JIMMA UNIVERSITY

JIMMA INSTITUTE OF TECHNOLOGY

SCHOOL OF COMPUTING

A THESIS SUBMITTED TO THE DEPARTMENT OF INFORMATION TECHNOLOGY IN PARTIAL FULFILLMENT FOR THE DEGREE OF MASTER OF SCIENCE IN INFORMATION TECHNOLOGY

Designing Dictionary Based Spelling Checker for Afaan Oromoo

Abduljebar Kedir

June, 2019

Jimma, Ethiopia

JIMMA UNIVERSITY

JIMMA INSTITUTE OF TECHNOLOGY

SCHOOL OF COMPUTING

This is to certify that the thesis prepared by *Abduljebar Kedir*, titled: *Designing Dictionary Based Spelling Checker for Afaan Oromoo* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Information Technology complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Abduljebar Kedir _____ _____

SIGNATURE DATE

APPROVED BY ADVISORS:

ADVISOR:

Wondwossen Mulugeta (PhD) _____ June 2019

SIGNATURE DATE

CO-ADVISOR:

_____ _____ _____

SIGNATURE DATE

APPROVED BY RESEARCH EXAMINATION MEMBERS

1. _____ _____ _____

SIGNATURE DATE

2. _____ _____ _____

SIGNATURE DATE

Contents

List of Tables

List of Figures

Acknowledgement

First of all I would like to thank the almighty God for helping me to accomplish my work. My next gratitude also goes to my advisors for the valuable, reasonable, constructive comments and in showing me clear research directions throughout the course of this work.

I would also like to thank my brothers Ibrahim Kedir and Tahir Kedir in providing me their research experiences, the challenges during conducting a research and how to confront these challenges.

Abstract

The focus of this thesis work is to design dictionary based spelling checker for Afaan Oromoo language. We proposed a dictionary lookup and n-gram approach to detect the misspelled word(s) and provide the correction suggestions. Accordingly, we built a dictionary from different domains consisting of correctly spelled words. The prepared dictionary is used as a point of reference in error detection and correction. The input word from the user would be cross-checked with the already built dictionary and we have employed a dictionary lookup approach in doing so. If the user input word is found in the dictionary, then the spelling checker considers it as correctly spelled. Otherwise, the word would be detected as misspelled and the next task will be generating the possible corrections for that particular word. To do so, we have used n-gram approach to generate those words having a shared or common bigrams as suggestions from the dictionary with respect to the wrongly spelled word(s). The generated suggestions will be ranked in accordance with their similarity relative to the invalid word(s) so that the most likely correction word to replace the wrongly spelled word would be at the top of the suggested list of words. To make this happen, we have employed the Dice's coefficient similarity measure. Accordingly, the candidate word with the highest dice's score would be the most likely to substitute the misspelled word. For evaluating the performance of the spelling checker, we have used precision and recall metrics as performance measurements. Consequently, a test dictionary of 3000 words collected from different domains of Afaan Oromoo language is used to test the capability of the spelling checker in error detection and providing corrections for those detected words. Accordingly, the results from evaluation reveal that the designed system scored a precision of 100% and 83.33% of recall. Additionally, we have also conducted an experiment to demonstrate the possible factors that could affect the performance of the designed spelling checker under selected three circumstances.

Chapter 1

Introduction

Now a day natural language processing and artificial intelligence are gaining focus of the researchers. The research done on these two areas reveal that there is still a lot need to be done; particularly for natural language processing, the research conducted so far has not got its depth [15, 16]. Moreover, the current state of the art indicates that the researchers are diverging towards the development of natural language processing tools and designing the intelligent systems (agents) that could gradually replace the work and ability of human being. Furthermore, from global perspective, some studies show that in the near future millions of people will be losing their job as a result of joint implementation of natural language processing and artificial intelligence to develop smart human-like decision making systems [17]. However, such a trend is at its early stage even for natural language processing tools from local point of view.

Natural Language processing (NLP) is a sub-field of Artificial Intelligence that is focused on enabling computers to understand and process human languages to get computers closer to a human-level understanding of natural languages. Yet, computers don't have the same intuitive recognition of natural languages to the best of human being. In a nutshell, computers can't read between lines and understand. However, recent advances in machine learning have made computers to perform things like language translation, semantic understanding, text summarization and others. One of the reasons that make Natural Language processing harder is the fact that the process of reading and understanding a particular text of the language as we do is far more complex than it seems. This can happen as a result of language contexts, structures and many more.

Afaan Oromoo is one of the local languages widely spoken in Ethiopia and some countries across Africa [18]. This language is a Cushitic family spoken by more than 50 million people in Ethiopia only and it ranks the third largest language in Africa. Furthermore, it uses Latin scripts like English language which are known as "*Qubee*" and additionally some language specific letters called "*Qubee Dachaa*". These "Qubee Dachaa" letters are the unique alphabetical features recognized in Afaan Oromoo language and they are five in number.

These scripts are formed by combining two consonant letters which are read as a single letter in this language. Moreover, Afaan Oromoo language is known by its rich morphological structures. The morphological richness and complexity of the language makes harder for the researchers to build a system of their interest as it requires having language expertise. It is from this fact that it is found to be a challenging task to design a complete and fully-fledged systems that bases on the language morphology. One of such challenging language processing for morphologically rich languages is spell checking.

The aim of this thesis is, then, on designing a dictionary based Afaan Oromoo spelling checker. Accordingly, we have designed a system incorporating a dictionary of correctly spelled words of Afaan Oromoo language. Therefore, we strongly believe that this approach will reduce the multifaceted challenge of morphological analysis of Afaan Oromoo language.

The history of spelling checker dates back to 1960s for international language like English [12]. Ralph Gorin, created the first spelling checker program called *SPELL* which was written as application program for general English text. Gorin wrote *SPELL* in assembly language and he made the first spelling corrector by searching the word list for plausible correct spellings that differ by a single letter or adjacent letter transpositions and presenting them to the user. Gorin made the application publicly accessible and soon spread around the world via ARPAnet, ten years before personal computers came into general use. Its algorithms and data structures inspired the unix *ispell* program.

The first spelling checkers for personal computers such as "*Word Check*" appeared in the late 1980s [19]. These spelling checkers were standalone programs, many of them could run from within word processing packages on personal computers. However, the market for standalone packages was short-lived and later on the developers of popular word processing packages like *WordStar* and *WordPerfect* had incorporated spelling checkers in their packages. Eventually, these packages expanded support from English to European and Asian languages. However, this required increasing sophistication in the morphology routines of the software, particularly with regard to heavily-agglutinative languages.

In recent years, spelling checkers have become increasingly sophisticated; some are now capable of recognizing simple grammatical errors. However, even at their best, they rarely catch all the errors in a particular text and flag neologisms and foreign words as misspellings.

1.1. Motivation

The initiative for this thesis is observation. While moving across different places like government institutions, malls, cafeterias and others for personal cases, we observed tremendous errors in spelling. These errors are highly reflected on the banners and electronic screen shows installed in different areas of work. Moreover, different documents written in Afaan Oromoo language and so many speakers of this language are often exposed to spelling errors. It is this fact that drives us to conduct a research on this issue.

1.2. Statement of The Problem

The maximum percentages of native Afaan Oromoo speakers do not spell correctly. Only a small number of them spell correctly. To support this claim, we have conducted a simple survey on spelling errors in Afaan Oromoo language. Accordingly, we have randomly selected 100 Facebook posts in Afaan Oromoo from 100 different users. Additionally, a sample text file and banners written in Afaan Oromoo language were considered as we have attached a sample data under *Appendix I*. Moreover, we have also attached a detailed statistics of our survey under *Appendix II*.

Consequently, from the randomly selected 100 Facebook posts, 65 of the users misspelled some of the words and even some users don't spell correctly at all as we have shown in the attached statistics under *Appendix II*. This implies that if we take the ratio between those users spelling correctly and the total users, only 35% of them spell correctly from the findings of our survey. Similarly, we could also notice that some of the words in the text file and those written on the banner are wrongly spelled as we have attached the statistics along with a survey from Facebook posts under *Appendix II*. Such a problem is being manifested in different areas of working environments and other places.

Apart from the earlier mentioned case, we could identify a gap in a previous research done on the problem under investigation [1]. The paper aimed at designing a spelling checker in terms of morphology of the language.

But, as we stated in the introductory part of the thesis, the task of developing systems based the morphological analysis of Afaan Oromoo language is very challenging. Because, the structure of the language itself needs its own expertise and it will not be easy to build any system through a supervisory consultation of language experts.

Accordingly, designing morphology based spelling checker for a researcher having only the knowledge of technology would definitely be tough. Consequently, the researcher has to deal with each and every structure of the language and implementing all the rules governing it. Moreover, such a rule developing using linguistics knowledge would not be exhaustive. Hence, the intention of this thesis is to come up with another approach named dictionary based technique to address the problem which has no deal with the analysis of language structure.

1.3. Research Questions

After the completion of this thesis the following questions need to be answered:

- How to handle the errors committed in spelling Afaan Oromoo words?
- How would dictionary based n-gram approach can be applied to correct misspellings in Afaan oromoo?
- To what extent would dictionary based n-gram approach is effective for spelling error correction?

1.4. Objectives of The Study

1.4.1. General Objective

The general objective of this thesis is to design dictionary based spelling checker for Afaan Oromoo language.

1.4.2. Specific Objectives

In order to accomplish the general objective of this research, we will be targeting the following central points:

- ✓ To investigate spelling errors in Afaan Oromoo language.
- ✓ Exploring word structures in Afaan Oromoo language.
- ✓ Explore the attempts on handling spelling errors for morphologically rich languages like Afaan Oromoo.
- ✓ To come up with a solution to the problem under investigation.
- ✓ Designing and implementation of the proposed solution for the problem being addressed.

1.5. Study Methodology

1.5.1. Data Collection

We have collected data to build a test dictionary for performance evaluation of the developed system. Accordingly, to build the dictionary, different types of Afaan materials and resources were referred. Consequently, a test dictionary consisting of correctly spelled words from different domains of Afaan Oromoo language is prepared. The data are primarily collected from the following sources:

- ➢ Written documents in Afaan Oromoo
- ➢ Afaan Oromoo medias: streaming and social media

We have covered this section and study methodology generally; in further detail in chapter 3 of this work.

1.5.2. Implementation and Design Tools

In implementing the proposed solution, we have used Java programming language in designing prototype and developing the spelling checker application. Moreover, to design the system architecture, we have also used Microsoft Office Visio and some other painting tools as per their requirements.

1.5.3. Algorithm

We can declare that algorithm is the heart and central managing unit that lets the components of system architecture be functional. We followed a dictionary lookup approach and n-gram algorithm in detecting the misspelled word and providing suggestions. Furthermore, the dictionary acts as a point of reference in spelling error detection and generating the corrections. These error correcting suggestions should appear in accordance with their degree of similarity to replace the wrongly spelled word. To do so, we have employed the dice's similarity coefficient measure primarily in ranking the suggestions. Accordingly, the algorithm computes the dice's similarity coefficient of each of the correction candidates with respect to the wrongly spelled word. Consequently, the candidate with the highest dice's score will be at the top of the suggestions and possibly be the replacing target word for the misspelled word.

1.5.4. Evaluation Metrics

The testing of the system suggested as a solution for the problem under investigation is a mandatory task to determine the degree of its performance and the efficiency of the model we have followed. Moreover, evaluation also helps ensure the results obtained from the proposed approach to indicate the significance of the designed framework. Accordingly, to test the performance of our system, we have used the two popular information retrieval metrics namely: precision and recall. Furthermore, we have considered different parameters of measurement in evaluating the developed spelling checker in terms of both metrics. Consequently, we have obtained a promising result from the evaluation.

1.6. Literature Review

For better understanding of the problem area, different literatures were gone through. The detailed analysis of literatures helps researchers to get an insight of the existing problem and identify the gap. Furthermore, it also helps contextualize the techniques and methodologies employed by the scholars. This will in turn be important to come up with a suitable and better approach for the problem being addressed. Accordingly, the literatures and related works pertaining to spelling checker were thoroughly reviewed from the standing point of *local and foreign* languages. We have covered it in the next coming chapter.

1.7. Scope and Limitations

It is always important for a researcher to define a playing boundary. Because, it helps know and do the planned objectives only. Likewise, this thesis tries to cover the most widely or commonly words being used in the community speaking Afaan Oromoo language. Hence, we would be dealing with those commonly used words in writing Afaan Oromoo texts, documents and etc. Because, it is not always compulsory to include words used only at speaking level but not necessarily used in writing. However, preparing a dictionary representing all words from many domains is a difficult task. The first ranking limitation is a shortage of time to build such all-inclusive corpus. Moreover, this thesis covers only non-word errors and the concept of real-word errors will not be dealt with in this paper.

The other limitation is lack of well-organized resources. The materials written in Afaan Oromoo from different domains are not abundantly available. Additionally, some resources may not exist as required. For instance, materials written in this language regarding science and technology are very rare.

1.8. Significance of The Work

Hopefully, the thesis will have a great importance in reducing spelling errors in Afaan Oromoo and also helps those vulnerable to this problem. Most importantly, the prepared dictionary can be an input for other natural language processing applications. Once we have a well prepared dictionary (all-inclusive), it paves the way for other researcher to develop different Afaan Oromoo and other local languages processing applications in parallel. For instance, it is possible to design bidirectional dictionary and voice recognizing translation systems of Afaan Oromoo language in other local languages. Similarly, it can also play an important role in reducing spelling errors that occur during searching on the Web where Afaan Oromoo search engines could be used.

