systems requires a significant amount of transcribed speech corpora, for the construction of prompts, language models and pronunciation dictionaries. For languages that are regarded as under-resourced, it is often a very difficult task to acquire such text. In this research, for a speech system to be developed for a certain language, it is required that the language of interest at least have a standardized orthography, and some presence on the World Wide Web. Our goal is to collect text data of sufficient quality to produce accurate overall system performance, which implies that the data does not need to be perfectly grammatical or monolingual.

2.2.1.1 ESTABLISHED METHODS

Wall Street Journal

In Paul *et al.* [10] emphasis is put on matching the type of text a developer is going to use to train the system to the testing text data. Here the developers acquired a large text database from the Dow Jones, Inc. The data was distributed in a form of tertiary storage. This concept is beneficial in that it does not require developers to use data crawling and it is subject-specific. Even though the text data provided was relatively clean (it did not, for example, include any HTML tags) it still required certain levels of preprocessing to be used.

The research conducted by Paul *et al.* focuses on two main modes, the verbalized mode with punctuation and without punctuation. The idea was to produce a version that would be read by the speaker as prompt, and a version used to train and score the system. Our focus is mainly on the former. The preprocessing is implemented by labelling each of the sentences, which makes sense if the developer wants to keep track of them. Next, spelling errors were located and fixed. If a developer is dealing with an under-resourced language appropriate resources such as spell checkers would not be available to perform such tasks. Hence, a linguist or a native speaker of that particular language may need to be asked for assistance.

Paul *et al.* also convert numbers into orthographics. For multilingual and multicultural developing countries, the numbers may be called out in a totally different language or more specifically in the lingua franca. For this reason it is sometimes better to leave the numbers as they are to get an idea of how the natives call them out. Then the processor removes punctuations from the given text; this is a delicate process in that the normalization of the text, when performed correctly, should not influence the meaning of the words. Paul *et al.* highlights the importance of case normalization if the text is going to be used for case-sensitive or case-insensitive recognition.

Different criteria may used in selecting text to be recorded by speakers. Paul *et al.* used an automated ”quality filtering” which filtered out unreadable sentences and paragraphs of the recorded text only. The filtering extended to limiting the paragraphs to a minimum of three sentences and a maximum of eight sentences. This is to construct easy to read and non-offensive prompts. Paul *et al.* performs a final check on the data by involving human intervention to improve the prompt quality.

Google’s Approach

There are many routes that speech system developers take in order to generate prompts that best suit their systems. This is dependent on the kind of resources that are available during that period. In Hughes *et al.* [4] a large pool of Google queries were harvested from the search engine and utilised

for prompt design. This approach enables the developers to have a large coverage of the current language that is used. Another advantage to this approach is that it saves the developer the process of having to remove Hypertext Mark-up Language (HTML) tags from the text. The queries are of a short length thus minimizing the need to perform sentence segmentation. Having mentioned this, the uncertainty of harvesting prompts in this manner is that due to the large variety collected, a lot of language polarization can be encountered.

Hughes *et al.* performed text data collection for a specific purpose, in that it harvested Google search queries for Google voice search. As mentioned in [4], this approach creates a similar distribution between the text data and voice search queries. This in effect improves the accuracy of the speech recognition system.

2.2.1.2 EMERGING METHODS

WGET method

The research described above was initiated with some form of text corpus already available. Most languages do not have this luxury: the speech system developers have to start with very little or no data, no tools to start text data harvesting. This scenario is common to developing environments, specifically on the African continent. The use of search engines to collect large text data is one alternative that has proven to work tremendously well for such environments. With this in mind, with the enormity of the data available on the internet, it is very difficult to locate sufficient sites for under-resourced languages to acquire the needed text data.

Kivaisi *et al.* [11] describes the use of very limited resources to collect a large amount of data from the internet. The idea was to use standard command-line tools to retrieve language-specific data – in particular, the *wget* command, which uses HyperText Transfer Protocol (HTTP) request protocols. Since the developers were unaware of the location or complete Universal Resource Locator (URL) of the sites, they used a search engine to search for common words in the language of interest. This then yielded links to pages that contained those words. The URLs for these pages were used as input to the bash command to download the content.

Text data collected in this manner goes through three main stages which are the web crawling that is described above, cleaning the text and normalizing the text. The data in [11] was cleaned by first filtering HTML tags that did not contain data of interest. Then the punctuation was removed using on line scripts. Most of the data that is crawled under these environments has a heavy influence of a world language such as English, as in Kivaisi *et al.* or French. A dictionary is used as a lookup to ignore sentences with more than half the words in such an identified “unwanted” language.

The normalization process is particular to a language and different regions where different dialects of a language are spoken. Words from English or French are borrowed most times to normalise characters that are not found in that particular language. This is one of the least complicated and cost effective ways of acquiring a clean text corpus. It can be implemented by developers without an in depth knowledge in the field of speech system development.

Rapid Language Adaptation Toolkit

Most text data collection efforts are made for the purpose of developing big language models with millions of words. Before the emergence of text data collection tools, to undertake such a task the developer needed to have extensive knowledge in the field. A tool like Rapid Language Adaptation Toolkit (RLAT) has been developed to bridge the gap between experts and beginners. Developed by Karlsruhe Institute of Technology, it enables speech system developers to collect text data from the web by using crawlers or web robots.

Schlippe *et al.* [12] describe the ease of use of RLAT. The user interface allows the user to manually normalize the crawled sentences. The tool also provides language independent text normalization, which involves the removal of tags, empty lines, punctuations and case normalization based on statistics. It also provides language dependent text normalization which is specific to a certain language. This type of normalization requires a user or native speaker of the language to give input. An attractive feature about a tool of this nature is the abstraction of the technical aspects from the user. The user can perform system evaluation using the tool itself. This is done in an efficient manner with very little cost incurred.

2.2.2 SPEECH DATA COLLECTION

Throughout the development of speech recognition systems, various methodologies of collecting clean speech data have been explored. Monitoring these developments significant improvements have been gradually made up to this point. The subsection describes these exciting advancements from the past few decades to the era of mobile smart phone technology.

2.2.2.1 ESTABLISHED METHODS

A Professional Studio Set up

Studio recordings have long been used for speech data collection, such as the venerable TIMIT [13] and WSJ corpora [10]. As a more recent example, Vlaj *et al.* [14] describe the acquisition of Slovenian Lombard Speech Database which employed the use of a studio. The deployment of such a studio provides the user with a more controllable recording environment. It is advantageous in that there is very limited uncontrollable noise factors that may adversely affect the quality of the overall recording. The environment can also be set up to simulate various noise backgrounds that the recognition system might need to be utilised. In the case of Vlaj *et al.* two types of controlled noise were utilised which were babble and car noise. The environment can also be equipped with a speech annotator tool that presents the orthographic transcriptions of the audio while its is being played back. This type of set up requires a substantial amount of capital to employ and maintain. The other factor is that it is generally in a single place and cannot be moved, thus users need to travel to the studio.

Telephone-Based speech collection

Another approach that has long been used to aid the effort of speech data collection is the use of telephone-based systems. This approach traditionally used a land line to gather either spoken dialogues between end users, or single-user responses to automated prompts. With the emergence of mobile telephones which are carried by large percentages of many communities, telephone-based data collection proves to be a viable option as discussed in Gee *et al.* [15]. Lerer *et al.* [16] explores interactive voice response (IVRs) systems as a way to collect speech data. The idea is to guide the user through interactive voice commands over the telephone. The process in itself does not require users to travel to studios to do recordings, so it is very convenient. One of the main concerns about such a system is for users not to use their own airtime to pay for the calls. To circumvent this, Lerer *et al.* informs the participant twenty four hours in advance that they will receive a call from the IVR to do the survey. This way the participants know when to expect the call and it is not charged from their account. Telephone-based systems are very beneficial, more so in communities that have a high level of illiteracy. With this being said, there are several precautions that need to be taken into account when setting up a telephone-based capturing and storing speech system. As described in de Wet *et al.* [17], these include the type of compression to use, format, the users' ability to respond to prompted commands and the system set up.

There are numerous disadvantages that accompany such a system. As highlighted in Gee *et al.*, recording done via mobile telephones requires it to be switched on at all times to avoid a call being missed. The concern especially in developing countries may be that of mobile network coverage to

get good quality reception and hence clear enough recording. Since most telephone-based systems are not supervised, the collected data is unpredictable and may yield in unexpected results.

Amazon Mechanical Turk

Amazon Mechanical Turk (AMT) is a popular online crowd-sourcing site by Amazon. The idea behind the site is to put tasks called Human Intelligent Tasks (HITs) on the website by ”requesters”, then have participants or ”workers” look through and perform the tasks. The participants get compensated based on the quality of the work submitted Lane *et al.* [18]. Unlike most web-based speech collection systems, the quality of the work can be evaluated and participants paid accordingly. The quality of the speech data collected through crowd-sourcing was found to be of similar quality to the speech data collected through professional means [19]. For this reason AMT has proven to be a reliable means of collecting speech data, but the developed-world focus is clear.

2.2.2.2 EMERGING METHODS

Speech data collection via the Web

One obstacle that is encountered during the process of collecting speech data is the issue of mobility. The introduction of temporary studios has made it less difficult to acquire speech data but this does not completely eradicate the problem. With high internet penetration, especially in the developed world, the idea of having participants record data via the web in the comfort of their own homes can be realized. This is often used in conjunction with crowd sourcing. Crowd sourcing is implemented in a variety of ways which are explained in the next paragraphs.

Schultz *et al.* [20] describe a web-based tool to rapidly collect transcribed speech corpora. The recording process can take place from the comfort of the donor’s own home. *SPICE* provides an interactive user interface to guide and facilitate the recording process. The recorded wavefiles are verified and uploaded to the server. *SPICE* is an example of an end-to-end speech system. It significantly reduces the time to develop and evaluate a fully functional high quality speech system.

Siebert *et al.* [21] elaborates on one variation which is to develop a web-based gaming environment which interacts with the user through voice commands. The commands or phrases are then saved on to the server. The method gives a distributed collection of speech data under different acoustic conditions. Of course the drawback might be that there is no real guarantee that the same person is not doing most of the recordings. Gruenstein *et al.* [22] employs an educational game, *Voice Scatter*, to collect orthographically transcribed continuous speech data. It uses flashcards to assist users to match the term with their definitions. Gruenstein *et al.* overcomes the problem of data labelling faced

by Siebert *et al.* through narrow-domain speech recognition, confidence scores and game constraints. The games are restricted to small domains to achieve high recognition results. The drawback of this type of method is that it requires reliable internet connection which is often an issue in developing countries.

