Development of a Web-based Service to Transcribe Between Multiple Orthographies of the Iu Mien Language

Robert P. Batzinger

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree

Master of Science in Applied Mathematics and Computer Science

in the Department of Computer and Information Science,
Indiana University
October 7, 2011

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Master of Science.

Michael Scheessele

Liquo Yu

Zhong Guan

Dana Vrajitoru

Abstract

The goal of this study was to explore the use of machine learning techniques in the development of a web-based application that transcribes between multiple orthographies of the same language. To this end, source text files used in the publishing of the Iu Mien Bible translation in 4 scripts were merged into a single textbase that served as a text corpus for this study.

All syllables in the corpus were combined into a list of parallel renderings which were subjected to ID3 and neural networks with the back propagation in an attempt to achieve machine learning of transcription between the different Iu Mien orthographies. The most effective set of neural net transcription rules were captured and incorporated into a web-based service where visitors could submit text in one writing system and receive a webpage containing the corresponding text rendered in the other writing systems of this language. Transcriptions that are in excess of 90% correct were achieved between a Roman script and another Roman script or between a non-Roman script and another non-Roman script. Transcriptions between a Roman script and a non-Roman yield output that were only 50% correct. This system is still being tested and improved by linguists and volunteers from various organizations associated with the target community within Thailand, Laos, Vietnam and the USA.

This study demonstrates the potential of this approach for developing written materials in languages with multiple scripts. This study also provides useful insights on how this technology might be improved.

Dedication

God can do anything, you know— far more than you could ever
imagine or guess or request in your wildest dreams!
He does it not by pushing us around but by working within us,
His Spirit deeply and gently within us.

Ephesians 3:20 (English, The Message)[1]

Tinb huvb, ei ninb lovg ninb Eei tomb Qa'q yem Bua Eei Hruq kaEuad zruj kov,
zuvq haib zruj kaub Zamb Bua haib Toq Eei fai Namq tu'q Tauj Eei.

e-fe-so 3:20 (Iu Mien – Old Roman Script)[2]

Tin-Hungh, ei ninh longc ninh nyei domh qawv yiem mbuo nyei hnyouv gu'nyuoz zoux gong,
zungv haih zoux gauh camv mbuo haih tov nyei fai hnamv duqv taux nyei.

E-fe-so 3:20 (Iu Mien – New Roman Script)[3]

ทิน-ฮู่ง, เอย นิ่น หล่ง นิ่น เญย ต้ม ชะ เยียม บัว เญย เฮญี้ยว กะญั๋ว โหฒว กง,
ฒู้ง ไฮ่ โหฒว เก้า ธั้ม บัว ไฮ่ โท้ เญย ไฟ ฮนั้ม ตุ๊ เถา เญย.

เอ^เฟ^โซ 3:20 *(Iu Mien – Thai Script)*[4]

ທິນ-ຮູ່ງ, ເອີຍ ນິ່ນ ຫລົ່ງ ນິ່ນ ເຍີຍ ຕົ້ມ ທຈະ ຢຽມ ບົວ ເຍີຍ ໂຮຍ້ວ ກະຍົ໊ວ ໂຕສ໋ວ ກົງ,
ຕສູ໊ງ ໄຮ່ ໂຕສ໋ວ ເກົ໊າ ທສໍ້າ ບົວ ໄຮ່ ໂທ້ ເຍີຍ ໄຟ ຮນໍ້າ ຕຸ໊ ເຖົາ ເຍີຍ.

ເອ-ເຟ-ໂຊ 3:20 *(Iu Mien – Lao Script)*[5]

Acknowledgements

This project could not have been completed without help. I am grateful for the time and effort various individuals have taken to assist me. Therefore, I want to acknowledge that their help was truly God sent and much appreciated.

I am greatly indebted to Ann Burgess for providing instruction about the Iu Mien orthographies and for her painstaking efforts, together with Bienh Gueix-Fonge, in editing and proofreading drafts of the Bible translation in all four orthographies. I will always remember them for their diligence to detail, patience with primitive computer systems, and perserverance on a difficult translation project that spanned two decades.

I am also grateful to OMF Publishers and the Thailand Bible Society, the co-publishers of the Iu Mien Bible, for the use of the source text files of the Iu Mien Bible translation for this studies. I also acknowledge the efforts of the TBS typesetter, Jongluck Tariyo, for her attempts to maintain markup standards in the electronic manuscripts despite the pressure of tight typesetting deadlines of this Bible transaltion.

I recognize and thank the staff of Heroku for their generous technical support and service in providing a free Rails hosting services and technical assistance on the Amazon Elastic Compute Cloud (Amazon EC2). This versatile service provided excellent opportunities to experiment with cutting edge technology.

This project was also greatly facilitated by some key software: the reliable non-Roman script text processing capabilities of the Ruby programming language by Yukihiro Matsumoto and the flexible non-Roman typography of Jonathan Kew's XƎTEX / XƎLATEX variants of Donald E. Knuth's TEX typesetting system. I am truly grateful for the willingness of these three eminent computer developers, not only for taking the time to respond to my inquiries, but also for their efforts to expand the Unicode support of their products.

I am grateful for the encouragement and feedback from members of the IU South Bend Department of Computer and Information Sciences. I am thankful for the guidance, patience and insight of my advisor, Dr. Michael Scheessele. The thesis was also shaped and improved by the regular informal encounters and hallway conversations with Drs. James Wolfer and Hossein Hakimzadeh who provided generous amounts of encouragement and intellectual prodding. I am also grateful for insights and feedback from the readers of this thesis, Drs. Liquo Yu and Zhong Guan.

I also acknowledge support of my wife Khajohn as well and my daughters, Joy and Faith who have been a constant source of encouragement and inspiration all the years I have been developing software to support publishing in various non-Roman scripts of Asia.

Lastly, I thank the Iu Mien people for their interest in this project and for their patience as the systems were developed and tested. It is rewarding to see this technology used as a communication bridge between members of a people group who have been separated by geopolitical borders and orthographic differences. May God continue to bless them with a spirit of cooperation as they build a better future together despite these barriers between them.

Contents

List of Tables

List of Figures

List of code fragments

CHAPTER 1

Introduction

1.1 Multiple orthographies of a language

All writing systems are an attempt to use orthographic symbols to represent various linguistic features of a language. However, individual writing systems differ in the level of phonetic and linguistic information captured, the range of symbols used and formal constraints governing the mapping between these entities. This is true even when multiple writing systems are used to describe the same language.[6]

Multiple orthographies commonly arise within a language to address specific needs of subgroups of readers. For example, the majority of readers of English are familiar with the Roman script used in most English publications. However, this is not the only orthography for English in current use. The visually impaired prefer Level 3 Braille which uses contractions rendered in 6 dot Braille to improve digital reading speeds. Linguists use the International Phonetic Alphabet (IPA) to record and describe the regional accents of spoken English. American dictionary publishers commonly use some variant of the pronunciation symbols derived by Noah Webster to describe the standard pronunciation of words. Greg short hand, speed writing and court stenographic systems are all different attempts to increase the speed and accuracy of manual transcription of English dictation. The ideographic system of emoticons and acronyms used in English text messaging has become very popular among Internet users and is still evolving. Each of these writing systems of English are unique in appearance and have been shaped by their function.

In Southeast Asia, local majority languages like Thai, Myanmar and Lao use derivatives of the Devanagari script which have been shaped centuries ago by the phonetics of the language and the writing surfaces they used.[6] However, for many minority groups, particularly those with rich oral traditions, written orthographies are a recent invention which is still evolving. As a minority community becomes literate, the orthography is often revised to correct mistakes made in the design of the original script as well as to alter the writing system so as to improve literacy.

Southeast Asia has also been a home to hundreds of small minority people groups that have lived in the deep forests and remote mountain tops along the ancient trade routes. Although they have lived for centuries in a manner that was unaffected by the rise and fall of the empires of Asia, the political and economic conditions stemming from the aftermath of World War II, colonial rule, the Cold War and Vietnam War have facilitated the emergence of independent countries. The corresponding national governments have attempted to seal their borders in order to reduce threats to national security arising from the smuggling of arms and opium.[7] This has introduced new barriers to travel and communication that never existed before.

In addition, the sovereign countries have promoted nationalism and have taken various measures to reduce the isolation of all minority groups.[8] To this end, countries like China, Vietnam and Thailand have attempted to relocate isolated villages and even to restrict publications in the vernacular scripts of minority groups. During the Vietnam War, governments of Southeast Asia encouraged, or even on occasions required, minority language groups to adopt a variant of the orthography of the national language.[9] This political pressure has also given rise to multiple orthographies for tribal people, particularly those living in regions that span national borders.

Given the turbulent history of political and cartographic changes in the past two centuries, there are numerous cases across Asia where these geo-political pressures have affected the writing systems of minority languages, some notable examples can be seen in Table 1.1. The Kamut who are scattered throughout Southeast Asia have six distinct scripts. The Pwo Karen however are confined to a single border and only have two discrete scripts. Multiple scripts for Hindi and Mandarin arose from attempts to include minority groups into the life of their respective nations. Recent regional politics of Indonesia has contributed to the amazing revival and rapid reintroduction of the Bali script which had been on the brink of extinction with only 10 readers in 1995 to over 5,000 in 2005. Over 50,000 Santali of India were contracted to build roads in East India and were stranded by the partition of India and subsequent war for Bangladeshi independence. In each case, publishers working with such groups must create multiple editions of their publications in order to reach all readers of the target language group.[10]

More recently, with the expansion of radio and television coverage of remote areas, oral traditions have begun losing their appeal as younger generations are introduced to the pop culture of the national majority. The extinction of tribal languages in the Region has been growing at an alarming rate. Governments of Asia have been known for their attempts to hasten this process by controlling vernacular publications and literacy education.[9]

However, with increasing revenue from international tourism particularly in regions with tribal people, governments have recently softened their position with local minorities and have even encouraged education of tribal culture and traditions in primary schools. When the education of tribal heritage has been embraced and practiced, governments have reported increased school enrollment rates, higher levels of compliance with local health and agricultural initiatives and significant improvement of national language literacy rates in those tribal areas.[11]

Table 1.1: Examples of spoken languages with multiple writing systems

Language	**Name of Script**	**Country**
Bali	Bali script	Bali, Indonesia
	Roman script	Indonesia
Hindi	Romanized script	Fiji
	Romanized script	Northeast India
	Devanagari script	Central India
Kamut	Khmer script	Cambodia
	Roman script	France
	Lao script	Laos
	Myanmar script	Myanmar
	Roman script	Vietnam
	Thai script	Thailand
Mandarin	Simplified Chinese script	China
	Romanized Pinyin script	China
	Complex Chinese script	Taiwan
	China Standard script	China
	Old Roman script	Northern Thailand
	New Roman script	China
	New Roman script	USA
	New Roman script	Vietnam
	Thai script	Northeast Thailand
Mongolian	Cyrillic script	Mongolia
	Mongolian script	Mongolia
Pwo Karen	Kayin script	Myanmar
	Thai script	Thailand
Santali	Bengali script	Bangladesh
	Orissa script	India

These benefits are less prominent when the phonemes of the tribal script have little overlap with those of the national script because the inconsistencies are a major source of frustration and confusion for both the teacher and the student.[12, 13] For example, American volunteer literacy workers often find that Mien refugees consistently misspell the word *gong* as *kov*. In this case, the refugees are merely being consistent with the script of their mother tongue.[14] This confusion adds additional pressure for the revision of orthographies of minority people.[15]

Although each individual orthography of a language provides distinct advantages for their users and promoters, the differences between the various orthographies quickly form barriers to effective communication and sharing of captured text. While the development of an application to transcribe between orthographies would provide a communication bridge between subpopulations, the process of developing transcription rules can be very time consuming.

Groups that are isolated by a political or international border also tend to develop their own local orthographic rules and spellings which further complicates attempts to communicate across borders. However, the isolation is being broken. Internet connections are currently available in nearly every village of Asia. It is common to see school children in remote villages of Thailand or India sending email, blogging and spending time in chat rooms via Cyber cafes and cell phones. Online services like Google Translate have become popular among educators and businessmen in these remote areas. Similarly, online automated transcription services should prove useful for helping minority groups to communicate with fellow tribesmen in a way that spans international borders and bridges differences in orthography.

While the groups described above may seem small in number, the problem of multiple orthographies is fairly common across Asia. In fact, the problem is not restricted to minority languages. Hindi and Mandarin, the two largest language groups in Asia and the world, were also included in the list of languages with multiple writing systems given in Table 1.1. The total number of languages with multiple scripts in current use has been estimated to be around 200 of the approximately 4,000 languages spoken in Asia[10, 16]. Publishers across Asia often print parallel publications to distribute the same content in multiple scripts of the same language. This current work aims to explore the use of machine learning technology applied to the text sources of parallel editions as a means for developing auto-transcription applications. The outcome of this work would have practical implications on efforts to link together minority communities separated by orthography.

1.2 The value of parallel texts

Parallel texts provide excellent opportunities for data-mining and machine learning. Ever since the Rosetta stone was used in the early 19th century to crack the secrets of hieroglyphics, parallel texts have been used for gleaning rules for translation and transcription. Lessons learned with hieroglyphics were quickly applied to other sets of parallel texts to determine the phonetics of various semitic languages.[17] With the dawn of computers and machine learning techniques, corresponding elements in pairs of text in large parallel text sets can be tagged, linked and analyzed as a means for unraveling the meaning of natural human language. However, a large corpus of text can also contain noise arising from inconsistencies in text entry, regional language variations, as well as human error. Because these errors and inconsistencies give rise to artifacts that lower the accuracy and efficiency of the machine learning, considerable effort is required to develop filters that would result in consistent parallel text.[18]

With the emergence of the Internet and a global economy, the growing need to communicate in multiple languages has fueled research to develop software that can compare dialects and languages, and apply this information in an attempt to translate between them.[19]. This field of research has given rise to automatic translation products like Alta Vista's Babelfish and Google's Translate which have become fairly popular despite frequent mistakes in the translations produced.

However, these mistakes in machine translation illustrate the complex array of linguistic differences between natural languages. Mere introspection of the surface differences is inadequate to determine all the changes in the grammar, semantics and lexicon required, as text in one language is translated into another. Incorporating various linguistic data into the data model greatly improves the effectiveness and efficiency of these comparative studies.[20]

The evolution of human languages has been generally driven by grammatical ancestry and the need to develop lexicons that can fully describe human experience. Differences in human experience are reflected in the differences in the lexicon. For example, the Yup'ik of Alaska have 16 discrete words to describe the various types of Arctic snow seen in their environment[21] while the Thai of tropical Southeast Asia only have one word. Similarly, the semantic word order and grammar of Dutch is closer to German (with whom there has been a long history of interaction) than to a language of a distant people like the Mandarin Chinese. Translation systems that would attempt to compete with human translation would require a complex hybrid data model that includes linguistic information and filters for human error within the machine learning model.[22]

In this study, parallel texts of the same language were used to develop transcription rules to convert between the multiple orthographies of the same language. While machine learning of transcription rules would appear to be relatively easier than the development of machine translation solutions, many of the linguistic and data mining issues encountered in parallel translations can also be seen in parallel transcriptions.

Although each orthography is an attempt to capture the phonetics of the same target language, the graphemes (or the symbols used) in the different transcription systems do not render the phonetic features (phonemes) in the same way. In addition, spelling of key concepts and proper names as well as the phonetics of the local majority community influences the orthography of minority languages and can give rise to exceptions to normal transcript rules. However, this research is hindered by the lack of large scale parallel transcriptions.

The Thailand Bible Society and the OMF Publishers in Thailand recently launched the Iu Mien Bible translation through the simultaneous publication in four scripts (Old Roman, New Roman, Thai, and Lao).[2, 3, 4, 5] The source text of these simultaneous publications was for the most part derived from a single textbase and represented a word by word parallel rendering of the Iu Mein text in four scripts. The text was held in a generic script that captured all features supported by the target orthographies. In addition, the textbase also captured the markup of textual objects like headings and verse numbering. The 6MB generic textbase and the corresponding proofread source files for each of the orthographies appeared promising as a source of a corpus that could be used for these studies in machine learning.

Because a good text corpus must have a collection of words in content that reflects the full range of word usage within a given language, it would be useful to ascertain the relevance of using a Bible translation as a text corpus for this language. While it is not entirely possible to address this issue for the Iu Mien without an exhaustive unabridged dictionary of the language, a look at English would provide some insight as to the order of magnitude of this problem. An unabridged English dictionary catalogues over 300K words.[23] The average college educated American adult is thought to have mastered between 13,200 and 20,000 English words.[24] Analysis of English words in the The authors of the Corpus of Contemporary American English showed that despite the large lexicon of English only 2000 words account for 90% of all communications.[25] If an international language like English has a total of several hundred thousand words, it would be reasonable to assume a smaller word count for Iu Mien who live in remote area of South East Asia.

At the same time, all translations of the Bible are attempts to render 4,000 proper names of individuals, people and locations, and 8,000 other Hebrew, Greek or Aramaic words into a target language. For English, there is a wide range in the number of lexical units used in the various English translations based on the intended audience of the translation. The Good News Bible with its 4K unique words was designed for readers of English as a second language. The literary scholars that wrote the King James Bible used nearly 8K words to convey the message. The Amplified Bible uses nearly 10K words. All of these English translations use all of the 2,000 most frequent words of the English language which represent over 90% of American communications.

In summary, an English Bible translation contains 1-3% of all words of the language and over 90% of those used in written communications. Although not as much is known about the Iu Mien language despite years of intensive study, one might expect a similar correlation between the number of words of the Bible and the number of words in common use in. With thousands of words from the Bible to draw on, one can assume that a corpus based on the Bible should be large enough to draw inferences about transcription rules between various scripts of the language.

1.3 The Iu Mien and their language

The Iu Mien generally live in remote isolated villages in the mountainous regions of southeast Asia extending from Thailand through Laos, Vietnam and southern China. Despite the distance between them, the Iu Mien share a common language and oral tradition. However, the local majority community in which they live greatly influences the use of the script in general. The most profound effects can be seen in the use of the majority language spelling of proper names despite the difference in phonetics. Figure 1.1 shows the geographic distribution of the orthographies used in Southeast Asia.

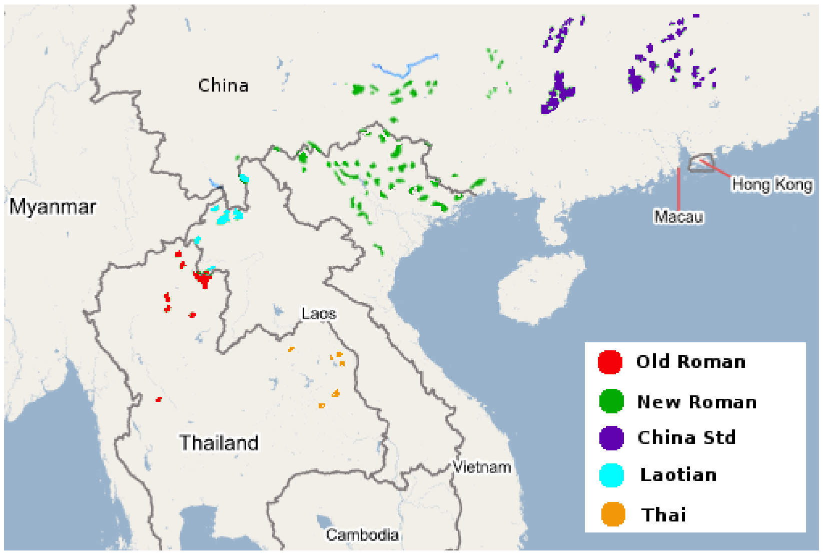


Figure 1.1: Southeast Asian regions where Iu Mien orthographies are used

According to the 2009 Ethnologue,[26] there are approximately 1.7 million Mien speakers worldwide, 1.3 million live in the south-western regions of China and 250,000 live in the mountainous region of northern Vietnam. The local governments in these two regions encourage the use of Roman orthography. Another 100,000 live in the mountainous regions of Laos where the Lao script is used. It is estimated that there are also more than 40,000 Mien speakers living in northern Thailand many of which read an old Roman script that was used in a dictionary published in 1960.[27] The remainder of the Iu Mien readers in Thailand use the Thai script. Because of the socio-political instability and economic hardships in Laos, Vietnam and China during the second half of the 20th century, many Iu Mien immigrated from their traditional homelands and approximately 250,000 have settled in Australia, North America, and France where a New Roman script is used. Therefore, publishers for this people group have had to develop publications in multiple scripts in order to support the different groups of readers.

The Iu Mien refer to themselves as *Kim Mien*, or “men of the mountains”, but as shown in Table 1.2, they are known by many names within the region. However, most of the other people groups in the region refer to them as *Yao* which is derived from a Chinese derogatory term for “barbarians.” It is for this reason that the term Iu Mien is used throughout this document to refer to both the people and the language.

Table 1.2: Various Names Used for the Iu Mien People

Dao	Highland Yao	Mien	Yao
Guangxi-Yunnan	Iu Mien	Min	Yiu Mien
Guoshan	Man	Myen	Youmian
Guoshan Yao	Mian	Pan Yao	Yumian

The Iu Mien language is a tonal language consisting mainly of monosyllable words. Some words are prefixed with an open syllable which some would consider to be a pre-syllable. Since many foreign names are multisyllabic, special marking is required to identify a sequence of syllables as a foreign proper name that should not be misunderstood or confused with phrases of Iu Mien words. In addition, there is a tone shift when a noun is modified by an adjective. Earlier attempts to develop orthographies explicitly marked these tone changes. However, this feature hindered the development of standard spelling of words in the lexicon. Therefore tone changes of nouns in context were not marked features in orthographies that were developed later.

1.4 Four major orthographies of Iu Mien

The following section provides a general description of each of the four major orthographies of Iu Mien studied in this project:

- **Old Roman script:** This script was developed and introduced by OMF Missionaries in northern Thailand and was meant as a replacement for an earlier script derived from the French orthography. This script attempted to capture all the phonetic features of this language, including tonal changes to nouns modified by an adjective, using individual characters found on a standard English typewriter. The marked tone is the true tone of the syllable.

  As such, this script is case sensitive with uppercase and lowercase characters used to mark different phonetic features. In many cases, capitalized consonants were used to mark aspirated consonants and lowercase consonants were used for unaspirated consonants. Because no digraphs were used in this script, numerous characters were arbitrarily mapped to phonemes in a way that would confuse native readers of Roman script.[28]

- **New Roman script:** As the name implies, this script is newer than the Old Roman script. It is used by Iu Mien refugees in America and France. It was an attempt to correct the problems Iu Mien readers of Old Roman script had in learning to read English and French in their new homes. Glyphs are reassigned to phonemes more consistent with French and English phonetics. Nouns were consistently spelled the same way regardless of whether they were modified by adjectives or not. Capitalization was used to mark proper names and start of sentences. Digraph sequences are used for some consonants. The script was also adopted by the Iu Mien communities in China and Vietnam.

- **Thai script:** The Thai script for Iu Mien arose as a response to local government pressure to promote Thai literacy among Iu Mien communities. In this script, the tone marker is actually a tone class change marker which alters both the class and reading of the initial consonant. The tone actually uttered must be determined from a complex set of rules that considers the class of the initial consonant, the final consonant, and the type and length of the vowel. More details of the Thai reading rules can be found in Appendix A. In addition, there is a repeated unit marker ๆ *(mai yamook)* which is used to alert the reader that the previous word is to be spoken twice.

- **Lao script:** The national government has been known for its strict control of all publications within Lao. All printed material must be approved by the government prior to printing or importation. Because it is difficult to gain such approval for publications in tribal languages especially if a non-Lao script is used, the Lao script is used for Iu Mien living in Laos. The complex rules for determining tones of syllables are similar to those used in Thai. More details can be found in Appendix A. Like Thai, the Lao script also has a repeated unit marker ໆ *(ko la)* which is used to mark words that are repeated. The tones actually uttered are determined by the Lao set of rules which are similar to those used in Thai.

Each of these phonetic writing systems of the Iu Mien was designed to be read as a stream of glyphs. From this stream, the reader is not only given clues to the pronunciation of the sequence sounds to be uttered but also provided with sufficient hints to allow correct decoding of the sentence, phrase and word units. Reading of written text is similar to the parsing of software compilers which is often modelled in Backus Naur Form (BNF).[29] BNF could be used to describe the decomposition of the graphemes within syllable units in a stream of Iu Mien text. This model could be used to generate both a computer parser of Iu Mien text as well as a train-tracks graphical rendering to enhance understanding of the decomposition process.

```
<paragraph>  ::=  <sentence>(<sentence>) ∗
 <sentence>  ::=  <phrase>(<whitespace><phrase>) ∗ <sentence.end.marker>
   <phrase>  ::=  <word>(<whitespace><word>) ∗ [<phrase.separator>]
     <word>  ::=  <syllable> ([<syllable.separator>] <syllable>) ∗
 <syllable>  ::=  [<initial.consonant>]<vowel>[<final.consonant>][<tone>]
```

Figure 1.2: BNF representation of Iu Mien in both Roman scripts

The general description of the BNF and train-track representations of Iu Mien in the Roman scripts are given in Figures 1.2 and 1.3, respectively. At this level of abstraction, the two scripts are essentially the same. It is noteworthy that in theory only a vowel and sentence terminator could define a whole utterance. In fact, such is the case with exclamations commonly used to express surprise, fear or pain.