1.9. Organization of The Thesis

We organize and present the remaining sections of this paper as follows: in section 2 we will be presenting the general overview of spelling checker and related works done so far. Section 3 deals with the system design and methodologies. Section 4 presents implementation and evaluation of the system. Finally, Section 5 concludes the paper with some remarks.

Chapter 2

Literature Review

In this section we would be presenting the overview of spelling checker, the general background information of Afaan Oromoo word structures and related works in relation to spelling errors.

2.1. Overview of Spelling Checker

Spelling checker is an application that helps check the spellings of words in a text file and from user input to validate them. For instance, text editing tools like word processors have such built in functions. In case the spelling checker has doubt about the spelling of word(s), it suggests the possible correction alternatives.

The spelling checker provides two core functionalities. The first one is the capability of error detection. This is to identify and verify the validity of word(s) of a particular language. The second functionality is the ability of forwarding the possible correction for the misspelled word(s) of the language. Spelling checker can be standalone application capable of operating on a block of text and user input or as part of larger applications.

2.2. Error Occurrences in Spelling

It is our observation that spelling errors are committed in Afaan Oromoo language in similar analogy with other languages. These errors can be classified into three categories namely: character(s) omission, appending character(s) and character swapping. These categories of errors are the major and primarily committed in Afaan Oromoo language in addition to some contextual spelling errors.

Character(s) omission error is the most apparently noticeable spelling error in Afaan Oromoo language. Such error occurs when people forget and omit the necessary character(s) that must be included in spelling a particular word(s). Moreover, the number of missed or omitted characters depends on one's ability in spelling. This also holds true for errors occurring as a result of extra character addition.

Appending character(s) is another kind of error in which unnecessary extra characters are added in spelling a given word(s). It is also one of the errors made widely in Afaan Oromoo.

Character swapping error happens very rarely compared to other types of errors in Afaan Oromoo. This kind of error is made by those individuals who know the correct spelling of the word(s). But, such errors occur as a result of typing fast and it happens by suddenly interchanging any adjacent characters.

The following table summarizes the above explained three categories of errors.

Correct word	*Error types*		
Mootummaa	*Character omission*	*Appending character*	*Character swapping*
Resulting erroneous word	*Motummaa*	*Mootuummaa*	*Mooutmmaa*

Table 1: Types of spelling errors

According to the work of Damerau and Peterson [10, 11], spelling error can be categorized into two basic types. These are typographic and cognitive errors. Typographic errors occur when a writer knows the correct spelling of a word, but mistakenly types the word incorrectly. Cognitive errors are generated when the writer doesn't know or forgotten the correct spelling of a word. The authors further classified these errors into the following four different categories.

- Insertion
- Deletion
- Substitution
- Transposition

Insertion error happens when a single extra letter is added mistakenly to change string X to string Y, whereas deletion error take place in omitting a single letter that could change string X to string Y. In case of substitution error, the intended letter is substituted by another single letter resulting in a change from string X to string Y. Transposition error happens when the neighbouring two letters are interchanged producing another different string from the intended one.

2.3. Afaan Oromoo Word Structures

Afaan Oromoo uses Latin based script called "*Qubee*" and it has 26 basic letters (*A-Z*). Like in other languages, there are two sets of symbols or letters in Afaan Oromoo language. These letters are categorized as Vowels and Consonants. The vowel letters include (*a, e, i, o, u*) and the rest are considered to be consonant letters.

Apart from the 26 basic letters, this language has some additional consonant letters called "*Qubee dachaa*" which include (*CH, DH, NY, SH, PH*) and these are specific to the language. Therefore, Afaan Oromoo language has 31 letters in general.

Word formations in Afaan Oromoo follow the rule: *Consonant + Vowel* like that of English language [13]. However, there are many language specifics in Afaan Oromoo which we will be highlighting hereunder. Unlike other languages, having one or two vowels in between consonants convey a different meaning which we call "*jecha gabaabaa*" *(a word with single vowel next to consonant letter)* and "*jecha dheeraa*" *(a word with two vowels in succession next to consonant letter)* based on the number of the vowels a word is consisting of.

However, it is not allowed to have more than two vowels next to each other unless otherwise separated by an apostrophe (') called "*hudhaa*" in Afaan Oromoo language. For instance:

Jecha gabaabaa: Laga, meaning *river*, but

Jecha dheeraa: Laagaa, will have another meaning which means *esophagus*.

Similarly, we cannot have more than two consecutive consonants one after the other. A single consonant and two serial consonants in a word will produce two different meanings of that word. Such structure is called "*jecha laafaa*" and "*jecha jabaa*" respectively in the language.

For example:

Jecha laafaa: Madaa, meaning "*wound*", but

Jecha jabaa: Madda, expressing "*origin*".

Another specific characteristic of Afaan Oromoo language is the formation of words by putting apostrophes in between vowels. This structure is termed as words having "*hudhaa or irra butaa*" in Afaan Oromoo which means glottal words. We summarize some of them as shown in the following table.

Words in Afaan Oromoo	**English Meaning**
Du'e	*Dead*
Hir'ina	*Deficiency*
Re'ee	*Goat*
Har'a	*Today*
Baay'ee	*Many*
Dhangala'aa	*Fluid*

Table 2: Glottal words

As stated in many literatures, there are two basic types of morphology in the language [13]. The first one is *inflectional morphology* which makes a single word to have various forms of morphological nature and it makes the language to have many different variants of the same word. Different morphological forms of such words are used to indicate person, numbers, gender, tense or case.

This allows having many words in a particular language all sharing a single stem and appearing in different structure of a given sentence and at the same time each word altering the purpose of that sentence. Nevertheless, all variants of these inflected words will not alter a class of that word.

The other type is a *derivative morphology* which entirely changes the lexical meaning of a word through changing the class of that word. For instance, adjective or verb might be derived from a verb. Furthermore, Afaan Oromoo classifies functional words as prepositions, postpositions and article markers which are often indicated through affixes.

Additionally, conjunctions can separate words and some of them are affixed. Every word comprises morphemes. In Afaan Oromoo, there are two categories of morphemes: free and bounded morphemes. Free morpheme can stand as a word on its own whereas bound morpheme cannot occur as a word on its own. Roots (stems) are bounded as they cannot occur on their own.

Similarly, an affix is also a bounded morpheme that cannot occur independently. It is attached in some manner to the root of a word which serves as a base. Broadly, it is possible to categorize the major types of suffixes into three basic groups as: derivational, inflectional and attached suffixes. Afaan Oromoo attached suffixes are particles or postpositions like: *-arra, -irra, -itti* and others. Inflectional suffixes comprise the most frequent and dominant affixes such as: *-lee, -een, -oota, -wwan, -icha* and many more. Afaan Oromoo derivational suffixes include affixes such as: *-achuu, -eenya, -ina and –ummaa* are often used in the formation of new words following the roots or base of words in the language.

There are many ways of word formations in Afaan Oromoo language [13]. These formations can possibly be organized into six categories. These include: nouns, verbs, adverbs, adjectives, functional words and conjunctions. Almost all nouns in a given text in this language have person, gender, number and possession markers which are concatenated and affixed to the stem.

Verbs are inflected to indicate gender, number, tenses and others. Adjectives in this language are also inflected to express gender and number. Moreover, adverbs can be classified into adverb of: time, place and manner in which some of the adverbs are affixed.

For the purpose of relevance and structural complexity of the language we will be giving an emphasis on some particular category of words. Accordingly, we would be briefing on some words expressing:

- Noun inflections
- Gender
- Definiteness (cases) and
- Derived nouns in Afaan Oromoo language.

2.3.1. Noun Inflections

Afaan Oromoo nouns are words used to name any categories of things, people, places or ideas. Nouns are inflected to indicate different grammatical functions such as: number, gender and definiteness or cases. Inflectional suffixes are combined with stem usually resulting in a word of the same class as original stem. There are various ways of expressing plural numbers in this language. This is accomplished mostly by adding suffix at the end of nouns. Different suffixes are added to different nouns to express pluralism.

According to [13], in comparison with the English plural markers (s or es), there are more than 12 major and very common plural markers in Afaan Oromoo nouns. For instance, some plural markers include: *-oota, -wwan, -lee, -een,* etc. The table below shows few examples of these.

Suffix	**Afaan Oromoo**		**English Meaning**
	Singular	Plural	
-oota	Nama	Namoota	*Men*
-wwan	Sa'a	Saawwan	*Cows*
-een	Gaara	Gaarreen	*Moutains*
-an	Ilma	Ilmaan	*Sons*
-olii	Haadha	Haadholii	*Mothers*

Table 3: Plural nouns with suffixes

2.3.2. Gender

There are also morphemes that indicate gender variations based on the suffixes attached to the noun. Although, there are no clear categories of such suffixes for marking gender, some common suffixes are agreed up on by the linguists. For instance, *-aa* is attached for *masculine* and *–tuu* for feminine as in barat***aa*** expressing *male student* and barat***tuu*** indicating *female student*. Additionally, we can express barsiis***aa*** representing *male teacher* and barsiis***tuu*** showing *female teacher*. Similarly, in ogees***sa*** revealing *male expert* and ogeet***tii*** indicating female expert as well.

Natural female gender grammatically corresponds to feminine. The sun, moon, stars and other astronomic bodies usually expressed in feminine gender. Furthermore, in some Afaan Oromoo dialects, geographical terms such as: towns, countries and others are considered as feminine. Example:

- Magaalli Jimmaa gudda***tte***, meaning *Jimma town is developed*.
- Aduun baa***te***, meaning *the sun is risen*.

Similarly, the same holds true for some other geographical terms taking masculine behavior.

2.3.3. Definiteness

Afaan Oromoo has no indefinite articles that correspond to English language i.e. ***a.*** However, definiteness similar to English '***the***' can be expressed through different ways in Afaan Oromoo. For instance, different authors of Afaan Oromoo books like [13], shows that definiteness (expressing cases) could be indicated by attaching suffixes like *–icha* and *–ittii* for expressing different cases. Vowel endings of nouns are dropped before adding these suffixes as shown in the table below.

Afaan Oromoo nouns	**English meaning**	**Definiteness in Afaan Oromoo**	**English meaning**
Mana	*House*	Manicha	*The house*
Nama	*Man*	Namicha	*The man*

Table 4: Definiteness expression

Definiteness expressions can be classified into the following three major categories.

- *Instrumental case:* instrumental cases in a language are used to indicate the means by which something is done (*by*), the tool used (*with*), the actors who did the action (*by whom*) and many more. Such cases are expressed in Afaan Oromoo with the help of suffixes like *–n, -iin, -aan, -an, -dhaan, -tiin* and others based on the number of vowels and consonants the noun is ending with.

Example:

Miilaan dhufe meaning *he came on foot*.

Gareedhaan hojjedhaa meaning *do in group*.

Konkolaataadhaan dhufte to say *she came by car.*

- *Locative case:* the locative case is used for nouns that represent general locations of events or states. Usually, Afaan Oromoo uses prepositions or postpositions for this purpose. These locative case expressions are formed by adding the suffix *–tti* at the end of words.

  For instance:

  Manatti indicating *at home*.

  Dugdatti meaning *on his/her back*.

  Bukkeetti to say *on his/her side.*

- *Ablative case:* we use ablative expressions to represent the source of an event. It closely corresponds to English *'from'.* Such cases also use different suffixes that are appended to the nouns. These ablative suffixes are attached based on the number of vowels found at the end of that noun which are shown as follows.
  - ✓ When a word ends in a short vowel, this vowel would be lengthened. For example, consider the words *biyya* and *biyyaa*. The first word represents English meaning *country* whereas the second word means *from country*. In similar way, the word *keessa* indicates *inside* and *keessaa* stands for English equivalent meaning *from inside.*
  - ✓ When a word ends in a long vowel, the suffix *–dhaa* is added on that particular word. For instance, the words *Finfinnee*(*Finfine*) and *Gabaa*(*market*) can express definiteness by adding the earlier mentioned suffix as: *Finfinneedhaa*(*from Finfine*) and *Gabaadhaa*(*from market*) respectively.

In addition to the above scenarios, an alternative way for *ablative expression* in Afaan Oromoo language is the postposition *irraa* which means *from*. The initial vowel may be dropped from it in the process. For example, *konkolaataa*(car) would be *konkolaataa irraa* or *konkolaataarraa* both expressing the same case i.e. *from car*.

2.3.4. Derived Nouns

Afaan Oromoo is very productive in word formations by different means. One of the methods is the use of different derivational suffixes. Derivational suffixes enable new words that are often with a different grammatical category to be built from the root of a word.

The most common derivational strategies are the derivations which is done by adding suffixes like *-eenya, -ina, -ummaa, -annoo, -ii, -ee, -a, -iinsa, -aa,-i(tii), -umsa, -oota, -aata, and –ooma* to the word.

As explained in [6], nouns can be derived from a noun itself by adding appropriate suffixes to the noun stem. The following table summarizes some of these.

Noun	**Derived nouns**
Goota	*Gootummaa*
Beekaa	*Beekumsa*
Saba	*Sabummaa*
Lammii	*Lammummaa*

Table 5: Derived nouns from noun

In addition, nouns can also be derived from verbs by adding one of the above suffixes. For instance, from the verb *rakkate,* it is possible to derive nouns like *rakkina, rakkataa* and many more others can be mentioned. Furthermore, like in other languages, compound nouns are common in Afaan Oromoo as well. In such cases, two independent nouns are compounded to form a noun having a single meaning. These two separate words are treated as a single noun even though they seem a phrase. Some of them are summarized in the following table.