Freitas *et al.* [23] describes a client/server architecture aimed at employing crowd-sourcing to donate speech through a quiz game. It uses a platform called *YourSpeech* which can be used on a desktop computer at very little cost to the user. One limitation of such a system is that it can rarely be used by people who are illiterate. To overcome this limitation, the platform employs a Text-To-Speech (TTS) system to guide such users through the recording process. The TTS functionality is a welcome feature when operating the system in the developing world. Most of the tools developed in this manner require specific platforms to run on. For environments with limited budget and a limited variety of available hardware, this could be a huge obstacle.

Draxler *et al.* [24] highlights the importance of platform independence through the use SpeechRecorder. It is a Java-based tool and therefore runs in any web browser. The tool is divided into the idle, pre-recording, recording and post-recording phases. To cater for different operating situations, *SpeechRecorder* allows plain text prompts, image prompts and audio prompts. The tool is multi purpose and cost effective, which makes it ideal for ASR system development for a new language. This type of speech data collection may not be practical for developing countries due to poor connectivity, high bandwidth requirements and high internet rates.

Mobile Studio Set up

A typical mobile studio setup is developed from off-the-shelf components. These mobile studios are constructed with the idea of decreasing the cost compared to large professional studios. The quality of the recordings is not comparable to that of professional studios. In Burnham *et al.* [25], one of the tools called the *Black Box* in *AusTalk* was developed with off-the-shelf elements which made it economical to construct. On a larger scale, *GlobalPhone* which is described in Schultz *et al.* [26] was used to collect very large amounts of speech data. In data collection ventures that are embarked on, field workers have to travel to remote locations where native speakers are to perform recordings. For this reason, the mobile set up is more practical when compared to a large professional studio.

2.3 SYSTEM RESOURCE DEVELOPMENT

2.3.1 PROMPT DESIGN AND GENERATION

The process of prompt design is an important step when creating an ASR system: the manner in which prompts are generated can greatly influence the accuracy of the system. Different methods and algorithms for generating prompts have been implemented in various studies. Important factors that need to be kept in mind are the domain in which the prompts will be used, acoustic patterns in a language and phonetic coverage of the prompts.

For specific domains, the prompts are generally required to cover the most frequently used words in that language domain. This is achieved by crawling text data and performing a word frequency count. A greedy algorithm is then used on the list to generate prompts. As Oliveira *et al.* [27] highlighted, the approach may yield unexpected results. Oliveira *et al.* selected the prompts to cover the most frequent word bi-grams and tri-grams. The system created from this approach is expected to give more accurate ASR results than one that is created for an open domain.

For open domain, a complete coverage cannot be achieved since a language can have countless number of words. Because of this, Oliveira *et al.* represents the prompts in three levels syllables, tri-phones and di-phones. This makes it easier for readers and it is beneficial for verifying purposes. For conjunctive languages, tri-grams work well to limit the length of the sentence but still get sufficient phone coverage.

2.3.2 PROMPT VERIFICATION

Our prompt selection process uses statistical algorithms that do not perform spell checking. For this reason, before the recording process could take place, the prompts have to be verified. This is to ensure that they do not contain spelling errors or inappropriate content such as abusive or obscene phrases. For under-resourced languages the luxury of a spell checker to correct the text in the prompts is not available. The verification process under these conditions requires manual verification from linguists or native speakers of the particular language.

2.3.3 PROMPT DATA RECORDING

At this stage of speech data development, respondents are recruited to perform recordings. The generated prompts are recorded using different forms of hardware under various conditions. The process may be conducted by field workers or a professional producer and sound engineers in a professional studio.

2.3.4 SPEECH ANNOTATION AND TRANSCRIPTION

In the process of collecting data for ASR development, one option is to collect text data and use the text to perform recordings; the prompted text can therefore be considered to be (approximate) transcriptions of the speech. However, if natural speech is used, a separate transcription process is required. In the presence of multiple speakers (e.g during radio interviews or spontaneous conversations), phenomena such as speaker overlap may increase the complexity of the transcription process. Barras *et al.* [28] addressed this problem by presenting a tool called *Transcriber* for creating speech corpora. Transcriber is well suited for long duration continuous news broadcasts. The tool is easy to use, low cost and intuitive.

2.4 CONCLUSION

In this chapter we provided an overview of the literature dealing with different strategies used to acquire transcribed speech corpora. We explored two separate avenues which were text and speech data collection. The literature includes both established and emerging strategies for each avenue. In Section 2.2.1, the research led us to conclude that we can incorporate an emerging text data collection strategy, embodied in the RLAT toolkit, which is aligned with our aim. Section 2.2.2 gave an overview of the speech data collection strategies in recent years, and suggested the use of a mobile-phone based strategy for our work. Though other strategies explored could provide better audio quality, they would not be practical under the conditions in which our end-to-end system is going to be operated. These literature findings provide the starting point for our own development.

CHAPTER THREE

TEXT DATA COLLECTION AND PREPARATION

3.1 INTRODUCTION

For the purposes of text data collection, a tool called Rapid Language adaptation Toolkit (RLAT) was incorporated into our end-to-end system. RLAT was developed by Karslruhe Institute of Technology (KIT). It allows speech system developers to rapidly crawl and clean text data from the World Wide Web. This Chapter details the different stages used to collect internet data. Section 3.2 describes the initial preparation of the crawling process. Thereafter Section 3.3 discusses the text normalization measures provided by RLAT. Section 3.4 evaluates the crawled text data for our Shona test case and gives a graphical representation of the results.

Note that RLAT can also be used for speech data collection, for the development of automatic speech recognition (ASR) and text-to-speech (TTS) systems. However, that functionality requires that audio data be recorded over the internet, which is often not feasible in developing countries, where sufficient internet connectivity is often an issue. We therefore only utilize the text-collection capabilities of RLAT in our development.

3.2 TEXT CRAWLING

To initiate the crawling process, a list of the 100 most frequently used words in the target language (Shona in our case) is compiled and sent to the RLAT team at the Karlsruhe Institute of Technology, in order to create a place holder for the target language on the RLAT web site. Next, a list of URLs pointing to websites in the target language is uploaded to the site. RLAT then crawls the internet, starting from those URLs, and collecting documents that contain a sufficient concentration of the

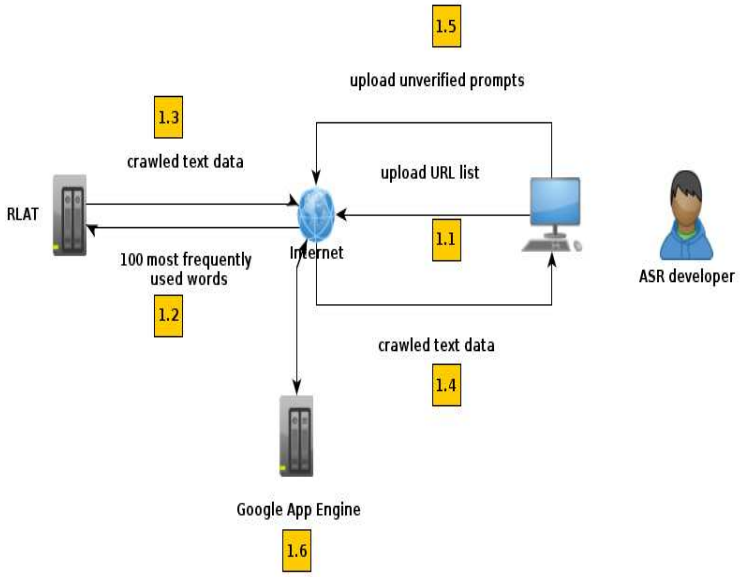

Figure 3.1: *A schematic diagram of the RLAT interaction process.*

100 common words. The process flow of the system's interaction with RLAT is depicted in Figure 3.1.

The crawling process may take several days or weeks to retrieve all the sites. For direct and robust web crawling, a text file with a list of eight URLs, shown in Table 3.1, was uploaded to the RLAT website, to initialize the crawling process. A total of 19 Megabytes of data was collected. The data contained approximately 267 000 sentences, which included over 2.6 million word tokens. The text from the web sites was found to contain numerous characters and words that needed to be cleaned and normalized.

Order	**URL**
1	http://mudararatinashemuchuri.blogspot.com
2	http://vashona.com/shona-news
3	http://www.watchtower.org/ca/jt/
4	http://www.kwayedza.co.zw/
5	http://www.voanews.com/shona
6	http://www.viva.org/downloads/pdf/wwp2012/
7	http://faraitose.wordpress.com
8	http://16dayscwgl.rutgers.edu

Table 3.1: *Shona URLs used to initiate crawling.*

3.3 TEXT NORMALIZATION

RLAT provides a data clean-up mechanism that removes HTML tags and punctuation marks and converts the text to lower case. This is termed language independent text normalization [12]. RLAT also provides the capability to perform language dependent text normalization. This process involves the removal of characters not occurring in the target language, digit normalization, converting text to lower case and refined punctuation mark removal. The process requires input from a linguist or a native speaker of the language. Since we operated in under resourced conditions, we did not have the luxury of having a large Shona words list. However, we assumed that words that were non-English were Shona until the prompt verification stage where a native speaker performed manual verification.

3.4 TEXT DATA EVALUATION

A graphical representation of the amount of English-to-Shona text data is shown in Figure 3.2. The numerics were left unchanged, to enable us to hear how native speakers call them out – we have previously found that numeric quantities in Southern African languages are often pronounced in English [29].

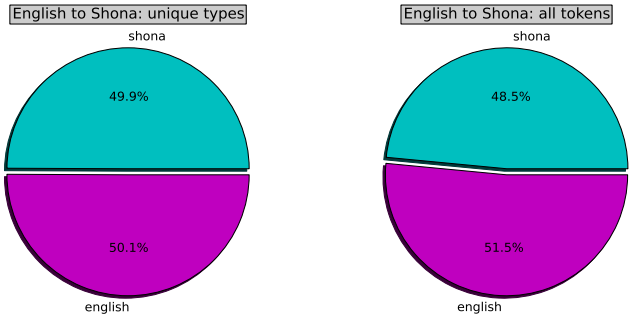

Figure 3.2: *A diagram of initial English to Shona text ratio.*

Figure 3.2 shows that even though the text was crawled from Shona web sites, the data was found to have a large portion of English content: for both word types (i.e each unique word is counted separately) and word tokens (i.e each word counted regardless of repetition) the ratio of English to Shona was approximately 1:1. Although some English data would be acceptable for our Shona development process, this ratio is too high - we therefore needed to perform additional processing. To control the amount of English text in our corpus, a list of English words was acquired by combining the CMU [30], Lwazi [31] and NCHLT English [5] pronunciation dictionaries. The list is used as a lookup table to remove sentences that contain English words only.

The list consisted of 65 thousand words, mostly in the South African dialect of English. Sentences that had a mix of English and Shona were included in the corpus, since such code-switched speech is commonly found in ASR applications in under-resources languages. Figure 3.3 shows the ratio of English to Shona text after the sentences were removed. Around 14 % of the words are now in Shona - a more acceptable starting point for corpus development.