The chief difference between the Roman scripts occurs in the definition of the primitives. As shown by Table 1.3, the Old Roman Script uses case-sensitivity to render the full range of initial and final consonants when the New Roman script employs multiple case-insensitive glyphs. In addition, the New Roman script uses character sequences that are closer to English phonetics in an attempt to facilitate the acquisition of English and French as a second language by the refugee community.

The Lao and Thai scripts share similar syllable decomposition patterns with the main difference occurring in the characters used and the range of sounds supported. These scripts are different from the Roman scripts described above in two ways. The vowels are broken into 3 segments that surround the initial consonant and an initial

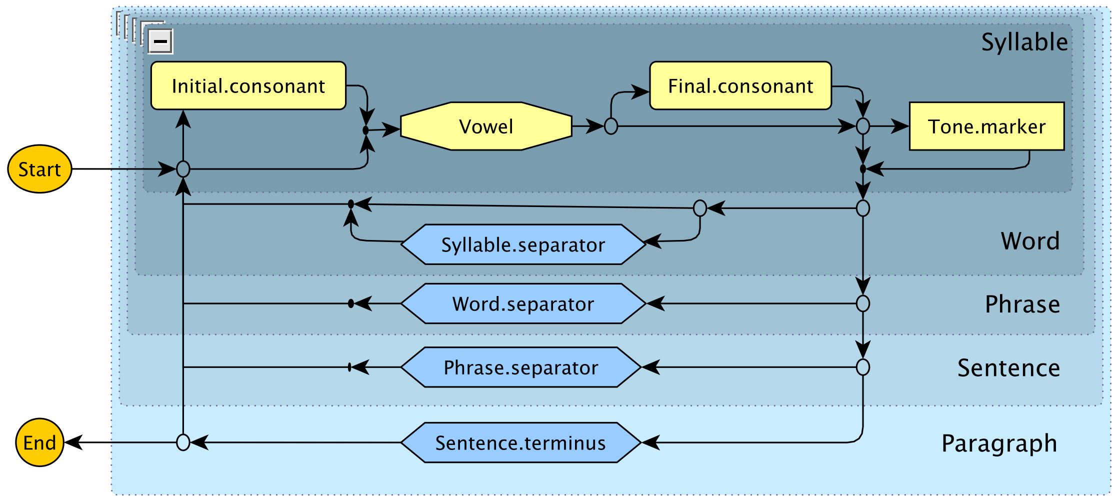


Figure 1.3: Train-tracks representation of decomposition of Iu Mien in Roman scripts

Table 1.3: Description of the primitives of Iu Mien syllables

Primitive	**Old Roman Script**	**New Roman Script**
<initial.consonant>	Single case-sensitive glyph	Multiple case-insensitive glyphs
<vowel>	Multiple case-sensitive glyphs	Multiple case-insensitive glyphs
<final.consonant>	Single case-sensitive glyph	Multiple case-insensitive glyphs
<tone>	Single character glyph	Single character glyph

consonant marker is required for each syllable. (Open syllables use a vowel marker as the initial consonant.) The BNF and train-tracks representations of Iu Mien in the Lao and Thai scripts are given in Figures 1.4 and 1.5. More details on these scripts can be found in Appendix A.

Successful writing systems tend to exhibit low levels of ambiguity in the decomposition of the text by a simple rule that symbols comprising in any one phoneme (depicted by a balloon in the train-tracks representation) cannot be members of any

<paragraph>	::=	<sentence>(<sentence>) ∗
<sentence>	::=	<phrase>(<whitespace><phrase>) ∗ <sentence.end.marker>
<phrase>	::=	<word>(<whitespace><word>) ∗ [<phrase.separator>]
<word>	::=	<syllable> ([<syllable.separator>] <syllable>) ∗
<syllable>	::=	[<leading.vowel>]<init.cons>[<vowel.tone>][<final.const>]
<vowel.tone>	::=	[<superimposing.vowel>][<tone.marker>][<trailing.vowel >]
< *init.cons* >	::=	[<consonant.class.modifier>]<initial.consonant>

Figure 1.4: BNF representation of Iu Mien in Thai and Lao scripts

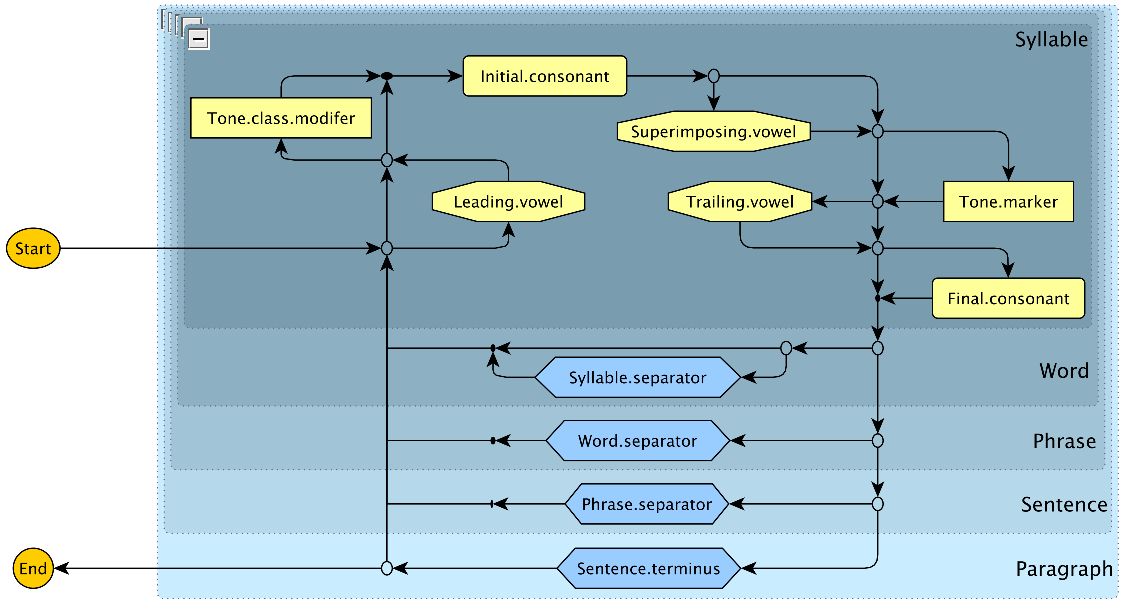


Figure 1.5: Train-tracks representation of decomposition of Iu Mien in Lao and Thai scripts

preceding or subsequent balloon.[6]. When this rule applies, it reduces the number of possible readings. However, this is not the case in all writing systems currently in use. For example, standard Thai and Lao text sequences exhibit a significant level of ambiguity. In these scripts, final consonant symbols are also used as initial consonant symbols. Without syllable or word boundary markers, alternative readings are possible as seen in the Thai text in Figure 1.6. Through the introduction of syllable and word break markers, the Iu Mien has eliminated this problem in their use of Thai and Lao scripts.

ตากลม

↙ ↘

ตา กลม	ตาก ลม
dha glom	*dhak lom*
(large eyes)	(airing in the wind)

Figure 1.6: Multiple readings of a given Thai text

Given the similarities mentioned above, it would appear that converting between two Roman scripts or between two Non-Roman scripts would be relatively easier than converting between a Roman script and a non-Roman script. The key differences between Roman and non-Roman scripts are described below. Each of these points increases the complexity of the transcription process and can be attributed to significant differences in phonology.

- **Complexity of the tone marking:** In the Roman script, the tone marker represents the actual tone to be read. In the Thai and Lao scripts, the tone marker generally modifies the effective class of the initial consonant. The spoken tone register is determined by the effective class of the initial consonant, vowel and final consonant.

- **Multiplicity of the initial consonants:** While each consonant of the Roman script is associated with a unique phoneme, many of the consonants in the Thai and Lao scripts share the same phoneme and differ only by the range of tones they support.

- **Vowel markers:** While the Roman scripts use a sequential and contiguous sequence of vowel markers to represent the different vowel sounds, the Thai and Lao scripts break the vowel into 3 separate segments that surround the initial consonant. All three parts must be read together to determine not only the intended vowel sound but the length and tone registers of the syllable as well.

The success of the Bible translation project was due to the selection of an internal representation that supported all the features of every script. Custom software provided the means to map from this format into the published scripts. As can be expected, the mapping was not trivial and did not correlate to a one-to-one mapping. Figure 1.7 illustrates the complex relationships between the phonemes of the internal representation and those of the various surface forms. For example, in the case of the vowels, the internal representation maps directly to single units in the Old and New Roman scripts, but upto 3 separate segments that surround the initial consonant when transcribed into Thai or Lao scripts.

1.5 A text corpus from the Iu Mien Bible translation

Having multiple scripts for Iu Mien has made it difficult for publishers to develop publications that reach all segments of this group of people. Although there had been several attempts to develop software to facilitate multi-script publications, the

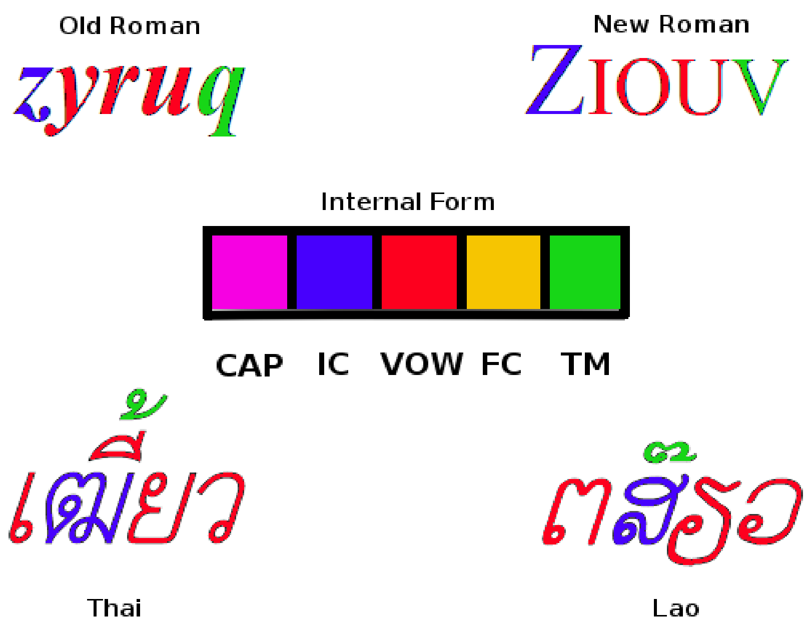


Figure 1.7: Mapping between the internal representation of the generic script to the surface forms of the other scripts

initial attempts failed because the programming environment used did not have a convenient way of handling the complex context sensitivity. When the Cat's Paw port of SNOBOL4 to MSDOS became available in 1983, a prototype of a transcription engine was developed in SNOBOL4 soon after. This transcription software was used to successfully support the simultaneous 1988 publication of a Iu Mien hymnbook in 3 script editions (Old Roman[30], New Roman[31] and Thai[32]), from a single manuscript rendered in the Old Roman script. With a few minor revisions, the prototype multiple script publishing system was upgraded to support the transcription of the Iu Mien New Testament which was published in 1995. During the publication of the full Bible translation in 2001, software was developed to support the transcription of the text into the Lao script for use among Iu Mien living in Laos. This allowed for the simultaneous publication of the Bible in 4 scripts of Iu Mien from a single manuscript.

The generic meta-script used for capturing the text of the Bible translation was primarily based on the Old Roman script with some additional markers added to support any features of the language that are rendered in the other Iu Mien transcription systems. These features included capitalization of proper names, standardized spelling of modified nouns and disambiguation of homographs unique to the Old Roman script. A system of software tools was developed to render and typeset the text in each target script. The correspondence between the different scripts in this internal form is shown in Figure 1.7.

Throughout most of the Bible translation project, all Iu Mien text was captured and edited in the generic script manuscript file. The publishing system developed for this project is shown in Figure 1.8. With this software, PDF review copies could be generated automatically in each of the target scripts and sent to the respective reviewers. All changes to the text were made in the generic script files. Throughout the life of the project, changes made to the generic script were seen in generated manuscript files which were in turn processed by TEX to generate the printed copies. Tables of exceptions were developed to support script-specific renderings that could not be handled by the rule-based software. By the end of the project there were about 100 entries in each exception table.

With the successful launch of the Iu Mien Bible translation, the separate manuscript files of 4 transcriptions comprise a significant corpus of text in multiple, parallel scripts. This valuable resource opens new opportunities for applying machine learning techniques to develop auto transcription software. It also facilitates research on the support of publishing from any of the 4 major scripts of this language. However, in order to meet publishing deadlines, late stage corrections and typesetting markups

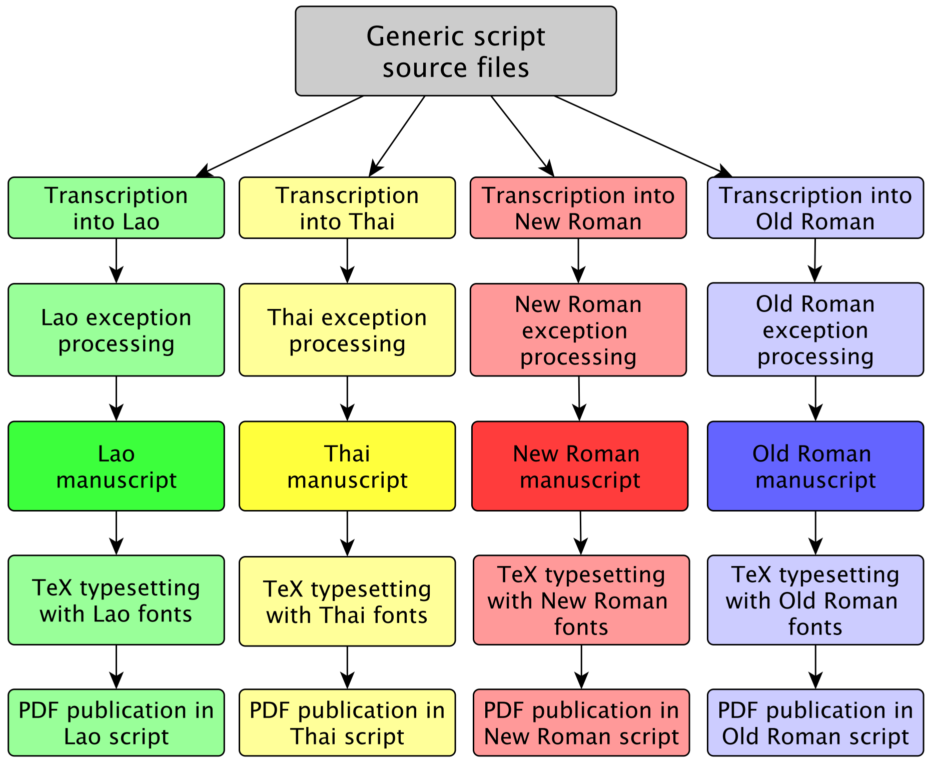

Generic script source files
Transcription into Lao
Transcription into Thai
Transcription into New Roman
Transcription into Old Roman
Lao exception processing
Thai exception processing
New Roman exception processing
Old Roman exception processing
Lao manuscript
Thai manuscript
New Roman manuscript
Old Roman manuscript
TeX typesetting with Lao fonts
TeX typesetting with Thai fonts
TeX typesetting with New Roman fonts
TeX typesetting with Old Roman fonts
PDF publication in Lao script
PDF publication in Thai script
PDF publication in New Roman script
PDF publication in Old Roman script

Figure 1.8: Bible publishing work flow from source text held in generic script.

were introduced manually into the individual manuscript files. This made it possible for various errors and inconsistencies to arise between the individual transcriptions. Given the amount of manual editing done under time pressure of the last stage of typesetting, one can expect that data cleansing would require a significant amount of effort before the separate text files can be merged into a single text corpus suitable for these studies.

Assembling the text corpus required loading the Bible text into a database and parsing the words and syllables into the 4 main components of a Iu Mien syllable (initial consonant, vowel, final consonant and tone marker). These entries in the text corpus facilitated the comparison of source glyphs to target glyphs. It also provided the resources for generating learning and test sets required by supervised machine learning technologies. Figure 1.9 shows a worst case scenario where every phoneme in a syllable can only be determined after analysis of the full context from the input script. This would require a neural network that could potentially require thousands of nodes and exhaust the available resources and/or exceed the statistical limits of the text corpus. Fortunately, techniques like decision tree analysis provide opportunities for pruning the neural network to a more compact and efficient network.

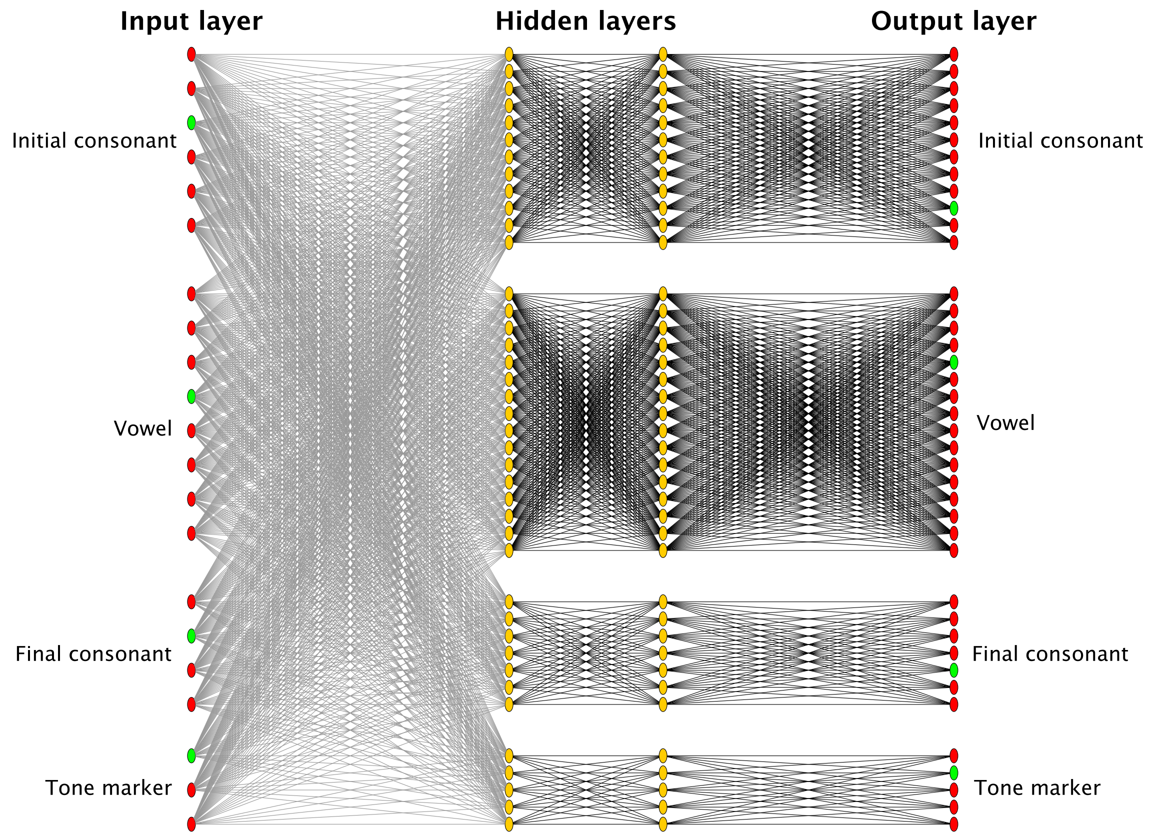


Figure 1.9: A neural network where the full input context is needed for each target phoneme.

This study also aims to determine which strategy shown in Figure 1.10 would be most appropriate for supporting the transcription of multiple scripts from any of the written scripts. A two step strategy involves converting an input script into a generic script before transcription. This strategy requires only 8 sets of transcription rules to support all 4 scripts. In addition, it would be easier for such a system to monitor accuracy by comparing the original text to reconstituted text in the same script. However, this introduces a 2-step transcription process which could increase the amount of processing time and decrease the accuracy of the output.

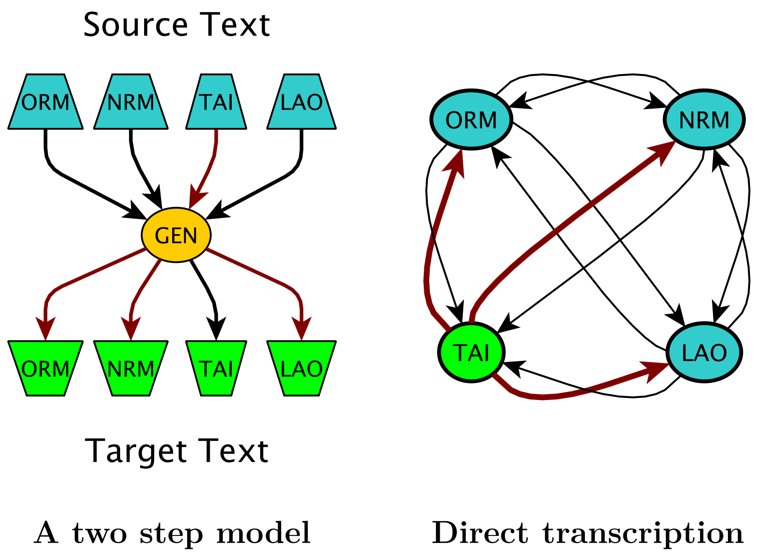


Figure 1.10: Two service models for transcribing between orthographies.

As shown in Figure 1.10, an alternative strategy would be to hold 12 sets of transcription rules that support direct transcription between any source script and its corresponding 3 target scripts. This approach has the potential of reducing the processing time, and should result in more accurate results especially when similarities between scripts can be exploited. By bypassing the generic script altogether, this strategy also holds better promise for ongoing development. The generic script was a by-product of the Bible translation project and currently has no stakeholders interested in further development of this part of the text corpus. Any strategy that does not require the generic script would help facilitate the inclusion of other new textual material into the text corpus from other multiple script publications.

The work flow for both strategies are shown in Figure 1.11. The two step transcript would follow the path of the red arrows and direct transcription would follow the blue arrows.

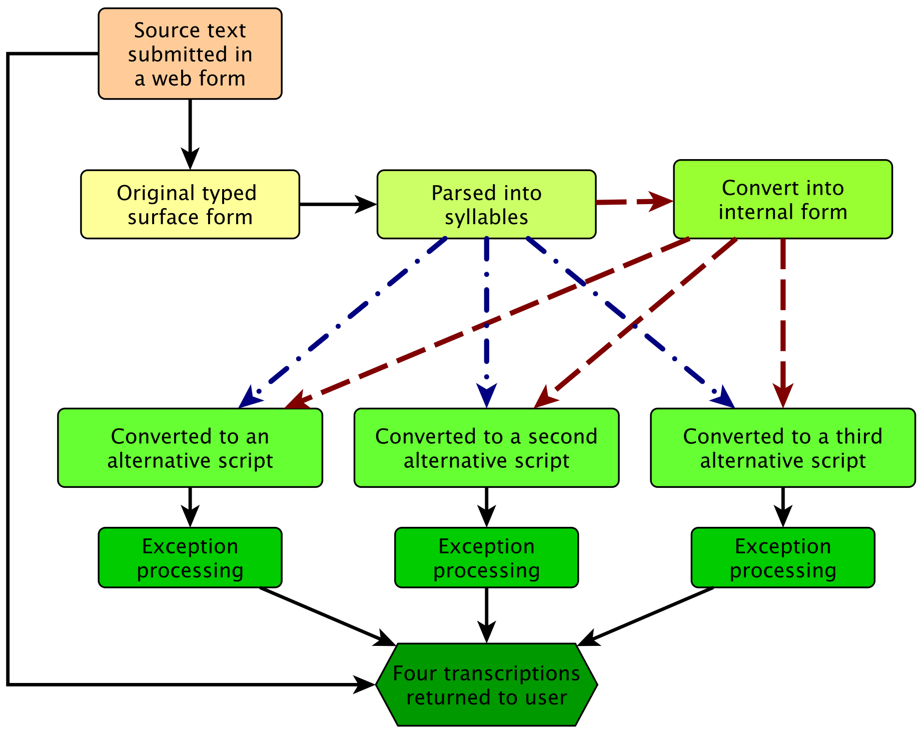

Source text submitted in a web form
Original typed surface form
Parsed into syllables
Convert into internal form
Converted to an alternative script
Converted to a second alternative script
Converted to a third alternative script
Exception processing
Exception processing
Exception processing
Four transcriptions returned to user

Figure 1.11: Work flow of an online system to transcribe between orthographies

1.6 Support for Unicode encodings

This project uses Roman, Thai and Lao scripts to encode Iu Mien words. The multi-script nature of this project makes this project vulnerable to undocumented features and bugs of both application software and operation systems especially since the source files were encoded in legacy 8 bit proprietary codepages. Editing software like Microsoft Word is designed to catch common English, Thai or Lao misspellings and non-standard characters. However, correct Iu Mien character sequences in the source files are often flagged as typos. While the spell checker can be turned off, there are some character sequences that cause Microsoft Office 2007 products to enter a mode of operation that prevents entry of additional characters until one of the preceding characters is removed.