Compound words	**English meaning**
Mata duree	*Headline*
Haal-duree	*Precondition*
Abbaa manaa	*Husband*
Abbaa gadaa	*Leader(in traditional Gadaa system)*
Seer-luga	*Grammar*

Table 6: Compound words

2.4. Related Work

Under this section we will be covering some of the relevant and related works done in relation to the problem area under investigation. Accordingly, we would be dealing with the works done both from local and foreign languages perspectives. We would explore the approaches adopted by the researcher and the challenges faced thoroughly.

2.4.1. Local Language Works

Literature reveals that spelling checker has been designed for Afaan Oromoo based on the morphological structures of the language. One of the local works related to spelling checker is the work of Gadisa Olani [1]. The author proposed the *design and implementation* of spelling checker on the basis of Afaan Oromoo language morphology.

The paper presented the approach named as *dictionary lookup* for the *root of* words and affixes with *morphological rules*. The paper also indicates that study is done to analyze the spelling error pattern of Afaan Oromoo by selecting a module prepared for teaching this language. The author used text analysis data gathering technique and the finding depicts that 1,342 words are misspelled. The study further indicates among those 1,342 words, 1,287 words were in the category of non-word errors. This paper classifies the errors into eight different classes. For instance, valid prefix, invalid root and invalid suffix, which means there would be eight classes of errors like this. In this work, the rules developed and the *Levenshtein Edit Distance* techniques are used as an approach for error correction.

The author also raised the challenge in his study. This challenge is the morphological richness and complexity of Afaan Oromoo language that is found to be difficult to develop complete spelling checker. Due to this fact, the author recommended that it has to be studied linguistically. Consequently, we could determine that the structural convolution of Afaan Oromoo creates a challenge to design a comprehensive spelling checker for this language. Accordingly, it is from this fact that we are suggesting a *dictionary based n-gram* approach to design a spelling checker for Afaan Oromoo. Hopefully, this method will reduce the task and challenge arising from the morphological richness of this language.

Because, our proposed method doesn't consider the analysis of language structure as we would be storing the correctly spelled words of this language in the system regardless of their formations and structures thereby generating the corrections for the misspelled words from the list of stored valid words.

Evaluations showed that the implemented system is tested using three metrics namely: lexical recall, error recall and precision. Accordingly, the results indicate that the score of the three metrics are 88.62%, 100% and 28.62% respectively.

The other Literatures also indicate that a research is done on spell checking for Amharic language which is one of the local languages dominantly used in Ethiopia. One of these literatures we would be discussing related to the problem being addressed is the work of Andargachew M., Andreas N and Binyam Ephrem [2].

This paper presented the idea of automatic spelling corrector for Amharic language in which the authors proposed a *corpus-driven* approach with the noisy channel for spelling correction. According to the approach proposed in this paper, to detect the error word, the input word would be checked in the term list which is compiled from text corpus and if the word is not in the compiled term list then it is considered as a word having spelling error. Candidate corrections that are closer to the misspelling are generated from the term list.

As stated in the paper, the authors used Damerau-Levenshtein edit distance to measure how closer the candidate corrections are to the misspelling. Accordingly, all the words in the term list having one up to two (1-2) edit distance from the misspelling will be selected. Then the candidates are scored and ranked according to their prior and likelihood probabilities with the help of noisy channel approach. In case there is no candidate correction, the misspelled term will be broken down. This is needed to correct the misspellings resulting from missed spaces between words.

For evaluation purpose, different word lists compiled from Contemporary Amharic Corpus (CACO) were used. Moreover, the paper mentioned precision, recall and F1 measure as the underlining evaluation metrics.

Accordingly, the evaluation results revealed the score of 89.4%, 80.6% and 84.8% respectively for the earlier mentioned metrics. Finally, the authors claimed that the proposed approach showed the outperforming result when comparing it to some of the baseline system like Aspell, Hunspell. Additionally, considering the correct spellings that appear at the top of first suggestions list, the suggested approach that uses the CACO corpus scored 9% higher than that uses HaBiT corpus.

The other research done from viewpoint of local works in relation to spelling checker is the work of Mekonnen Fentaw [3]. The paper presented the design and development of Amharic spelling checker based on morphology of the language.

The author employed the approach named as Xerox Finite State Transducer (XFST) for the development of morphological analyzer. The author further mentioned the importance of XFST in natural language processing and discussed two benefits in analyzing the morphology of Amharic language by using XFST analyzer. The first stated benefit was that XFST based morphological analyzer demands small amount of memory and processor speed for computation and its ability to handle concatinative and non-concatinative morphotactics, compactness, high speed and efficiency. Second, XFST based morphological analyzer is bidirectional which makes it suitable for morphological analysis and generation. Accordingly, the paper has dealt with Amharic language morphologies. Moreover, the author also used a rule based approach in which the rules are developed to govern the inflection and derivation of Amharic words based on inflectional and derivational behaviors. Consequently, the paper discussed the rules by dividing into *affixation* and *alteration* categories.

To evaluate the effectiveness of the designed spelling checker, the paper prepared two experimental cases where the first case consists of 150 words and the second case 75 words. Accordingly, by using *precision and recall* measurements, the experimental results indicate that the system has scored a precision of 96% and a recall of 55.17% for the first case. Similarly, the performance result obtained from the second case indicates a precision of 100% and a recall of 78.7%.

However, in this paper we could observe that the developed system fails to detect some misspelled words due to the incomplete development of rules governing all morphologies of the language. Additionally, rules developed for Amharic morphology cannot apply in some cases for other languages like Afaan Oromoo due to a distinct structural difference between the languages. For instance, in Afaan Oromoo there is a case that does not differentiate in a word while expressing a gender; we may use similar word indicating both gender.

Example:

- *bilbiliif (call him/her)*: this word applies for both male and female in Afaan Oromoo. But, such a case will not hold true for Amharic language as we use "ደውልለት" for male and "ደውልላት" for female. Therefore, we cannot necessarily develop similar rules for Afaan Oromoo and Amharic using XFST as we would not always have similar network diagram resulting from XFST analyzer.

Moreover, the author also mentioned that the system fails to recognize some words transcripted from foreign language like English and technology terms. But, such scenario will not be a problematic in the approach we have followed.

The other work in relation to spelling errors is also conducted for Amharic language based on its orthography [4]. The paper discussed a *Metaphone algorithm* which has been highly successful for English language. This algorithm uses phonetic encoding to simplify words prior to comparison. Accordingly, the author has modified this algorithm to the way it can work for Amharic language. This is because; the two languages (English, Amharic) differ in their alphabetical usage. Consequently, the paper presented the simplifications of some letters which are used redundantly in Amharic language to their phonetically equivalent syllables to reduce errors happening as a result of redundantly used symbols. Moreover, the paper has shown Amharic Metaphone algorithm through encoding and simplifications of words phonetically by employing the procedures (rules) of: simplification, vowel removal, checking phonological problems, checking for glyph mismatch and checking input method problems. To evaluate the algorithm, a test was conducted on 116 words with their corresponding misspellings. Accordingly, 90% of the words were found matching the misspelled words by using Amharic Metaphone.

Furthermore, the author compared Amharic Metaphone with a double Metaphone by using transliteration systems in which Amharic scripts are stored in English representations. Consequently, the matching results indicated a lower score for the transliterated system with double Metaphone. Later on, Amharic Metaphone results were improved by applying standard Metaphone rule of treating 'w' and 'y' (analogous to Amharic ዉ and ዩ) under vowels rules resulting in a success rate of 96% matching words were found from the test set. However, in this paper it is possible to imagine that how many possible phonetic encoding will be taking place only for a single word by using the above stated 5 rules and we believe that this will result in a larger computation for languages like Amharic.

2.4.2. Related Works on Foreign Languages

Based on the literatures reviewed, different works have been done regarding spelling checker for many languages globally. We would be discussing some of the attempts hereunder.

The first attempt we would be highlighting is the work of Youssef Bassil and Mohammad Alwani [5]. The study was conducted on *context-sensitive spelling correction* by using Google Web 1T 5-gram information. The authors proposed a new context sensitive spelling error correction method for detecting and correcting non-word and real-word errors in text documents. Fundamentally, the paper came up with a methodology which is a blend of three algorithms each having a particular purpose and task. The task of the first algorithm was to detect non-word errors using Google Web 1T unigram data set (a subset of Google Web 1T 5-gram data set). The second algorithm was to generate a list of candidate spellings for every detected error in the text using Google Web 1T unigram data set and a character based 2-gram model. The third algorithm performs context sensitive error correction and select the best appropriate spelling candidate using 5-gram word counts from Google Web 1T 5-gram data set. Accordingly, the candidate that belongs to a particular sentence with the highest count or frequency in Google Web 1T 5-gram data set will be selected as a replacement for originally detected error word.

The evaluation results in this paper indicate that 99% of non-word errors and 70% of real-word errors were corrected successfully.

The authors further added that as a result of the proposed methods, the average evaluations score shows that around 93% of total errors were corrected from the total number of identified errors in the text.

Finally, the paper claimed that the proposed method was able to detect and correct 2.4 times more errors than the previous works done. The authors mentioned the major reason behind this result is the integration of Google Web 1T data set into the proposed algorithm as it embraces a wide-ranging set of words and precise statistics about word associations that cover domain specific terms.

According to [6], study is also conducted on spell checking in which the authors proposed a method named as *dictionary clustering* approach. This paper is targeted at reducing the number of times distances have to be calculated when finding the target words for misspellings. The introduced method combines the application of pattern initialization and partition around medoids (PAM). The authors presented this technique to serve as an alternative to the Finite State Automata (FSA) based dictionaries that reduce the number of distances to be calculated for each misspellings and thereby improving the processing speed.

The partition around medoids (PAM) algorithm divides a data set into a given clusters. Each cluster would be represented by a medoid. PAM creates compact clusters by iteratively minimizing the distances between medoids. The authors used pattern initialization and partition around medoids (PAM) method as described below.

- Applying pattern initialization to the dictionary, finding the number of clusters and a set of initial medoids.
- Using the set initial medoids and applying PAM to the dictionary to find clusters. This would produce a final set of medoids.
- Given a misspelling word w, the algorithm calculates its distance to each medoid in the final set of medoids.
- The medoids that have the distance to w equal to the minimum found plus a constant c would be saved to the final set of medoids. They added this constant c to increase the chances of the algorithm finding the target word. In their experiment, they used this constant by setting it to c = 1, along with *Levenshtein* distance.

- Finally, the algorithm calculates the distance between w and each word in the clusters represented by the medoids in final set. This would result in outputting the words having the minimum possible distance to w.

In evaluations, the authors used a dictionary containing list of words extracted from a Birkbeck corpus to test the performance of the proposed methodology. The result indicates that 88.42% of the cases, the method returned a cluster containing the target word or a word with smaller distance to the misspelling. Moreover, it is stated that as this result has shown a performance improvement of 11.18% increase relative to the previous works done.

Another research done on spell checking is the work of Hasan M. and Rasha Al-Tarawneh [7]. This paper presented the study of spelling checker for Arabic language based on n-gram scores. Basically, the authors implemented this method by employing a *matrix* approach. Accordingly, eleven matrices are built, each of the matrices having the dimension of 28 by 28. The eleven matrices are built on the ground that the longest valid Arabic language word has 12 letters. Therefore, these matrices represent all the words in a particular corpus.

The matrix approach deals with each word within the text separately and extracts the 2-gram set for it by assigning value 1 or 2 according to the following rules stated in the paper:

- If the item is the last one in 2-gram set, then the value 2 is stored in the corresponding matrix and it indicates that this word is ended by these two letters.
- If the item is not the last one in 2-gram set, then the value 1 is stored in the corresponding matrix and it indicates that this word is not ended by these two letters.
- When the value stored in the corresponding matrix is zero for the item in 2-gram set, then the spelling checker considers the tested word as wrong and colors it by red.

For evaluation purpose, the authors adopted the Muaidi corpus as a dataset to test the performance of the proposed spelling checker. They divided the dataset into training and testing datasets. The training dataset is used to build the matrices and the testing dataset is used to evaluate the performance of the spelling checker. Furthermore, the authors used the testing dataset to measure the accuracy of the system. Specifically, the success rate (S_R) measure is used in calculating the accuracy of the spelling checker as it is compatible with the proposed approaches. Accordingly, the results from the two testing stages indicate that the system scored the overall performance of 98.99% accuracy.

However, it is explained in the paper that the approach followed sometimes fails to detect the misspelled word. Such a scenario happens when the index value of the matrix is not having zero value for the invalid word. In such a case the spelling checker considers a particular wrongly spelled word as a correct word.

The work of Yilmaz Ince [8], has also came up with a paper focusing on spell checking and correction for Turkish language. Turkish language is also known by its agglutinative nature. The author has followed the methods named as n-gram and edit distance to design and implement the spelling corrector for Turkish that bases on the language structure. Accordingly, the author used a corpus labeled as nZemberek for spelling error checking and correcting of Turkish words. The nZemberek corpus contains a letter file, suffix file, root words and special case tags. Additionally, the corpus has a suffix production class, a class for root word special cases, helper for parsing operation, class containing basic information about the language specific classes and optional syllable finder. Consequently, a Turkish morphological analysis has been completed with the corpus according to Turkish suffix order rules.