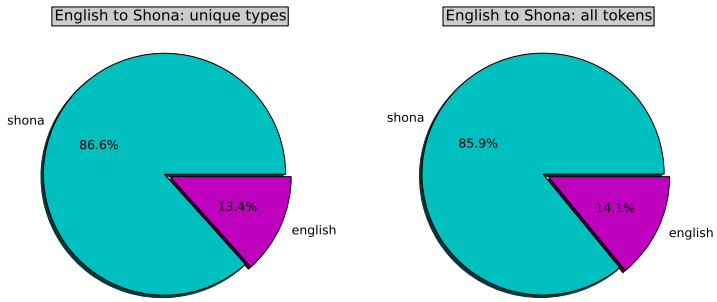

Figure 3.3: *A diagram of filtered English to Shona text ratio.*

The crawled text data is then returned to the developer, who uses a Perl script to segment the text into three-word prompts. This is because the Shona language is morphologically complex (agglutinative) language with a conjunctive writing style, its words tend to be long. This made the prompts not too long to read but still semantically meaningful.A list of five hundred prompts was generated and prepared to be recorded. The Perl scripts use a greedy algorithm which does not perform any spelling checks. Before the recording process, the prompts need to be verified. The verify process is explained in more detail in the next chapters.

3.5 CONCLUSION

We have managed to collect clean text data from the internet for a typically under-resourced African language. The process was efficient and cost effective. The main motivation for choosing RLAT was the ease of use and less reliance on internet connectivity when acquiring text data. However the text data needed a fair amount of post processing due to the amount of English found in the text. The process managed to collect sufficient text data to generate prompts that were used for recording which is the next stage of our ASR development.

CHAPTER FOUR

WDOWNLOAD : AN ANDROID TOOL

4.1 INTRODUCTION

The literature discussed in Chapter 2 highlighted the importance of acquiring clean text data rapidly and cost effectively. Once the text data has been collected, processed and segmented, it needs to be downloaded onto a device and prepared for recording. The download process is usually done manually by copying the files onto the secure digital (SD) card [5].

The goal of developing an end-to-end system is to make the complete process intuitive by easily combining different existing tools. However this required the development of new tools to interface existing tools. This chapter introduces a tool called *WDownload*. The aim of the tool is to facilitate and automate the prompt download process. The chapter starts by highlighting primary application requirements in Section 4.2. Section 4.3 discusses the design process followed. Thereafter we detail the software architecture in Section 4.4 and perform software testing. We conclude the chapter in Section 4.6 by discussing our findings and giving recommendations.

4.2 PRODUCT REQUIREMENTS

The concept behind WDownload is that it should be freely available to anyone who wants to use it independently or incorporate it into their system. For this reason the tool is released as open-source. This will allow easy extension and customizability. Currently, WDownload requires internet connection to communicate with the remote server using asynchronous HTTP requests. WDownload was designed to be easily portable and hardware-independent i.e the software must be able to run on different hand-held devices. It is required to fetch large text files from the cloud and save them in standard UTF-8 variable width encoding.

4.3 SOFTWARE DESIGN

The overall system was designed and implemented using the linear sequential model. This model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing and support [32]. These approaches are discussed in detail below.

4.3.1 DESIGN CONSTRAINTS

Even though the WDownload is hardware-independent, its operations are constrained to devices that run Android Operation System. The application was developed to work in conjunction with Woefzela [8]. WDownload is constrained to using the file tree created by Woefzela on the device. For the purposes of our project, the application is constrained to downloading only a previously verified prompt file.

4.3.2 CONCEPTUAL MODEL

The conceptual model for the tool was simple and forthright. In essence the tool was divided into three distinct operations. The system interaction of WDownload conceptual model and interaction is shown in Figure 4.1.

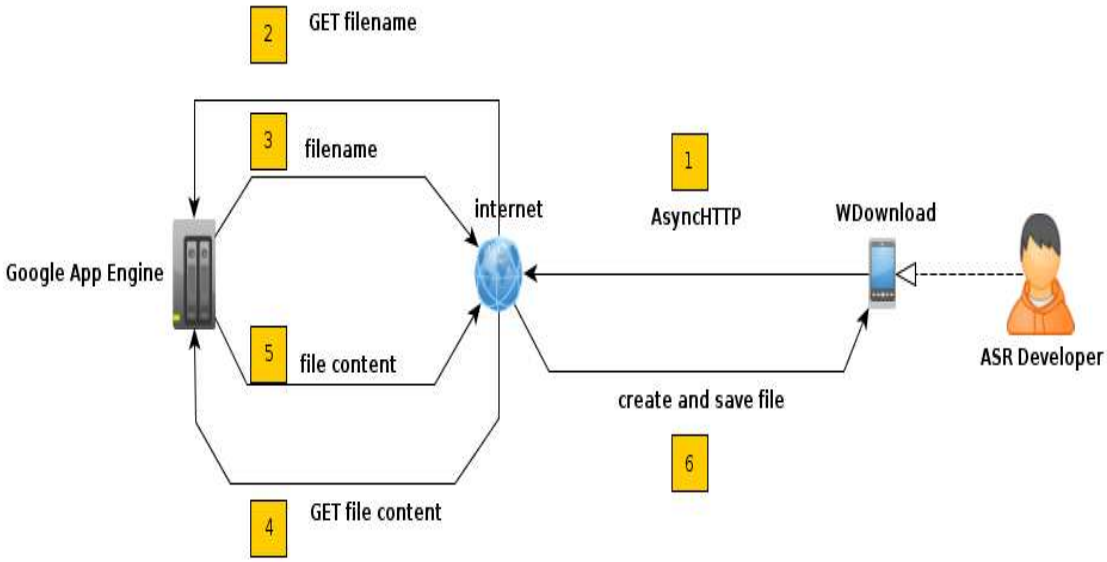

Figure 4.1: *A diagram of WDownload system interaction.*

The first operation that is performed by the application is to establish an hypertext transfer protocol (HTTP) request to initiate and close the connection. After the connection has been established the application issues an HTTP GET to retrieve and return the file ID to prepare for download. The application then issues an HTTP POST to the file name and the contents of the file from the server

database. These operations are performed on the recently verified prompt file. A new file is created with the contents retrieved form the server and saved on the SD card. The prompt file is loaded by Woefzela before the recording process commences.

4.4 SOFTWARE ARCHITECTURE

4.4.1 COMPONENT LEVEL DESCRIPTION

Figure 4.2 below shows an Android runtime OS components. This is the architecture around which an Android application is modelled. Android applications are written in the Java language. They run on the Dalvik virtual machine which is similar but not a Java Virtual Machine. Dalvik works well on mobile devices because it can run on slow CPUs,uses very little RAM and will run on OS without swap space. WDownload did not require a lot of resources to implement; the tool was constructed using one Activity application component

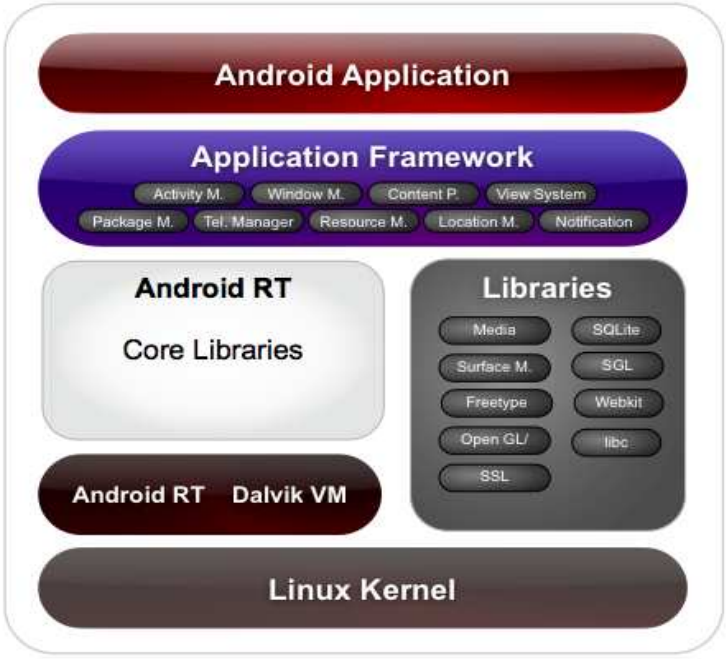

Figure 4.2: *A diagram of Android runtime OS components [33].*

4.4.2 SOFTWARE IMPLEMENTATION

As mentioned in Section 4.4.1, WDownload was constructed using only one Activity, the reason being that it was designed to perform limited tasks. An Activity is a separate focused operation that a user may perform. In particular, activities may be utilised for user interfaces for easy interaction with the user. Our design was a simple single activity application that was required to download only recently verified prompts at a click of a button. The detailed operations of these methods is described

below.

AsyncHttpClient:filename

filename creates an instance of *AsyncHttpResponseHandler()* post to the server which accepts the server URL as an argument. Method *onSuccess()* returns the filename from Google App Engine.

AsyncHttpClient:fileId

fileId also creates an instance of *AsyncHttpResponseHandler()* get to the server which accepts the server URL as an argument. Method *onSuccess()* returns the file ID from Google App Engine.

AsyncHttpClient:client

client creates an instance of *BinaryHttpResponseHandler()* which accept a complete server download URL and content-type as arguments. Hidden method *onSuccess()* receives the contents of the file from Google App Engine. The method creates a new file with the contents downloaded.

4.5 TESTING

The testing process was conducted on an HTC and Samsung hand-held devices for variability. To verify that the file downloaded is identical to the one on the server, a checksum is calculated and compared on both ends. If the checksum is the same, the file is saved on the SD card. However, if the checksums do not match an error message is displayed on the screen. The system only makes one attempt to download the file to prevent overhead. The application allows the user to view the download process by displaying tokens for every stage completed.

4.6 CONCLUSION

The chapter described the steps undertaken to develop a tool to facilitate a connection between two components of our end-to-end system. We have managed to develop an open-source tool called WDownload. The tool managed to automate the process of loading prompts onto the smart phone.

CHAPTER FIVE

COLLECTING AND STORING SPEECH RECORDINGS

5.1 INTRODUCTION

The emergence and use of smart hand-held devices as a method of speech data collection has rapidly advanced the state of speech data collection for under-resourced languages [4, 34]. These devices have limited storage and processing capability, thus the collected data needs to be combined and stored on a centralized location such as a server for further processing, this is often a tedious process. The data migration poses a challenge to data storage infrastructure development and maintenance. It is this challenge that has prompted the need to provide dynamic computing for speech resources.

Google App Engine (GAE) [35] is employed as the core environment for data verification, storage and distribution. The tool is used in conjunction with existing tools for gathering text data and for speech data recording. GAE is used as a data repository and an environment for cloud computing. It is also hosts the web site around which the project is centred. GAE provides all the benefits that come with the Google platform to the developer and user.