However, this is not the only problem requiring attention. At this time, not all programming languages and application software provide full support for Unicode characters and extended ASCII character sets that use the full 8 bits of a byte (8-bit ASCII). The ability to handle Thai and Lao characters as distinct entities is essential to this project and can be demonstrated with a simple, 3-character regular expression (regex) as shown in Code Frag. 1.1.

```
initial_consonant = string.match/[ก-ฮ]/
```

Code Frag. 1.1: A Regex to retrieve the initial consonant of a Thai syllable

As per regular expression parsing of character ranges, any misinterpretation of the entities on either side of the hyphen character will change both the content and range of characters selected. Unfortunately, various interpretations of standard character encoding make it hard to capture a non-Roman regular expression as shown in

Table 1.4.

Table 1.4: Interpretation of regex parameters in different character encodings (See Code Frag. 1.1)

Input Encoding	Interpretation	Character representation ก	-	ฮ
8-bit ASCII	8-bit ASCII (bytewise)	`\xa1`	`\x2d`	`\xce`
8-bit ASCII	Unicode (Thai codepage)	`\x0e01`	`\x2d`	`\x0e2e`
8-bit ASCII	Unicode (Latin I codepage)	`\xc2a1`	`\x2d`	`\c38e`
UTF-8	8-bit ASCII (bytewise)	`\xe0\xb8\x81`	`\x2d`	`\xe0\xb8\xae`
UTF-8	8-bit Thai codepage	`\xa1`	`\x2d`	`\xce`
UTF-8	8-bit Latin I codepage	`\x04`	`\x2d`	`\x04`
UTF-8	Unicode (characterwise)	`\x0e01`	`\x2d`	`\x0e2e`

As shown in Table 1.4, if software was designed to support Unicode, the 3-byte sequence of UTF-8 would be correctly interpreted as a single code point in the Unicode set. If Unicode support is lacking all together, each Thai UTF-8 encoded letter will actually be handled as a string of 3 bytes. Partial support usually means converting Unicode character codepoints to the corresponding codepoints in the default codepage. Thus, if a Thai code page is in operation, this conversation results in mapping Thai characters from the Unicode standard to the corresponding character within the TIS620 standard code page. However, most installations of Windows, Linux and Mac used in America, Australia, Africa and Europe default to Latin I which is a collection of accents used with the Roman script. In these cases, the Thai letters will be lost as they map to a missing character code point. Under these conditions, it is possible to lose the Thai text if the file is saved or updated.

A similar pattern can be seen with 8-bit ASCII set. If these letters are handled as 8-bit units, there is no change of the code point values. However, most operating system services attempt to convert the upper ASCII characters to their Unicode equivalence depending on the default codepage. Saving or updating text under these conditions can also result in data corruption and loss.

This project will depend on careful selection of the components of the development environment to insure that none of them will introduce anomalies into the database and software developed. The reliability of the software can be verified with test files.

1.7 Machine learning of transcription

The text corpus of the parallel Iu Mien text can be broken down to create parallel lists of words in each script. Each word can then be broken down to create a parallel list of syllables. In turn the syllables give rise to a parallel list of phonemes. The phonemes form the basis for supervised machine learning of transcription. This study will focus on the use of the following two common techniques:

- **Decision tree learning:** Decision tree learning is capable of rule induction from the data. In this form of machine learning, various combinations of input attributes are paired to corresponding outcomes. This selection process is then reordered to produce the most efficient selection of outcomes based on the inputs. Decision tree learning algorithms order the conditions according to their corresponding entropies which is used as a measure of doubt about the possible conclusions. Entropy is determined according to measured probabilities, as shown in Equation 1.1.

  $$\text{entropy} = -\sum_{i=1}^{n} p(c_i|a_j)\log_2 p(c_i|a_j) \tag{1.1}$$

  The induction of the decision tree is achieved by multiple iterations which remove high entropy attributes so as to identify the next sub-tree that represents the most number of leaves of a common outcome. However, this iterative process often proves wasteful and impractical when applied to real world problems. The Iterative Dichotomiser 3 (ID3) algorithm developed by Ross Quinlan[33] improves the efficiency of this search by creating an initial decision tree from a sampling of the data. The initial decision tree is then used to identify new attribute vectors in the rest of the training set that were not handled by the initial rules. The newly discovered attribute vectors are then added to the sampled training set of vectors to generate a new decision tree. The final decision tree can be used to simulate transcription between phoneme markers of different scripts.

- **Neural network with backpropagation:** Neural network infers a function through the use of weighted links in hidden layers which connect inputs to expected outcomes. In 1974, Paul Werbos devised a method in which errors could be backpropagated within a learning mode of the neural network.[34] This mathematical operation would adjust the weights appropriately thereby improving the accuracy of the network output. After many iterations, the adjustments result in a network that models the expected outcome. The resulting network can be used to calculate the most likely equivalent phoneme marker in a target script given the source script phonemes of a syllable.

In both machine learning techniques, the technology was designed to select a single outcome from multiple choices. However, there are potentially thousands of discrete syllables in the Iu Mien, and it would be impractical to come up with a single decision tree or neural network to map the syllable transcription rules directly from one script to another. However, automated transcription could be simulated by generating separate decision trees or neural networks for each phoneme of an Iu Mien syllable. Assembling the outcomes for each phoneme network would result in a predicted rendering of the syllable, even for syllables that do not occur in the corpus used in this study. Using the resulting decision trees or neural network in a web application would provide the general public with access to this technology and would help to determine if the rules generated from this corpus have wider application within other literary domains of Iu Mien.

1.8 Online service

This study aims to deliver the transcription service online in the form of a Ruby on Rails application running on top of the web services of Heroku which was founded in 2007 as a cloud application platform for Ruby and was built upon the services provided by the Amazon Elastic Compute Cloud (Amazon EC2). The system was set up so that Ruby on Rails applications could be designed and developed locally. Once the applications are written and tested, they could be deployed to the cloud using version control commands of GIT. As a cloud based solution, the system monitor provides practical tools for measuring performance and use of computing resources. It also has the potential for handling bottlenecks and future expansion if the service becomes possible.

The Rails framework was chosen for developing an online transcription application because of its clear and consistent design which facilitates web development.[35] Rails has been implemented as a 3 part MVC architecture consisting of the following:

- **model (M) :** which captures the class definition of the data objects held in a database.
- **view (V) :** which renders data in an appropriate format.
- **controller (C) :** which interprets the user's request and heralds a response by

querying appropriate resources (both data objects and view renderings).

The Rails development framework also provides the developer with design tools and data structures that facilitate both object-oriented and behavior driven design. In addition, links between objects are fully supported by the relational attributes of Rails, such as `has_many` and `belongs_to`. Once the data of an application has been defined, command scripts are used to generate much of the required code automatically.

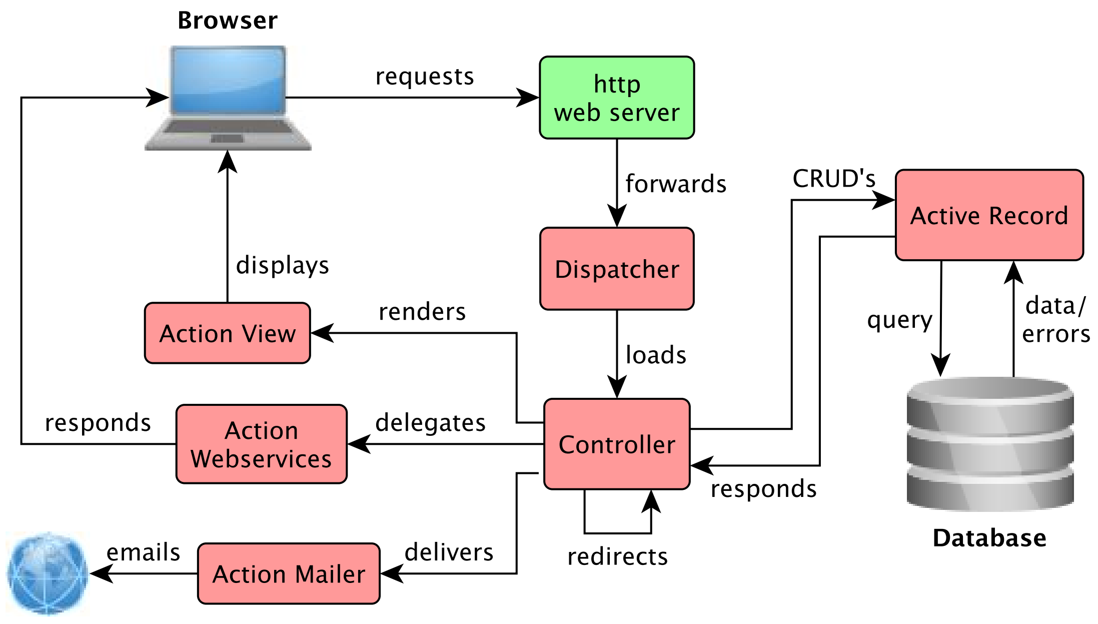


Figure 1.12: Interaction between the framework components within a Rails applications

The resulting web application leverages the MVC framework to respond to user commands as shown in Figure 1.12. When users issue a request from their browser, the hosting server forwards the request to the dispatcher that routes the request to appropriate controller which may redirect the request to another controller. A controller can also herald the relevant data by issuing queries to databases via the Active Record. The responses are collected and sent to the rendering engines that respond via standard http, AJAX, or email.

In this way, a Rails web development programmer has the advantage of an easy-to-use, powerful and flexible web development framework that focuses on the data classes and application behavior, instead of centering on the details of the individual web page objects as is common to other traditional web development platforms, such as PHP and Perl.[1] By leveraging this new technology, transcription rules developed through offline experiments can be easily adapted into a web application for testing by the larger Iu Mien community. [36]

1.9 Software documentation

The long term goal of this project is to make the databases and software resulting from this project available to the Iu Mien community for continued use and development by those who provide technical support to its publishers. As such, every effort has been made to document the working copies of the software and data structures in a fashion similar to literal programming.[37] The goal is to provide insights not only into how the software works but also into the reasoning behind the programming decisions made. While literate programming promotes the creation of better software, it works best when all components of a system are kept in a single source file in which the author has both described and defined the software in an pedagogical order that is consistent with what Knuth calls "a continuous stream of consciousness".[38] While

[1]It should be mentioned that the value of Rails has not gone unnoticed by web developers using Perl, PHP or Python. *cakePHP* for PHP, *Django* for Python and *Catalyst* for Perl were inspired by Rails and have seen popular and rapidly growing support within their respective communities.

the components are developed in human logical order, software utilities are required to restructure the source code into the order required by the compiler.

However, the chief benefit of using the Rails framework comes from allowing the associated Rails scripts to automatically generate hundreds of files, many of which will require only minor editing and updating. At this time, literate programming tools for Rails still do not exist. Instead, Ruby is shipped with RDOC, a built-in documentation module, which automatically generates a web of documentation from the class definition libraries. While overviews would require separate text files, description classes of objects as well as the attributes and methods are gleaned directly from the comments embedded in the code. While this is not exactly literate programming, RDOC does provide new readers of the code rapid access to the thoughts of the programmer and the implementation of the solution in code.

Ruby also supports both integrity and unit testing which are powerful paradigms for ensuring correctness of behavior from even the earliest stages of the project.[39] While units are tested as a series of assertions about expected values attributes or responses of methods, integrity tests deal with the responses seen at the user interface and can demonstrate the behavior of the system. These tools help to enforce a useful discipline of regular and frequent testing which is needed to ensure regular measured progress while minimizing unwanted surprises at the time of launch.[40]

One of the most recent additions to the Rails utilities is a behavior testing framework known as Cucumber[41]. Cucumber is built on a language called Gherkin which has only eight key words. The Gherkin interpreter was designed to be able to parse a detailed description of expected software behavior written in natural language. In this way, Cucumber documents serve as both system design documentation as well as specifications for automated testing. A simple description of login behavior captured in Cucumber is shown in Code Frag. 1.2.

```
Feature: Login authentication
As administrator to the site
I want to restrict access to the system configuration pages
In order to secure and protect the online service

Scenario: Unauthorized request for an admin page
Given I have logged in as 'testuser'
When I request 'admin services'
Then I should see 'You do not have permission to open this page'

Scenario: Authorized request for an admin page
Given I have logged in as 'adminuser'
When I request 'admin services'
Then I should see 'Welcome to Admin Services'
```

Code Frag. 1.2: A sample behavior specification in Cucumber

Cucumber specifications provide an opportunity for the programmer, system designer and end user to capture and share use cases and behavior descriptions in a human readable form. At the same time, each scenario in the specifications is read and executed as a test case by the system which is used to validate the behavior of the end product. This approach has great value not only for verifying that all major features have been included and tested, but it also helps to ensure correct operation at all phases of the project. This test framework has proven to be invaluable for ensuring continued operation even during major upgrades and refactoring of the source code.[42]

From the onset of this project, the objective has been to use best practices to develop a useful online transcription service that draws inferences from published texts and is built on reliable, documented and tested software. The following chapters will describe the measures taken and how well these goals have been realized.

CHAPTER 2

Methodology

2.1 Selection of the development environment

The source files of the Iu Mien Bible text used for this project were encoded in either ASCII (in the case of both Old and New Roman scripts) or in legacy 8-bit character encodings for the Thai and Lao script versions. In the case of the Thai script source files, the encoding of the Thai characters are identical to the current standard codepage TIS-620. However, the Thai fonts used with the source files also used upper ASCII code-points (i.e., those with values greater than 127). These non-standard code points were used to encode the no-break space, dash and bullet ligatures, Thai vowel and tone variant glyphs, and smart quote characters. Fortunately for the purposes of this study, the Iu Mien translation project primarily used character encodings instead of remapped glyph codepoint values.

However, some non-standard glyphs and character sequence of Thai vowels and tones were introduced by Thai word processors during the late stages of proofreading of the Bible. While this problem does not occur often and can be overcome by converting the non-standard glyphs to their UTF-8 equivalent character, most window systems will totally reject both non-standard characters or standard characters out of sequence.

In contrast, the Lao character set was a proprietary codeset, and a full remapping was required to convert it to the current Windows Lao codepage and UTF-8 code points. However, the standard Windows 7 text interface generally rejected 12 of the Lao characters and remapped the rest to accented Roman UTF-8 codepoints. In MS Office products, these characters were subject to even further modification due to autocorrection of the case accented of Roman characters.

Given this situation, a series of test files were generated to ensure that all systems and software applications supported the full range of characters without changing or omitting any. The following files were generated in binary mode using a program written in Ruby and their values were confirmed by inspection of a hexadecimal dump of the contents and by visual inspection of the characters in the Firefox html browser with the appropriate setting of the character encoding.

- **allcodetest.txt:** a sequential set of 256 bytes ranging in values between 0 and 255, useful for testing 8-bit ASCII handling.

- **unicodetest.txt:** a Unicode-encoded listing of ASCII characters, and a complete set of Thai and Lao consonants. Common combinations of Iu Mien consonants vowels and tones are also included to verify the proper handling of non-standard character sequences.

- **thaichrtest.txt:** an 8 bit-encoded listing of ASCII characters, Thai consonants with common combinations of vowels. Common combinations of Iu Mien consonants, vowels, and tones are also included to verify the proper handling of non-standard character sequences.

- **laochrtest.txt:** an 8 bit-encoded listing of ASCII characters, Thai consonants with common combinations of vowels. Common combinations of Iu Mien consonants, vowels, and tones are also included to verify the proper handling of non-standard character sequences.

The following test protocol was developed to ensure that systems and software could handle these files reliably. At the shell level, the files were copied as files as well as displayed text dump that was redirected to another file. The output of these files was compared with the original test files. The cut and paste operations were also used and the saved text was compared bytewise. Applications were tested by opening the text files, introducing a couple of spaces and then saving back to another file. The accuracy of the cursor control was tested by conveying the cursor over the common sequences before adding a space. The position of the spaces introduced into the saved files was compared with the intended position. Likewise, attempts were made to delete specific tone marks and vowels from multicharacter sequences to determine whether deletion of vowel and tones in multicharacter units was designed as a character by character operation or handled as a stack of diacritical marks.

The results of this preliminary study were used to select the software and operating systems used for this project. Although the initial decisions were made over five years ago, these tests had to be repeated regularly to ensure that upgrades to the software development environment did not bear unwanted surprises.

2.2 Development of a text corpus

Attempts were made to merge the original generic script file with the edited source files of the Old Roman, New Roman, Lao and Thai script editions of the Iu Mien Bible. Samples containing the corresponding introduction and initial 5 verses of 3 John are included in Appendix B along with more detail of the nature of the file structure of these textbases.

These 5 sets of text files were combined into a single text corpus that could be used for this study. This involved writing software filters to read the text and its associated text markup in order to break the text at appropriate places. Because all editions were originally derived programmatically from the same generic script source files, the punctuation and line breaks are generally located at the same place in all samples. These markers were used to synchronize the parallel text from each script.

However, there was a significant amount of manual correction of individual files during the last stages of the copy editing of the Bible manuscript. This process introduced a number of anomalies into the source files. Even the short samples given in Appendix B contain some anomalies between the versions. (Figure B.4 has new markers (`\gb`, and `\ths`) that were added only to the Thai script version.)

The process of merging the separate text files into a single text corps was accomplished with a series of object classes designed to progressively decompose the original source text into sets of smaller and smaller parallel fragments using specific textual elements as break points and delimiters. Within each stage of decomposition, the resulting fragments were sorted and combined together into an intermediate textbase which contained the corresponding text fragment for each of the 5 source files and the reference citation. The work flow of this process is shown in Figure 2.1.

Methods to check the resulting textbase for discrepancies and to ensure consistency were also added. The content of parallel units was tested for completeness by comparing the relative string length of the parallel entities. Strings exceeding by 1 standard deviation of the relative string length were manually inspected and corrected to reduce the possibility of having missing or mis-allocated text. This approach was used to remove comments, typesetter remarks and similar anomalies as well as to correct for frame shifts in the text base due to missing synchronization markers. Any errors and coding inconsistency were corrected before applying a subsequent class to the resulting textbase.

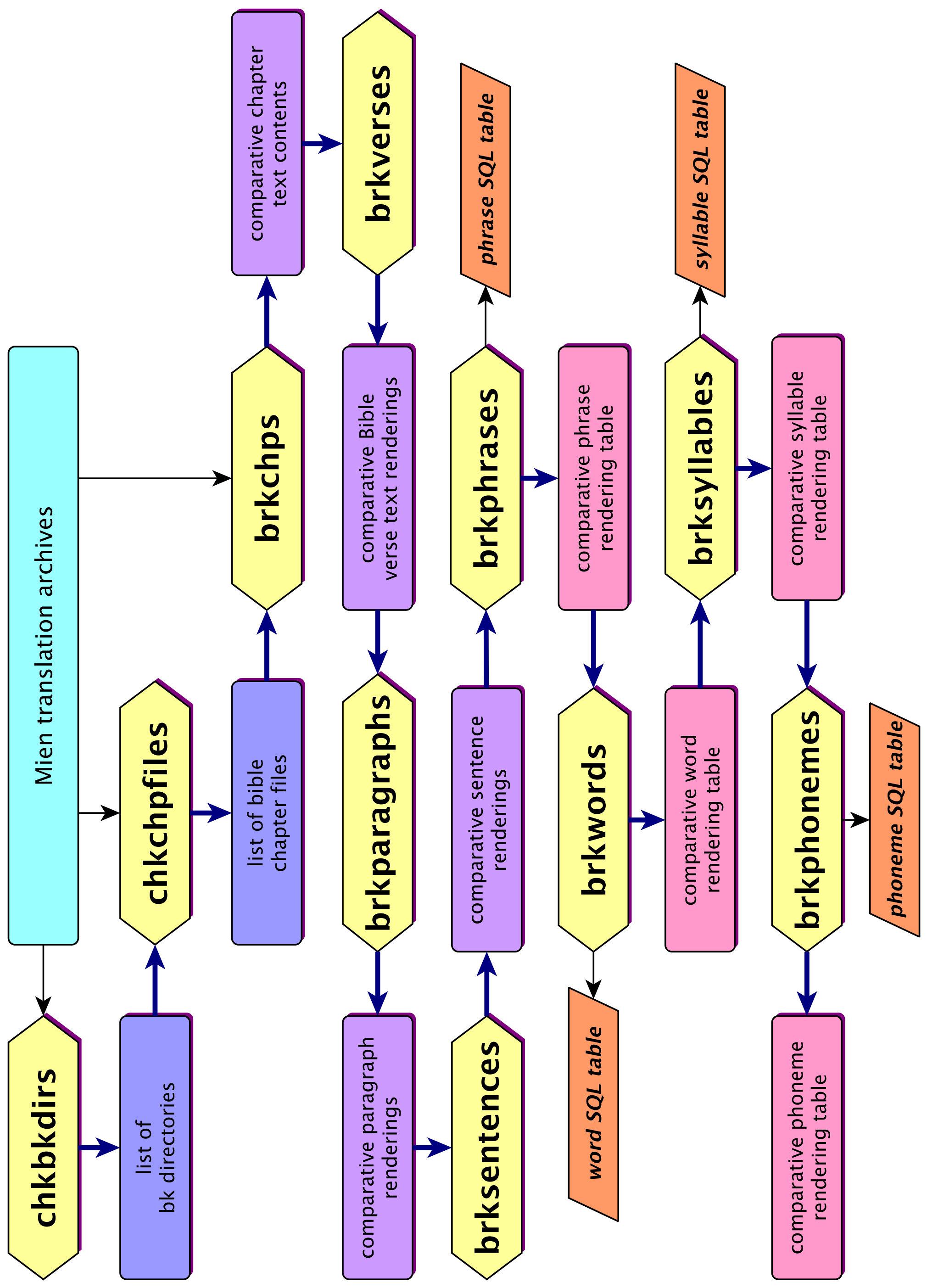

Mien translation archives
chkbkdirs
list of bk directories
chkchpfiles
list of bible chapter files
brkchps
comparative chapter text contents
brkverses
comparative Bible verse text renderings
brkparagraphs
comparative paragraph renderings
brksentences
comparative sentence renderings
brkphrases
phrase SQL table
comparative phrase rendering table
brkwords
word SQL table
comparative word rendering table
brksyllables
syllable SQL table
comparative syllable rendering table
brkphonemes
phoneme SQL table
comparative phoneme rendering table

Figure 2.1: Work flow used to build the Iu Mien corpus from archived text files

In this way, it was possible to create a corpus with a verifiable level of correctness. The different classes used in this break and merge process are shown in Figure 2.1 and are described below:

1. **chkbookdir:** The text was checked to ensure that the directories for each book of the Bible were available for processing.

2. **chkchpfiles:** The archive was checked to ensure that all source text files were accounted for. File naming inconsistencies are handled; missing text files were replaced.

3. **brkchps:** The text was broken down by chapters and the text for each chapter was combined together. Introductions were marked as Chapter 0 in order to be able to separate Biblical text from commentary with the hopes of later studying the influence of Hebrew, Aramaic and Greek phonetics on the machine learning at a later time.

4. **brkverses:** The text was separated into the corresponding verse text units. This approach not only provided a set of reference markers but also created milestones to provide a frame of reference for checking the text with that of the printed copies.

5. **brkparagraphs:** For any given verse, the text was broken by paragraph tags which corresponded to section headers, divisions of paragraphs and top level stanzas of poetry.

6. **brksentences:** This level broke the text by sentence-terminal punctuation such as `!?.` Some adjustments were required to ensure that all parallel units were present.

7. **brkphrases:** This level of separation broke on the text on all phrase terminal punctuation such as , ; : Some adjustments were required to resynch units that were entered manually without corresponding punctuation.

8. **brkwords:** This level of separation broke the text into parallel units of either proper names or white-space delimited words. The resulting textbase formed the basis of a SQL database of unique words and proper names used for testing word level processing.

9. **brksyllables:** This separated words and proper names into a list of syllables. The resulting textbase formed the basis of a list of unique parallel units of syllables.

10. **brkphonemes:** Corresponding syllables were broken down into an object oriented database of graphic units used to render the basic phonemes of Iu Mien syllables, (*i.e,* initial consonant, vowel, final consonant and tone marker). The results of this step were used for the parallel units of source and target text needed for supervised learning of auto-transcription. The records of this textbase were randomly divided between test and training sets for each attempt at supervised learning.