In this paper, n-gram method is used in order to learn the rules of letters of a word sequence on the whole words. As a result, the misspelled word length is determined and the value would be saved as n value. Accordingly, n-grams, (n-1) grams and (n-2) grams were used as attributes of the text that are based on the misspelled word, length n. Similarly, edit distance algorithm is used in selecting the suggestion candidate words from the corpus where a threshold value is determined to be 2 in order to select the neighbouring words. The neighbouring words are found by the help of nZemberek direct acyclic word graph tree. Consequently, the candidate words selected by the edit distance algorithm with the nearest 2 distances were added to the suggestion list thereby sorting according to the frequency of the candidate words.

Moreover, the evaluation results indicated that the algorithm shows 95% of realizing spell and suggests the right candidate correction with 86% success rate. The author also explained that this paper is the first work implementing the earlier stated methods and scoring a successful result.

Chapter 3

System Design and Methodology

This chapter focuses on designing the system architecture of the spelling checker and the underlining methodologies.

3.1. System Design

Under system design subsection, we present the proposed architecture or model as a solution to the problem being addressed. We have designed a system incorporating different components, each having a separate and interrelated task. In the architecture, the central component playing a vital role is a *dictionary*. Consequently, the system uses this component as a reference both in detecting the spelling error of a word and generating the possible corrections for that particular word.

Moreover, the system is designed to operate on the inputs accepted from the user at word level. The system checks for the spelling of a word once we finished typing a particular word and hit the space bar. We built the system in such a way that to handle the errors during typing.

We have used different shapes in our architecture to represent system interface, processes, data storage and flow of operations. Bidirectional arrow is employed in the design of the system to indicate that there would be two operations between the two linked components. Both bidirectional and unidirectional arrows are used to reflect how the system works and the operational linkage between each of the components. Accordingly, we came up with the following system architecture to represent our proposed solution and would be presenting its detailed description of the components as follows:

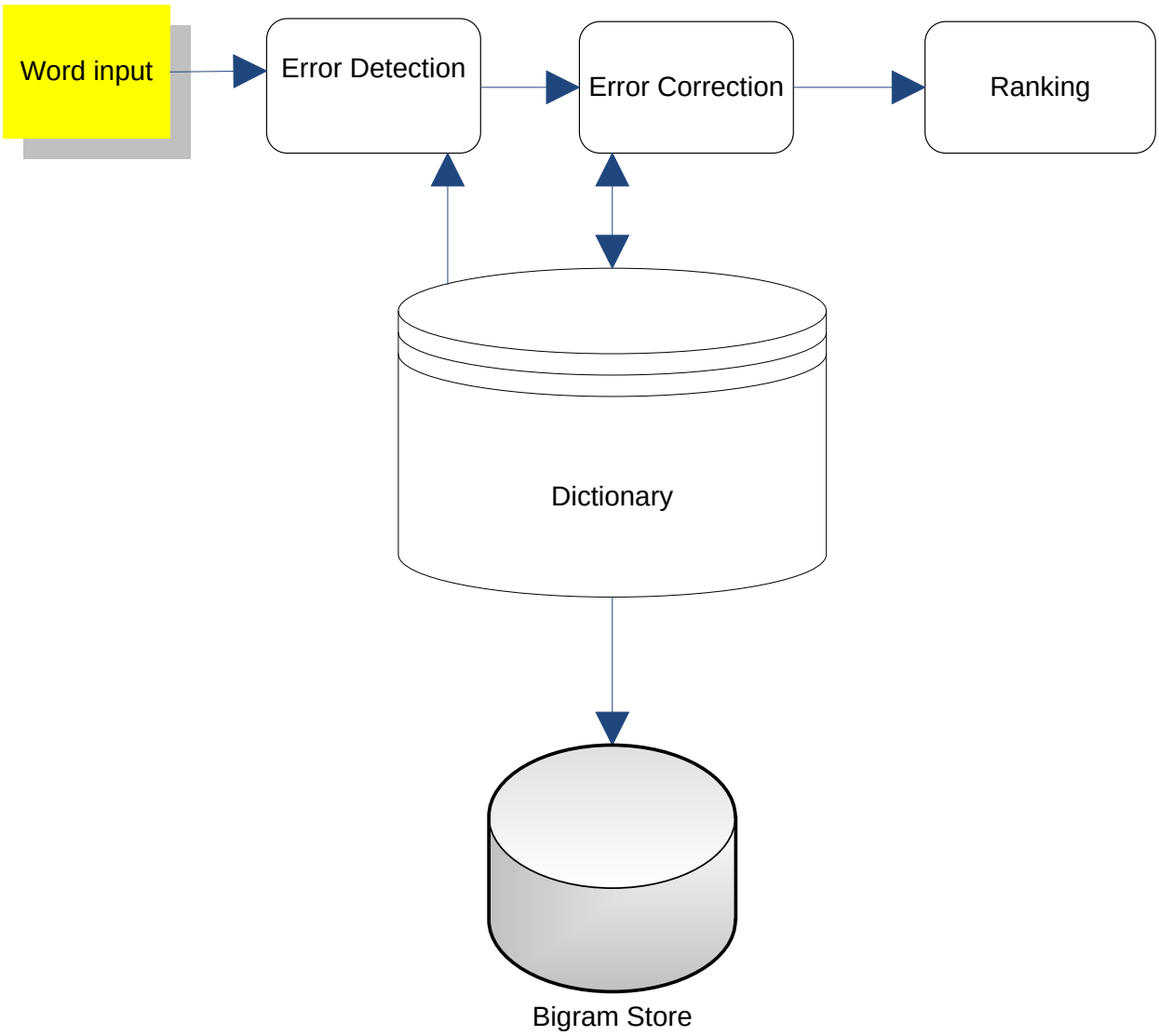


Figure 1: System architecture

The architecture consists of six basic components namely: word input, error detection, dictionary, bigram store, error correction and ranking. The following descriptions illustrate the details of each of the components.

i. *Word input*: this component reads the input word from the user which is to be checked for the spelling error. This unit acts as a representative for user interface of the designed system. Moreover, it passes the word accepted from user to the next component (i.e., error detection) of the system to identify weather the word is correctly spelled or not.

ii. *Error detection*: as the name indicates it is used for error lookup of the input word(s). It checks weather the word is valid or not in reference to the designed dictionary. This component looks for the existence of a particular word in the prepared list of words (i.e., *dictionary*) to verify that the word is either valid or invalid. If a word accepted from the user is found in the list of words, then this component recognizes that particular word as correctly spelled and does nothing. However, if the accepted word does not exist in the dictionary, then it marks that word as misspelled by underlining with red zigzag line which is shown in the implementation prototype. After marking the word, it passes to the error correction component as the task of generating possible suggestions is given to this unit.

iii. *Dictionary*: is the collection of correctly spelt Afaan Oromoo words. This component is designed to cross-check the spelling of input word(s) from the user. It is the frame of reference both for error detection and error correction components. These two components refer to the dictionary to validate or provide suggestions (erroneous word) for the input word from the user. We built it by collecting correctly spelled words from different domains that are mentioned under the methodology section of the thesis. These collected words are those words we commonly use in writing Afaan Oromoo documents, texts and etc. Because, as we stated earlier under scope part of this work, there are some words that we do not use in writing. However, such words are used at spoken level than in document writing in Afaan Oromoo language. For instance, the words '*bicuu'* (*small*) and '*xiqqaa'*(small) have similar meaning. But, the second word is the most commonly and preferably used in writing than the first word which we use at speaking level. Therefore, the claim here is that our dictionary includes words we always use in writing than those words used at speaking level. But, what matters is that if the corpus's inclusion percentage of words we do not use in writing is higher than we commonly use in writing, then it results in performance degradation of the system. For example, reconsider the earlier mentioned case: the words '*bicuu*' and '*xiqqaa*', if we had the first word in our dictionary and the second word is missing, then the user will be missing the word that is preferably used which should not be the case. However, we can store both words in the dictionary.

But, this would definitely result in the bigger size of the dictionary and the problem would be the question of memory management and it is easy to imagine how many millions of words should be stored. Therefore, we came to deal with the dictionary of commonly used words in writing Afaan Oromoo.

iv. *Bigram store*: this component is a subcomponent of the *dictionary* unit where the bigrams of the words are stored. The stored bigrams are used in comparing the bigrams of the misspelled word from user input to identify the matching word from the dictionary in generating the suggestions for that particular incorrectly spelled word. Accordingly, the words having a common bigrams would be listed as the possible suggestion candidates.

v. *Error correction*: once the error is caught, it is important to provide the possible corrections for the incorrectly spelled word(s). Hence, this component suggests relatively correct words from the dictionary for the word(s) detected as wrongly spelled. Similar to error detection component, error correction component also uses the dictionary to generate the possible suggestion candidates for the word (s) detected as misspelled. To do so, this component compares the wrongly spelled word with respect to the words in the prepared list. Those words in the list having a matching bigrams would be stored as suggestions for the misspelled word. Once the suggestions are determined, the error correction component passes those suggestions to the ranking component to sort the provided corrections according to the similarity these suggestions have relative to the misspelled word. Another important issue is that we may not be able to prepare the entire list (i.e., dictionary) of Afaan Oromoo words within a defined short period of time. Therefore, from this fact it is possible to determine that if the correctly spelled word doesn't exist in the dictionary, the system recognizes that word as wrongly spelled which is in actual sense a false argument. For such a scenario, the system provides an alternative solution for the user. For instance, if a user has no doubt about the spelling of a word, then there would be the option to add that specific word to the dictionary. This in turn helps update and have the missing words in our list. We have used the bidirectional

arrow in our architecture pointing both directions between error correction component and the dictionary to indicate the mentioned alternative feature of the system.

vi. *Ranking*: in making suggestions for wrongly spelled word(s), the list of suggestions has to be arranged in accordance with the degree of similarity. This component is used to rank the given suggestions in descending (from high to low) order according to the percentage of similarity with respect to misspelled word. To rank the correction candidates, the dice's similarity coefficients for each of the candidates would be computed. This implies that the word having the highest percentage of similarity with the input word (wrongly spelled) gets the top priority relative to other words in the suggestions list. Therefore, suggestion ranking component does such a task. The general and detailed view of the approaches and algorithm demonstration is covered under the *methodology* section of this work.

3.2. Study Methodologies

Under this section we would be elaborating a thorough investigation of the approaches employed in error detection and correction of the misspelled word(s). Furthermore, the techniques of providing suggestions for the word marked as wrongly spelled and ranking those suggestions are discussed in detail. These subsection further deals with data collection and how we built a dictionary. Moreover, the algorithms we have used in error detection, error correction and ranking are also discussed explicitly and separately.

3.2.1. Dictionary Preparation

As the intention of this thesis is to design a dictionary based spelling checker for Afaan Oromoo language, accordingly we prepared a dictionary of correctly spelled words of the language from a collection of different domains. The prepared dictionary included the words from the following listed domains:-

- ✓ News
- ✓ Sport
- ✓ Politics
- ✓ Transport
- ✓ Culture
- ✓ Governance

- ✓ Tourism
- ✓ Geography
- ✓ Religion
- ✓ Medicine
- ✓ Science and technology
- ✓ Business and economics
- ✓ Arts
- ✓ Academics
- ✓ History

The words collected from the above mentioned categories are obtained from Afaan Oromoo sources like Facebook pages of streaming Medias, websites, popular personal blogs and others. Accordingly, we have accessed many sources to collect Afaan oromoo words that fall under the above listed separate domains.

For instance, we have collected words categorized under *Sport* domain from specific sources like OBN – Ispoortii, oroIsport. Similarly, words for the rest of other domains are gathered from the below listed sources:

- ➢ FBCAfan Oromo
- ➢ OBN Afaan Oromoo
- ➢ VOA Afaan Oromoo
- ➢ BBC News Afaan Oromoo
- ➢ Oromia Media Network
- ➢ GadaaUniverse Vision
- ➢ OROMO ICT
- ➢ www.fanabc.com
- ➢ www.oroict.com
- ➢ Oromia Bloggers Network
- ➢ Raadiyoo Daandii Haqaa
- ➢ Dr.Gurmeessaa (personal blog)
- ➢ Seenaa Gootota Oromoo fi Kaan (book)
- ➢ Kallacha Oromiyaa and many more

We have used the above listed sources to prepare a dictionary consisting of different variety of words for the earlier listed domains. In general, we have prepared a category of 15 different domains. Each one of the domains comprises 200 words and this has resulted in a total of 3000 words collected from the fifteen separate domains.

In fact this number is not much to declare that the dictionary is complete and a representative for the list of Afaan Oromoo words that we use in writing. However, the list is prepared from different domains for the purpose of evaluating our system.

3.2.2. Error Detection Approach

This method checks the correct spelling of input word(s) from the user with respect to the prepared dictionary. The algorithm follows a lookup technique for the input word to be matched with the one in the list. If it doesn't exist in the dictionary, then the algorithm detects and marks it as misspelled word by flagging with a wavy red line as shown under implementation prototype.

In checking the existence of the input word, possibly we can employ different searching algorithms. However, every searching algorithm is not efficient to accomplish and hit the target of the problem under investigation. This is due to the fact that searching algorithms depend on the size of the data and its structure. From these standing points, we choose a binary searching algorithm as long as the size of the dictionary is concerned.