Another added advantage of incorporating GAE into our end-to-end system is to enhance system distribution. The tool allows concurrent users to execute different CPU intensive tasks without performance degradation. This project focuses mainly on three parts provided by GAE namely: the runtime environment, data store and scalable services. The GAE runtime environment is responsible for handling HTTP requests made through the web. GAE provides both Java and Python development environments. For this project, we use the Python environment that employs an open-source web application framework called Django.

5.2 SOFTWARE REQUIREMENTS

Due to the proposed distributed nature of our system, the application was required to handle concurrent users from different domains. Another requirement for the application was that it should be open-source. The application was also required to permit large file upload and download using a database. Users and system developers are required to have internet access to perform various tasks related to GAE. Because the application is web based, it can be run on any web browser on any machine regardless of the hardware or OS.

5.3 SOFTWARE DESIGN

5.3.1 CONCEPTUAL DESIGN

The conceptual stage of the system was designed with strict adherence to the scope and requirements of the project. The system was required to efficiently operate in two streams which were user interface via the web and data storage.

5.3.1.1 WEB INTERFACE DESIGN

The system's web interface was required to provide simplicity and efficient navigation. Additionally, it should be able to be accessed remotely via the World Wide Web by multiple users. The site should provide an interface that allows the user to upload a text file containing UTF-8 encoded prompts to the system's database. However the database only allows the original file and the recently uploaded file to be saved which allows minimum usage of the database. The tool should allow automatic data upload from smart phones via HTTP POST.

Another capability envisaged was to allow users to verify uploaded prompts on-line. The user would be given the luxury of choosing which prompt file to verify. Since there would only exist two prompt files on the database, the user would be given a choice between the ”original” or the ”recent” file. After the user chooses either one, a verify page would allow the user to select the prompts they wanted. The user would then provide a file name and save the file.

For added usability, the web interface should provide a navigation frame to permit users to switch between pages. Links to associated project documentation and tools should be easily accessible on the page. Lastly, a short description of the system should be provided to give the user an idea of what the website site does.

5.3.1.2 AUTOMATIC DATA UPLOAD

Collected data could be copied directly from the SD card to limit reliance on the internet (a significant concern in the developing world). However, to further enhance the end-to-end functionality of the system, data upload from mobile phones after recording should be automated. The collected data should be uploaded directly to the cloud as soon as internet connection is established. To ensure that the files would not be duplicated, a checksum is returned from the server and if it matches that on the phone, the file on the SD card is deleted. The uploaded data is stored in a blob-oriented database for easy retrieval.

5.3.1.3 DATA DOWNLOAD

Data saved in the cloud should be downloaded directly to the user's computer when specifically requested. A simple python script that could be executed on any Linux machine should also be provided for download from the website. To limit the use of resources on the server, the system should use task queues to download the data in the background. The data should be uniquely named by speaker to allow efficient statistical analysis.

5.4 COMPONENT LEVEL DESCRIPTION

The high-level operation of GAE is summarized in Figure 5.1 below. The figure highlights the main components that form the GAE architecture. When a developer registers an application on the GAE, they get assigned a domain name that uniquely points to their application. Frontends are the entry point to the architecture when a request is made. They ascertain the application for which the request is targeted by identifying the unique domain name. If an invalid URL is provided an HTTP 404 error response is given to the client. However, if the URL is valid, the Frontends redirect the request to the Static file severs. Static file servers provide a path to the application's static files and handle resources that do not change often [7].

A configuration file that contains valid patterns of the URL determines the application that is invoked (when a valid URL matches one of the patterns). The Frontends, which interface the user/client to the server-side or backend, send the client request to the application handler. Application handlers handle the load balancing and allocation of unused resources to requests. The application server allows application code to run in a runtime environment introduced in the beginning of the chapter. Other applications that were utilised in the implementation of the system are Datastore and Task queue. Both are explained in more detail later in the section.

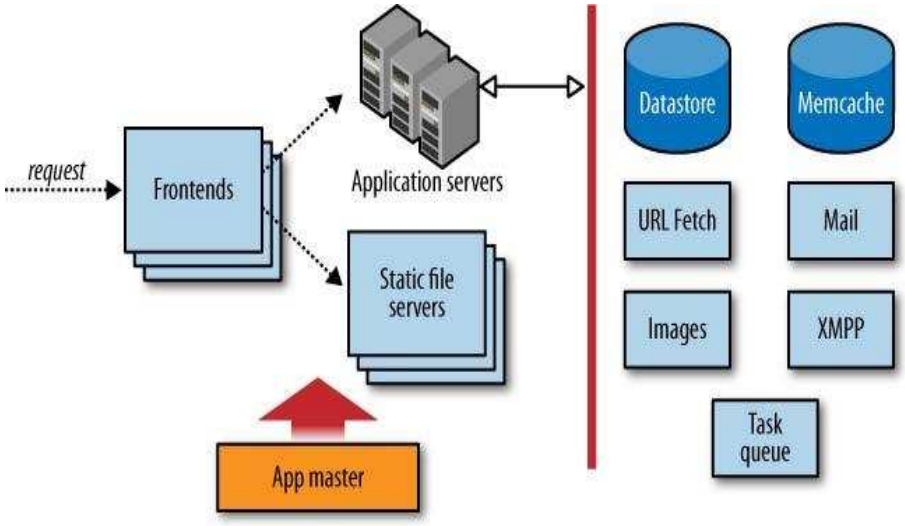

Figure 5.1: *A diagram of App Engine architecture [7].*

5.4.1 COMPONENT INTERACTION

As discussed in Section 5.1, our Python environment employs a web application framework called Django. Django allows Python developers to incorporate web applications using Hypertext Mark-up Language (HTML) scripting with their Python code. Component interaction is accomplished through the clicking of buttons and HTML links. Figure 5.2 shows how different components interact with each other to enable complete system functionality.

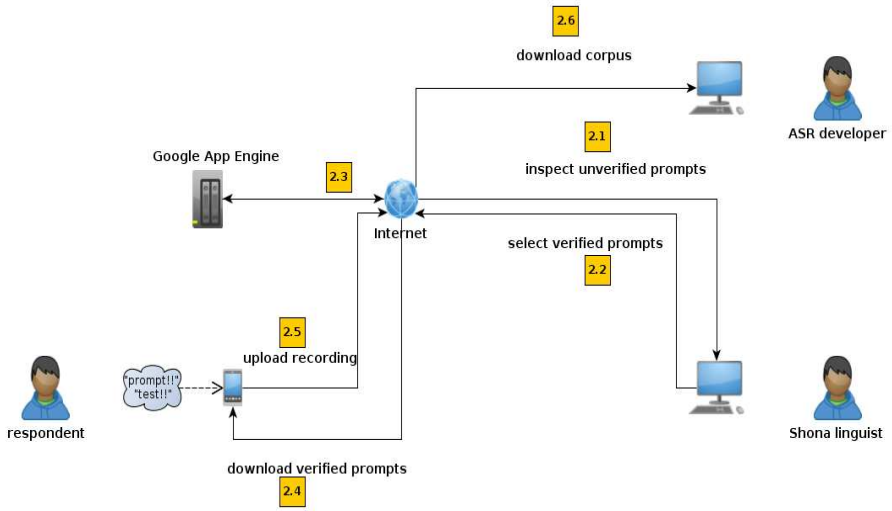

Figure 5.2: *A schematic diagram of data collection and training processes.*

Process 2.1 (*inspect unverified prompts*) and 2.2 (*select verified prompts*) in the figure allow users to inspect and verify prompts on-line. These two processes are discussed in more detail in Section 5.5. Process 2.4 (*download verified prompts*) and 2.5 (*upload recording*) form part of speech data collection. They will be discussed in the next chapter with the introduction of Woefzela. Process

2.3 is what the overall component interaction is centred around. It is mainly represented by a web interface which is hosted on GAE.

5.5 SYSTEM FUNCTIONALITY

The functionality describes responses of the system to user actions. System functionality is guided by the requirements proposed in Section 5.2. The following subsections discuss in detail the operations and functionality that each component adds to the overall system.

5.5.1 MAIN PAGE

The front page shown in Figure 5.3, which can be accessed at: *http://under-resourced.appspot.com/main.html*, gives the user a brief background of the different tools that are incorporated into the system. It is designed to provide simplicity and efficient usability. For each major tool that is introduced in the background an associated link is added for convenience if the user requires further information. The page also provides a left Links frame to facilitate easy navigation and to download tools required to automate the end-to-end process.

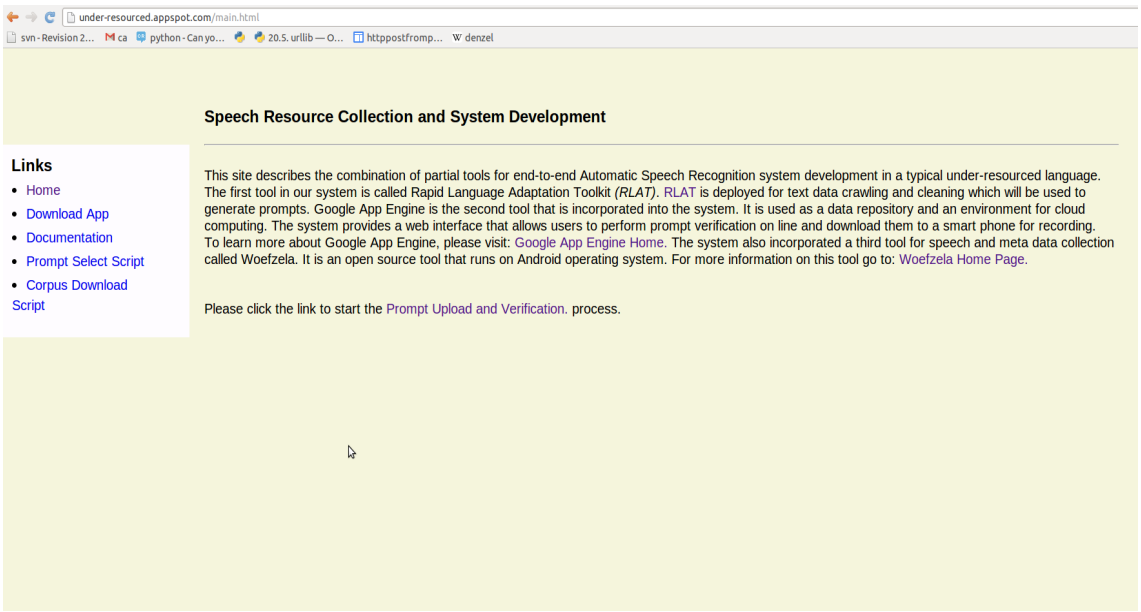

Figure 5.3: *A screen shot of the system main page.*

The *Download App* link on the Links frame allows the user to download *WDownload.apk*, which is an Android installation file for WDownload application discussed in Chapter 4. *Documentation* links to the project user manual/README that guides the development process of the end-to-end system. The fourth link *Prompt Select Script*, permits the user to download a Perl script used for

prompt segmentation. Finally the *Corpus Download Script* links to a Python script used to download the entire corpus from GAE. The verification and upload processes can be initiated by clicking the *Prompt Upload and Verification* link at the bottom of the page.