To simplify the development, testing and refinement of the software code used in this text process, each of the above stages represents a separate class of processing. This allowed for better control over the appropriate rules and exceptions that were needed at each stage. To illustrate this modular structure, the code for the first class in this process is given in Code Frag. 2.1

As shown in Code Frag. 2.1, this sample code of a class definition illustrates the way Ruby encapsulates related constants, attributes, getter and setter functions

```
#! /usr/ruby
# Class Bookcheck - checks availability of source text directories
# (c) Copyright 2011 by Robert Batzinger
class Bookcheck

  attr_accessor :rootdir, # root directory of archive
     :scripts, # array of subdirectory names for each script
    :dirlist, # Hash of book subdirectories found for each script
     :err  # collection of errors found

# A list of a standard Bible Book abbreviations
BIBLEBKS = "GEN|EXO|LEV|NUM|DEU|JOS|JDG|RUT|1SA|2SA|1KI|2KI|" +
   "1CH|2CH|EZR|NEH|EST|JOB|PSA|PRO|ECC|SNG|ISA|JER|LAM|EZK|" +
   "DAN|HOS|JOL|AMO|OBA|JON|MIC|NAM|HAB|ZEP|HAG|ZEC|MAL|MAT|" +
   "MRK|LUK|JHN|ACT|ROM|1CO|2CO|GAL|EPH|PHP|COL|1TH|2TH|1TI|" +
    "2TI|TIT|PHM|HEB|JAS|1PE|2PE|1JN|2JN|3JN|JUD|REV"

  # Constructor input parameters:
  # * rootdir: root directory of the project archive
  # * dirs: array of subdirectory names for each script
  def initialize(rootdir,dirs)
    @rootdir = rootdir
    @scripts = dirs
    @dirlist = Hash.new()
    @errs = Array.new()
  end

  # CLASS METHODS WOULD BE INCLUDED HERE ...

end
```

Code Frag. 2.1: Class Bookcheck definition within the code of the class definition. In addition, all comments, attributes, methods and parameters were automatically collected by the RDOC utility into a network of web pages which document the system.

At the same time, additional code was appended to the end of each class definition file to create an instance of the class and test the key methods provided. This approach facilitated the development by creating a comprehensive test of each class method within the definition of each class of this project. In this way, running the class definition file as a program resulted in an exhaustive test of the various methods of a class. However, including a class definition within another Ruby file (via the `require` operator) would result in the test code being ignored. Code Frag. 2.4 contains the code from the bottom of the Bookcheck Class (whose class definition is given in Code Frag. 2.1). The conditional statement on the first line of this fragment is used to determine if the class definition was loaded in test mode or not.

To complete this example, the actual code to run the book check is given in Code Frag. 2.4. The Ruby `require` directive was used to load the class definition into memory during script program execution. The Ruby utility RDOC was used to assemble the programming documentation into a web of pages.

2.3 Parsing syllables

Regular expressions were developed, tested and used for syllable parsing in these studies. Some preliminary studies were undertaken to attempt a machine learning of the parsing of Old Roman syllables. As shown in Figure 2.2, the Old Roman script is relatively easier to parse because of the low number of character sets involved.

The following three basic approaches to machine learning of parsing rules were attempted and compared against the results obtained by hand-crafted regular ex-

```
# finddirs - Makes a hashed list of all source directories found
# and compiles a list of errors for follow up

def finddirs
   if Dir.exist?(@rootdir)
      scripts.each {|script|
         BIBLEBKS.split('|').each {|bk|
            puts "Processing: #{bk}"
            workdir = [@rootdir,script, bk].join('/')
            worklabel = "#{bk}\|#{scrip}"
            if Dir.exist?(workdir)
               @dirlist[worklabel] = workdir
            else
               @errs.push("#{worklabel}: (#{workdir}) Missing")
            end
         }
      }
   else
      @errs.push("Root dir: (#{@rootdir}) Missing")
   end
end
```

Code Frag. 2.2: Definition of the finddirs method with the Bookcheck class

$$\textbf{ORM}\quad \underbrace{\overbrace{\Big[(cons_p)(vowel_p)\Big]?}^{\text{presyllable}}} \qquad \underbrace{\overbrace{(cons_i)(vowel+)(cons_f)?(tone)?}^{\text{syllable}}}$$

$$\textbf{NRM}\quad \underbrace{\overbrace{\Big[((cons_p)(vowel_p)\Big]?}} \qquad \underbrace{\overbrace{(cons_i+)(vowel+)(cons_f)?(tone)?}}$$

$$\textbf{TAI}\quad \underbrace{\overbrace{\Big[(cons_p)(vowel_p)\Big]?}} \quad \underbrace{\overbrace{(vowel_i?)(cons_i+)(vowel_s?)(tone?)(vowel_p*)(cons_f)?}}$$

$$\textbf{LAO}\quad \underbrace{\overbrace{\Big[(cons_p)(vowel_p)\Big]?}} \quad \underbrace{\overbrace{(vowel_i?)(cons_i+)(vowel_s?)(tone?)(vowel_p*)(cons_f)?}}$$

Figure 2.2: A regular expression definition of a Iu Mien Syllable

```
if __FILE__ == $0
# Test hash
 TESTUNITS = {'AAA' => ["GEN","EXO","LEV","NUM","DEU"],
  'BBB' => ["GEN","EXO","NUM","DEU"],   # missing one
   'CCC' => ["GEN","EXO","LEV","NUM","DEU"]}
  TESTDIR = 'testdirectory$$$'
  chk = Bookcheck.new(TESTDIR, ['AAA','BBB','CCC'])
  chk.finddirs
  assert("Root dir.+missing", chk.errs[0])

# Create test units
  Dir.mkdir('testdirectory$$$')
  TESTUNIT.keys.each {|inx|
     Dir.mkdir([TESTDIR,inx].join('/'))
     TESTUNIT[inx].each {|bk|
      Dir.mkdir([TESTDIR,inx, bk].join('/'))
     }
}
  chk = Bookcheck.new(TESTDIR, ['AAA','BBB','CCC'])
  chk.finddirs
  assert(5,chk.dirlist.grep(/AAA/).length)
  assert(4,chk.dirlist.grep(/BBB/).length)
 assert("#{TESTDIR}/BBB/LEV",chk.errs.grep(/BBB/)[0])
  assert(5,chk.dirlist.grep(/CCC/).length)

# Kill the test directory
  TESTUNIT.keys.each {|inx|
     TESTUNIT[inx].each {|bk|
      Dir.rmdir([TESTDIR,inx, bk].join('/'))
     }
     Dir.rmdir([TESTDIR,inx].join('/'))
  }
  Dir.rmdir('testdirectory$$$')
end
```

Code Frag. 2.3: Test routines for Class Bookcheck pressions.

- **Genetic algorithm:** Random attempts at developing the sets of characters that make up a standard Iu Mien syllable in Old Roman script.
- **Positional analysis:** Analysis of the data to determine which symbols only exist in the initial, medial or trailing positions of a syllable.
- **Hybrid approach:** Using the genetic algorithm constrained by positional rules.

2.4 Machine learning of transcription rules

In these studies, a Ruby implementation of the ID3[43] was used to generate decision trees. Each syllable of the unique word list was broken into a vector list of phonemes which were used as pre-classified examples. ID3 was then applied to generate a top-down induction of the corresponding decision trees.

Neural networks in this study were generated using a Ruby implementation of a multilayer perceptron with back-propagation learning.[43] Each syllable of the unique word list was broken into a vector list of phonemes. The unique elements for each phoneme were catalogued and enumerated over the full range of possibilities. The rank order of each element was then used to determine the corresponding outcome or input bit that should be set for the transcription engine, based respectively on whether the phoneme was part of the target or source syllable.

The bit field values for each phoneme were also combined together to create a list of input values. The bit fields of the target phonemes were used to represent the expected outcomes. These bit patterns were then used as respective outputs and inputs to the multi-layered perceptons which were train the neural networks. To illustrate this, Figure 2.3 shows how 3 bits of input might be processed in by a neural network of 2 hidden layers connected by links of varying weights to determine which of 2 bits is to be selected.

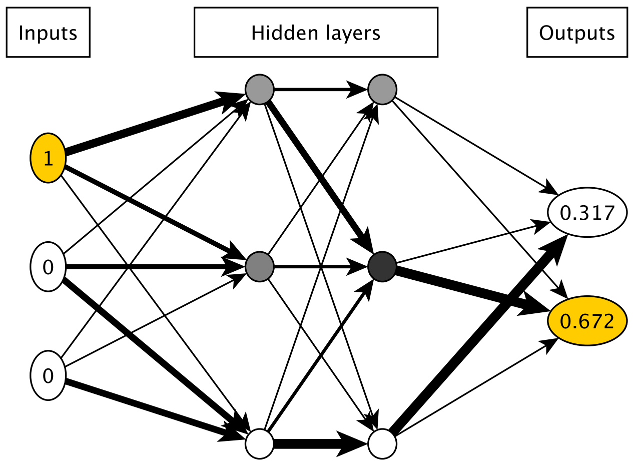


Figure 2.3: An example of a neural work

Although many linguistic rules are best modelled by a step function, the sigmoid function given in Eq. 2.1 was used in the hopes with the expectation that it was better suited towards the discovery of rules from datasets with known exceptions and typos. As shown in Figure 2.4, a step function is unforgiving at a threshold while a sigmoid function exhibits some smoothing of the transition making it better suited for the back propagation of errors when typos are present.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

With both machine learning techniques, the Ruby function shown in Code Frag. 2.5 was used to assign individual syllables randomly between the training set and test sets according to a given portion of all possible syllables. The portion of the size of

```
#! /usr/ruby
# Book check program - check the directory structure of the Iu
# Mien Bible text archives
# (c) Copyright 2011 by Robert Batzinger. All rights reserved.
require "bookcheck.rb"

puts "Checking the source directories"
src = Bookcheck.new('~/mientext',
        ['GEN', 'MNR', 'ORM', 'TAI', 'LAO'])
src.finddirs
if src.errs == []
    require 'yaml'

    fout = open('bookdirs.yml','w')
    fout.puts src.dirlist.to_yaml
    fout.close
    puts "Processing completed"
else
    puts "Errors found:"
    puts src.errs.join("\n")
end
```

Code Frag. 2.4: Running Class Bookcheck from within a program

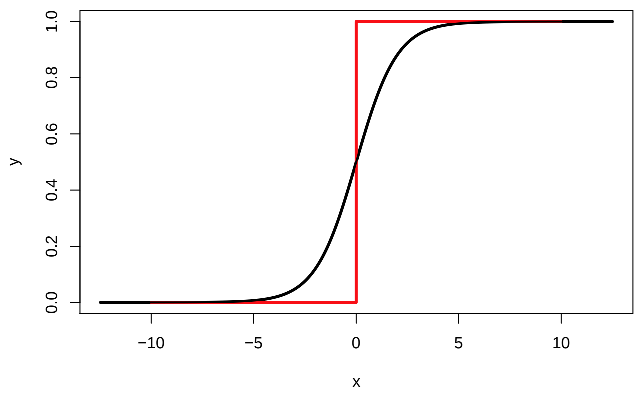


Figure 2.4: Sigmoid vs step function

the training set to that of the test set was varied to gain insight as to how much of the language would have to be sampled in order to develop practical and dependable rules. All studies were conducted in triplicate in an attempt to block sampling errors.

```
def choose_set(portion=0.5)
   rand < fraction ? :trainingset : :testset
end
```

Code Frag. 2.5: Method for assigning samples between the training and test set

Machine learning of the transcription rules was attempted through the use of two AI4R modules: ID3 (for decision teaching) and neural networks with backpropagation. The number of hidden layers needed in neural networks was empirically determined in an experiment that tested the outcome with the number of hidden layers ranging between 0 and 3. The AI4R package also gives Ruby programmers the ability to save the learned logic of ID3 and the generated weighted links of the neural networks in a format that could be loaded and used in application software.

2.5 Implementing transcription as a web service

The final stage of this project was to embed the transcription rules in a web application. The goal was to provide an opportunity for the Iu Mien community to have access to this technology. As all previous development had been done using Ruby as the programming language, it seemed natural to use Ruby on Rails as the development framework environment for developing the web application. Because Rails development framework implements solutions that are designed to manage the core data objects and related processes, the key data objects and processes of the web application were identified and documented within the Rails model.

The analysis began with the description and specification of the required behavior. This was recorded in plain English as per the standards of Cucumber. The specifications of the revised website have been included in Appendix C. Collection formal descriptions of the intended behaviors helped to elucidate the basic requirements of the web application. Analysis of these specifications suggested that the web application must manage the data needed to support the following two systems of data objects:

- **User accounts:** Authentication facilitates authorization of the range of services appropriate for each particular user via role-based permissions. Similarly, establishing formal roles makes it easier to establish views of the system in which users are only provided links and access only to services to which they have permission. The entity relation diagram (ERD) of the user accounts is given in Figure 2.5.

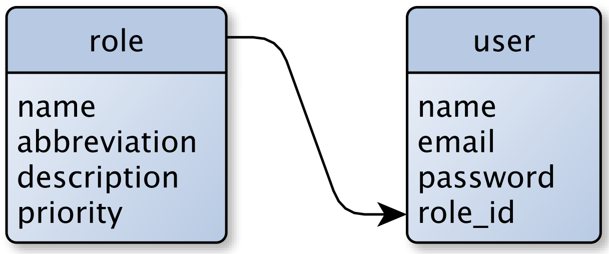


Figure 2.5: Entity relationship diagram of the user account management

- **Transcription jobs:** The ERD is given in Figure 2.6.

  The submitted text and its script identifier are accepted as input parameters to the job. The processor then converts the text to the other scripts, upgrading the

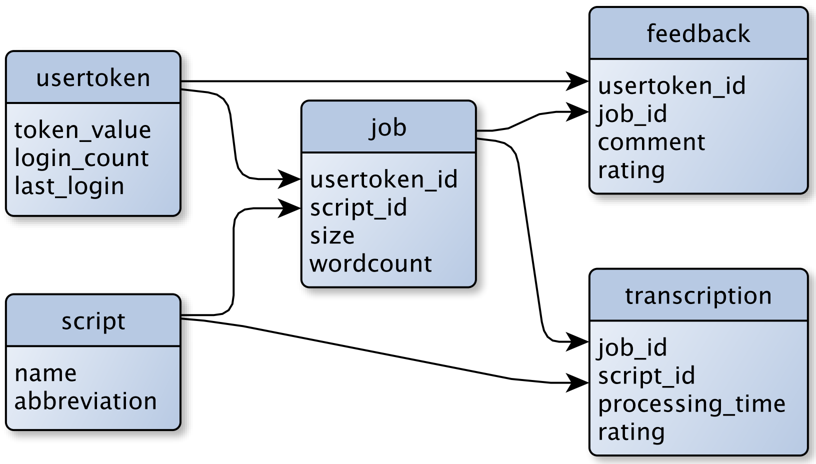


Figure 2.6: Entity relationship diagram of the transcription job management

the temporary page via AJAX for every thousand characters of text converted. Upon completion, the output page changes to display the converted text in each script.

In this project, data migration tables were used to specify the attributes of data objects. Attributes were assigned standard Rails data types which were automatically converted at the time of application deployment to SQL tables with corresponding fields of appropriate data types. A sample of the data migration file for the user is given in Code Frag.2.6. The `up` and `down` methods are executed during deployment and revision of database schemas.

The flexibility of this system was demonstrated when Heroku made its decision to switch its database support from mySQL to postgreSQL. In traditional web application development environments, this change would result in the rewriting database connection methods and operators. However, the migration could be accomplished by dumping the data to a yaml object file, changing the database configuration file

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table :users do |t|
      t.string :name
      t.string :email
      t.string :password
      t.integer :login_count
      t.integer :role_id
      t.timestamp :last_login
    end
  end

  def self.down
    drop_table :users
  end
end
```

Code Frag. 2.6: User data object

and then uploading the data. The changes made to the database file are shown in Code Frag. 2.7. It should be noted that the web application was developed and tested locally using a SQLite database. The Rails system was installed to automatically update the production database from the development database and upload the production system to the Heroku server cloud.

Analysis of the Cucumber specs also revealed the range of processes that would be required by the application. The links between them are shown in Figure 2.7. The corresponding web site was implemented in Rails and tested locally before uploading to the Heroku cloud providing public access to this service. The online service maintains a log file to record the frequency and type of use. Feedback is forwarded to a project email address.

using mySQL

```
development:
  adapter: sqlite3
  database: db/devel.sqlite3
  pool: 5
  timeout: 5000

test:
  adapter: sqlite3
  database: db/test.sqlite3
  pool: 5
  timeout: 5000

production:
  adapter: mysql
  host: localhost
  username: miendev
  password: 65a7f82d841...
```

$\longrightarrow$

using PostgreSQL

```
development:
  adapter: sqlite3
  database: db/devel.sqlite3
  pool: 5
  timeout: 5000

test:
  adapter: sqlite3
  database: db/test.sqlite3
  pool: 5
  timeout: 5000

production:
  encoding: unicode
  adapter: postgresql
  port: 5432
  host: localhost
  username: miendev
  database: mien
  password: 65a7f82d841...
```

Code Frag. 2.7: User data object

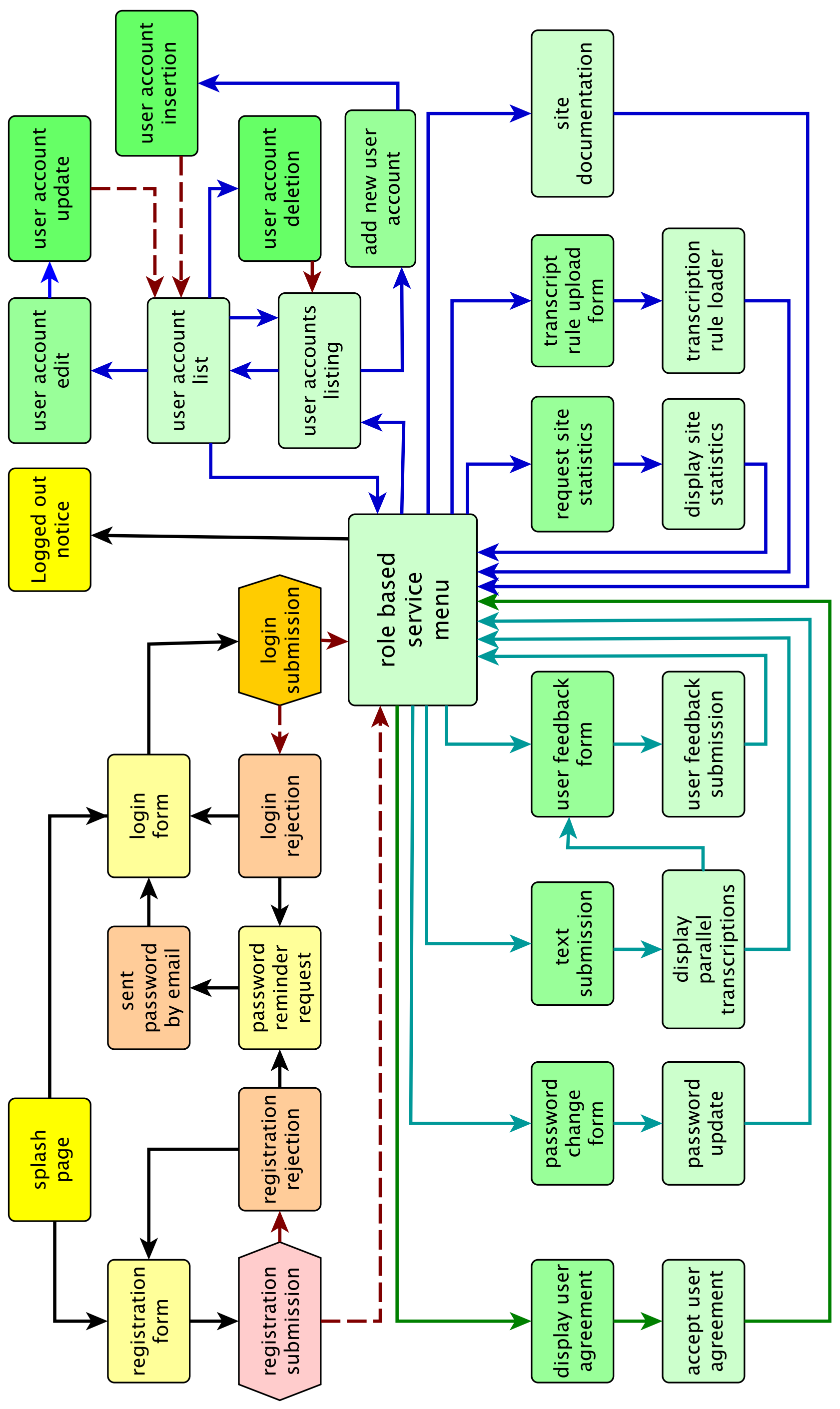


Figure 2.7: Webpage navigation map of the online transcription service

CHAPTER 3

Results

3.1 Fidelity of character handling

The source textfiles for this project were initially keyboarded and processed with software that ran on MSDOS 3.1. However, the software originally used to edit the files in Thai and Lao no longer works in modern version of Microsoft Windows because several of the DOS BIOS calls and the direct addressing of graphic memory have changed. However, it was quickly discovered that modern versions of Windows word processing software were changing the contents of the Thai and Lao text files.

Comparison of common codepages[1] provides the first hints of the source of this apparent lack of fidelity in handling legacy 8-bit character encodings. As shown in Table 3.1, approximately 25% of the 8-bit code space of a typical standard codepage are ignored as unknown letters. Ever since Windows 2000, the default behavior of the Microsoft Windows operating systems and much of the software programs that run on them is to replace characters of unknown codepoints with that of a box symbol. While this was meant to alert the user to character encoding problems, it does result in a loss of data. In Mac OsX, the default behavior has been to display the box mark but leave the code point untouched. The situation was made even more dire by the fact that different codepages have different regions of unknown character making it

[1]Codepages are tables of the underlying code numbers (or codepoints) that correspond to each character. For most languages of the world, codepages are registered as ISO standards and cover codepoints in a range of values between 32 and 255. Although most operating systems attempt to support Unicode, keyboard mappings and font glyphs are still linked codepages.

possible to lose data as default fonts and/or system codepages are changed.

Additionally, attempts to copying text into Windows text files are started with the detection of the file format. Those formats foreign to Windows and would automatically transform the contents into a Windows standards codepage. Experiments using the text files with encoded in different 8-bit codepages showed that this process worked well if the computer correctly detected the codepage in use and has full support for the corresponding codepage. However, most computer workstations on the Indiana University - South Bend campus are devoid of codepages of Asian languages. In addition, plain text files are merely a capture of the sequence of codepoints corresponding to the character sequence in the document with any codepage signature or identifier. Under these conditions, the Windows copy attempted to guess but would often use the default Roman codepage.[2] In addition, the system attempted to convert the characters to Unicode equivalents of the codepage. Therefore, files could not be given file names ending in `.txt` extensions and could only be copied in binary mode. In practice, the risk of losing Thai and Lao legacy-encoded characters on Windows 7 was high enough that the results of the initial months of processing was corrupt and had to be discarded. The decision was made to port the legacy text files to Mac OsX where file operations were more reliable.

However, a second problem arose when the text processing was attempted using Perl version 5.10. Although binary mode file operations were used, it was found that the Perl language assumed either standard codepage or Unicode encoding in the string operations and regular expressions. At this point, the project embraced the Ruby language which provided better control of encoding by allowing specification of the codepage for file, memory and string operations. In Ruby, there was even built-in support for forcing strings loaded in one codepage to be either interpreted

[2]In Microsoft Office products, the user is prompted for the underlying codepage.

Table 3.1: 8-bit Codepoints used in various codepage encodings

Filled circles represent standard codepoints the code page.

x	x0 x8	x1 x9	x2 xA	x3 xB	x4 xC	x5 xD	x6 xE	x7 xF
0	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○
1	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○	○○○○○ ○○○○○
2	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●
3	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●
4	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●
5	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●
6	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●
7	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ○●○○○
8	○○○○○ ○●○○○	○○○○○ ○●○○○	○●○○○ ○●○○○	○●○○○ ○●○○○	○●○○○ ○●○○○	○●○○○ ○○○○○	○●○○○ ○○○○○	○●○○○ ○○○○○
9	○○○○○ ○●○●○	○●○●○ ○●○●○	○●○●○ ○●○●○	○●○●○ ○●○●○	○●○●○ ○●○●○	○●○●○ ○○○●○	○●○●○ ○○○○○	○●○●○ ○○○○○
A	○○●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●
B	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●○● ●●●●●
C	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●
D	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●○●
E	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●○	●●●●● ●●●●●	●●●●● ●●●●○	●●●●● ●●●○○	●●●●● ●●●○●
F	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●●	●●●●● ●●●●○	●●●●● ●●●●○	●●●●● ●●●●○	●●●●● ●●●○○

Codepages: Latin1: ●○ UTF-8: ●○ ISO-8859-1: ●○ Lao: ●○ TIS-620: ●○

and/or converted into another codepage. The support for Unicode characters in Ruby was so good that even identifiers of objects, methods and attributes in programs could also be specified as Unicode text strings. To illustrate this, a Thai version of a recursive implementation of Euclid's algorithm to find the greatest common denominator (GCD) is shown in Code Frag. 3.1