Binary searching algorithm best fits for large data size. This algorithm plays important role in reducing the time and memory space in dealing with large volume of data. Accordingly, we would make use of this algorithm thorough out the thesis for error detection. As per binary searching algorithm, collections of words (i.e., dictionary) are arranged in their alphabetical order. Hence, to search for the input word to determine its spelling, the collections would be divided into left and right window (*LRW*) there by calculating the middle value. Once the middle value is known, and it is not the input word we are searching for, then the algorithm searches for the word in either of the two windows depending on the conditions. Therefore, at a time only one window would be active. This strategy results in the reduction of time and search space which the other way round indicates the effectiveness of the approach. We would represent this algorithm in figure as follows:

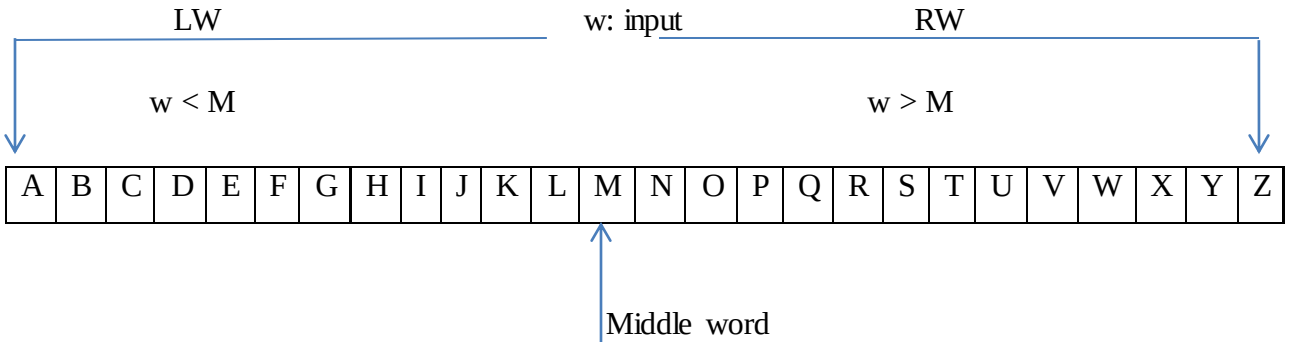


Figure 2: Error detection approach

By the definition of binary searching algorithm, one of the preconditions is always rearranging the list of words in their alphabetical order. In Figure 2 above, we have used the letters 'A' through 'Z' to represent the alphabetical order of list of words in our dictionary.

Accordingly, the letters 'A' and 'Z' denote the first and the last entries in the list we have prepared respectively. After such sorting, the algorithm divides the list into left window (LW) and right window (RW) there by finding the middle word (as shown in Figure 2) and such a process is repetitive throughout the algorithm depending on the conditions.

Furthermore, in checking the existence of a word (w in Figure 2) from user input, the algorithm starts by comparing it with the word at the middle (M from Figure 2) of the list. Moreover, the algorithm functions in similar analogy with string comparison. This implies that if the user input and the middle word are identical (i.e. return value = 0), then the input word is considered to be correctly spelled. When the user input is not similar with the middle word, the algorithm further checks in either left window (for w < M) or right window (for w > M) until it reaches the last possible index.

However, if the matching word is not found in the list of words, then the algorithm marks the user input as misspelled. After the word is determined as wrongly spelled, the next task of the algorithm would be generating the possible suggestions and we have explained this under *error correction approach* of this work.

The following pseudo code explains how error detection algorithm works.

```
read words // list of words from dictionary

arrayName [] = words

initialize first_index = 0

initialize last_index = size-1

declare middle_index

initialize positive = 1

initialize negative = -1

set found = false

read word // the input word to be checked

loop: while(first_index <= last_index)

{

  middle_index = first_index + last_index/ 2

 if(word.compareToIgnoreCase(arrayName[middle_index]) == 0)

{

   found = true

}



}
```

```
else if(word.compareToIgnoreCase(arrayName[middle_index]) >= positive)
{
  first_index = middle_index + 1
 goto loop
else if(word.compareToIgnoreCase(arrayName[middle_index]) <= negative)
{
  last_index = middle_index -1
 goto loop
}
else
{
  found = false
  mark word
}
}
```

Summary of the above pseudo code

To determine whether the word is misspelled or not, the algorithm starts looking up that particular word by dividing the list into two. Then, it compares the input word with the word at the middle of the list. As we have explained earlier, the comparison works in similar way with comparing the strings. Consequently, the algorithm considers three cases in comparing the input word with words in the list. The first case is checking if the user input is identical with the middle word and in this case the return value would be zero. The second case is by comparison, if the word from user input is less than the word at the middle index, the resulting return value is always a negative number. The third case is from comparison, if the user input is greater than the middle word, this time the return value would always be a positive number. Basically, the negative return values indicate that the user input precedes the word at the middle index alphabetically and hence the algorithm checks that particular word in the left window. Similarly, positive return values show that the user input comes after the middle word in terms of its alphabetical order and hence searching would be to the right of the window. Accordingly, If the return value is true (i.e. input word == middle word), then the word is found which means that it's correctly spelled and the algorithm does nothing. If the word is not identical with the word at the middle (i.e. return value is false, return value ≠ 0), it checks in the left or right side of the list depending on the following two conditions:

1) *If input word < middle word, algorithm checks in the left side of the list by further dividing into two. This time the first index of the list would be zero and the last index is the index just before the middle index.*
2) *If the input word > middle word, the algorithm checks in the right of the list and again it continues dividing. In this case, the first index is going to be the index just next to the middle index and the last index would be dictionary size – 1. These operations are repeatedly done until the last index is reached. Finally, if the word is not found in the entire list, then the algorithm marks it as misspelled.*

4.2.3. Error Correction Approach

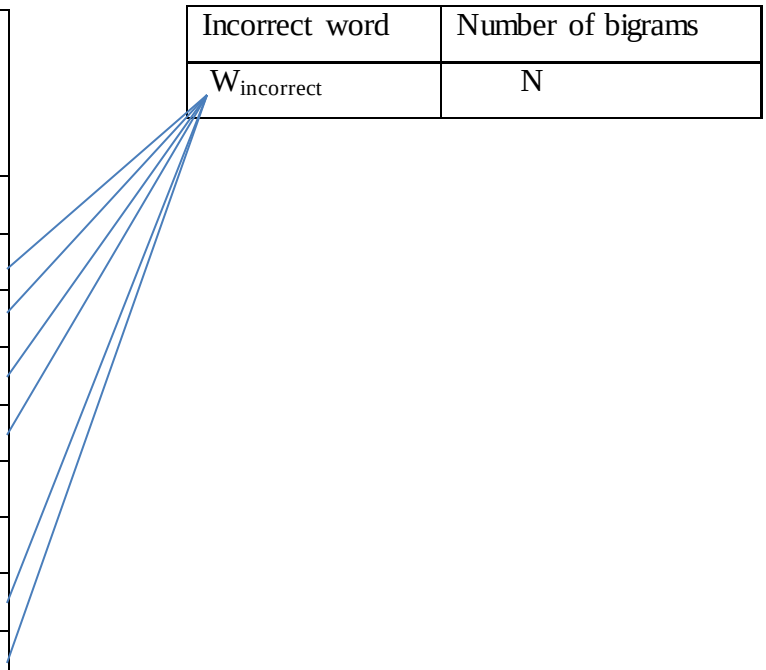
Error correction mechanism focuses on how to provide suggestions for wrongly spelled word(s). After the word is detected to be incorrectly spelled, the way out would be making a relatively similar lists of correctly spelled words from the dictionary as suggestions.

In fact there are different methods to make corrections for the word(s) identified as misspelled. The decisive task is to figure out the most efficient and effective techniques to handle the problem being addressed. In this thesis we would employ one of the n-gram approaches (i.e. bigram) in correcting the word(s) found as wrongly spelt. Accordingly, the dictionary is preprocessed and the bigrams of each word is stored. Furthermore, the misspelled words would also be broken down into their respective bigrams. This helps determine those words sharing common bigrams with the misspelled word. Later on, words that share common bigrams would be listed as the possible suggestions for the wrongly spelled user input.

Most importantly, it is not the total number of bigrams that matters in selecting the correction candidates. However, the deal is with the maximum number of common or shared bigrams to identify the suggestions. For instance, consider the words '*qorannoo' (research)* and '*kaleessa' (yesterday)*. These two words have equal number of bigrams. But, these words never share a common bigrams. Hence, the first word cannot be the correction candidate for the second word and the vice versa also holds true. Therefore, only words having similar or shared bigrams with incorrect word(s) are stored as correction candidates. These correction candidates share at least one bigram and n bigrams at most where n is the maximum number of shared bigrams.

The following tables illustrate a general fact behind the approach by considering symbols labeled from{w1,w2,…,w10} representing correctly spelled words in the dictionary and {$w_{incorrect}$} representing the misspelled word from the user. The column '*Words having intersection*' (Table 7) indicates the words that share a common bigrams with erroneous word (i.e. $w_{incorrect}$) which is indicated in (Table 8). Moreover, we have used the letter 'C' in our table to label *candidate* words sharing a similar bigrams and the (---) symbol showing words in the list that do not have a shared bigram. Consequently, it is only these words (labeled as C) that would be in the category of list of suggestions. Furthermore, the letter 'N' is also used to denote the total number of bigrams.

Correct words	Number of bigrams	Words having intersection
W1	N	---
W2	N	C
W3	N	C
W4	N	C
W5	N	C
W6	N	---
W7	N	---
W8	N	C
W9	N	C
W10	N	---



Incorrect word	Number of bigrams
$W_{incorrect}$	N

Table 7: Valid words

Table 8: Erroneous word

From the above tables we can clearly summarize the following facts:

i. The set of correction candidates are the list of words {w2, w3, w4, w5, w8, w9}. These words are the only suggestions that have a shared bigrams. Hence, the set represents the correction and possible replacements for wrongly spelled word $\{w_{incorrect}\}$.

ii. The remaining set of words {w1, w6, w7, w10} would not be in the category of correction candidates for $\{w_{incorrect}\}$. Because, these words do not have a common intersection with the erroneous word as shown with (---) line in the above table (Table 7).

iii. Among the set of {w2, w3, w4, w5, w8, w9}, the word with the highest shared bigrams is the most likely to replace the wrongly spelled word $\{w_{incorrect}\}$. The higher the number of shared bigrams (not total bigrams), the higher the probability of a word to replace the incorrect word.

For instance, from the correction candidates {w2, w3, w4, w5, w8, w9}, if w2 is having a greater common bigrams with $\{w_{incorrect}\}$ (Table 8) relative to {w3, w4, w5, w8, w9}, then it is the most likely word to replace the erroneous word. This would be possible by computing the similarity of each of the candidates with respect to the misspelled word based on the number of shared bigrams. We have explained this with demonstration example under *ranking suggestions* part of the thesis.

Writing error correction algorithm in pseudo code, we have the following:

```
for(i = 0; i <= arrayName.length() -1; i++  )

{

  declare String s

 s = arrayname[i]

loop: for(j = 0; j < s.length() -1; j++) // split every entry and store

{

  declare char c1, c2

 c1= s.charAt(j)

 c2 = s.charAt(j+1)

create Set Bigram1

Bigram1.add(c1 + c2)

}

for(k = 0; k < word.length() -1; k++) // split the marked word and store

{

  declare char c11, c22

 c11 = word.charAt(k)
```

```
c22 = word.charAt(k + 1)

create Set Bigram2

Bigram2.add(c11 + c22)

}

   create Set intersection

 intersection = new HashSet(Bigram2)

 intersection.retainAll(Bigram1)

if(intersection.isEmpty() == false)

{

  create Set suggestions

 suggestions.add(s)

}

else if(intersection.isEmpty() == true && i <= arrayName.length() -1)

{

  goto loop

}

else if(intersection.isEmpty() == true && i > arrayName.length() -1)

{

  save word to dictionary

}

}
```

Summary of the pseudo code

To generate the corrections for the wrongly spelled word from user input, first of all, the algorithm would break down the list of words (i.e. words from dictionary) into their respective bigrams and store it. Similarly, the algorithm also reads and splits the misspelled word from user input into its bigrams. Now, it checks the bigrams of the invalid word across the already stored bigrams of dictionary. Then, words having similar bigrams (intersection point) would be generated as the correction candidates. However, if no words are found having a common bigrams in the stored list, then the algorithm generates no suggestion for the input word. This time it should be a new word missing from our dictionary. Because, having all the words correctly spelled in the list, the algorithm always generates a suggestion for a particular incorrectly spelled word. In such cases the algorithm provides the other alternative for the user to add that particular word to the dictionary of words. This is helpful to add more missing words from the dictionary when the user is undoubted about the spelling of specific word.

3.2.4. Ranking Suggestions

This subsection deals with the mechanism that we have used in sorting the potential suggestions generated as correction candidates. In essence, all words having common bigrams should not be listed as a replacement for error correction. Moreover, the algorithm should not be generating the entire words (even words with the least similarity) that share a common bigram with the wrongly spelled word. For example, we could have 20 or more correction candidate words. Therefore, it is not important to display all the 20 candidates as a list of suggestions; rather it would be helpful to select only ***top highly similar*** words as replacements. Furthermore, these replacing words are ranked according to their shared bigrams (similarity) resulting in the most likely word to replace the misspelled word would be at the top of the correction list.

Accordingly, in ranking the potential suggestions, we would compute the similarity measure of words having a shared bigrams with respect to the word(s) detected as incorrect. Ultimately, we use the dice's similarity measure where the dice coefficients of suggestions are computed relative to the wrongly spelled word(s). The computation of dice's coefficients helps determine the percentage similarity of the candidate words to replace the word detected as erroneous. This implies that the word with the highest dice's coefficient is the most likely to replace the incorrectly spelled word from user input. The more shared or common number of bigrams the higher dice's coefficient (i.e. higher similarity) of a particular candidate word.