5.5.2 PROMPT UPLOAD

5.5.2.1 FILE PREPARATION

In order for the user to verify customised prompts, they need to upload a prompt text file of their choice. The text file should have one prompt per line in standard UTF-8. The blob-oriented database incorporated into the system requires the file to be limited to a size of 32 Megabytes. Figure 5.4 shows the contents of a sample 3-gram prompt file.

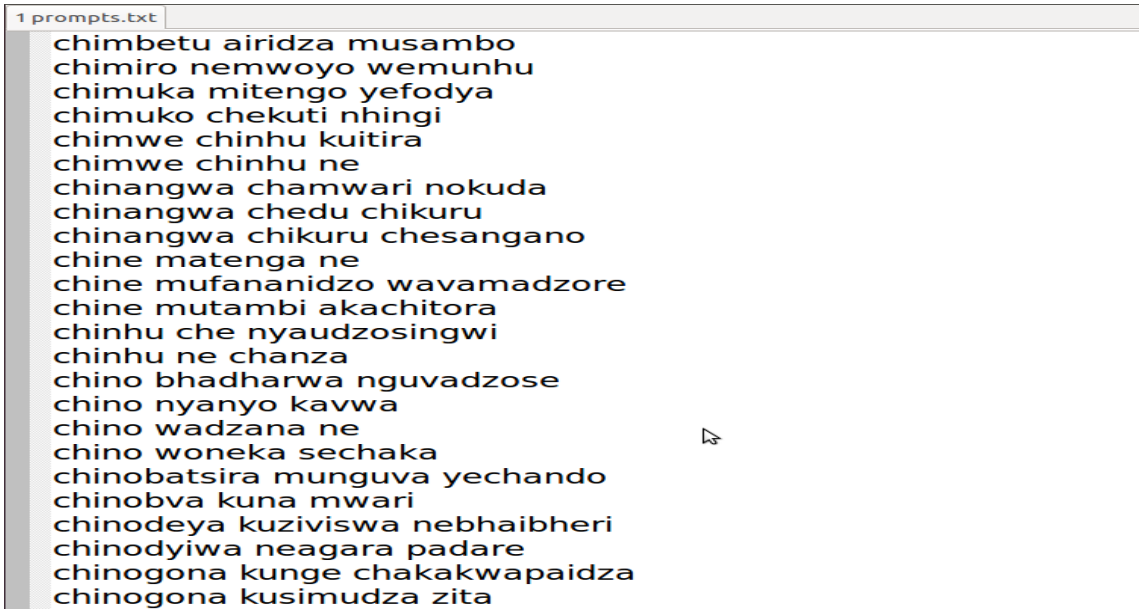

Figure 5.4: *An example of a prompt text file.*

5.5.2.2 PROMPT FILE UPLOAD

After the prompt file has been processed to meet the system's requirements, it can be uploaded to the GAE. The web interface provides an interface to perform file upload easily. Figure 5.5 shows an upload page with a file selection button *Choose File* and an *Upload* button. The former allows the user to select the prompt file from the local machine. The *Upload* button uploads the file to the GAE server for verification.

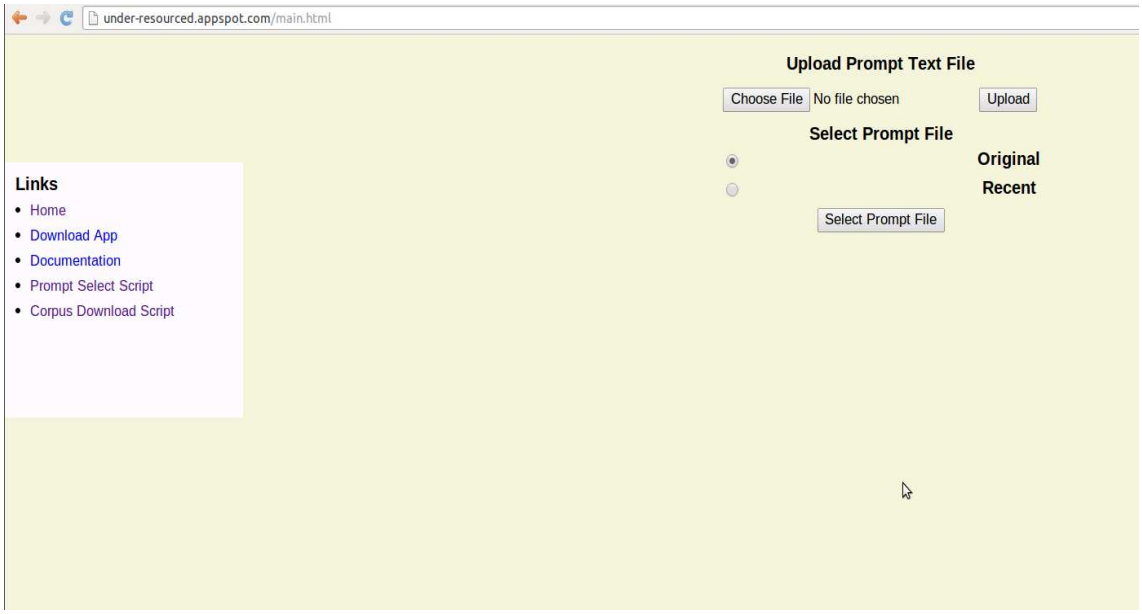

Figure 5.5: *A screen shot of a prompt text file.*

5.5.2.3 PROMPT FILE SELECTION

In Figure 5.5, the prompt selection process allows users to select and retrieve the desired uploaded prompt file. The original file and the most recently uploaded file to the system are retained; these are labelled as *original* and *recent* files. To retrieve the recently uploaded prompt file, select the *Recent* radio button to verify recently uploaded prompt file. Thereafter, click the *Select Prompt File* button to retrieve the file and display the verify page shown in Figure 5.6. Similarly, if the user prefers the original prompt file, they can select *Original* radio button.

5.5.2.4 PROMPT VERIFICATION

The verification process allows the user to verify a prompt by selecting check boxes to indicate which prompts are valid. The aim is to select prompts that are not offensive or use derogatory language; depending on the goals of the system being developed, foreign-language prompts may also be rejected at this stage. A prompt is considered valid if it meets the criteria mentioned.

Figure 5.6 shows a prompt with its associated check box. If a prompt is valid, the corresponding check box is checked to be incorporated into the generated prompt file. After the selection, the user should enter the name of the text file that would contain the valid prompts in the text field *Project Name*. The process is completed by clicking *Select Verified Prompts*. A “success” message is displayed afterwards.

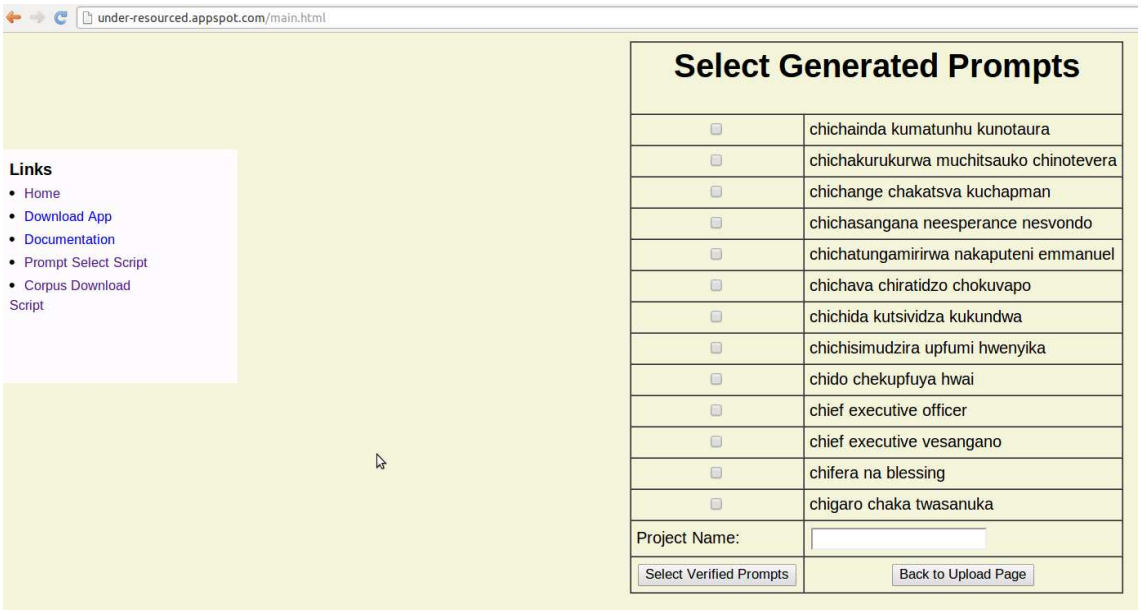

Figure 5.6: *A screen shot of App Engine prompt verifier.*

5.5.3 SYSTEM DATABASE

GAE provides a data store to house collected speech and text corpora. It employs modern data models, which provide persistent data storage similar to tables in traditional databases such as MySQL [36] and Oracle [37, 38]. A new model Python or Java class has to be created every time additional types of data are added to the GAE data store. The model contains elements that comprise the object class which are similar to records used in Structured Query Language (SQL) [35]. However, instead of SQL, GAE employs the services of Google Query Language [39] to query the database.

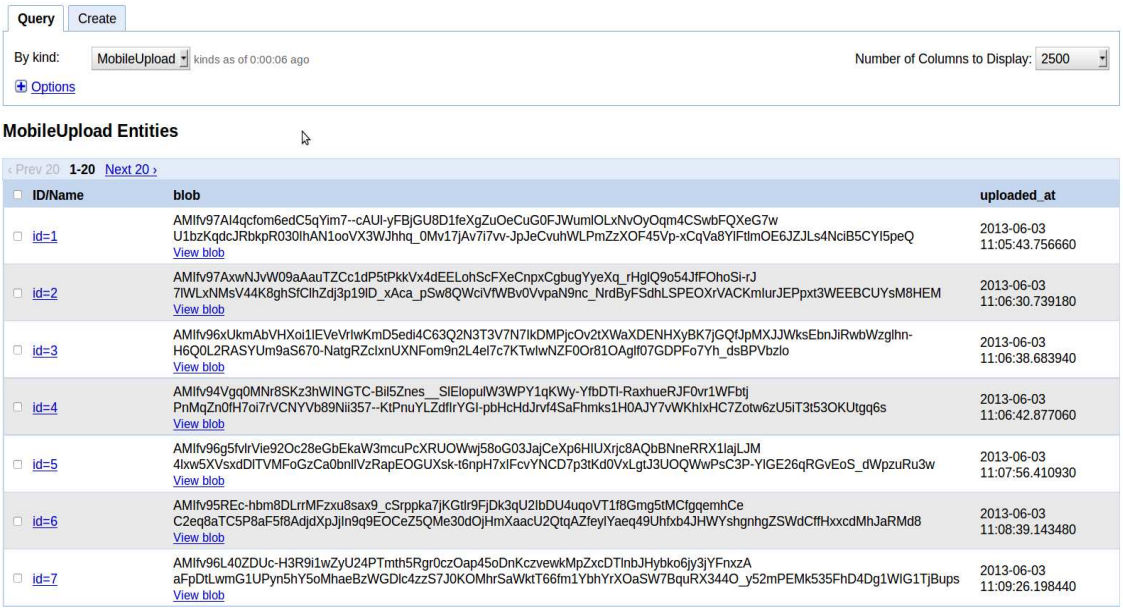

Figure 5.7: *Example of entities stored in GAE data store.*

Although the Engine provides limited database storage, large files can be uploaded using a binary large objects (BLOBs). The maximum file size currently allowed for a blob is 32 Megabytes. Figure 5.7 shows different entities created by a Python class from the *MobileUpload* model. The three entities `ID`, `blob` and `updated_at` are defined when creating a model class.