```
# encoding: utf-8

# หรม: ตัวหารร่วมมาก
# (อังกฤษ: Greatest Common Divisor: gcd)
# ของจำนวนเต็มสองจำนวนซึ่งไม่เป็นศูนย์พร้อมกัน
# คือจำนวนเต็มที่มากที่สุดที่หารทั้งสองจำนวนลงตัว
# โดยขั้นตอนวิธียุคลิด (Euclid’s algorithm)
# (จาก http://th.wikipedia.org/wiki/ตัวหารร่วมมาก)
# ตัวอย่างเช่น
# 45/60 = 15 x (3/4) = 3/4
# ตัวเศษ (หรือ ต.ศ.) = 45
# ตัวส่วน (หรือ ต.ส.) = 60
# ตัวหารร่วมมาก (หรือ ห.ร.ม.) = 15

def หรม(ตศ, ตส)
    (ตส == 0)? ตศ : หรม(ตส, ตศ mod ตส)
end
```

Code Frag. 3.1: A recursive implementation of Euclid's GCD algorithm in Thai

With the wide range of character encodings supported by Ruby, it was possible to write code fragments that modeled the character confusion that had occurred in Windows and to write filters to unscramble unwanted character remapping. The Ruby code fragments shown in Code Frag. 3.2 were used to compare the difference between byte-wise and character-wise decomposition of a string in Thai script.

Code Frag. 3.2 was used in an experiment in which the default code page of the Ruby interpreter was set to one of three common codepages, *i.e.,* ISO-8859-1 (a

```
text = ″เย^ซู″

puts 'Method 1: Byte-wise iteration through a string: '
text.each_byte {|c| print "#{c.ord.to_s(16)} " }

puts 'Method 2: Character-wise iteration across a string: '
text.chars.each {|c| print "#{c.ord.to_s(16)} " }

puts 'Method 3: Indexed character iteration across a string: '
0.upto(text.length - 1) {|inx| print "#{text[inx].ord.to_s(16)} "}
```

Code Frag. 3.2: Hexadecimal dump of characters found by different string iterations

common accented Roman codepage of Windows 7, also known as Roman I), TIS-620 (a standard Thai codepage) and ASCII-8bit (an extended ASCII codepage). The tests were conducted in Windows 7 on a system with the locale set to Thai. In two test runs, the string was declared in one encoding and then converted to another encoding. The results are shown in Table 3.2.

The effect of the default system codepage can be seen in the ASCII-8bit results where the default system codepage was used to initially set the string. As the codepoints in this string exist in all three codepages tested, the byte-wise interpretation of the string was unaltered by switching codepages. However, UTF-8 remapped the codepoints to the corresponding Unicode character values according to the default system codepage. However, if the string was forced to assume the character mapping of ISO-8859-1, subsequent conversion to UTF-8 resulted in the remapping of Thai characters to accented Roman characters even if UPC Thai fonts were used as the default font. Fortunately, the system default codepage setting had no effect of text encoded in UTF-8.

Based on these studies, it was decided that conversion of the source text to UTF-8

Table 3.2: Effects of character encoding settings on the output
(See Code Frag. 3.2)
Colors indicate correct 8-bit Thai or **Unicode** encoding.

String encoding	**Method 1**	**Method 2**	**Method 3**
`ASCII-8bit`	`e0 c2 5e ab d9`[3]	`e0 c2 5e ab d9`[3]	`e0 c2 5e ab d9`[3]
`TIS-620`	`e0 c2 5e ab d9`[3]	`e0 c2 5e ab d9`[3]	`e0 c2 5e ab d9`[3]
`ISO-8859-1`	`e0 c2 5e ab d9`[3]	`e0 c2 5e ab d9`[3]	`e0 c2 5e ab d9`[3]
`UTF-8`	`e0 b9 80 e0 b8 a2 5e e0 b8 8b e0 b8 b9`[2]	**e40 e22 5e e0b e39**[2]	**e40 e22 5e e0b e39**[2]
`ISO-8859-1 → UTF-8` [1]	`c3 a0 c3 82 5e c2 ab c3 99`[4]	`e0 c2 5e ab d9`[3]	`e0 c2 5e ab d9`[3]
`TIS-620 → UTF-8` [1]	`e0 b9 80 e0 b8 a2 5e e0 b8 8b e0 b8 b9`[2]	**e40 e22 5e e0b e39**[2]	**e40 e22 5e e0b e39**[2]

[1] The string was specified in one encoding and then converted to another encoding.
[2] The string can be viewed as Thai using a Thai font encoded in Unicode,
[3] The string can be viewed as Thai only with a UPC Thai font encoded in TIS-620.
[4] The string has been converted to accented Roman script characters.

was well worth the effort in terms of reliability. In addition, UTF-8 text could be displayed in programs like Emacs and Eclipse making it easier to create regular expressions that could be edited in character form instead of hexadecimal representation that had been used previously. For the most part, converting the source text files to UTF-8 required a byte-wise conversion from the legacy coding to the corresponding UTF-8 codes as shown in Appendix A.

However, in some cases, the Thai script archived files had been changed to Roman I encoding by the Windows software used in the publishing process. In these cases, the Roman I UTF-8 encoded characters had to be remapped back the ISO-8859-1 codepage. At this point, the encoding attribute was changed to TIS-620 in order to force the text to be interpreted by the Ruby as text that can be remapped to Thai Unicode.

Similar processing was required of the Lao encoded text that was displaying as Roman I characters. However, at the time of this text processing, the Lao character set had not been fully accepted into the Unicode standard. As such, not all Unicode aware software would support characters in the Lao range.[44] To get around this, a Unicode to custom UTF-8 converter method shown in Code Frag. 3.3 was developed to support the proposed Lao codepoints that had been submitted to the Unicode Consortium. The Lao proposal was incorporated into the Unicode 5.0 standard[45] and full support for the Lao Unicode codepoints became available in Ruby in 2008. However, the corresponding ISO standard Lao codepage is still lacking as of this writing.

Code Frag. 3.3 calculates the multi-byte rendering of a codepoint by iterative bit shifting to strip off the least significant 6 bits at a time. The leading byte is used to identify the range of the Unicode character, the most significant digits and the number subsequent of data bytes. This routine makes it possible to work with proposed Unicode codepoints as well as user defined characters in the surrogate user planes in the Unicode codespace that ranges between 0x00 and 0x10FFFF and encodes for over 1 million characters.

3.2 Merging source text into a text corpus

The Iu Mien Bible translation source texts were obtained from the OMF translation team headed by Ann Burgess. The process of extracting the Iu Mien text from the source files in order to create a text corpus was described in Section 2.2 Sample text is shown in Appendix B. The markers used in these source files were a means to

```
# to_utf8 : Converts the vector of integers representing
# unicode code values byte-wise into a UTF-8 string via
# bit shift of the Unicode value creating Big-Endian UTF-8.

#  input parameter:  a list of Unicode code values
#  returned value: UTF-8 string

def to_utf8(uvector)
   output = ''
   @uvector.each {|c|
      case c

        # Append 7Bit ASCII characters to string
        when 0..127
          output = output + c.chr

       # Assemble and append multiple byte character code
        else
          chr = ''
          mask = 0b00011111
          offset = 0b10000000

          #  Little Endian processing
          while c >= mask
             x = (c & 0b111111) + 0b10000000
             # Packed as a Big Endian string of bytes
             chr = x.chr + chr
             c = c >> 6
             offset = (offset >> 1) + 0b10000000
             mask = mask >> 1
          end