Furthermore, to compute the dice's similarity coefficients of the generated corrections, we need to count the number of common bigrams of each of the candidates with their corresponding total number of bigrams with respect to the misspelled word. Consequently, the dice's coefficient of specific candidate ($w_{candidate}$) is calculated as a ratio between twice of the common bigrams (C) and the sum of total number of bigrams of the candidate word ($Bcandidate$) and the wrongly spelled word ($Bincorrect$). Mathematically, it is expressed as follows:

$$Dice\ [w_{candidate}] = \frac{2C}{Bcandidate + Bincorrect} \quad \ldots\ldots (1)$$

Where, C stands for total number of common bigrams, $Bcandidate$ and $Bincorrect$ stand for total number of bigrams of the candidate word and the misspelled one respectively.

The following example demonstrates the above description of ranking algorithm through the calculation of dice's coefficients. In this example, we would intentionally consider one misspelled word and randomly select some words as correction candidates to show how the earlier formula works (equation (1)).

Illustration example:

Let the incorrect word be ["gaari"] and the correction candidates ["gaarii", "gaara", "gaarreen", "gaarri"]. Before computing the dice's coefficient of the selected candidates, first the number of common bigrams of each of the given candidates with respect to the misspelled word (i.e. gaari) would be identified and counted (we have done this by manual counting). Every intersection point of the candidate word and the wrongly spelled word would be counted as common bigram (s) as we have shown hereunder.

Bigrams of incorrect word ["gaari"] = ["'ga", "aa ", "ar ", "ri "],

Bigrams of candidate ["gaarii"] = ["'ga", "aa ", "ar ", "ri ", "ii"],

Therefore, common bigrams of candidate ["gaarii"] and incorrect word ["gaari"] would be the set containing {"'ga", "aa ", "ar ", "ri "}. This implies that the total number of common bigrams (C) between the two words (i.e. candidate and the incorrect one) is 4. Similarly, this applies for the remaining candidates.

Bigrams of candidate ["gaara"] = ["'ga", "aa ", "ar ", "ra "],

Bigrams of candidate ["gaarreen"] = ["'ga", "aa ", "ar ", "rr ", "re", "ee", "en"],

Bigrams of candidate ["gaarri"] = ["'ga", "aa ", "ar ", "rr ", "ri"].

For ranking these candidates, their dice's similarity coefficients are given below:

Dice ["gaarii"] = 2*4/5+4 = 0.89

Dice ["gaara"] = 2*3/4+4 = 0.75

Dice ["gaarreen"] = 2*3/7+4 = 0.55

Dice ["gaarri"] = 2*4/5+4 = 0.89

Accordingly, from the above computations the suggestions for the misspelled word [‘’gaari”] are ranked as follows on the basis of their percentage of relative similarity.

Incorrect word	Candidates	Dice coefficient (%)	Rank
Gaari	Gaarii	89 %	1
	Gaarri	89 %	2
	Gaara	75 %	3
	Gaarreen	55 %	4

Table 9: Ranking suggestions

The results indicate that the candidate word with the highest dice’s coefficient is the most likely word to replace the incorrect word. Moreover, the above calculation scores depict that two candidate words have the same dice’s coefficient.

However, such a scenario happens very rarely. Because, for two candidates to have the same dice’s score, always two conditions must hold true. The first condition is, the two candidates must have equal number of common bigrams with respect to the misspelled word. The second condition is, the two candidates again must have equal number of total bigrams. Therefore, for such a scenario to take place, the two explained conditions must be true at the same time. For instance, consider two candidate words [“gaara”] and [“gaarreen”] from the above computations. These two correction candidates have equal number of common bigrams with the incorrect word [“gaari”], but fail to have equal number of total bigrams (see Illustration example above). Thus, it has resulted in different dice’s score. This implies that either equal number of common bigrams alone or equal number of total bigrams of correction candidates doesn’t indicate equal dice’s score. Hence, the two conditions we have discussed earlier must be always true at the same time; which is a rare case.

Perhaps, if such a scenario happens, the possible way out would be sorting those specific candidates scoring equal dice’s coefficient based on their alphabetical order. Accordingly, words having similar score (i.e. dice’s coefficient) would be ranked one after the other as we did in the above table (Table 9).

The remaining words listed as suggestions would get the next priority (to be chosen as replacement) according to the intentions of the user.

In general, it is not necessary to generate all the words having a common similarity including those with the least shared bigrams as corrections. For instance, we can have 'n' number of words sharing a common bigram with the misspelled word; where some of the words possibly have the minimum shared bigram (s).

Example: Consider the word ["maaliif"] as a candidate and the earlier incorrect word ["gaari"] that we have discussed in our earlier illustration example; the new word ["maaliif"] shares only a sinlge common bigram (i.e. {"aa"}) with the incorrect word which indicates the lower similarity between the two words and hence the lower dice's score. Yet, according to our algorithm this word can be in the list of correction candidates. But, it is not logical to display such words showing lower percentage of similarity as a replacing candidate along with the candidates scoring highly similar dice's coefficient. Our claim here is that every word sharing a common bigrams should not be displayed as a replacement. This implies that the likelihood of those candidates with lower dice's score to replace the incorrect word is very low when compared to the candidates having higher dice's score.

Therefore, by eliminating words with lower percentage of similarity, it is more important to display top *highly similar* words having a tendency to replace the misspelled word(s). This can be done by selecting a certain number of words from the generated list of candidates. Accordingly, we have made our algorithm to display top ten words as the possible replacing suggestions. Furthermore, providing ten words as a suggestion could be a considerable number to meet the user's intentions *possibly*. In fact, it is impossible to determine the user's need to decide the number of suggestions we should be generating. Most importantly, it is clear that the words with lesser similarity score would not necessarily be a replacing candidate (as shown in the above example) and the defining point is the capability of the algorithm being able to put the most likely word to replace the incorrect word at the top of the listed suggestions. We have explained this suggestion ranking algorithm in pseudo code as follows:

```
declare TotComBigrams

declare result

TotComBigrams = intersection.size()

 result = 2* TotComBigrams/Bigram2.size()  + Bigram1.size()

create HashMap corrections

for(c:suggestions)

{

   corrections.put(s, result)

}

sort by value

for(Map.Entry<datat ype, data type> entry:corrections.entrySet())

{

  create HashMap sorted

  sorted=
corrections.entrySet().stream().sorted(collections.reverseOrder(Map.Entry.comparingByValue()))
.collect(toMap(Map.Entry::getKey, Map.Entry::getValue))

create List Top_10

Top_10 = new ArrayList<Map.Entry<data type, data type>>(sorted.entrySet().getKey())

}

for(Map.Entry<data type, data type> entry:Top_10.subList(0,10))

{


display entry } }
```

Pseudo code summary

To rank those candidates generated as suggestions for the misspelled word, first we would compute the dice's similarity coefficients of those candidates. Moreover, in computing the dice's similarity of each of the candidates, we need to count the total number of common bigrams of particular candidates with respect to the wrongly spelled word with their corresponding total number of bigrams. The total number of common bigrams of a given candidate relative to the incorrect word is obtained from the size of intersection points of the two words (i.e. candidate, misspelled). Additionally, the total number of bigrams of a candidate word and the incorrect one is given from their size of bigrams. Once we have counted the total number of common bigrams and total number of bigrams of each of the candidates, we would be calculating the dice's coefficients of the entire listed suggestions by using the formula we have given in equation (1) and store them as key and value pairs. In this case, key refers to the candidate word and value refers to its dice's coefficient. After storing in the form of key and value pairs, now we need to sort based on their values (i.e. dice's coefficient) to list them in descending order (from high to low) and this in turn helps rearrange according to their similarity leaving the candidate with the highest similarity score to be at the top of the listed suggestions. Finally, the algorithm finishes by displaying top ten words from the list as the possible suggestions.

Chapter 4

Implementation of The System

This chapter covers the implementation prototype, evaluation of the system and the results. The algorithms introduced in the methodology section of this thesis are implemented using java programming language. Accordingly, we developed Afaan Oromoo spelling checker application.

4.1. Implementation Prototype

This subsection presents the prototype (user interface) of the developed system and the way it works in error detection and correction of the word (s) from user input. We have used java programming language both in implementing our algorithms and developing prototype of the system. The system accepts a word from a user and checks for its validity and verifies accordingly. If the word from user input is valid, then the system recognizes that particular word as correctly spelled. If the input word is invalid, the system marks it as incorrectly spelled by underlining with a red zigzag line as shown in Figure 3. Alternatively, the user can look for suggestions by right clicking on the word detected as wrongly spelled. Accordingly, the system displays top similar words as suggestions for that specific word as we have shown in Figure 4.

Moreover, the system doesn't have a feature to check for the misspellings in a text file or document. This is due to the fact that the system is designed to detect and correct only those words from the user input specifically and it doesn't operate on browsed files. We have developed the system in such a way intentionally to handle spelling errors during typing and this could be a hands-on mechanism. Because, it is the reality that people always make spelling error while typing a given word, text, document and etc. Additionally, it also made possible in the reduction of language preprocessing like stemming, stop word removals, tokenization and etc. Because, if our system also includes checking the misspellings in a file, it would be necessary to deal with all the language preprocessing we raised earlier and other related language preprocessing.

The following user interfaces reflect spelling error detection and providing suggestions for the misspelled word respectively.

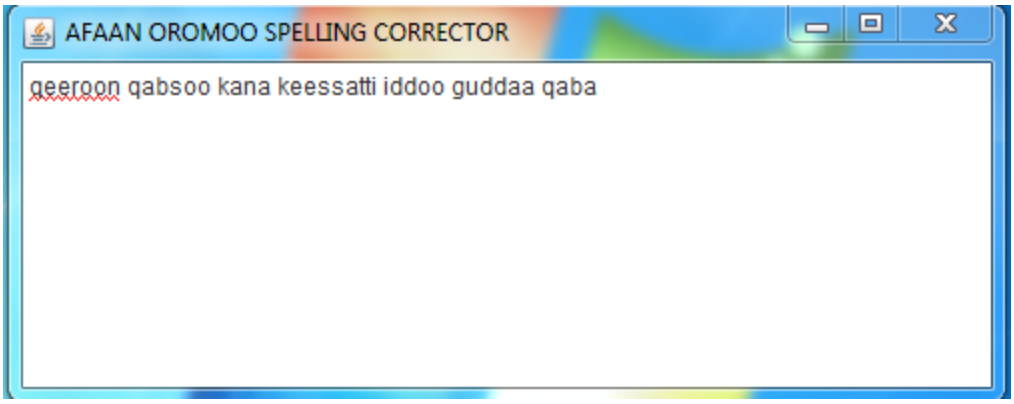


Figure 3: Error detection prototype

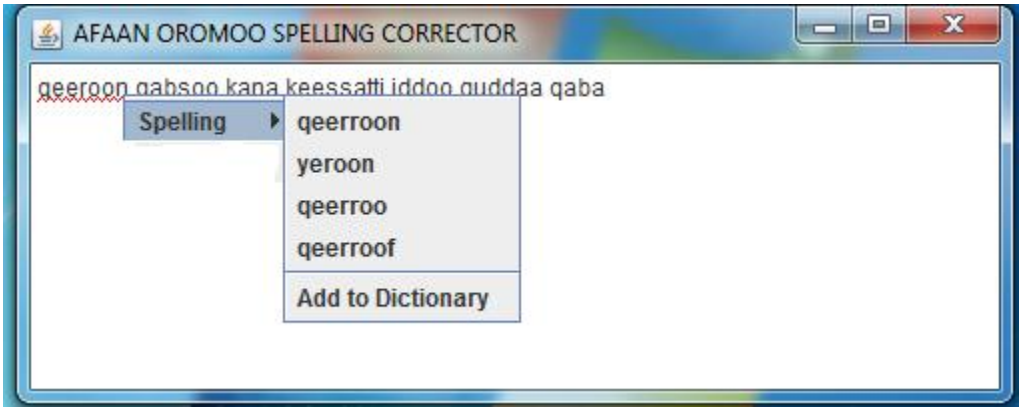


Figure 4: Error correction prototype

4.2. Performance Evaluation

We evaluated the performance of the spelling checker by using the IR (Information Retrieval) popular metrics namely: precision and recall measures. We would be discussing how the two mentioned metrics work in terms of spell checking and the possible procedures to be followed in evaluating the performance of the system. Furthermore, we would also be presenting the resulting scores of the system; computed by using the precision and recall measures.

4.2.1. Precision Measure

In measuring the performance of the spelling checker, we dealt with finding out the capability of the system in error detection or flagging the incorrect word(s) and thereby suggesting the possible corrections. Moreover, precision tells the possible suggestions which are the most likely to replace those words detected as incorrectly spelled.

Furthermore, precision measure indicates exactness of the spelling checker in catching up and responding to the erroneous words. For every single invalid word detected, there should be a correction generated by the spelling checker as a suggestion. This is to make sure that for every flagged word as misspelled; there must always be a correction for that particular word. For instance, if a user wrongly spelled one word from a given ten words, there will always be one correction to replace the misspelled word from the available generated correction options.