5.6 SOFTWARE IMPLEMENTATION

The software was constructed in GAE's Python environment which supports the programming language of choice for this project. It adheres to the conditions and requirements set out in Section 5.2. The development process followed a simple linear sequential model in which the coding process was performed at the end of the specification and design steps.

5.6.1 SOFTWARE OBJECTS AND ACTIONS

The software is composed of different objects for each tool that make up the overall system. The following table lists the different objects and gives a brief description of their respective actions.

Object	**Action and Description**
PromptsModel	A database model class to save uploaded prompts.
DictionaryModel	Database model class used to save file names and ID's
UploadModel	A database model class to save automatically uploaded files to a corpus.
ApplicationModel	A database model class to save WDownload install file.
VerifiedModel	A database model class to save verified prompts.
BlobUploader	A class that provides user authentication and removes old uploaded files.
PromptVerify	A class that validates the selected prompts and saves them to the database.
TaskQueues	A class that queues up tasks to perform downloads.
FileDownloadHandler	Facilitates total corpus download.
DictionaryHandler	Create a file with all entities to be downloaded and passed to the client.
IdParser	The class returns the recently uploaded file ID to WDownload
PromptCreator	Creates a Woefzela prompt file on the database using the *Verified model.*
MobileDataHandler	Handles uploaded URL encoded data and reconstructs uploaded files.

Table 5.1: *GAE object and action description.*

5.7 SOFTWARE TESTING

5.7.1 FILE UPLOAD

Automatic file upload is performed from a smart phone directly to the server. For the upload process to be completed, a checksum is returned from the server and if it matched that on the phone, the file

on the SD card is deleted to prevent multiple uploads. If the checksums do not match, the file does not get uploaded and remains on the SD card.

5.7.2 ONLINE LOG

GAE provides a dynamic real-time log interface to monitor every activity on the server. The developer can monitor activities as they occur and make changes accordingly. Different events can be sorted and filtered to the developer's preference. This functionality provides developers with an interactive debug interface to easily locate code faults instead of using a terminal.

5.8 SOFTWARE LIMITATIONS

The system's interaction with GAE is dependent on internet connectivity. Currently, development may only be done in Java, Python or PHP. GAE provides free resources for developers to start the development process. These resources include CPU usage, incoming and outgoing bandwidth and data storage. Once the resources are exhausted, App Engine refreshes the resources at the beginning of each calender day. If a developer requires more resources, GAE provides a billing system that allows resource customization to match the developer's needs. Currently, the system does not allow users to manually edit prompts on-line during the verification process. The user is only allowed to verify existing uploaded prompts.

5.9 CONCLUSION

In this chapter we have explored a platform that houses the collected corpus in a reliable secure location. GAE is a tool incorporated into the end-to-end system to perform various tasks, including: (i) data upload; (ii) data download; (iii) data repository; and (iv) hosting of the website around which the project is centred. We have developed assisting tools to assist in performance of all the tasks mentioned.

We have developed a method to upload a speech corpus from mobile phones to App Engine through the use of an Android application, as mentioned in Section 5.5.2.2. The download mechanisms were also performed through executing a Python script for corpus download and through running an Android application for verified prompt download of which the latter is briefly described in Section 5.3.1.3. Most of the manual operations such as prompt upload and prompt verification are performed on the website. Very little maintenance is required from the developer's side since GAE provides all the benefits that come with the Google platform to the user.

CHAPTER SIX

SHONA : CASE STUDY

6.1 INTRODUCTION

As discussed in Chapter 3, Shona was the language used to validate partial and end results of our end-to-end system. To achieve this, we built our first automatic speech recognizer for the Shona language. In this chapter, we start by providing a brief background on the language of choice in Section 6.2. Section 6.3 discusses in detail the different phases involved in collecting speech data. Thereafter, we present an overview of the experiment and the set up. Section 6.6 and Section 6.7 present the results of the first and second experiment respectively. We then give a summary and discussion of the results followed by the conclusion in Section 6.8.

6.2 BACKGROUND

The Shona language is a language on the Bantu branch of the Niger-Congo language family, native to the Shona people of Zimbabwe, southern Zambia, Botswana and parts of Mozambique. Shona is used as an umbrella term to identify people who speak one of the Shona language dialects, namely Zezuru, Karanga, Manyika, Ndau, and Korekore. Zezuru, mainly spoken in Mashonaland, is regarded as standard Shona dialect [40]. Shona is also spoken unofficially in South Africa and is closely related to the Venda language (one of the official languages of South Africa).

The language has more than 10.8 million first-language speakers across Southern Africa. Shona is a tonal language with two tones (high and low); the tones are not indicated in the script form of the language, which uses the Roman alphabet with a fairly regular relationship between orthography and pronunciation. The Shona language comprises of five vowels and thirty five consonants. Table 6.1

lists the phonetic pronunciations of the vowels and Table 6.2 lists the consonant pronunciations.

Table 6.1: Shona vowels: orthography and pronunciation

Vowel	IPA
a	/a/
e	/e/
i	/i/
o	/o/
u	/u/

Table 6.2: Shona consonants: orthography and pronunciation

Consonant	IPA	Consonant	IPA	Consonant	IPA	Consonant	IPA
b	/b/	bh	/$\ddot{b}$	ch	/tf/	d	/d/
dh	/$\ddot{d}$/	dzv	/$\widehat{\text{d}\beta}$z	dy	/dg/	f	/f/
g	/ǧ/	h	/h/	j	/dz/	k	/k/
l	/l/	m	/m/	mbw	/mbeg/	mh	/$\ddot{m}$/
n	/n/	ng	/η/	p	/p/	r	/r/
s	/s/	sv	/$\widehat{\phi}$s/	sw	/skw/	t	/t/
ty	/tk/	tsv	/$\widehat{\text{t}\phi}$s/	v	/β/	vh	/v/
w	/w/	y	/j/	z	/z/	zv	/$\widehat{\beta}$z/

6.3 SPEECH DATA COLLECTION

The final stage of our data collection efforts was the collection of speech data. As discussed in Chapter 2, various factors need to be taken into consideration when deciding which speech data collection method to use. For our typically under-resourced conditions, we opted to incorporate a mobile phone application called Woefzela to facilitate the speech data collection process. Woefzela is an open-source tool that runs on the Android operating system. It does not rely on internet connection to perform audio data collection, but does require that text prompts be loaded on the phone manually. The main reason for using the application is due to its open-source nature and ease of use.

In Chapter 4, we introduced a tool called WDownload used to download previously verified prompts from GAE to the mobile phone. WDownload integrates and uses the file structure created by Woefzela on the mobile phone to direct the download. The downloaded prompt file is loaded when Woefzela is started. However, before the recording process could commence, there were several measures that

were taken to ensure that we collected high quality speech data. These measures are discussed in detail in the next subsections.

6.3.1 RESPONDENT CANVASSING AND SCREENING

For the recording process to start, native speakers of the language have to be recruited to perform verification and to do the recordings. A Shona native speaker was hired to be in every recording session to screen the respondents. The screening process was done by assessing the fluency and accuracy with which respondents could read fifteen Shona sentences that were randomly selected from the prompts text file.

The respondents included students and domestic workers, and were rewarded with token wards for their participation. However, this turned out to be surprisingly controversial – many potential respondents wished for substantial payments in order to participate, which was not compatible with the limited budget and open-source approach of the current project. Amongst the students, there was a greater receptivity for the open-source style; we were able to collect with greater success in that population, but only a limited number of students were available in Pretoria, where our collection was being performed.

Another challenge that was faced by field workers during the collection process was getting many respondents in a single location to record. This was the unexpected result of Xenophobic attacks that had occurred previously around the location of the recordings. The recordings therefore had to be done with one or two respondents at a time in different locations, and again limited our ability to collect a substantial number of speakers.

6.3.2 RESPONDENT REGISTERING

After the screening process was concluded, the respondents were required to sign a consent form to allow their voices to be used for our project; afterwards they received their tokens of appreciation. They were also required to fill in a profile form which included their age, gender, phone numbers and identity or passport numbers. The reason for requiring respondents to provide their identity number is to ensure that they are above the age of consent. However, the field worker may use the respondents phone numbers if more recording is required. The recording process using Woefzela (see below) was very intuitive for students; very little training was required to operate the application. The older generation needed more assistance on how the application should be operated.

6.3.3 PROMPT RECORDING

Six inexpensive mobile telephones running the Android operating system were used to perform recordings. The phones had to be fully charged and running all the software required. Woefzela was used for audio and meta-data collection. It provides a practical manner to collect speech data, especially in under-resourced environments.

Each respondent was required to record about 500 prompts initially; this was later reduced to 300 when frequent respondent fatigue and loss of concentration was noticed. Depending on how fast the respondent could read prompts, the recording session could take between 45 minutes to an hour. The recording process allowed the user to have multiple recording sessions. If the user needed to rest, the recording session could be stopped and restarted at a later stage. The respondents removed on average 63 prompts during the prompt verification process based on their incorrect spelling and their obscene content. Other prompts were also flagged during the recording process by respondents who thought they were too long or were a different Shona dialect. The recordings with the associated meta data were then saved onto the SD card. The collection effort initially aimed at recording 20 Shona speakers, based on performance against speaker-number results previously obtained [41].

Collected data may be copied directly from the SD cards to limit reliance on the internet, which is a significant concern in the developing world. However, phones can also be moved to an area with internet and directly upload all the files on the SD cards to the server. WUpload is an Android application that is responsible for data upload to the GAE. To ensure that files are not duplicated, a checksum is returned from the server and if it matches that on the phone, the file in the SD card is deleted. The data is stored in a blob-oriented database for easy retrieval.