          x = offset + c
          output = output + x.chr + chr
      end
   }
   output
end
```

Code Frag. 3.3: Converting a list of Unicode values into a UTF-8 string identify the textual elements in the Bible translation and conformed to a regional standard format for Bible manuscripts.[46]

A number of textual units in the source files were ignored for the purposes of this study because they commonly contained either non-phonetic sequences of abbreviations (as are common to footnotes, cross-references) or non-Iu Mien text (as found in file meta-data which referred to the processing status of the text in the file). A listing of the markers which were omitted in this study and the corresponding text patterns are shown in Table 3.3.

Table 3.3: Features of the source files which were excluded from this study

Feature description	**Marking**
Identification line	`\id ...`
Running page headers	`\h ...`
Footnotes	`\f . * ... *`
Cross-references	`\x . * ... *`
Table contents	`\tb ... \te`

While the number of Bible books processed shown in Table 3.4 matches the canonical Biblical count of 66 books, the number of chapters and verses found in the source text files did not match the chapter and verse counts of a standard Protestant Bible. Many of these discrepancies stem from the conventions used for the purpose of this study. First of all, the text of Bible book introductions was labeled and referenced as an additional chapter, referenced as Chapter 0 (Hence the extra 66 chapters over the canonical count of 1,189). Although the Iu Mien Bible text references the full set of 31,102 verses found in a standard protestant Bible, numerous sections of the Iu Mien translation of the Bible were translated as a cluster of verses which were counted as a single verse text unit instead of the corresponding range of verses. In addition, the text of introductions were also considered as a single verse unit.

Table 3.4: Processing statistics in the development of the Iu Mien Corpus

Process class name	**Total number found**	**Discrepancies found**	**Hrs required to complete**
chkbookdir	66	0	5
chkchpfiles	1,189	3	4
brkchps	1,255	25	5
brkverses	30,987	15	8
brkparagraphs	39,626	207	32
brkphrases	117,495	1,580	80
brkwords	1,023,320	76,793	169
uniqwords	11,224	3	2
brksyllables	34,770	117	2
uniqsyllables	3,320	987	10
brkphonemes	420,123	-	12
uniqphonemes	166	-	10

The processing of the source text took place part-time and was completed over several years of works. The source text was broken down into smaller units of text which were stored, tested and managed as units of parallel text. One of the unique properties of these parallel text units is that the sequence order of words and punctuation was consistent for all scripts. This provided addition referencing of text from the standard Bible book, chapter and verse to an extended system used for this project: Bible book, chapter, verse, paragraph, phrase, word and syllable. Sentence and phrase boundary punctuation were used as delimiters as an attempt to minimize alignment problem. The referencing system provided the precise referencing needed to identify and re-align misplaced text units when missing or extra punctuation were discovered. The statistics of the phrases found are given in Table 3.5

The times given in Table 3.4 represent the total amount of time spent developing and testing class definitions, processing text and handling exceptions. The alignment of paragraphs, phrases and words represented the greatest challenge to the process.

Table 3.5: Phrase break units found

Punctuation	**Count**
.	45,979
,	12,222
?	3,323
!	2,917
Total	64,441

The number of exceptions discovered on the first pass of each step are also given in the table.

Although the Iu Mien share a common language, local conventions in spacing were discovered. One source of difference is in the use of a hyphenation character to join adjectives to the noun they modify. This is clearly shown in Table 3.6 where the adjective *(new)* has been linked to the name of the city *(Jerusalem)* for the New Roman, Thai and Lao fonts but broken into separate whitespace delimited units in the Old Roman script. This kind of alignment error posed a major problem to subsequent processing because the word count would be off: Old Roman returns 3 words while the other 3 published scripts would return a word count of 2. These words were realigned by the whitespace to an underscore in all cases where the whitespace had been replaced by a hyphenation characters in any of the other scripts.

Table 3.6: A word alignment error of New Jerusalem City from Rev 21:2:1:3:2-3

Script	**text rendering**		
Generic	syav=	[ye-lu-saa-lem	zivb]
Old Roman	syavb	ye-lu-saa-lem	zivb
New Roman	siang-Ye^lu^saa^lem		Zingh
Thai	เซียง-เย^ลู^ซา^เลม		ฒี่ง
Lao	ຊຢັງ-ເຢ^ລູ^ຊາ^ເລມ		ຕສີ້ງ

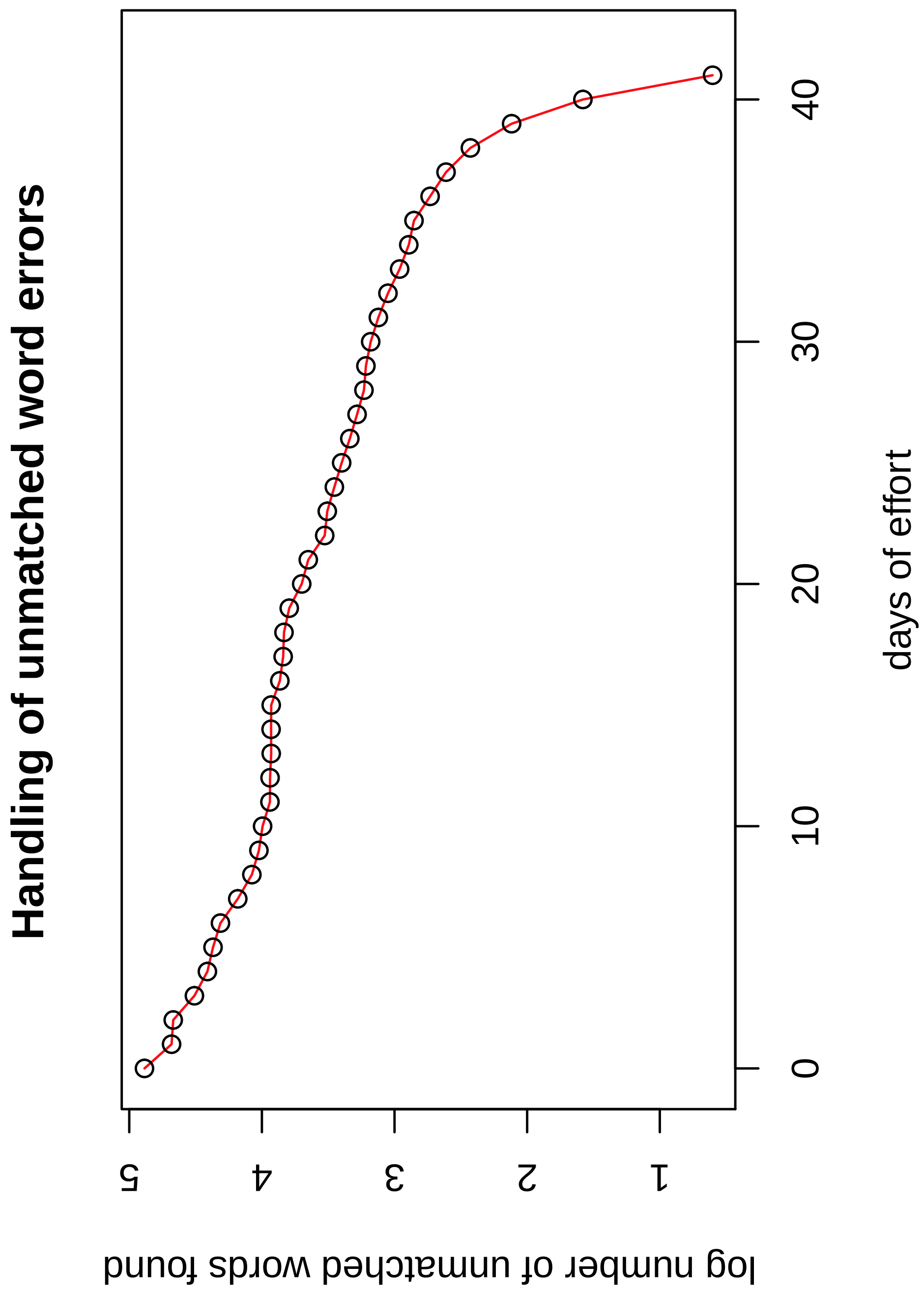


Figure 3.1: Effort required to align of words units in corpus

Because the alignment of the text into parallel text units was essential for building a corpus that could be used for supervised learning, considerable effort was spent on developing procedural and programmed methods to align the text and to verify the alignment. Figure 3.1 traces the efforts required to align words during the development of the word list. While the first attempts were able to quickly handle thousands of issues, there was an exponential growth in the effort to find and remove the last remaining detectable errors which often requiring new paradigms to effectively and efficiently handle low frequency issues with complex or multiple alignment shifts. In fact, this effort to create a word list without detectable word alignment errors proved to be generally log-linear.

Once a word list had been achieved the development of a syllable list was relatively easier. However, numerous ambiguities appeared in the syllable list. Table 3.7 shows one case where there were 4 entries for the New Roman and Old Roman syllable *yu*. In the majority of these entries, the long vowels are used. However, the Thai and Lao short vowels were also used occasionally. This raised concerns whether these unusual syllable patterns might actually be typographic errors especially because they appeared as single occurrences in a list of 1 million words and that the long ູ and short ຸ vowels are located on the same key of the keyboard.

The 17 words which contained the syllable *yu* are given in Table 3.9. First, it was noted that this syllable only occurred in proper names. The Lao ຢຸ໊ດາ was clearly a misspelled reference to Judah son of Jacob. The other occurrences represented the name Justus which occurs 3 times in the New Testament. The use of a short vowel in this proper name would be consistent with the short vowels of presyllables in Iu Mien words that have them. If this is true, then it would appear that the use of the long vowel in the Lao version of Titus Justus is a typo that is inconsistent with the

Table 3.7: Ambiguity in the rendering the syllable `yu`

Long vowels are printed in black and short vowels in red

Rendering by specified script					
Gen	Orm	Nrm	Tai	Lao	Frequency
yu	yu	yu	ยุ	ຢຸ	1
yu	yu	yu	ยุ	ຢູ	1
yu	yu	yu	ยู	ຢຸ	1
yu	yu	yu	ยู	ຢູ	14

other occurrences of this name.[3]

It is also known that in the late stages of the editing of this translation, separate focus groups proofread the draft editions of each script and submitted their recommendations to the translators and editors. This input resulted in numerous changes in the spelling of proper names which often reflected the spelling of the majority community more than common Iu Mien pronunciation. The words in the corpus were checked for consistency by checking for multiple entries for any word of any script. The differences that were merely vowel length changes were grouped separately from other types of changes. The cross tabulation is shown in Table 3.8

Table 3.8: Word inconsistencies

	Consistent rendering	Vowel length changes	Other changes	Total words
Mien words	3,418	17	47	3482
Proper names	7,154	269	315	7,738
Total	10,572	286	362	11,220

[3]Confirmation of this observation by native speakers of language is still pending.

Table 3.9: Words that contain the yu syllable

Long u vowels are printed in black and short u vowels in red

Generic	**Old Roman**	**New Roman**	**Thai**	**Lao**	Word Count
[yu-Baan]	yu-Baan	Yuˆmbaan	ยู^บาน	ຢູ^ບານ	2
[yu-Bu-latq]	yu-Bu-latq	Yuˆmbuˆlatv	ยู^บู^ลัด	ຢູ^ບູ^ລັດ	1
[yu-Daa]	yu-Daa	Yuˆndaa	ยู^ดา	ຢູ^ດາ	1,100
[yu-Daa]	yu-Daa	Yuˆndaa	ยู^ดา	ຢຸ^ດາ	1*
[yu-Daatg]	yu-Daatg	Yun̂daatc	ยู^ดาด	ຢູ^ດາດ	56
[yu-Dia]	yu-Dia	Yuˆndie	ยู^เดีย	ຢູ^ເດຍ	49
[yu-o-Dia]	yu-o-Dia	Yuˆoˆndie	ยู^โอ^เดีย	ຢູ^ໂອ^ເດຍ	1
[yu-Ditg]	yu-Ditg	Yuˆnditc	ยู^ดิด	ຢູ^ດິດ	1
[yu-Ti-katg]	yu-Ti-katg	Yuˆtiˆgatc	ยู^ท^กัด	ຢຸ^ທິ^ກັດ	2
[yu-fe-titg]	yu-fe-titg	Yuˆfeˆditc	ยู^เฟ^ติด	ຢູ^ເຟ^ຕິດ	59
[yu-lia]	yu-lia	Yulie	ยู^เลีย	ຢູ^ເລຍ	1
[yu-lopg]	yu-lopg	Yulopc	ยู^โหลบ	ຢູ^ຫລົບ	1
[yu-lyetq]	yu-lyetq	Yulietv	ยู^เลี้ยด	ຢູ^ລຽດ	2
[yu-ni-atg]	yu-ni-atg	Yuˆniˆatc	ยู^นี^อัด	ຢູ^ນີ^ອັດ	1
[yu-nitq]	yu-nitq	Yunitv	ยู^นิด	ຢູ^ນິດ	1
[yu-saa-Tatq]	yu-saa-Tatq	Yuˆsaaˆtatv	ยุ^สะ^ทัด	ຢຸ^ສະ^ທັດ	2*
[yu-sapq-he-setg]	yu-sapq-he-setg	Yusapv Hesetc	ยู^ซับ^เฮ^เสด	ຢູ^ຊັບ^ເຮ^ເສດ	1
[Ti-Ti-atg-yu-saa-Tatq]	Ti-Ti-atg-yu-saa-Tatq	Tiˆtiˆatc Yuˆ saaˆtatv	ทิ^ทิ^อัด^ยุ^ สะ^ทัด	ທີ^ທີ^ອັດ^ຢູ^ ຊາ^ທັດ	1*

* Indicates the use of the short yu vowel

The results would suggest a highly statistically significant increase in discrepancies among proper names over what was observed with common Iu Mien words. While many of these discrepancies seem to be vowel length shifts especially of proper names, it was also suspected that some of these discrepancies could be attributed to typographic errors that occurred during manual correction of individual occurrences of each altered proper name under the pressure of approaching deadlines.[4]

3.3 Characteristics of the Iu Mien text corpus

The resulting word list was separated into two lists: one containing all proper names used in the Bible and the other containing Iu Mien words. To compare these two lists, both lists were sorted by the normalized rank order and plotted against the normalized accumulative sum of the frequency for each word unit. Normalization was achieved by dividing rank by the total number of unique units and the accumulative sum by the total number of units found in the Bible. The resulting graph is shown in Figure 3.2.

Figure 3.2 clearly showed that the frequency distribution of proper names was different from that of the rest of the Iu Mien text. In fact, the corresponding histograms of the curves are statistically significant (with $p < 0.001$). As shown in Table 3.10, the most frequent proper name is the word for Lord, which alone represented nearly 12% of all proper names in this Bible translation. The 5 most frequent words together represented over 23% of all proper names. At the other end of the spectrum, there were 4,825 proper names (or 62% of all proper names) that only occurred only once in the entire Bible .