Similarly, if we have five incorrectly spelled words, the spelling checker replaces those flagged words with five separate correction words. In general, if there are 'n' misspelled words, accordingly the spelling checker would suggest 'n' different corrections. From this fact, clearly we can have a ratio of one to one (1:1) which results in 100% accuracy of the spelling checker in terms of precision measure reasonably. This could happen from the fact that given the entire list of words that are widely used in writing Afaan Oromoo, for a particular misspelled word of which the intended word exists in the prepared list, then there would always be a suggestion generated by the system. For instance, if our list contains correctly spelled words: "seenaa", "aadaa", "maatii", etc.; and a user mistakenly types "seenaa" as "seena" which is wrongly spelled in this case, then it is always true that the spelling checker provides a suggestion from the given correctly spelled list of words. Accordingly, the word "seenaa" would be the best suggestion for the earlier mentioned misspelled word (i.e. "seena") from those words in the list we have given above and this is due to the fact that it ("seenaa") is the most highly similar word among the provided words. This implies that for every misspelled word there should be a target correctly spelled word (s) in the dictionary which is (are) likely to replace it. Moreover, the dictionary must be inclusive of the words from different domains. For instance, if the words from *Sport* domain are included, a user should not miss words from *governance* domain and etc. we would be explaining this idea further under *recall measure*.

4.2.2. Recall Measure

In evaluating the performance of the developed application in terms of recall measure, we deal with its completeness and language covering from different domains. Additionally, apart from the inclusion of words from separate domains, the system should further cover all the terms within a specific domain. For instance, if some terms are missing from a particular domain, definitely it affects the performance of the system and we would be showing this in our evaluation procedures. Most importantly, the terms included from different domains should be the words we often use in writing Afaan Oromoo; but not those words which are used only at speaking level as we have explained it under system architecture of this work (chapter 3). Because, the inclusion of such words (words used only at speaking level) in place of commonly used terms also affects the performance of the system. This implies that a user will be missing those words which are widely used in writing other than the terms used only at spoken level and this should not hold true.

Furthermore, the system has a capability of detecting and verifying any Afaan word regardless of its frequency of use in writing. But, the claim here is that if we ignore those words which are not commonly used in writing, it is possible to imagine the amount of memory we would be saving.

It is also important to put into consideration about novel words to check if the application can guarantee those missing from the dictionary. This is to reflect the fact that it would not be easy to include each and every word used in Afaan Oromoo language in the dictionary. Because, the vast nature of Afaan Oromoo words from one place to the other across the region disallows to prepare the entire list and a representative dictionary of the language. Moreover, some words having similar semantic are used in this language interchangeably. For instance, the words: “xumure”, “raawwate”, both indicate similar meaning. Accordingly, some users might use the first word whereas some could use the second word or else interchangeably. This implies that the system needs to include many words with similar semantics in Afaan Oromoo. For such a case, we have provided an alternative for a user to add any correctly spelled word which is not included in the system. In the other way round, this feature helps update the list and more inclusion of words of Afaan Oromoo language.

4.2.3. Evaluation Procedures

The quality of the spelling checker is defined by its ability to recognize valid words as correct and invalid words as incorrect. Furthermore, the spelling checker should also be capable of providing the possible corrections for a specific invalid word to correct it to the intended word. Accordingly, we would consider three aspects (parameters) of the spelling checker to evaluate its performance. These parameters include:

i. True positive
ii. False positive
iii. False negative

The following descriptions reflect how the above mentioned three conditions hold in evaluating the system.

i. *True positive*: is a capability of a spelling checker in detecting all the invalid words as incorrect. This implies that from our approaches discussed under methodology section of this work, a word is said to be invalid if doesn't exist in the list and the vice versa is also true.
ii. *False positive*: is a condition where there would be invalid words; but the spelling checker fails to detect it. However, such a scenario will not happen in our system unless that particular invalid word is included in the application itself. This will not hold true in actual sense.
iii. *False negative*: this scenario happens when a user spelled a particular word (s) correctly; but the spelling checker fails to recognize it as valid. This condition can hold true when there could be missing words from the developed system.

Hence, we would be evaluating the performance of our system based on the above explained conditions. Accordingly, the precision and recall measures of the system would be given as follows:

$$Precision = \frac{True\ positive}{True\ positive + False\ positive} \quad \text{.....(2)}$$

$$\text{Recall} = \frac{True\ positive}{True\ positive + False\ negative} \quad \text{.....(3)}$$

Therefore, in computing both precision and recall of the spelling checker, we need to count the total number of *True positive, False positive and False negative* from the test data and we would be demonstrating this manually as follows:

Consequently, to measure the system in terms of precision, we have taken sample words from 5 different domains (namely: academic, history, sport, news and governance) among the 15 domains we have introduced in the methodology part of our work. These words are incorrectly spelled randomly for the selected 5 domains where their target word is stored in the dictionary. This results in a total of 1000 words which are wrongly spelled. This implies that the total number of *True positive* would be 1000 and this is due to the fact that each domain consists of 200 words. Accordingly, the spelling checker is found detecting all those words and capable of replacing them with the intended word. Because, the original words of those words randomly made invalid already exist in the system and hence upon providing suggestions; the system could replace all the words which are detected as invalid. From the evaluation, no word (among 1000 invalid words) is left undetected having misspellings and this has evaluated the number of *False positive* to zero. Consequently, by using equation (2), the precision measure scored a 100% performance of the spelling checker from the sample test data. The result from computation implies that the spelling checker fails to detect the misspelled word if and only if that particular invalid word has been included within the system itself. Because, the wrongly spelled word will be left undetected only if its erroneous form exits in the system and this would not be true in relation to the approach we have followed.

Similarly, to evaluate the performance of spelling checker from recall point of view, we have dropped half the lists of words from the selected 5 domains. This implies that a total of 500 valid words are eliminated from the system. Hence, the spelling checker can now recognize only those words that are remaining in the system as valid. These recognizable words are the lists from 10 domains and half the lists of words from the selected 5 domains since we have dropped half of the words. Therefore, all valid words dropped from the 5 domains are going to be considered as incorrectly spelled by the system which is false in actual sense. This evaluates the number of *False negative* to be 500. Similarly, the number of *True positive* would be the total number of words remaining in the system which is 2500. Thus, by employing equation (3) above, the performance of the system is computed scoring 83.33% of recall capability.

The result indicates that the higher number of words included the higher the recall of the spelling checker. The following table summarizes performance evaluation of the system.

Measurement units	**Parameters**	
	True positive	*False positive*
	1000	*0*
Precision	*100%*	
	True positive	*False negative*
	2500	*500*
Recall	*83.33%*	

Table 10: Performance evaluation

The score of both metrics from the above computations reveal a substantial result in relation to the performance of the spelling checker. Moreover, the lower score of one of evaluation measurements affects the overall performance of the system. For instance, if we compute the average performance of our system using *F1-score measurement*, it falls down to 90.91% having scored a 100% in precision measurement. This implies that the lower score of *recall measurement* has affected the average performance of the spelling checker and the vice versa will also be true. Most importantly, from our evaluation, the spelling checker showed a significant result in detecting errors and providing the possible corrections from the sample test data. However, we have also observed from the evaluation that some words are yet recognized by the system as invalid despite the fact that the words are correctly spelled. This is a result we have obtained in measuring the spelling checker in terms of its recall capability where we have randomly dropped some valid words from the system. Accordingly, the system detected those valid words as wrongly spelled. This implies that the system should be inclusive of all necessarily used words of Afaan Oromoo language. Furthermore, as we explained in the earlier sections of this work, the system should include words from different domains and even those terms within a particular domain should be complete. Because, the lesser number of words included the lower performance of the spelling checker and we would be demonstrating this in further detail in the following section.

4.3. Results and Analysis

In our thesis we have considered different samples to evaluate the error detection and correction ability of the spelling checker in terms of precision and recall metrics. Accordingly, we found out that the performance evaluation indicates 100% of precision and 83.33% of recall capability. The test result from precision measure shows that the system is at its peak performance in detecting the invalid word and replacing it with the anticipated word. This implies that if a particular word is determined as misspelled, the system always generates a correction. Accordingly, for every incorrectly spelled word, there would always be its correct form within the system. However, the problem arises when a given invalid word is included within the system itself. In this case, the spelling checker would fail to identify that specific invalid word and the system would rather consider that word as correctly spelled word. Because, from the standing point of the approach followed in this thesis, the system is built to contain only the valid form of words of Afaan Oromoo language. Therefore, a great emphasis should always be given when storing the words in the system.

Although the system indicated a considerable performance result in terms of precision measure, it has revealed a lower score from recall perspective relatively. The evaluation in terms of recall ability of the spelling checker has considered the persistence of the system given correctly spelled words which are missing from the system. Consequently, the spelling checker has failed to recognize those missing; but correctly spelled words as valid. This test result indicates that a system should be complete; consisting of all the required list of words. Furthermore, the included words should be those we commonly use in writing than words used at speaking level. Because, in Afaan Oromoo language, there are words that we don't commonly use in writing and those words are used only at verbal level. Hence, the inclusion of those verbal words in the system in place of words used in script would ruin the performance of the spelling checker. This implies that a user would miss from the system those words that are preferably used in script and by far the system would also recognize those missing words used in writing as misspelled which is false in actual sense. Therefore, this in turn would result in performance reduction of the system.

Additionally, it is not only the incorporation of verbal words over the script that will result in performance degradation of the system, but the missing of words used in writing also matters as we have demonstrated in the evaluation.

Moreover, in comparison with the literatures we have gone through, the approach we followed has shown a significant score from our evaluation particularly when compared to spelling checker designed based on morphology of Afaan Oromoo language. This implies that our approach can result in a considerable performance regardless of making all the necessary rules and the contemplation of all Afaan Oromoo language structures to design spelling checker. Furthermore, the results from evaluation reveal that the higher language coverage the higher the accuracy of the spelling checker and vice versa. This ensures that once we have a complete dictionary designed touching all the domains of the language, the performance of the suggested system would definitely be perfect. However, it would not be easy to build a complete list that covers the entire words in Afaan Oromoo language. This is because, the massive nature of words in this language (Afaan Oromoo). Moreover, for a particular word used in specific area (for instance Jimma), there would be another word with similar usage in another area (for instance Harar). Similarly, there would be some words which are used interchangeably. Consequently, these all make challenging to build a complete dictionary covering all the terms used in Afaan Oromoo and thereby reducing the performance of the spelling checker. Therefore, it would be important to include all the words that are widely or commonly used in writing Afaan Oromoo texts, documents and etc. Accordingly, the degree of efficiency of the spelling checker depends on its inclusiveness of large enough words and by far from different domains of the language.

Another interesting result is that the most likely replacing candidate always appears at the top of the rank among the possible list of suggestions. This is also another significant result obtained from the proposed algorithm. Perhaps, the user's preference could be different in choosing a correction word among the given alternatives as replacement.

4.3.1. Experiment On Performance Affecting Factors

This subsection discusses and presents the factors affecting the performance of spelling checker with the corresponding experimental results. The performance of spelling checker can be affected by the following three conditions:

i. *Huge number of inclusion of words*: this is a scenario where the size of the dictionary is big enough consisting of enormous amount of words covering all the language domains. Consequently, the performance of the spelling checker would increase.

ii. *Less number of words included*: this condition happens when the size of the dictionary is considerably less. This implies that the more number of words missing from the system the more words recognized as invalid; even if correctly spelled. Because, the spelling checker cannot guarantee those words which are not included in the prepared list. Therefore, such a condition results in the performance drop of spelling checker.

iii. *Inclusion of rarely used words*: this is a case when some words which are used rarely or verbally are included in the system over those commonly used words in script. The implication is therefore; a user would miss some widely used words in writing. Accordingly, this condition will also affect the performance of the spelling checker. Therefore, the best method is including both rarely and commonly used words in the dictionary despite the disadvantage we have raised in the earlier sections.

Accordingly, to demonstrate the above described conditions we have taken three different spelling checkers labeled as *spelling checker A, spelling checker B and spelling checker C* reflecting the three conditions respectively and some randomly used testing data from our dictionary. The test data is taken from our 15 domains listed under *dictionary preparation* section of this work. For spelling checker A, we have dropped half the lists of words from two domains. This implies that a total of 200 words are omitted from the dictionary. The assumption here is that our dictionary of 3000 words is considered as big sized. For spelling checker B, from the 15 domains, half the lists of words from eight domains are dropped resulting in a total of 800 eliminated words from our list. Similarly, we have made half the lists of words from 7 domains to be rarely used words for spelling checker C. This implies that 700 commonly used words in writing would be missing from the dictionary.

We have used equation (2), equation (3) and equation (4) to compute the resulting performance score of the three spelling checkers and the results are summarized in the following table.

Moreover, we have also used a harmonic mean measurement unit (i.e. F1-score) to compute the average performance of the three spelling checkers. F1-score is given by the following formula:

$$F1\text{-}score = \frac{2*Precision * Recall}{Precision + Recall} \quad \ldots\ldots (4)$$

Spelling checkers	Performance		
	Precision	Recall	F1-score
Spelling checker A	100%	93.33%	96.55%
Spelling checker B	100%	73.33%	84.61%
Spelling checker C	100%	76.7%	86.81%

Table 11: Performance comparison

Therefore, from the above obtained results, it is possible to generalize that the efficiency of spelling checker highly depends on the size of the dictionary. Similarly, the lesser size of the dictionary and the inclusion of rarely used words over commonly used words in writing can result in lower performance of spelling checker as shown in Table 11.

Chapter 5

Conclusion and Recommendation

This chapter summarizes the key and major issues addressed in this paper. Finally, we would be recommending some suggestions in relation to the problem and the solutions presented in this thesis.