6.4 EXPERIMENT OVERVIEW

In order to evaluate the partial solutions that make up the end-to-end system, we have conducted various experiments as described in this section. We present the quality of the recorded transcribed corpora through various quality control measures. Finally, we report on the results obtained with an acoustic model that was trained to perform ASR.

6.5 EXPERIMENTAL SETUP

6.5.1 SPEECH DATA QUALITY CONTROL AND ANALYSIS

The speech data and associated eXtensible Mark-up Language (XML) files can be downloaded directly from the Google App Engine using a Python script. In order to complete the evaluation of our system, we have downloaded the data and developed a grapheme-based Shona ASR system.

Because of the complications described in Section 6.3.1 above, the collected corpus was smaller than we had initially intended. We have recordings from eight female voices and twelve male voices, and a total of just over six and a half hours of speech. This data had to go through quality measures both on the phone [8] and during off-line post processing [5]. The post processing scripts use the meta data from the mobile phone to tag the audio files. The process extracts the text prompts from XML files and creates associated transcriptions.

The off-line quality control examines the volume levels and the stop/start errors of the recording. Table 6.3 shows the results of the quality control process before any training is performed. From a total of 7602 recorded utterances, only 3204 were usable to train acoustic models. The quality control measure rejected the recordings due to a lot of background noise present in the recordings. The recorded prompts amounted to 13398 word tokens, containing 3871 unique types. The recording process managed to acquire 6.01 hours of speech data.

Table 6.3: *Quality control data results.*

Respondent	**Recordings**	**Usable**	**Respondent**	**Recordings**	**Usable**
000	305	3	010	494	65
001	395	59	011	390	231
002	478	321	012	369	95
003	679	259	013	516	186
004	328	13	014	199	111
005	520	276	015	378	211
006	375	223	016	197	87
007	424	331	017	185	11
008	377	205	018	370	228
009	393	119	019	230	170
TOTAL	**7602**	**3204**			

6.5.2 TRAINING AND TESTING CORPORA

The quality-control process verifies that prompts have appropriate durations and energy levels, but does not contain any mechanisms to verify that the recorded audio files correspond to the prompted transcriptions. For a better understanding of the collected corpus, we randomly split data into 80% and 20% training and test data respectively. For purposes of these experiments, the length of the test data was not taken into account, hence it was not kept constant throughout the different folds. For twenty speakers, we used five-fold cross validation, which means there were four speakers in the test set on each iteration. The selection process and speaker tags were generated so that no speaker would

appear in both the test and training sets.

6.5.3 PRONUNCIATION DICTIONARY

The development of phone-based ASR systems for new under-resourced languages normally requires the intervention of a native speaker of that particular language. To train our system, we opted to utilise a grapheme-based pronunciation dictionary. The pronunciation dictionary is assembled by presenting a word with its associated grapheme representation. All the words from the word list are acquired from the crawled text corpus.

6.5.4 FEATURE EXTRACTION AND ACOUSTIC MODELLING

The recogniser employed a standard Hidden Markov Model (HMM) based system. For feature extraction, 39 (13 static, 13 delta and 13 delta-delta) dimensional Mel Frequency Cepstral Coefficient (MFCC) features were generated using HTK [42]. The MFCCs were extracted from a 25 milliseconds frame every 10 milliseconds [43]. Channel normalisation was performed by means of cepstral mean normalization (CMN) per speaker. Eight Gaussian mixtures per HMM state were incorporated to model the cepstral densities. A flat grapheme-based, context dependent language model was used for grapheme recognition.

6.6 EXPERIMENT 1: SHONA AND ENGLISH + SHONA

In this section we present the results of the overall corpus. The independent test and training sets contain both English and Shona content. Table 6.4 shows the overall amount of data used and the accuracy of the grapheme-based system with both English and Shona. Table 6.4 presents the accuracy and correctness of the five-fold cross validation procedure.

Table 6.4: *Overall English + Shona results.*

Language	% Correct	% Accuracy	Amount of Data
English + Shona	72.87	63.03	6.01 hours

The percentage correctness is calculated in Equation 6.1 using *deletion* errors (D), *substitution* errors (S) and (N) which represents the number of tokens in the reference transcriptions.

$$Precentage\,Correct = \frac{N - D - S}{N} \times 100 \tag{6.1}$$

The percentage accuracy is calculated using the parameters defined above with the addition of the *insertion* errors (I). Equation 6.2 below is a general representation of the recogniser performance.

$$Precentage\,Accuracy = \frac{N - D - S - I}{N} \times 100 \tag{6.2}$$

We also present experiment results per speaker. Table 6.5 shows the results of all twenty speakers for the English and Shona combined corpus.

Table 6.5: *English + Shona ASR results per speaker.*

	English + Shona				
Respondent	**% Correct**	**% Accuracy**	**Respondent**	**% Correct**	**% Accuracy**
Speaker 000	66.42	55.03	Speaker 010	66.23	50.76
Speaker 001	66.18	53.54	Speaker 011	68.98	57.62
Speaker 002	61.18	49.88	Speaker 012	65.01	53.71
Speaker 003	70.39	59.34	Speaker 013	73.64	61.06
Speaker 004	65.14	52.44	Speaker 014	69.72	57.10
Speaker 005	47.91	65.85	Speaker 015	67.50	50.38
Speaker 006	75.68	67.70	Speaker 016	56.85	41.72
Speaker 007	72.33	63.40	Speaker 017	75.53	67.49
Speaker 008	63.97	51.13	Speaker 018	72.06	63.76
Speaker 009	75.74	65.83	Speaker 019	72.98	63.14

6.6.1 RESULTS OF DISCUSSION

The results presented in Table 6.5 and Table 6.4 are in the same range of accuracies presented by Basson *et al.* [44] for a grapheme-based system trained on a corpus of similar size. The results can however be improved by using phones instead of graphs since there is a significant amount of English content as highlighted in Chapter 3. The possible extent of such an improvement is investigated next.

6.7 EXPERIMENT 2: SHONA-ONLY

The training and test data contained English, which has a highly irregular mapping between graphemes and phonemes. The grapheme-based recognition results for such languages are invariably poor [45] – especially for the case where the majority of the speech data are written in the orthography of another language. To investigate this further, the English content from the training and test data were removed and the system was retrained. The experiments were also conducted using independent test sets and five-fold cross validation. Table 6.6 shows the overall amount of data used and the accuracy of the grapheme-based system with Shona-Only data.

Table 6.6: *Overall Shona-Only results.*

Language	% Correct	% Accuracy	Amount of Data
Shona-Only	75.30	64.56	5.67 hours

To compare the effect of removing the English content from both the training and test sets, we retrain and present the results. Table 6.7 shows the per speaker accuracy of the grapheme-based system with Shona-Only data.

Table 6.7: *Shona-Only ASR results per speaker.*

	Shona-Only				
Respondent	**% Correct**	**% Accuracy**	**Respondent**	**% Correct**	**% Accuracy**
Speaker 000	69.29	58.73	Speaker 010	69.47	56.11
Speaker 001	70.32	59.36	Speaker 011	72.90	64.54
Speaker 002	65.37	55.06	Speaker 012	68.80	58.16
Speaker 003	73.73	64.03	Speaker 013	78.16	65.80
Speaker 004	67.43	57.85	Speaker 014	73.69	63.60
Speaker 005	77.61	69.18	Speaker 015	69.59	53.08
Speaker 006	80.14	72.36	Speaker 016	64.16	48.99
Speaker 007	75.98	67.91	Speaker 017	77.94	70.34
Speaker 008	67.68	56.17	Speaker 018	76.78	68.19
Speaker 009	78.32	69.65	Speaker 019	75.80	65.25

6.7.1 RESULTS AND DISCUSSION

The Shona-Only grapheme-based system also produces accuracies in the range presented by [44]. Although our corpus was limited in size and speaker variability, the grapheme accuracy achieved is acceptable. From Table 6.7, Speaker 016 was found to have a high number of sentences that had English content, hence the low accuracy. The accuracy of the system can be improved by incorporating a manually verified pronunciation dictionary. However, a grapheme-based system for a language with regular spelling system such as Shona may yield accuracies comparable those of systems using manually verified pronunciation dictionaries [44].

6.8 SUMMARY AND CONCLUSION

It was observed from our experiments that all the speaker results improved, showing that even small amounts of English data in our corpus hurts grapheme-based performance substantially. The accuracies achieved in both 6.6 and 6.7 are in the same range as those achieved for phoneme recognition on

the 11 official South African languages during Lwazi project [31]. Of course, this is only a starting point for Shona ASR development and a number of measures that are likely to improve recognition accuracy are discussed in the next chapter.

CHAPTER SEVEN

CONCLUSION

7.1 INTRODUCTION

In this dissertation, we have investigated and evaluated different partial tools that we integrated to enable the end-to-end development of an Automatic Speech Recognition system in a typical under-resourced language. We investigated: (i) text data collection measures that were cost effective and less reliant on internet connectivity; (ii) prompt generation and on-line verification; (iii) automatic prompt downloading using open-source software; (iv) a secure hosted environment for data verification, storage and distribution used in conjunction with existing tools for speech data recording; and (v) speech data collection.

7.2 SUMMARY OF CONCLUSIONS

7.2.1 TEXT DATA COLLECTION MEASURES

In Chapter 3, we incorporated a web-based data collection tool called RLAT which was used for text data crawling and data cleaning. We thus acquired internet data for a Zimbabwean language called Shona. RLAT provided language independent normalization for the crawled data. However, there were large amounts of English content in the text data which needed to be removed. Therefore the text data were cleaned to contain more Shona words than English words. The process managed to acquire enough data to generate prompts that were utilised for recording.

7.2.2 PROMPT GENERATION AND ON-LINE VERIFICATION

Prompt generation was performed after taking several factors into consideration. The domain in which the prompts would be used and the phonetic coverage of the prompts were among the most

important factors taken into account. We used a Perl script to generate over five hundred 3-word prompts; however, the script did not perform any spell checking. For this reason, an on-line prompt verifier was developed to let native speakers remove inappropriate content.

7.2.3 AUTOMATIC PROMPT DOWNLOAD

Even though the aim of the end-to-end system was to combine existing partial tools to develop a complete system, the different tools needed to be interfaced. This prompted the development of new tools such as WDownload to facilitate prompt download from the server to the mobile phone. The tool was made consistent with the concept of the project by publishing it as open-source software. It runs on smart phones running Android OS.

7.2.4 STORAGE AND DISTRIBUTION

Chapter 5 details the significance of incorporating GAE into the end-to-end system. It is used as a data repository and an environment for cloud computing. GAE does not require maintenance from the developer and provides all the benefits that come with the Google platform to the user. GAE can be easily interfaced with most applications which makes it an ideal tool for open-source development.

7.2.5 SPEECH DATA RECORDING TOOLS

One of the existing tools that we incorporated into our system was Woefzela, an open-source tool that also runs on Android OS. The tools worked in conjunction with WUpload which facilitated automatic data upload to the GAE whenever there was internet connectivity. This process, though reliant on internet connectivity, reduced the tedious process of uploading each session to the server manually.