By contrast, Table 3.11 shows that the most frequent Iu Mien words represented less than 8% of all words and the 5 most frequent words together accounted for about

[4]Confirmation of the spellings by native speakers of language is still pending.

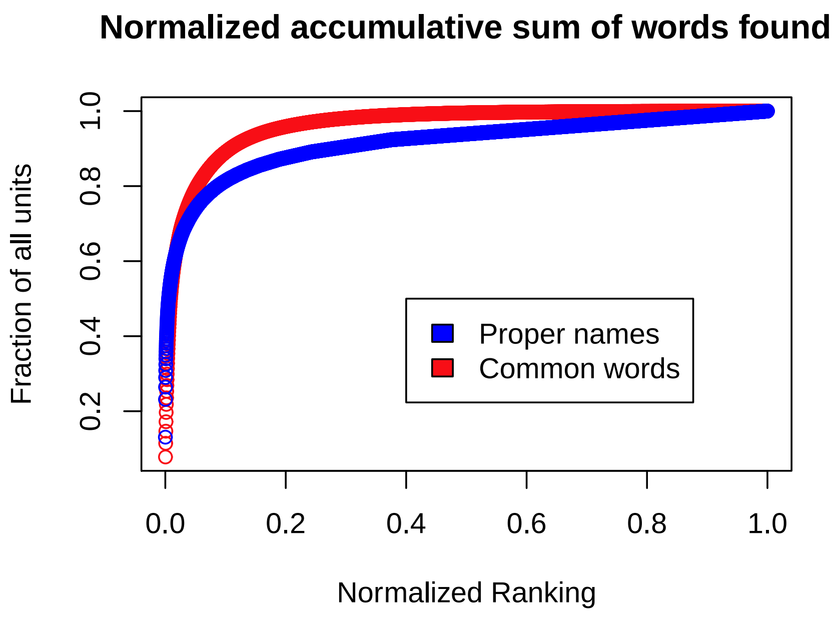


Figure 3.2: Comparison of normalized accumulative sum of unit frequencies

Table 3.10: The five most frequent proper names in the Iu Mien Bible

Old Roman	New Roman	Total count	Accum fraction
Tinb huvb	Tin-Hungh	6,375	0.116
ye-su	Yesu	2,083	0.154
i-saa-laa-en myenb	I^saa^laa^en Mienh	1,621	0.183
Daa-witq	Ndaawitv	1,177	0.204
i-saa-laa-en	I^saa^laa^en	1,019	0.223

22% of all words. Only 827 words occurred only once in the Bible, representing approximately 24% of the words used.

Table 3.11: The five most frequent Iu Mien words in the Bible

Old Roman	New Roman	Total count	Accum fraction
Eei	nyei	63,832	0.077
ninb	ninh	29,833	0.113
Bua	mbov	26,363	0.145
yia	yie	20,629	0.170
meib	meih	20,300	0.194
myenb	mienh	17,463	0.216

While the proper name distribution was heavily weighted for the extreme ends, i.e. the most and least frequent, the Iu Mien word distribution is a steady progression throughout the whole range. In fact, plotting log of the frequency of the Iu Mien distribution against the log of the rank resulted in a log linear graph that is consistent with Zapf's rule (which has been applied to many literary works in many languages).[47] However, analysis of the proper names in this way does not yield a linear relationship.

Table 3.12 provides some basic metrics on the text corpus retrieved from the Iu Mien Bible. It was interesting to note how 25 MBytes of files yielded only 3,320 unique parallel units of syllables. In addition, nearly 350 syllables found among the proper names were not seen in the rest of the Iu Mien text.

However, a number of statistics were calculated to better understand the differences between the Iu Mien words and the collection of Bible proper names transcribed into Iu Mien. However, some of the simplest equations also turned out to be the most revealing.

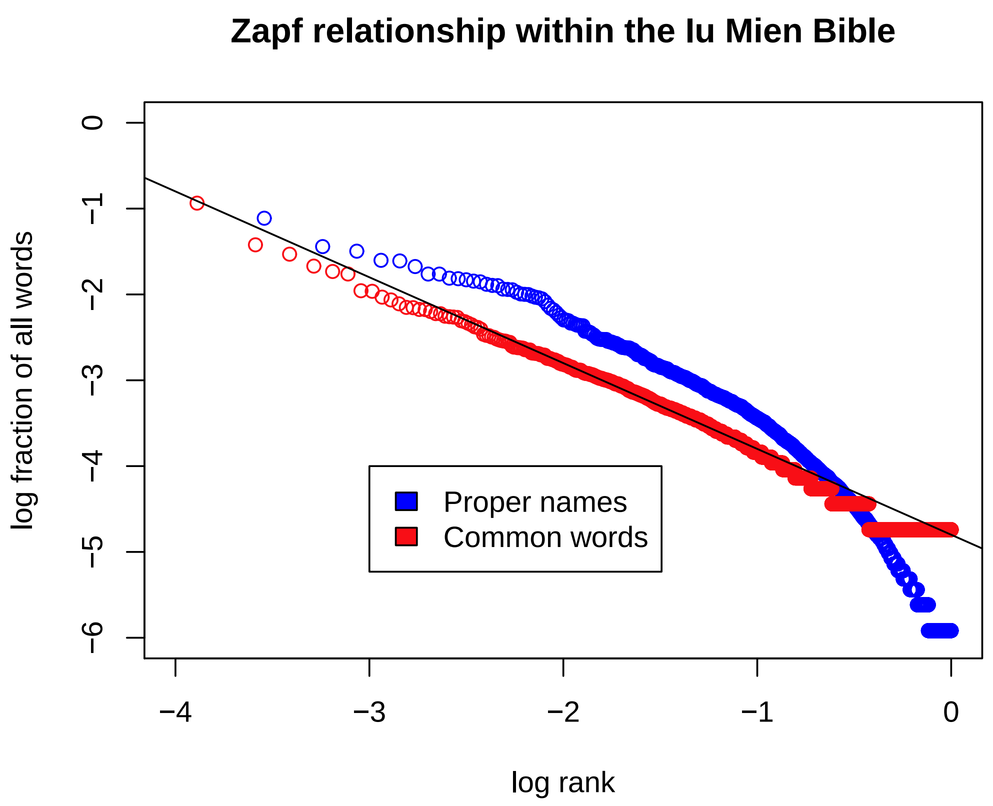


Figure 3.3: Normalized Zapf analysis of word frequencies in the Iu Mien corpus

Table 3.12: Raw metrics of the Iu Mien text corpus

Description	**Symbol**	**Proper names**	**Common words**	**All text**
Total byte count of the source files	N_{chr}	-	-	25,612,609
Verses found	N_{vs}	-	-	30,987
Sentences found	N_{sen}	-	-	52,219
Phrases found	N_{phr}	-	-	64,441
Word units found	N_{wrd}	87,747	817,918	905,665
Unique words units found	n_{wrd}	7,738	3,719	9,823
Syllables found in the unique word set	S_{wrd}	21,971	4,123	25,282
Unique syllables in the unique word set	s_{wrd}	871	3,001	3,320

Word-wise statistics

$$\text{Words per phrase} = \frac{N_{wrd}}{N_{phr}} \quad (3.1)$$

$$\text{Repetition of words} = \frac{N_{wrd}}{n_{wrd}} \quad (3.2)$$

Syllable-wise statistics

$$\text{Syllables per unit} = \frac{n_{wrd}}{S_{wrd}} \quad (3.3)$$

$$\text{Repetition of syllables} = \frac{S_{wrd}}{s_{wrd}} \quad (3.4)$$

Table 3.13: Basic statistics on the corpus retrieved from the Iu Mien Bible manuscript

Statistic	**Formula**	**Proper names**	**Common words**	**All words**	**PN fract.**[1]
Words per phrase	Eq. 3.1	1.36	12.69	14.05	0.097
Average word repetition	Eq. 3.2	11.34	219.93	92.20	0.388
Syllables per unit	Eq. 3.3	2.84	1.11	2.57	0.847
Average syllable repetition	Eq. 3.4	25.23	1.37	7.61	0.262

[1] Fraction of the outcome influenced by proper names

The statistics generated by these formulae are shown in Table 3.13. The differences between proper names and standard Iu Mien text can be clearly seen by these results. Proper names represent a minority of the text and have more syllables per unit than Iu Mien words some of which are associated with a presyllable. Through extrapolation it is possible to determine the amount of influence the proper names have on the statistics of the entire word list and corresponding syllable list.

A comparison of the distribution frequencies of syllables extracted from unique word lists is shown in Figure 3.4. These distributions were very different than those seen with the words in Figure 3.2. In addition, the distribution frequencies of syllables from proper names differed from those from common words. Syllables that occur only

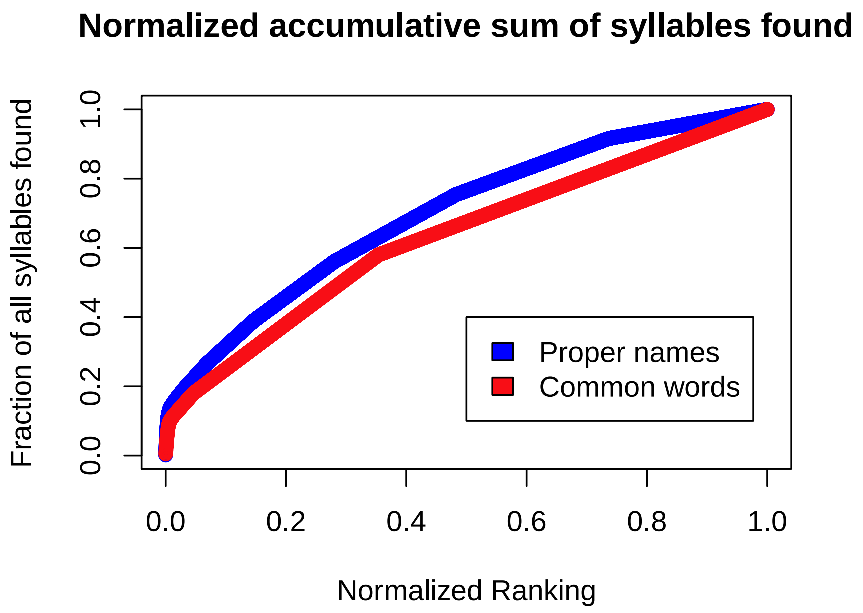


Figure 3.4: Comparison of normalized accumulative sum of syllable frequencies

once account for 65% of the syllables from proper names and 26% of the syllables from common words.

After discussion with various Iu Mien publishers, it was generally felt that it would be good to leave the proper names in the sample set used for supervised learning despite their differences from standard Iu Mien. The rationale was that Biblical proper names were an integral part of the kind of text documents that they would likely use with an automated transcription service. They would prefer a system that would be able to handle both text and Biblical proper names. Therefore, the project proceeded with the combined syllable lists.

3.4 Parsing the syllables

As described in Section 2.3, parsing of syllables was achieved using regular expressions developed, tested and run Ruby. The resulting segments were stored in a list of parallel tokens that could be randomly distributed between test and training sets. The list of tokens was analyzed to develop a complete catalogue of tokens used in each phoneme of the syllables. This catalogue of tokens was developed for each script. The results are given in Figures 3.5 to 3.9.

The complete catalogue of tokens was used as key to a map that replaced the characters tokens with a vector composed of a string of binary values (one bit string that is used as input to the neural network that determines the corresponding token in the outcome vector of the target script. A summary of the syllable input vectors and output token selection for each script is given in Table 3.14

Table 3.14: Size of input and outcome vectors for each script
Numbers represent the bit size of each vector

	Source	**Individual target vectors**			
Script	**vector**	**icon**	**vow**	**fcons**	**ton**
Gen	119	31	75	7	6
Orm	122	31	78	7	6
Nrm	123	38	71	8	6
Tai	179	129	37	7	6
Lao	220	160	47	7	6

The Lao input vector with 220 separate tokens was nearly twice the the size of the Roman script vectors. As seen earlier, initial consonants and vowels represent the tokens of greatest variability in terms of numbers. However, It should be noted that while the number of syllable final tone markers are the same across all scripts, there is a significant difference in the interpretation of tone marks in each script.

$$\begin{array}{cccc}
\textit{icons} & \textit{vowel} & \textit{fcons} & \textit{tone} \\
\begin{pmatrix}
\texttt{""| "B"|} \\
\texttt{"D"| "G"|} \\
\texttt{"K"|"P"|} \\
\texttt{"Q"| "R"|} \\
\texttt{"Z"|"f"|} \\
\texttt{"h"| "k"|} \\
\texttt{"l"| "m"|} \\
\texttt{"p"| "s"|} \\
\texttt{"t"| "z"|} \\
\texttt{"E"| "F"|} \\
\texttt{"H"| "J"|} \\
\texttt{"L"| "M"|} \\
\texttt{"N"| "T"|} \\
\texttt{"V"| "W"|} \\
\texttt{"Y"| "n"|} \\
\texttt{"v"}
\end{pmatrix} &
\begin{pmatrix}
\texttt{""| "a'"| "i"| "u'"| "a"|} \\
\texttt{"a'ai"| "a'ei"| "a'i"| "a'o"|} \\
\texttt{"a'waa"| "a'ye"| "aa"| "aa'"|} \\
\texttt{"aai"| "aau"| "ai"| "au"| "c"|} \\
\texttt{"c'"| "e"| "e'"| "ei"| "eu"|} \\
\texttt{"i'"| "ia"| "ia'"| "iu"| "o"|} \\
\texttt{"o'"| "oi"| "r"| "ru"| "u"|} \\
\texttt{"ua"| "ua'"| "wa"| "wa'"|"waa"|} \\
\texttt{"waai"| "waau"| "wai"| "wc"|} \\
\texttt{"wc'"| "we"| "wei"|"wei'"| "wi"|} \\
\texttt{"wo"| "wr"| "wu"| "x"| "x'"|} \\
\texttt{"xi"| "ya"| "yaa"|"yaau"| "yai"|} \\
\texttt{"yau"| "yc"| "ye"| "yei"| "yi"|} \\
\texttt{"yia"| "yia'"| "yiu"| "yo"|} \\
\texttt{"yru"| "yu"| "yu'"| "yua'"|} \\
\texttt{"ywa"| "ywi"| "ywr"| "yx"| "yx'"}
\end{pmatrix} &
\begin{pmatrix}
\texttt{""|} \\
\texttt{"k"|} \\
\texttt{"m"|} \\
\texttt{"n"|} \\
\texttt{"p"|} \\
\texttt{"t"|} \\
\texttt{"v"}
\end{pmatrix} &
\begin{pmatrix}
\texttt{""|} \\
\texttt{"b"|} \\
\texttt{"d"|} \\
\texttt{"g"|} \\
\texttt{"j"|} \\
\texttt{"q"}
\end{pmatrix}
\end{array}$$

Figure 3.5: Tokens in a Generic script syllable input vector

$$\begin{array}{cccc}
\textit{icons} & \textit{vowel} & \textit{fcons} & \textit{tone} \\
\begin{pmatrix}
\texttt{""| "B"|} \\
\texttt{"D"| "G"|} \\
\texttt{"K"| "P"|} \\
\texttt{"Q"| "R"|} \\
\texttt{"Z"| "f"|} \\
\texttt{"h"| "k"|} \\
\texttt{"l"| "m"|} \\
\texttt{"p"| "s"|} \\
\texttt{"t"| "z"|} \\
\texttt{"E"| "F"|} \\
\texttt{"H"| "J"|} \\
\texttt{"L"| "M"|} \\
\texttt{"N"| "T"|} \\
\texttt{"V"| "W"|} \\
\texttt{"Y"| "n"|} \\
\texttt{"v"}
\end{pmatrix} &
\begin{pmatrix}
\texttt{""| "'"| "a"| "a'"| "aa"|} \\
\texttt{"c"| "e"| "i"| "o"| "u"|} \\
\texttt{"u'"| "ua"| "wa"| "waa"|} \\
\texttt{"wc"| "we"| "wi"| "wo"|} \\
\texttt{"wr"| "x"| "xye"| "ya"|} \\
\texttt{"yaa"| "ye"| "yi"| "yo"|} \\
\texttt{"yu"| "ywa"| "ywr"| "yx"|} \\
\texttt{"a'ai"| "a'ei"| "a'i"|} \\
\texttt{"a'o"| "a'waa"| "a'ye"|} \\
\texttt{"aa'"| "aai"| "aau"| "ai"|} \\
\texttt{"au"| "aye"| "c'"| "e'"|} \\
\texttt{"ei"| "eu"| "i'"| "ia"|} \\
\texttt{"ia'"| "iu"| "o'"| "oi"|} \\
\texttt{"r"| "ru"| "ua'"| "wa'"|} \\
\texttt{"waai"| "waau"| "wai"|} \\
\texttt{"wc'"| "wei"| "wei'"| "wu"|} \\
\texttt{"x'"| "xi"| "yaau"| "yai"|} \\
\texttt{"yau"| "yc"| "yei"| "yia"|} \\
\texttt{"yia'"| "yiu"| "yru"| "yu'"|} \\
\texttt{"yua'"| "ywi"| "yx'"}
\end{pmatrix} &
\begin{pmatrix}
\texttt{""|} \\
\texttt{"k"|} \\
\texttt{"m"|} \\
\texttt{"n"|} \\
\texttt{"p"|} \\
\texttt{"t"|} \\
\texttt{"v"}
\end{pmatrix} &
\begin{pmatrix}
\texttt{""|} \\
\texttt{"b"|} \\
\texttt{"d"|} \\
\texttt{"g"|} \\
\texttt{"j"|} \\
\texttt{"q"}
\end{pmatrix}
\end{array}$$

Figure 3.6: Tokens in an Old Roman script syllable input vector

$$\begin{array}{cccc}
\textit{icons} & \textit{vowel} & \textit{fcons} & \textit{tone} \\
\begin{pmatrix}
\texttt{""| "b"|} \\
\texttt{"c"| "d"|} \\
\texttt{"f"| "g"|} \\
\texttt{"h"| "k"|} \\
\texttt{"l"| "m"|} \\
\texttt{"mb"| "nd"|} \\
\texttt{"nq"| "nz"|} \\
\texttt{"p"| "q"|} \\
\texttt{"s"| "z"|} \\
\texttt{"hl"| "hm"|} \\
\texttt{"hn"| "hng"|} \\
\texttt{"j"| "mc"|} \\
\texttt{"mh"| "mv"|} \\
\texttt{"mx"| "n"|} \\
\texttt{"nc"| "ng"|} \\
\texttt{"nh"| "nj"|} \\
\texttt{"nv"| "nx"|} \\
\texttt{"t"| "v"|} \\
\texttt{"w"| "x"}
\end{pmatrix}
&
\begin{pmatrix}
\texttt{""| "'"| "a"| "a'"| "aa"|} \\
\texttt{"aau"| "ae"| "ai"| "au"|} \\
\texttt{"e"| "ei"| "i"| "o"| "oei"|} \\
\texttt{"oi"| "or"| "u"| "u'"|} \\
\texttt{"ui"| "ya"| "yaa"| "yaau"|} \\
\texttt{"yae"| "ye"| "yi"| "yie"|} \\
\texttt{"yo"| "yor"| "you"| "yu"|} \\
\texttt{"yua"| "yuo"| "a'ai"|} \\
\texttt{"a'ei"| "a'i"| "a'o"|} \\
\texttt{"a'waa"| "a'yie"| "aai"|} \\
\texttt{"er"| "eu"| "i'"| "ia"|} \\
\texttt{"iaa"| "iaau"| "iae"| "iau"|} \\