5.1. Conclusion and Future Work

This thesis is emphasized and targeted to design a dictionary based spelling checker for Afaan Oromoo language. Basically, we addressed core issues: detection of erroneous word(s), providing suggestions and ranking those suggestions. We followed the approaches named dictionary lookup and n-gram to address the problem raised in the paper. Finally, we designed and developed a spelling checker system. In essence, the spelling checker performs the major tasks in detecting the misspelled words and providing the precise corrections for those words detected as misspelled. Furthermore, we evaluated the developed system in terms of *precision* and *recall* measurements. The results obtained from performance evaluation of the spelling checker indicated a promising result in handling the spelling errors of a word. However, this would not hold true if we consider word contexts due to the fact that the system is not designed in such a way that it is capable of correcting contextual errors. For instance, let's consider the following two scenarios:

String1: "Humnaan dhibee biyya keenya irraa nu baase".

String2: "Humnaan dhiibee biyya keenya irraa nu baase".

From these two scenarios, it is clear that the word ["*dhibee*"] is wrongly used in **String1** contextually. But, the context is to mean **String2**. However, these two words ["*dhibee*", "*dhiibee*"] are correctly spelled at word level. Hence, such a problem dives into a future work. Finally, we have conducted the experiment to demonstrate the factors affecting the performance of spelling checker by considering three circumstances namely: inclusion of more words, inclusion of less words and inclusion of rarely or verbally used words in the dictionary. Consequently, the experimental results indicated that the first case has increased the performance whereas the latter two cases have resulted in performance reduction of spelling checker.

5.2. Recommendation

As far as our observation is concerned, the misspelling of Afaan Oromoo language is boldly visible. Different banners written in this language and installed across government offices, public squares and private organizations and other documents reflect spelling error. Apart from this, we have also noticed that the maximum percentage of users of popular social medias commit a tremendous errors in writing Afaan Oromoo. Therefore, we recommend that it is helpful to use this spelling checker in the places it deserves to be used. It can be possible by integrating the application along with text editing tools and other applications. Finally, to implement the spelling checker in a real environment, the dictionary has to be complete and representative of all Afaan Oromoo words including from different domains of the language as well as all the words across the region (i.e. Oromia) that are commonly used in writing Afaan Oromoo documents, texts and etc.

References

[1] Gadissa Olani Ganfure, Dr. Dida Midekso, "Design and Implementation of morphology based spell checker", International Journal of Scientific & Technology Research, Issue 12, December 2014, ISSN 2277-8616.

[2] Andargachew Mekonnen Gezmu, Andreas Nürnberger, Binyam Ephrem Seyoum," Portable Spelling Corrector for a Less-Resourced Language: Amharic".

[3] Mekonnen Fentaw, "Designing and Developing Amharic Spell Checker and Integrating It with word Processor", BiT Research, Bulletin, Volume 3, issue 1, June, 2017.

[4] Daniel Yacob, "Application of the Double Metaphone Algorithm to Amharic Orthography", International Conference of Ethiopian Studies XV, 2006.

[5] Youssef Bassil & Mohammad Alwani, "Context-sensitive Spelling Correction Using Google Web 1T 5-Gram Information", Vol. 5, No. 3; May 2012.

[6] Renato Cordeiro de Amorim, Marcos Zampieri, "Effective Spell Checking Methods Using Clustering Algorithms", Proceedings of Recent Advances in Natural Language Processing, pages 172–178, Hissar, Bulgaria, 7-13 September 2014.

[7] Hasan Muaidi, Rasha Al-Tarawneh, "Towards Arabic Spell-Checker Based on N-Gram Scores", International Journal of Computer Applications (0975 - 8887), Volume 53 - No. 3, September 2013.

[8] E. Yilmaz Ince, "Spell Checking and Error Correcting Application for Turkish", International Journal of Information and Electronics Engineering, Vol. 7, No. 2, March 2017.

[9] Zhongye Jia, Peilu Wang, Hai Zhao, "Graph Model for Chinese Spell Checking", Proceedings of the Seventh SIGHAN Workshop on Chinese Language Processing (SIGHAN-7), pages 88–92, Nagoya, Japan, 14 October 2013.

[10] F.J. Damerau, "A Technique for Computer Detection and Correction of Spelling Errors", Communication of ACM, 7(3), 171-176, 1964.

[11] Peterson, L.J, "A Note on Undetected Typing Errors" In Communications of ACM, 29(7): 633-637, 1986 .

[12] Ralph Gorin, "SPELL for the DEC PDP-10", Stanford University's Artificial Intelligence Laboratory, February 1971.

[13] Geetaachoo Rabbirraa, "Furtuu Seerluga Afaan Oromoo", Finfinnee Oromiyaa press, 2014.

[14] Shallamaa Kabbee, "Seenaa Gootota Oromoo fi Kaan", maxxansa 2^{ffaa}, March, 20018.

[15] Marek Bardonski, "Natural Language Processing in Artificial Intelligence", 2017.

[16] Ernest Davis, "The Technological Singularity", an ACM publication October 2014.

[17] Visha Marria, "The Future of Artificial Intelligence In The Work Place: Friend or foe?", Forbes, January, 2019.

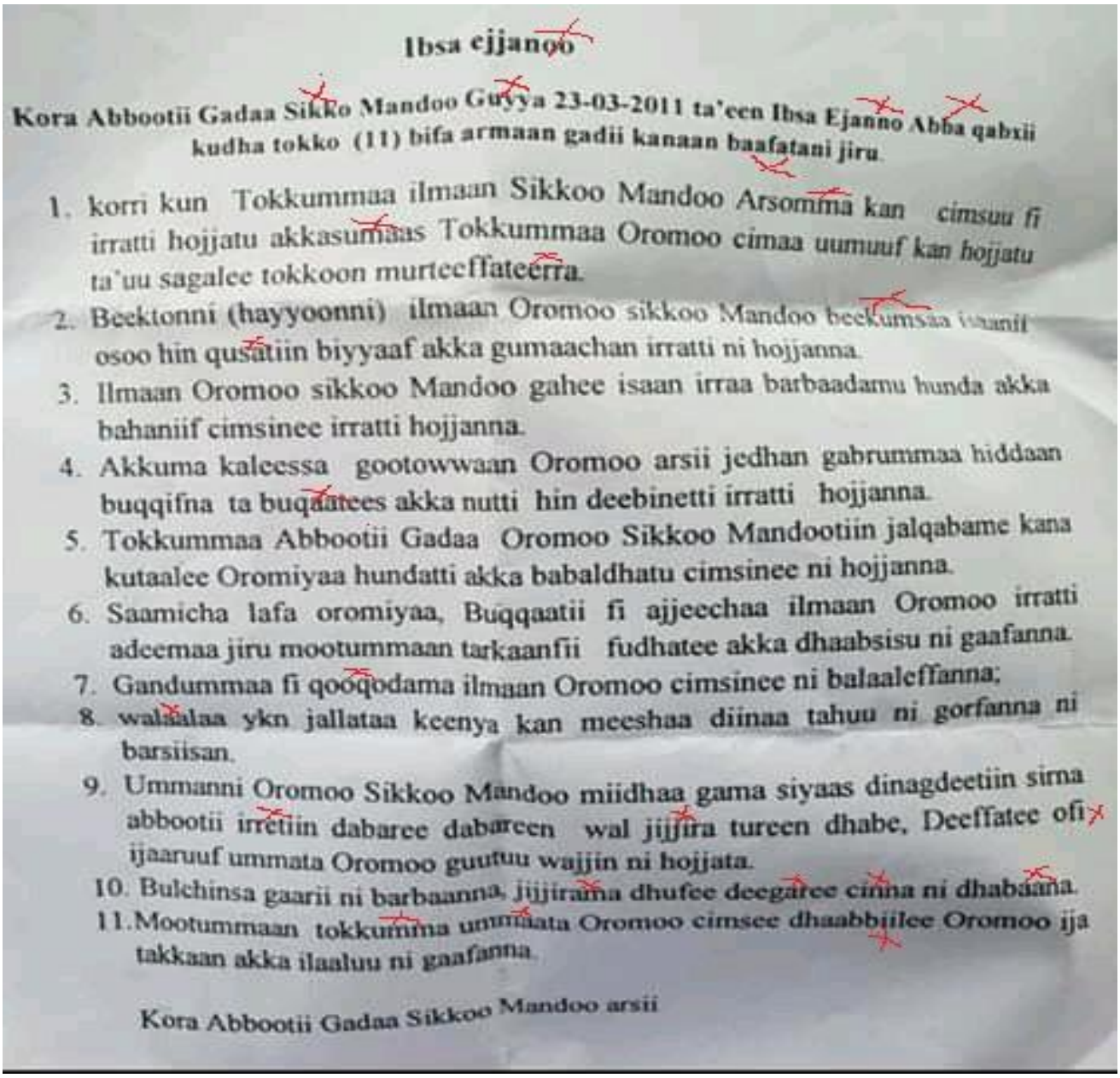
[18] Beekan G. Erena, "Oromo language (Afaan Oromoo)", 2018.

[19] Maria Mariani, Random House, "The Spelling Bee is Over", November, 1982.

Appendices

Appendix I

Sample Spelling Errors (A Survey)



Ibsa ejjanoo

Kora Abbootii Gadaa Sikko Mandoo Guyya 23-03-2011 ta'een Ibsa Ejanno Abba qabxii kudha tokko (11) bifa armaan gadii kanaan baafatani jiru.

1. korri kun Tokkummaa ilmaan Sikkoo Mandoo Arsomma kan *cimsuu fi* irratti hojjatu akkasumaas Tokkummaa Oromoo cimaa uumuuf kan *hojjatu* ta'uu sagalee tokkoon murteeffateerra.
2. Beektonni (hayyoonni) ilmaan Oromoo sikkoo Mandoo beekumsaa isaanii osoo hin qusatiin biyyaaf akka gumaachan irratti ni hojjanna.
3. Ilmaan Oromoo sikkoo Mandoo gahee isaan irraa barbaadamu hunda akka bahaniif cimsinee irratti hojjanna.
4. Akkuma kaleessa gootowwaan Oromoo arsii jedhan gabrummaa hiddaan buqqifna ta buqaatees akka nutti hin deebinetti irratti hojjanna.
5. Tokkummaa Abbootii Gadaa Oromoo Sikkoo Mandootiin jalqabame kana kutaalee Oromiyaa hundatti akka babaldhatu cimsinee ni hojjanna.
6. Saamicha lafa oromiyaa, Buqqaatii fi ajjeechaa ilmaan Oromoo irratti adeemaa jiru mootummaan tarkaanfii fudhatee akka dhaabsisu ni gaafanna.
7. Gandummaa fi qooqodama ilmaan Oromoo cimsinee ni balaaleffanna;
8. walaalaa ykn jallataa keenya kan meeshaa diinaa tahuu ni gorfanna ni barsiisan.
9. Ummanni Oromoo Sikkoo Mandoo miidhaa gama siyaas dinagdeetiin sirna abbootii irretiin dabaree dabareen wal jijjira tureen dhabe. Deeffatee ofi ijaaruuf ummata Oromoo guutuu wajjin ni hojjata.
10. Bulchinsa gaarii ni barbaanna, jijjirama dhufee deegaree cinna ni dhabaana.
11. Mootummaan tokkumma ummaata Oromoo cimsee dhaabbilee Oromoo ija takkaan akka ilaaluu ni gaafanna.

Kora Abbootii Gadaa Sikkoo Mandoo arsii

Figure 5: Sample spelling errors from text file

Figure 6: Sample spelling errors from banner

Appendix II

Spelling Error Statistics (A Survey)

Users	Total spelled words	Number of misspellings
User1	26	8
User2	27	7
User3	14	2
User4	11	5
User5	15	7
User6	20	3
User7	37	0
User8	28	0
User9	20	0
User10	38	0
User11	23	4
User12	28	0

User13	43	0
User14	17	10
User15	52	5
User16	7	3
User17	7	2
User18	6	0
User19	15	5
User20	17	2
User21	15	2
User22	12	2
User23	15	2
User24	2	1
User25	7	4
User26	6	5
User27	10	9
User28	14	0
User29	16	1
User30	9	0
User31	19	1
User32	11	1
User33	5	5
User34	10	0
User35	5	3
User36	25	15
User37	54	17
User38	17	4
User39	11	0
User40	22	0
User41	25	0
User42	16	0
User43	7	7

User44	9	0
User45	30	7
User46	32	3
User47	23	2
User48	21	16
User49	12	0
User50	14	5
User51	17	7
User52	31	0
User53	17	0
User54	24	1
User55	64	4
User56	23	0
User57	17	6
User58	83	12
User59	20	1
User60	85	0
User61	85	7
User62	18	0
User63	18	3
User64	10	2
User65	9	0
User66	18	0
User67	6	6
User68	24	1
User69	19	3
User70	104	5
User71	20	0
User72	4	1
User73	12	3
User74	10	0

User75	8	0
User76	43	0
User77	38	3
User78	18	10
User79	29	18
User80	53	6
User81	5	3
User82	47	1
User83	10	0
User84	2	0
User85	5	5
User86	8	0
User87	16	14
User88	14	2
User89	40	3
User90	98	12
User91	10	1
User92	9	3
User93	13	0
User94	2	0
User95	9	1
User96	18	0
User97	6	5
User98	2	0
User99	8	1
User100	33	13
Additional sources		
Text files	298	28
Banners	14	5

Table 12: Error statistics