7.3 FURTHER APPLICATION AND FUTURE WORK

The accuracy of the ASR system can be improved in a number of ways. For example, a manually verified pronunciation dictionary – especially of the English words – would be useful. Also, during the recording process it was found that there were several inconsistencies in the pronunciation of certain numerals: the reading of the years and large numbers, particularly, varied from respondent to respondent. This led to a degradation in word accuracy. Most importantly, more speech from a larger number of respondents will greatly enhance the accuracy of our recognizer. The logistic challenges that limited the size of our corpus were both unexpected and highly dependent on the local context. We hope that others will use our tools to perform ASR system development in under-resourced languages and that they will not be plagued by similar logistical issues.

7.4 CONCLUSION

We have explored the development of a set of tools that can be used for rapid end-to-end ASR development. The process is used for rapid end-to-end ASR system development. The process was tested and validated using the Shona language native to Zimbabwe. The system uses the web-based RLAT to acquire text data. The text data were cleaned to contain words in a 86% to 14% Shona-to-English ratio. Text data were segmented into prompts and uploaded to GAE. The prompts were verified on-line through a web-based system. To automate the end-to-end process, we also developed and Android application, WDownload, to download verified prompts to a mobile phone. Woefzela was used for recording and meta data collection. The recorded speech data was uploaded to the GAE through an Android application called WUpload. The data can be fetched at any time to develop an ASR system. With the combination of all partial solutions, end-to-end system development is fast, easy to use, and cost effective; its open-source availability will hopefully stimulate ASR development in many additional languages.

REFERENCES

[1] Etienne Barnard, Laurens Cloete, and Hina Patel, "Language and technology literacy barriers to accessing government services," in *Electronic Government*. 2003, pp. 37–42, Springer.

[2] Etienne Barnard, Johan Schalkwyk, Charl Van Heerden, and Pedro J Moreno, "Voice search for development," in *INTERSPEECH*, Makuhari, Japan, Sept 2010, pp. 282–285.

[3] Madelaine Plauche, Udhyakumar Nallasamy, Joyojeet Pal, Chuck Wooters, and Divya Ramachandran, "Speech recognition for illiterate access to information and technology," in *Information and Communication Technologies and Development, 2006. ICTD'06. International Conference on*. IEEE, May 2006, pp. 83–92.

[4] Thad Hughes, Kaisuke Nakajima, Linne Ha, Atul Vasu, Pedro J Moreno, and Mike LeBeau, "Building transcribed speech corpora quickly and cheaply for many languages.," in *INTERSPEECH*, Makuhari, Japan, Sept 2010, pp. 1914–1917.

[5] Nic J De Vries, Jaco Badenhorst, Marelie H Davel, Etienne Barnard, and Alta De Waal, "Woefzela-an open-source platform for ASR data collection in the developing world," in *INTERSPEECH*, Florence, Italy, Aug 2011, pp. 3177–3180.

[6] Tim Schlippe, Sebastian Ochs, and Tanja Schultz, "Wiktionary as a source for automatic pronunciation extraction.," in *INTERSPEECH*, Makuhari, Japan, Sept 2010, pp. 2290–2293.

[7] Dan Sanderson, *Programming Google app engine*, O'Reilly, 2010.

[8] Nic J de Vries, Marelie H Davel, Jaco Badenhorst, Willem D Basson, Etienne Barnard, et al., "A smartphone-based ASR data collection tool for under-resourced languages," 2013, Elsevier.

[9] Jaco Badenhorst, Alta De Waal, and Febe De Wet, "Quality measurements for mobile data collection in the developing world," in *3rd workshop on Spoken Languages Technologies for Under-resourced languages*, Cape Town, South Africa, May 2012, pp. 139–146.

[10] Douglas B Paul and Janet M Baker, "The design for the wall street journal-based csr corpus," in *Proceedings of the workshop on Speech and Natural Language*. Association for Computational Linguistics, 1992, pp. 357–362.

[11] Alexander Kivaisi and Audrey Mbogho, "Web-based corpus acquisition for Swahili language modelling," in *3rd workshop on Spoken Languages Technologies for Under-resourced languages*, Cape Town, South Africa, May 2012, pp. 42–47.

[12] Tim Schlippe, Chenfei Zhu, Jan Gebhardt, and Tanja Schultz, "Text normalization based on statistical machine translation and internet user support.," in *INTERSPEECH*, Makuhari, Japan, Sept 2010, pp. 1816–1819.

[13] JS Garofolo, LF Lamel, WM Fisher, JG Fiscus, DS Pallett, and NL Dahlgren, "Darpa timit acoustic phonetic speech corpus," 1993.

[14] Damjan Vlaj, Aleksandra Zögling Markus, Marko Kos, and Zdravko Kacic, "Acquisition and annotation of slovenian lombard speech database.," in *LREC*, Valletta, Malta, May 2010, pp. 595–600.

[15] Philip Gee, "Mobile telephones and behavioural research," 2007, vol. 8, pp. 193–201, NAFO.

[16] Adam Lerer, Molly Ward, and Saman Amarasinghe, "Evaluation of ivr data collection uis for untrained rural users," in *Proceedings of the First ACM Symposium on Computing for Development*, London, United Kingdom, Dec 2010, ACM, p. 2.

[17] Febe de Wet, Pippa Louw, and Thomas Niesler, "The design, collection and annotation of speech databases in south africa," Parys, South Africa, 29 November-1 December 2006, pp. 184–187.

[18] Ian Lane, Alex Waibel, Matthias Eck, and Kay Rottmann, "Tools for collecting speech corpora via mechanical-turk," in *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*. Association for Computational Linguistics, 2010, pp. 184–187.

[19] Keelan Evanini, Derrick Higgins, and Klaus Zechner, "Using amazon mechanical turk for transcription of non-native speech," in *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*. Association for Computational Linguistics, 2010, pp. 53–56.

[20] Tanja Schultz, Alan W Black, Sameer Badaskar, Matthew Hornyak, and John Kominek, "Interspeech 2007," in *INTERSPEECH*, Antwerp, Belgium, Aug 2007, pp. 2125–2128.

[21] Alexander Siebert, David Schlangen, and Raquel Fernández, "An implemented method for distributed collection and assessment of speech data," in *The 8th SIGdial Workshop on Discourse and Dialogue*, Antwerp, Belgium, Sept 2007.

[22] Alex Gruenstein, Ian McGraw, and Andrew Sutherland, "A self-transcribing speech corpus: collecting continuous speech with an online educational game," in *SLaTE Workshop*, Dallas,Texas,USA, March 2009, pp. 1520–5273.

[23] João Freitas, António Calado, Daniela Braga, Pedro Silva, and M Dias, "Crowdsourcing platform for large-scale speech data collection," Nov 2010.

[24] Christoph Draxler and Klaus Jänsch, "Speechrecorder-a universal platform independent multichannel audio recording software.," in *LREC*, Lisbon, Portugal, May 2004.

[25] Denis Burnham, Dominique Estival, Steven Fazio, Jette Viethen, Felicity Cox, Robert Dale, Steve Cassidy, Julien Epps, Roberto Togneri, Michael Wagner, et al., "Building an audio-visual corpus of australian english: Large corpus collection with an economical portable and replicable black box.," in *INTERSPEECH*, Florence, Italy, Aug 2011, pp. 841–844.

[26] Tanja Schultz, "Globalphone: a multilingual speech and text database developed at karlsruhe university.," in *INTERSPEECH*, Denver, Colorado, USA, Sept 2002, pp. 345–348.

[27] Luís C Oliveira, Sérgio Paulo, Luís Figueira, Carlos Mendes, Ana Nunes, and Joaquim Godinho, "Methodologies for designing and recording speech databases for corpus based synthesis.," in *LREC*, Marrakech, Morocco, May 2008.

[28] Claude Barras, Edouard Geoffrois, Zhibiao Wu, and Mark Liberman, "Transcriber: development and use of a tool for assisting speech corpora production," 2001, vol. 33, pp. 5–22, Elsevier.

[29] TJ Ndwe, E Barnard, and MR De Villiers, "Admixture practises in South African languages: Impact on speech-enabled technology design," in *IST-Africa Conference Proceedings*. IEEE, 2011, pp. 1–8.

[30] R Weide, "The CMU pronunciation dictionary, release 0.6," 1998, Carnegie Mellon University.

[31] Linsen Loots, Marelie Davel, Etienne Barnard, and Thomas Niesler, "Comparing manually-developed and data-driven rules for p2p learning," in *20th Annual Symposium of the Pattern Recognition Association of South Africa*. Nov 2009, pp. 35–39, PRASA 2009.

[32] Roger S Pressman, "Software engineering: a practitioner's approach," 2001, pp. 28–30, McGraw-Hill.

[33] ShrikantAndroidInfo, "Android architecture," Jan 2013.

[34] Swaran Lata and Somnath Chandra Vijay Kumar, "Development of linguistic resources and tools for providing multilingual solutions in indian languages a report on national initiative," in *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May 2010, pp. 2851–2854, European Language Resources Association (ELRA).

[35] Charles Severance, *Using Google App Engine*, O'Reilly, 2009.

[36] Ross Mistry and Stacia Misner, *Introducing Microsoft® SQL Server® 2012*, O'Reilly Media, Inc., 2012.

[37] Bill Pribyl and Steven Feuerstein, *Learning oracle pl/sql*, O'Reilly, 2002.

[38] Steven Feuerstein and Bill Pribyl, *Oracle Pl/SQL Programming*, O'Reilly Media, Inc., 2009.

[39] Eugene Ciurana, *Developing with Google App Engine*, Apress, 2009.

[40] Calisto Mudzingwa, *Shona morphophonemics: Repair strategies in Karanga and Zezuru*, Ph.D. thesis, University of British Columbia, 2010.

[41] Etienne Barnard, Marelie Davel, and Charl Van Heerden, "ASR corpus design for resource-scarce languages," in *INTERSPEECH*, Brighton, UK, Sept 2009, pp. 2847–2850.

[42] Steve Young, Gunnar Evermann, Dan Kershaw, Gareth Moore, Julian Odell, Dave Ollason, Valtcho Valtchev, and Phil Woodland, "The HTK book," 2002, vol. 3, p. 175.

[43] Neil Kleynhans, Raymond Molapo, and Febe De Wet, "Acoustic model optimisation for a call routing system," Pretoria, South Africa, Nov 2012, pp. 165–172, PRASA.

[44] Willem D Basson and Marelie H Davel, "Comparing grapheme-based and phoneme-based speech recognition for Afrikaans," in *23rd Annual Symposium of the Pattern Recognition Association of South Africa*. 2012, pp. 144–148, PRASA 2012.

[45] MH Davel and Etienne Barnard, "Default-and-refinement approach to pronunciation prediction," Grabouw, South Africa, Nov 2004, pp. 374–393, PRASA 2004.