\texttt{"ie"| "iei"| "io"| "ior"|} \\
\texttt{"iou"| "iu"| "iua"| "iui"|} \\
\texttt{"ou"| "ua"| "uaa"| "uaai"|} \\
\texttt{"uae"| "uai"| "ue"| "uei"|} \\
\texttt{"uo"| "yaai"| "yai"| "yau"|} \\
\texttt{"yei"| "yiu"| "yuei"| "yui"}
\end{pmatrix}
&
\begin{pmatrix}
\texttt{""|} \\
\texttt{"k"|} \\
\texttt{"m"|} \\
\texttt{"n"|} \\
\texttt{"ng"|} \\
\texttt{"p"|} \\
\texttt{"q"|} \\
\texttt{"t"}
\end{pmatrix}
&
\begin{pmatrix}
\texttt{""|} \\
\texttt{"c"|} \\
\texttt{"h"|} \\
\texttt{"v"|} \\
\texttt{"x"|} \\
\texttt{"z"}
\end{pmatrix}
\end{array}$$

Figure 3.7: Tokens in a New Roman script syllable input vector

icons

″″| ″ก″| ″ข″| ″ฆ″| ″จ″| ″ฉ″| ″ด″|
″ต″| ″บ″| ″ป″| ″ผ″| ″พ″| ″ม″|
″ล″| ″ส″| ″ห″| ″หฑ″| ″หฒ″| ″หธ″|
″หล″| ″อ″| ″กว″| ″ขว″| ″ค″| ″คว″|
″ฆว″| ″ง″| ″งว″| ″จย″| ″จว″| ″ช″|
″ชว″| ″ซ″| ″ซย″| ″ซว″| ″ฌ″| ″ญ″|
″ญย″| ″ญว″| ″ฑ″| ″ฑย″| ″ฑว″| ″ฒ″|
″ฒย″| ″ฒว″| ″ตว″| ″ถ″| ″ถย″| ″ถว″|
″ท″| ″ทย″| ″ธ″| ″ธย″| ″ธว″| ″น″|
″บย″| ″บว″| ″ปย″| ″ปว″| ″ผ″| ″ผย″|
″ผว″| ″ฝย″| ″ฝว″| ″พย″| ″พว″| ″ฟ″|
″ฟย″| ″ฟว″| ″มย″| ″มล″| ″มว″| ″ย″|
″ยว″| ″ลย″| ″ลว″| ″ว″| ″วว″| ″สย″|
″สว″| ″หฆ″| ″หฆว″| ″หง″| ″หงว″|
″หฌ″| ″หญ″| ″หฑย″| ″หฑว″| ″หฒย″|
″หฒว″| ″หด″| ″หธย″| ″หธว″| ″หน″|
″หนย″| ″หบ″| ″หม″| ″หมย″| ″หมว″|
″หย″| ″หยว″| ″หลว″| ″หว″| ″หวย″|
″หวว″| ″หอ″| ″หฮง″| ″หฮญ″| ″หฮน″|
″หฮมว″| ″หฮย″| ″หฮล″| ″หฮลย″| ″หฮว″|
″อย″| ″ฮ″| ″ฮง″| ″ฮญ″| ″ฮน″| ″ฮบ″|
″ฮม″| ″ฮมย″| ″ฮมว″| ″ฮย″| ″ฮยว″|
″ฮล″| ″ฮว″| ″ฮวว″| ″ฯฒฯ″

vowel

″″| ″..″|
″..ะ″| ″..า″|
″.ั.วะ″| ″.ื.″|
″.ุ.″| ″..ว″|
″..อ″| ″..อย″|
″..ะอ″| ″..าย″|
″..าว″| ″..าะ″|
″.ั.″| ″.ั.ว″|
″.ิ.″| ″.ิ.ว″|
″.ื.อา″| ″.ุ.ย″|
″.ู.″| ″เ..″|
″เ..ย″|″เ..ว″|
″เ..ะ″| ″เ..า″|
″เ..าะ″| ″เ.ี.ย″|
″เ.ี.ยว″| ″เ.ี.ยะ″|
″แ..″| ″แ..ะ″|
″โ..″| ″โ..ย″|
″โ..ว″| ″โ..ะ″|
″ไ..″

fcons

″″|
″ก″|
″ง″|
″ด″|
″น″|
″บ″|
″ม″

tone

″″|
″x่″|
″x้″|
″x๊″|
″x๋″|
″x์″

Figure 3.8: Tokens in a Thai script syllable input vector

icons

”” | ”ກ”| ”ຂ”| ”ຈ”| ”ດ”| ”ດສ”| ”ຕ”| ”ຕສ”|
”ຖຈ”| ”ຖສ”| ”ບ”| ”ປ”| ”ຜ”| ”ຝ”| ”ພ”| ”ມ”|
”ສ”| ”ຫ”| ”ຫລ”| ”ອ”| ”ອກ”| ”ກວ”| ”ຂວ”| ”ຄ”|
”ຄຍ”| ”ຄວ”| ”ງ”| ”ງວ”| ”ຈຍ”| ”ຈວ”| ”ຊ”|
”ຊຍ”| ”ຊຢ”| ”ຊວ”| ”ຍ”| ”ຍຍ”| ”ຍວ”| ”ດກ”|
”ດງ”| ”ດຍ”| ”ດດ”| ”ດນ”| ”ດມ”| ”ດຢ”|
”ດສຍ”| ”ດສຢ”| ”ດສວ”| ”ດອນ”| ”ດອບ”| ”ດອມ”|
”ຕກ”| ”ຕງ”| ”ຕຍ”| ”ຕວ”| ”ຕສຢ”| ”ຕສວ”|
”ຕອງ”| ”ຕອນ”| ”ຖ”| ”ຖງ”| ”ຖຈຍ”| ”ຖດ”| ”ຖວ”|
”ຖສຍ”| ”ຖສວ”| ”ຖອງ”| ”ທ”| ”ທກ”| ”ທຈ”|
”ທຈວ”| ”ທຍ”| ”ທດ”| ”ທນ”| ”ທບ”| ”ທມ”| ”ທສ”|
”ທສຍ”| ”ທສຢ”| ”ທສວ”| ”ທອງ”| ”ທອຍ”|
”ນ”| ”ນຍ”| ”ບຍ”| ”ບຢ”| ”ບວ”| ”ປຍ”| ”ປຢ”|
”ປວ”| ”ຜຍ”| ”ຜວ”| ”ຝຍ”| ”ຝວ”| ”ພຢ”| ”ພວ”|
”ຟ”| ”ຟຍ”| ”ຟຢ”| ”ຟວ”| ”ມຍ”| ”ມຢ”| ”ມລ”|
”ມວ”| ”ຢ”| ”ຢຍ”| ”ຢວ”| ”ລ”| ”ລຍ”| ”ລຢ”|
”ລວ”| ”ວ”| ”ວວ”| ”ສຍ”| ”ສຢ”| ”ສວ”| ”ຫງ”|
”ຫງວ”| ”ຫຍ”| ”ຫຍຍ”| ”ຫດ”| ”ຫນ”| ”ຫບ”|
”ຫມ”| ”ຫລຢ”| ”ຫລວ”| ”ຫວ”| ”ຫວວ”| ”ຫອບ”|
”ຫຮງ”| ”ຫຮຍ”| ”ຫຮນ”| ”ຫຮມວ”| ”ຫຮຢ”| ”ຫຮລ”|
”ຫຮວ”| ”ອກວ”| ”ອຈ”| ”ອດ”| ”ອນ”| ”ອບ”| ”ອມ”|
”ອອນ”| ”ຮ”| ”ຮງ”| ”ຮຍ”| ”ຮດ”| ”ຮນ”|
”ຮບ”| ”ຮມ”| ”ຮມວ”| ”ຮຢ”| ”ຮຢວ”| ”ຮລ”|
”ຮລຢ”| ”ຮລວ”| ”ຮວ”| ”ຮວວ”| ”ໜ”| ”ໝ”| ”ໝວ”

vowel

””| ”..”| ”..ອ”|
”..ະ”| ”..ະອ”|
”.ີ.”| ”.ຸ.”|
”..ຍ”| ”..ວ”|
”..ວຍ”| ”..ອຍ”|
”..ະຍາ”| ”..າ”|
”..າຍ”| ”..າວ”|
”..າະ”| ”..ຽ”|
”..ຽວ”| ”.ັ.”|
”.ັ.ວ”| ”.ຳ.”|
”.ິ.”| ”.ິ.ວ”|
”.ີ.ອາ”| ”.ຸ.ຍ”|
”.ູ.”| ”.ຶ.”|
”.ຶ.ວ”| ”.ຶ.ວະ”|
”.ົ.”| ”.ົ.າ”|
”ເ..”| ”ເ..ຍ”|
”ເ..ວ”| ”ເ..ະ”|
”ເ..າະ”| ”ເ.ັ.ຍ”|
”ເ.ິ.”| ”ເ.ີ.ຍ”|
”ເ.ົ.າ”| ”ແ..”|
”ແ..ະ”| ”ໂ..”|
”ໂ..ຍ”| ”ໂ..ວ”|
”ໂ..ະ”| ”ໄ..”

fcons

””|
”ກ”|
”ງ”|
”ດ”|
”ນ”|
”ບ”|
”ມ”

tone

””|
”x່”|
”x້”|
”x໊”|
”x໋”|
”x໌”

Figure 3.9: Tokens in a Lao script syllable input vector

3.5 Mapping between the scripts

As a first step in mapping between the scripts, the each input token of each script were given a corresponding value equal to its frequency rank order. The most frequent token was given the value of 1 and the least frequent token was given the largest value. The tokens were then paired up in frequency rank order with tokens from other scripts. The charts were drawn with histogram of the frequency distribution of each segment of a syllable, a standard scatterplot with a central line and a smooth scatter plot. The resulting charts are given in Figures 3.10 to 3.15.

Together, these charts provide a graphic indication of the degree of complexity for the conversion process. In the simplest cases, like the initial consonants of the Gen and Orm scripts, the most frequent tokens of the source script matched the most frequent tokens in the target script. However, in more complex cases like that between Gen and Lao vowels, there is no simple correlation between the tokens of the source and target scripts.

Table 3.15: Correlation of rank ordering in different combinations of presyllable segments

	Pre-syllable initial consonant			
GEN	1.00	0.98	0.83	0.81
0.41	**ORM**	0.98	0.83	0.81
0.32	0.73	**NRM**	0.80	0.81
0.22	0.51	0.61	**TAI**	0.93
0.04	0.40	0.54	0.60	**LAO**
Pre-syllable vowel				

Correlation coefficients were also calculated for the rank order of each pair of presyllables and syllable segments. The results are shown in Tables 3.15 and 3.16, respectively. Presyllable consonants and final consonants exhibited a high correlation between corresponding segments in the other scripts. This was also reflected by the simple linear relationship seen graphically in Figures 3.10 and 3.12. There was also a strong correlation observed between the rank orders of vowels and tone markers of the Thai and Lao scripts. The tone marking of the generic script had a high correlation with that of the New Roman script. Similarly, Thai and Lao tone marking also had a high degree of correlation. Generic script vowels also were closely correlated with those of Old Roman script. The smooth scatter plots suggest that while many elements of the other combination of segments tend to correlate, the relationship between the scripts is fairly complex. This was particularly evident in comparing the relationship between the Roman to the nonRoman initial consonants and vowels in Figures 3.12 and 3.13.

Table 3.16: Correlation of rank ordering of different combination of syllable segments

	Initial consonant			
GEN	0.83	0.40	0.42	0.35
0.92	**ORM**	0.65	0.47	0.36
0.54	0.61	**NRM**	0.39	0.20
0.51	0.54	0.28	**TAI**	0.59
0.48	0.51	0.26	0.80	**LAO**
Vowel				

	Final consonant			
GEN	0.88	0.73	0.96	0.91
0.59	**ORM**	0.81	0.89	0.84
0.95	0.55	**NRM**	0.75	0.70
0.53	0.40	0.50	**TAI**	0.91
0.53	0.43	0.49	0.80	**LAO**
Tone				

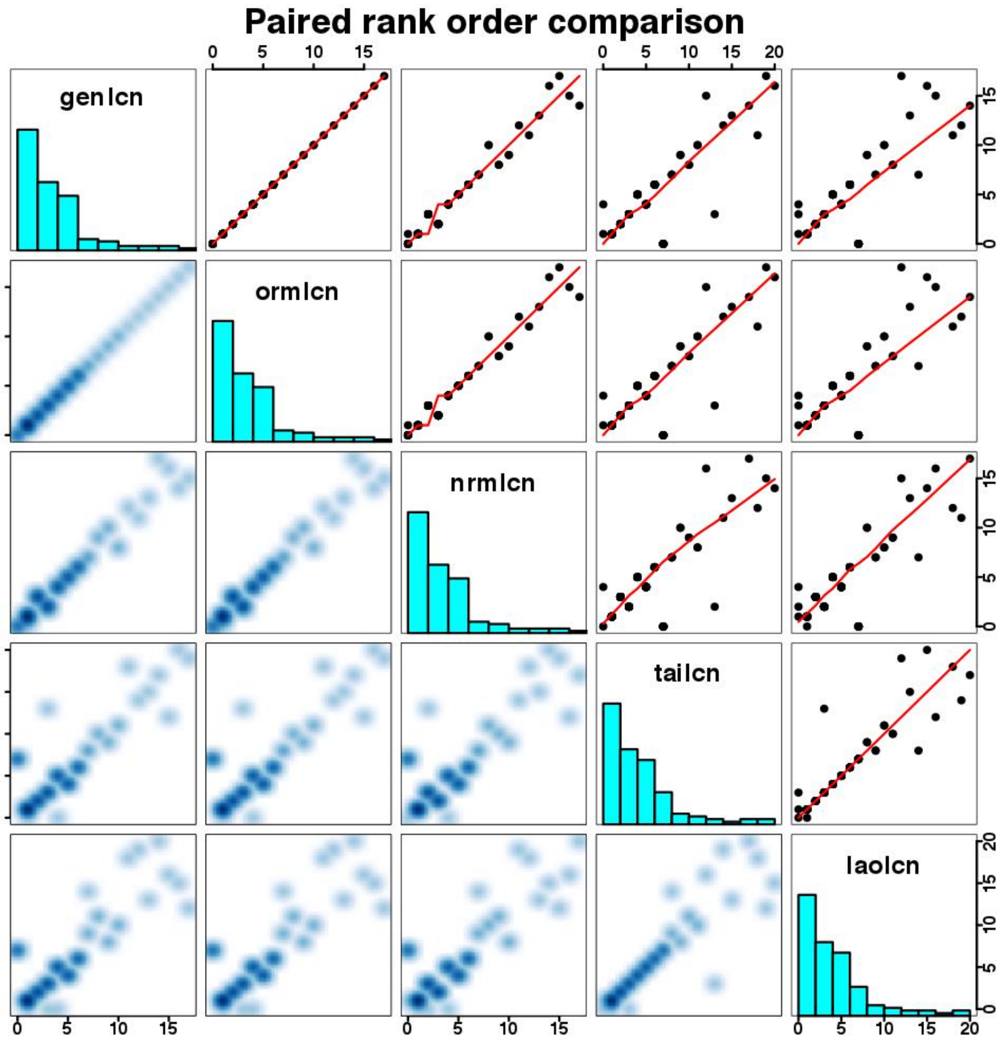


Figure 3.10: Comparison of initial consonants of pre-syllables (Pre-Icn)

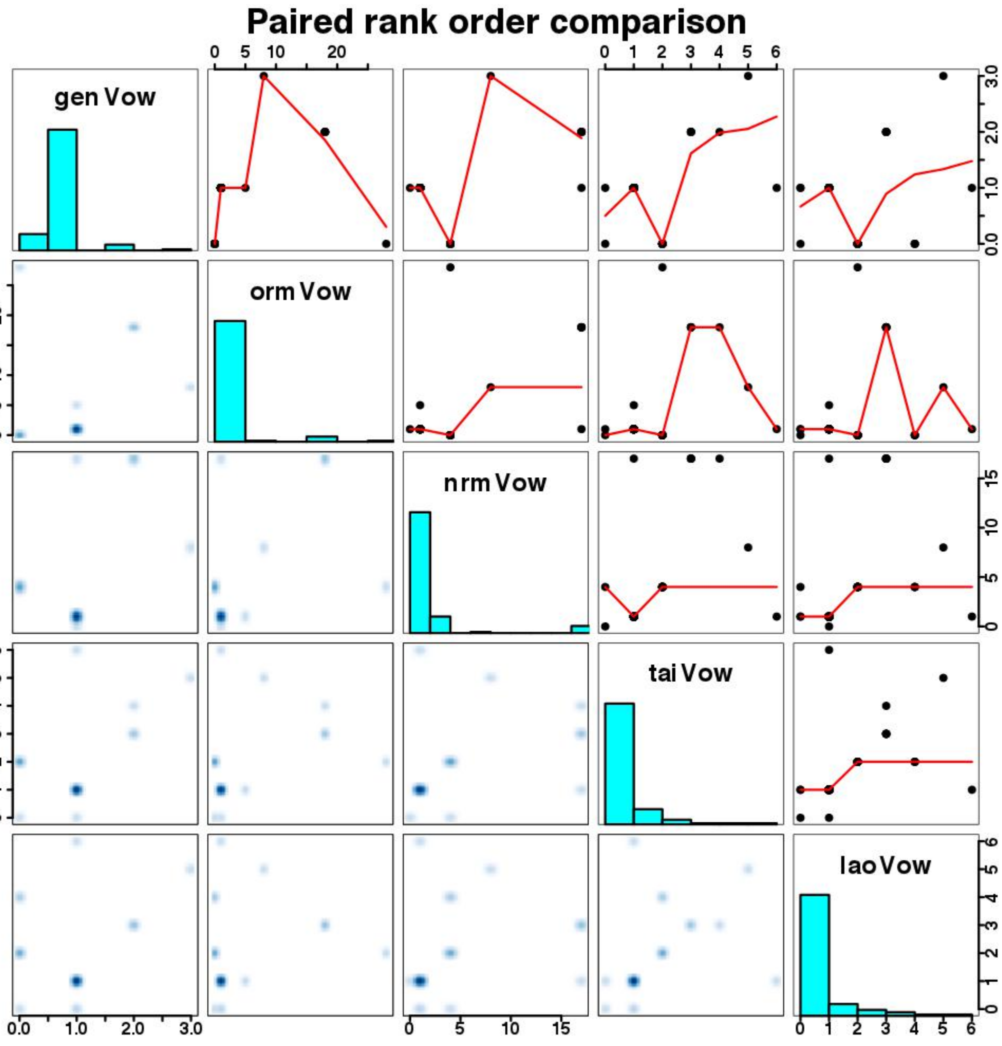


Figure 3.11: Comparison of vowels of pre-syllables (Pre-Vow)

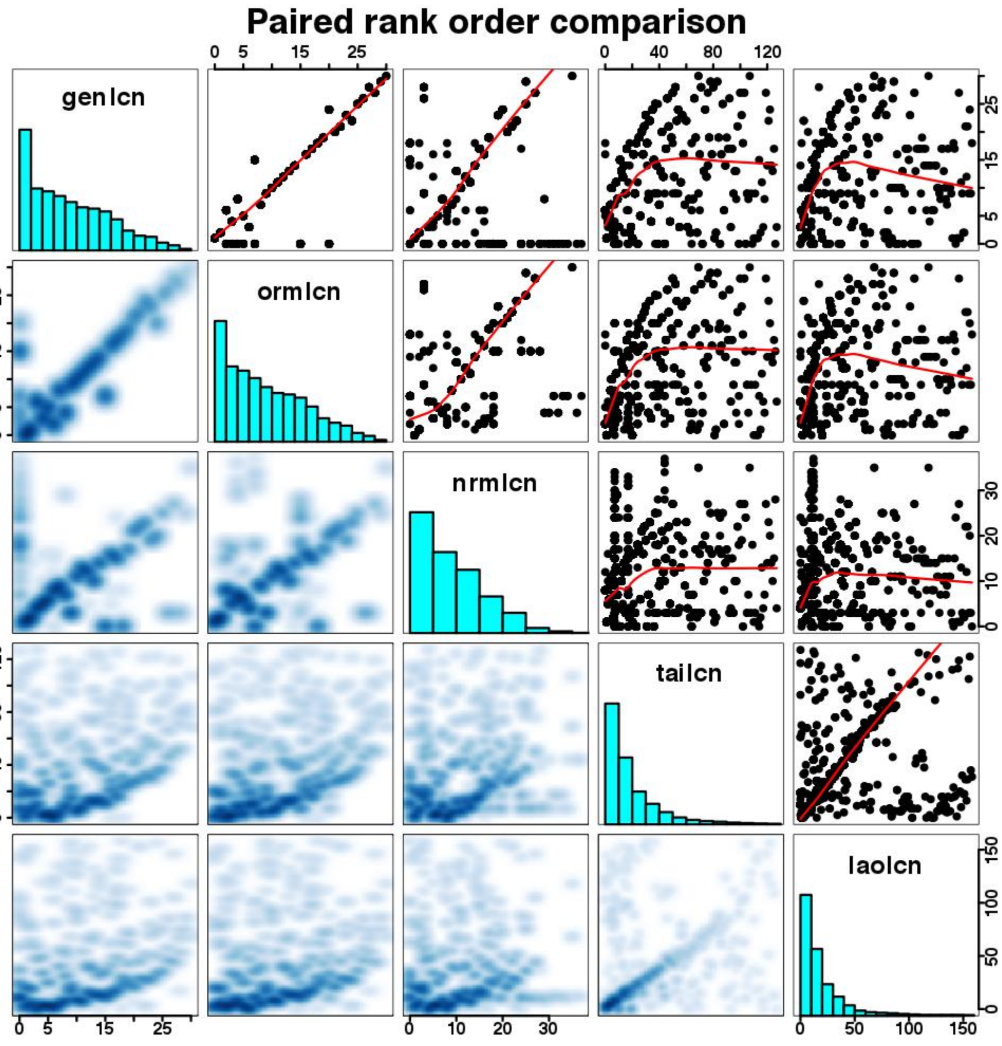


Figure 3.12: Comparison of initial consonants of syllables (Icns)

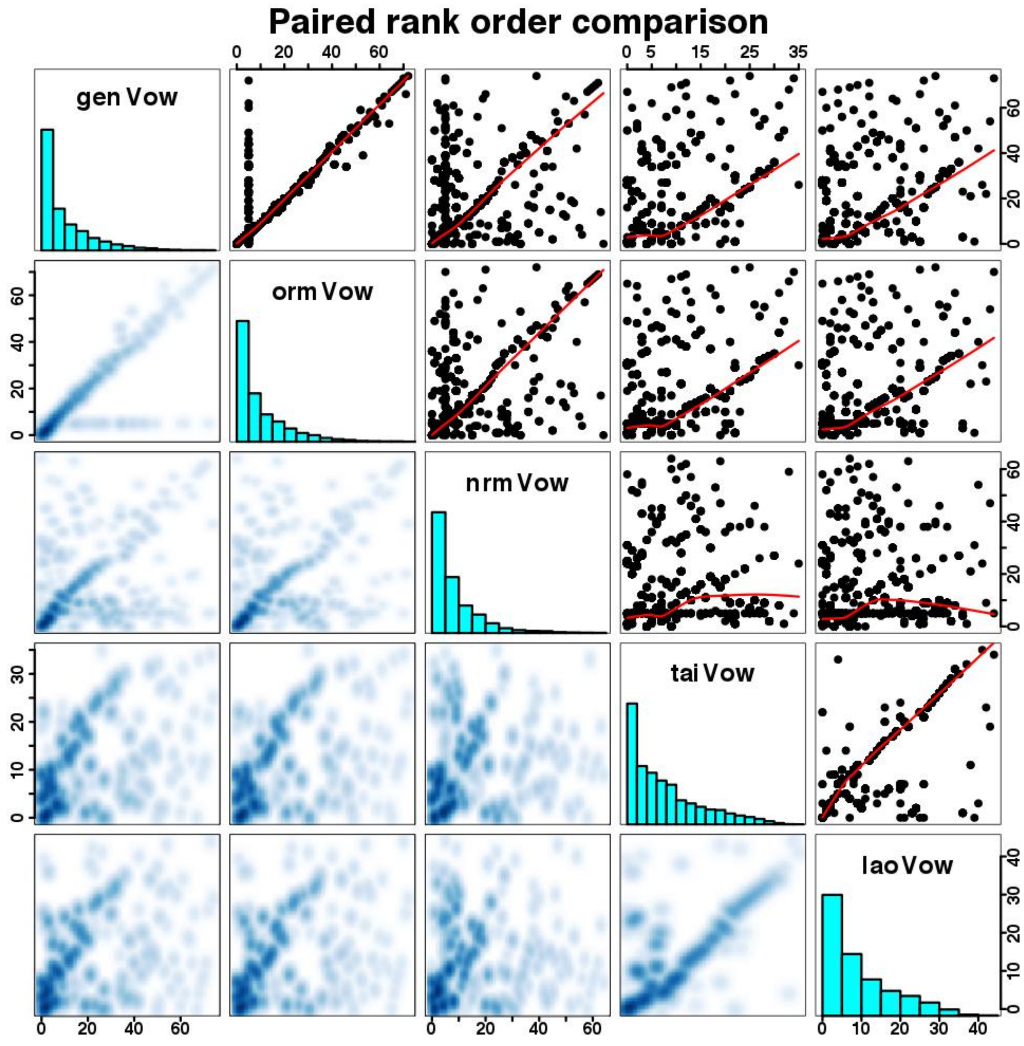


Figure 3.13: Comparison of vowels of syllables (Vow)

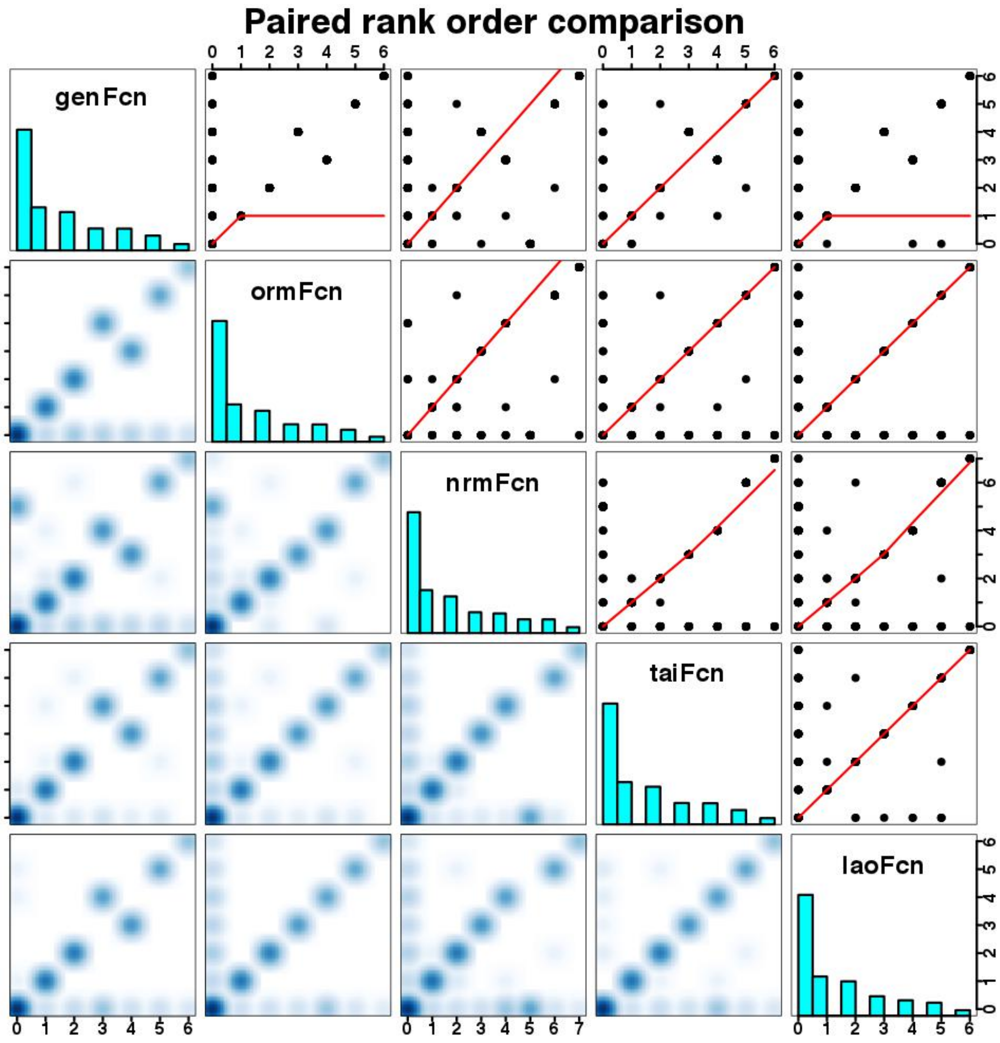


Figure 3.14: Comparison of final consonants of syllables (Fcns

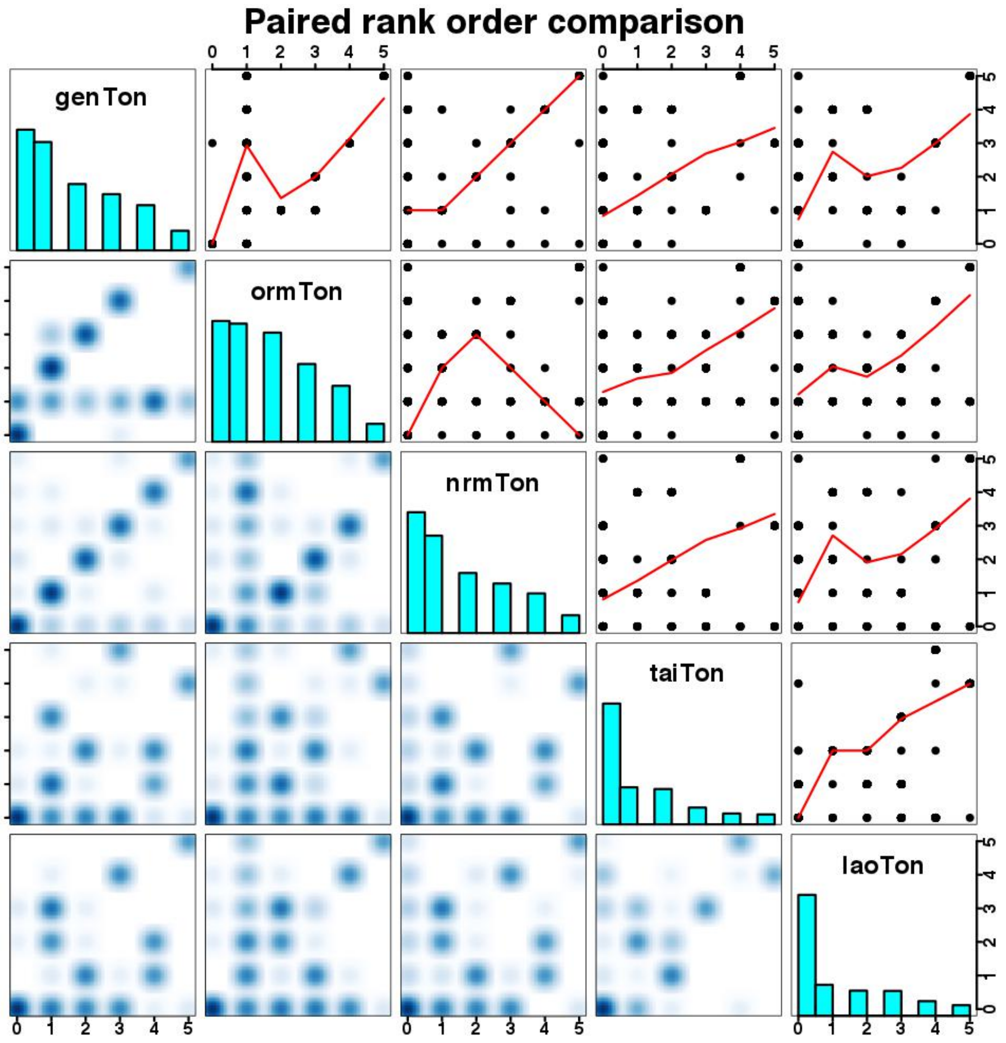


Figure 3.15: Comparison of syllable tone markers (Ton)

3.5.1 IC3 Learning

The effectiveness of IC3 learning of transcription was tested in an experiment where increasing portions of the parallel phonemes list were subjected to ID3 decision tree learning. The results of 5 separate runs were averaged together and are shown in Figures 3.16 to 3.20. Each point on these graphs represent an average of the outcome for 3 test runs. Standard deviation of these values was typically ± 0.05.

Without exception, the transcription of the final consonants was the most accurate of all phonemes. Likewise, the predicted transcription of the tone mark and the initial consonant were the most suspect.

Generally, the accuracy of the ID3 based transcription increased as fraction of the text sample increased. However in the case of converting initial consonants of the generic script into Thai appeared to diminish with the increased portion of the syllabus list. As expected, transcription between two Roman scripts or two non-Roman script produced more accurate results than attempts to transcribe between a Roman script and a non-Roman script.

3.5.2 Neural networks with back propagation

A number of preliminary experiments were run to determine the learning rate of the neural networks. The neural network routines of the AI3R package provided the sum of the total propagated error for each iteration as an indicator of progress in the learning process.

Initial studies were conducted with 1 hidden layer of the same size as the input vector. After 300 iterations, the residual propagated error was compared to the accuracy of the predicted outcomes. As shown in Figure 3.21, the training set exhibited a clear correlation between the size of the propogated error and the accuracy of the

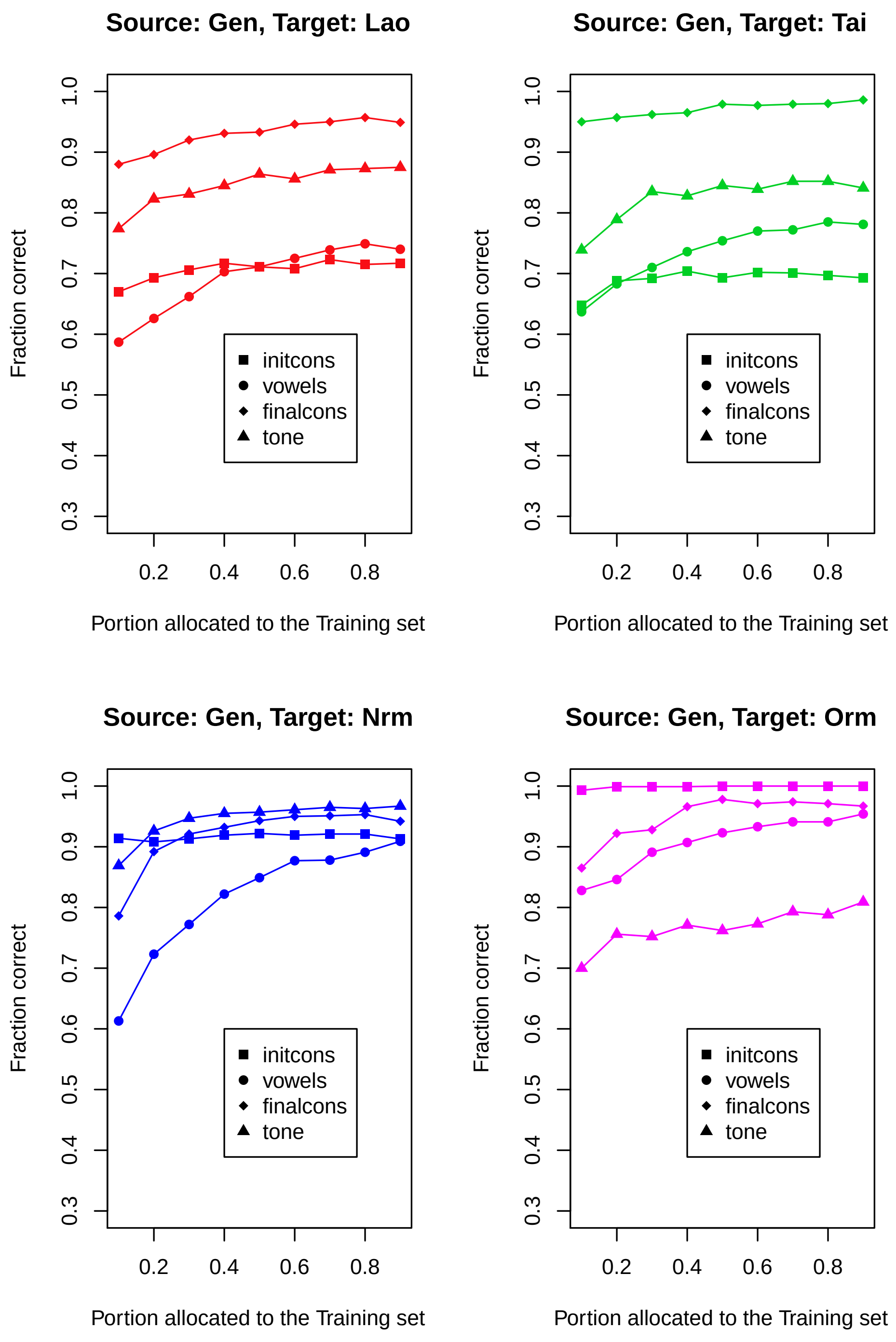


Figure 3.16: ID3 machine learning of transcription from the Generic script

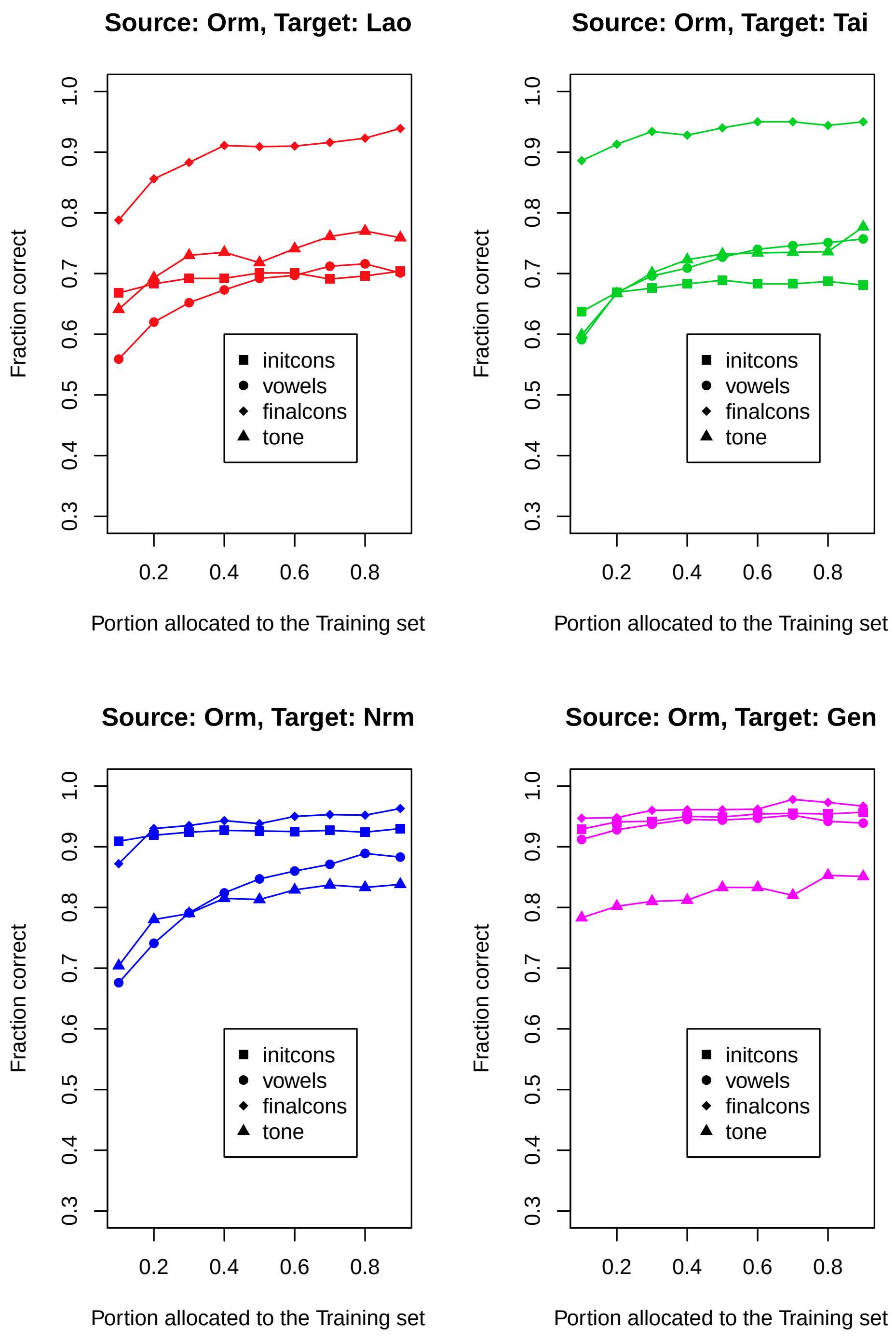


Figure 3.17: ID3 machine learning of transcription from the Old Roman script

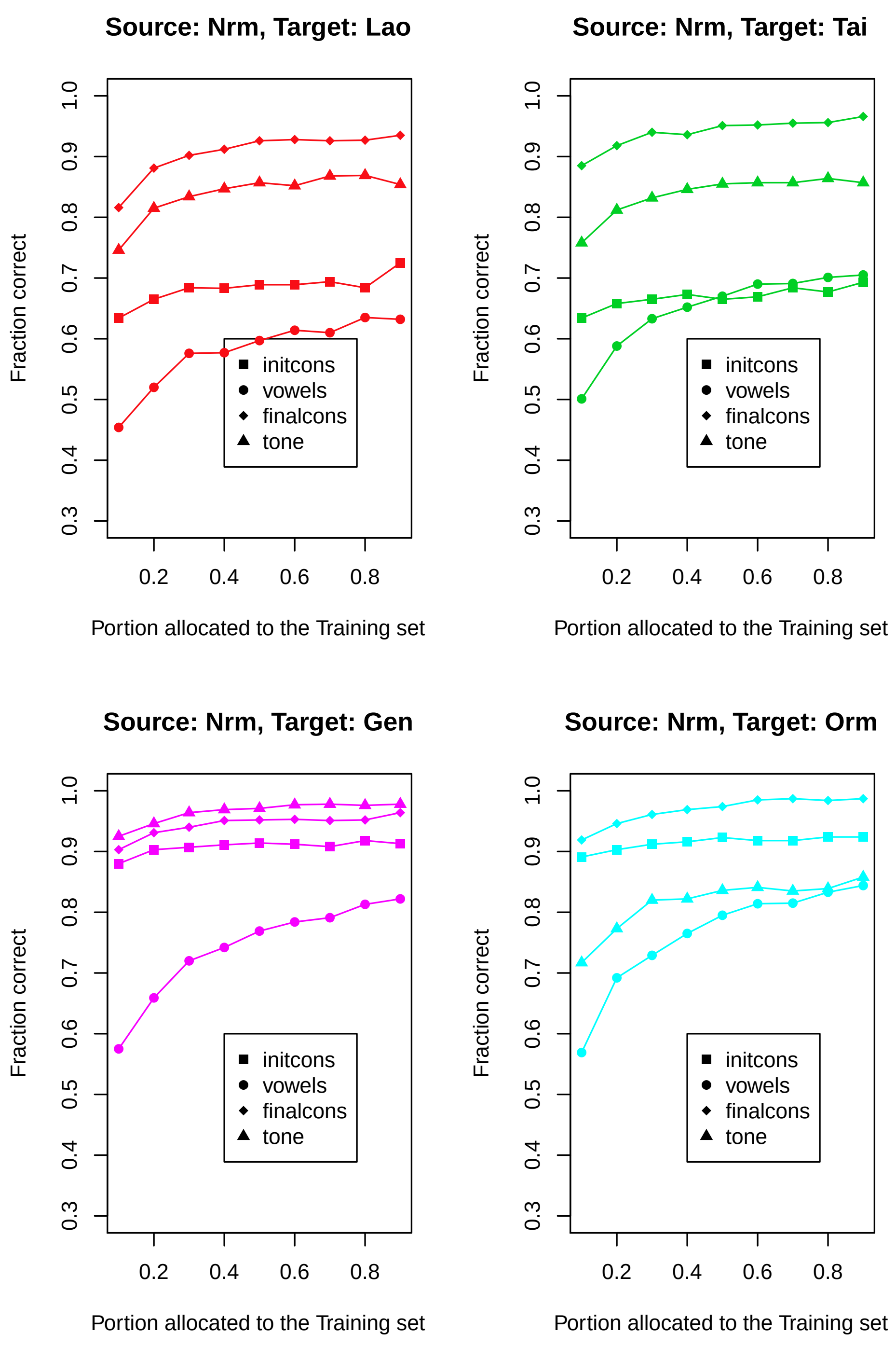


Figure 3.18: ID3 machine learning of transcription from the New Roman script

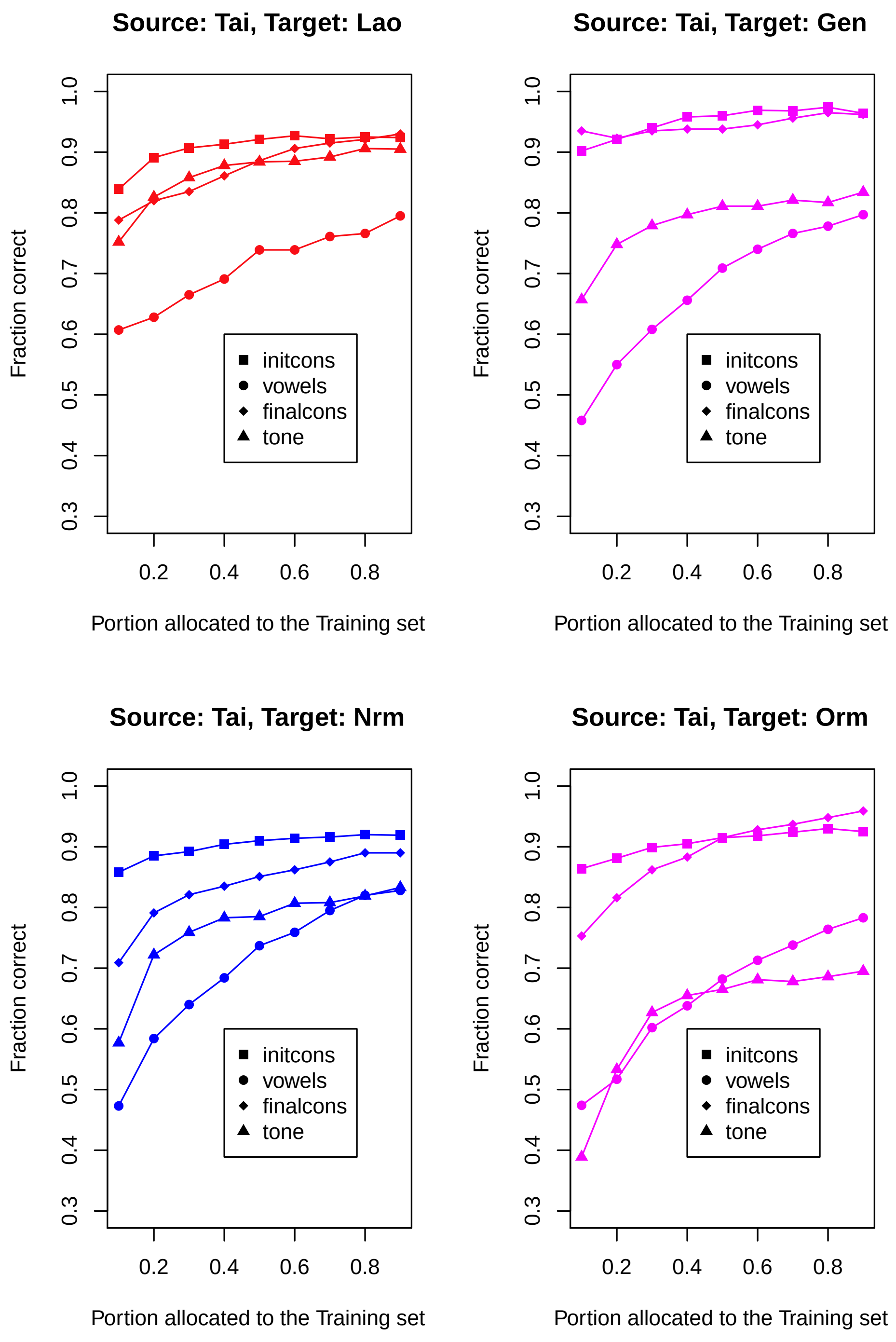


Figure 3.19: ID3 machine learning of transcription from the Thai script

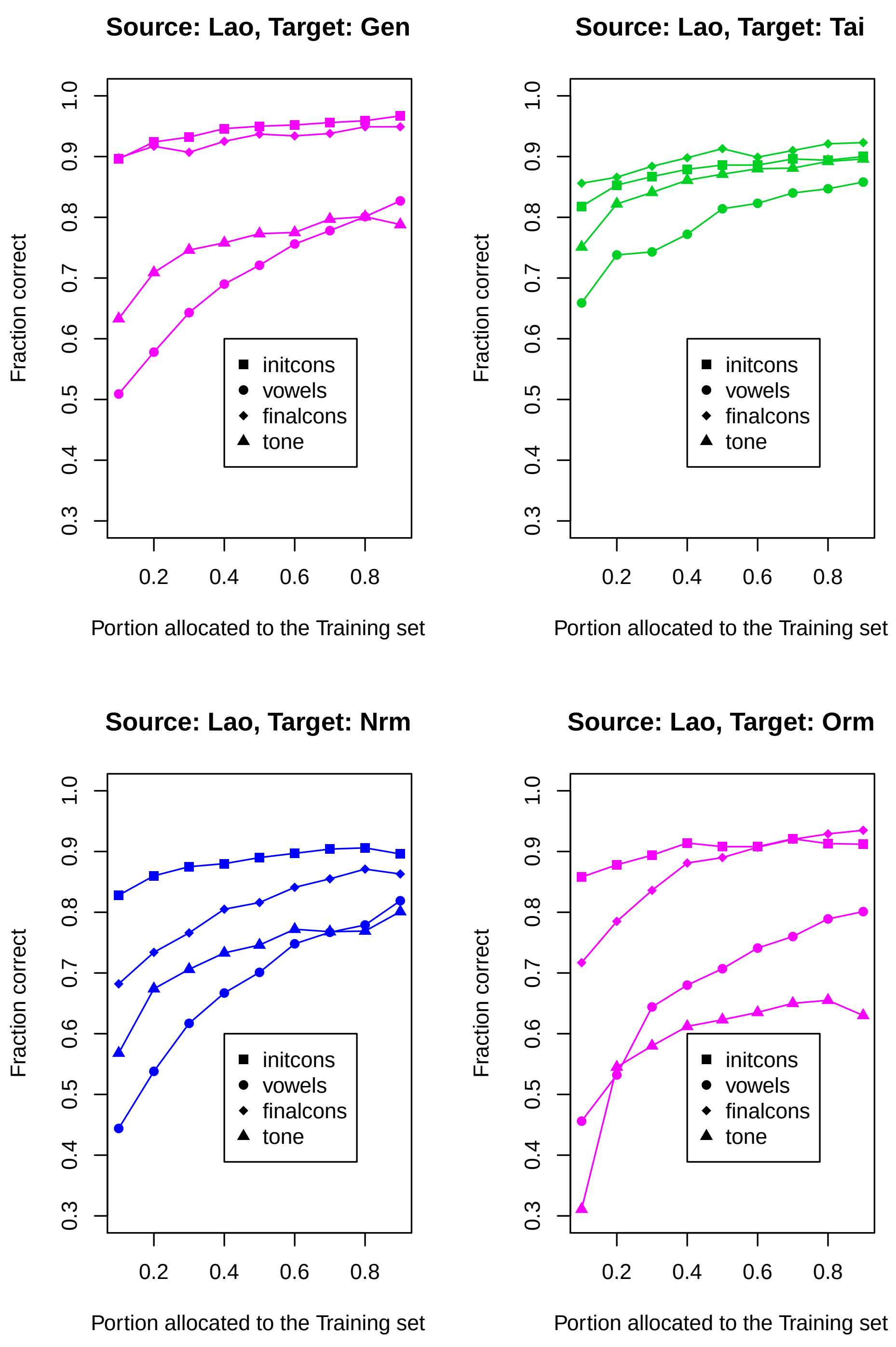


Figure 3.20: ID3 machine learning of transcription from the Lao script

transcription. It was interesting to note that when Lao and Gen were used as the source script, the outcomes were nearly correct despite propagated error.

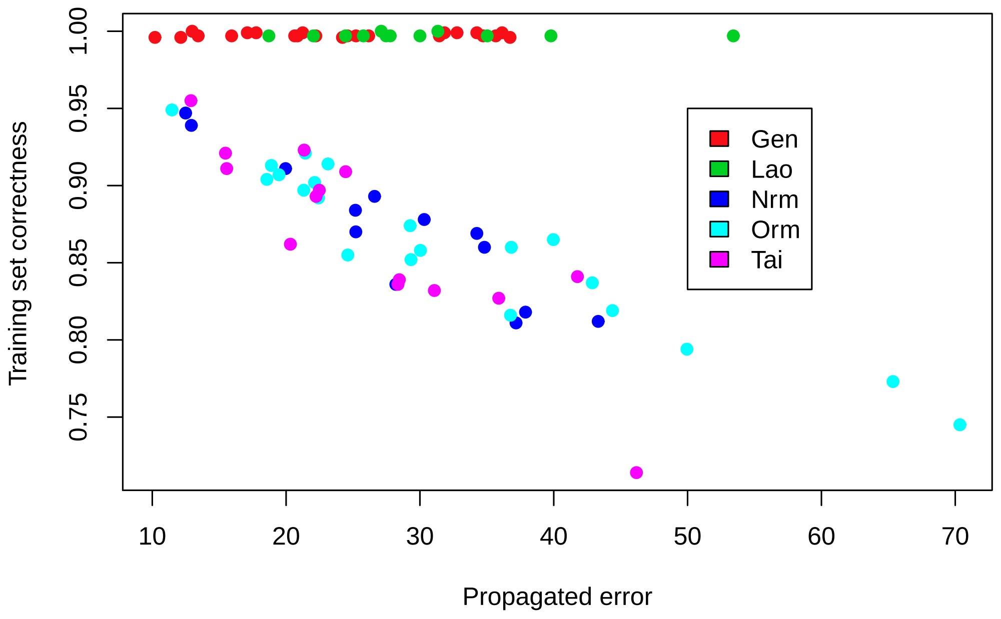


Figure 3.21: Training set vs propagated errors

However, the residual propagated error appeared to be nearly independant of the errors in transcription observed with the test set. The scatterplot comparing residual propagated error to errors of transcription of the test set is shown in Figure 3.22. These results demonstrate the need to test outcomes directly.

Some initial results for learning Nrm and Tai scripts are given in Figures 3.23 and 3.24, respectively. These figures show the total error back proprogated on each iteration. These initial studies were performed with one hidden layer that was same size as the input vector. These learning curves suggest that vowels and final consonants

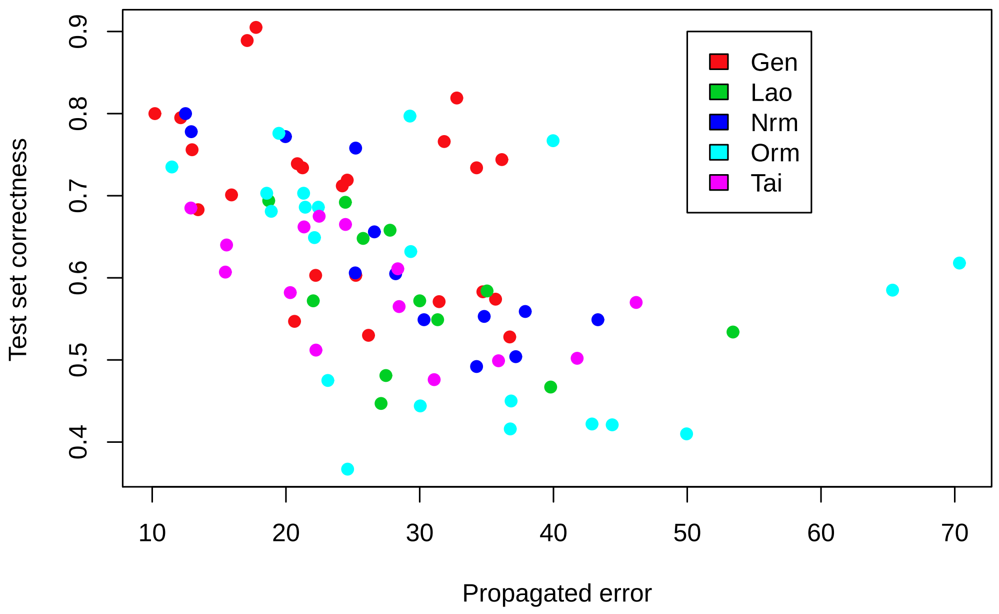


Figure 3.22: Test vs propagated errors

were rapidly learned, while tone marks and initial consonants took longer to learn. These results were consistent with the correlations seen in Section 3.5. It was found that 300 iterations achieved statistically the same level of precision as was seen with after 500, 1000, and 5000 iterations.

In another experiment, neural networks were trained with between 0 and 3 hidden layers for each source-target script pair. The hidden layers had the same number of nodes as the input vector of the source script. (The size of the input vector for each script is given in Table 3.14. Each trial attempted to develop 6 separate networks that would transcribe the input script tokens into one of the phonemes of the target script. A set of neural networks was created for each combination of source and

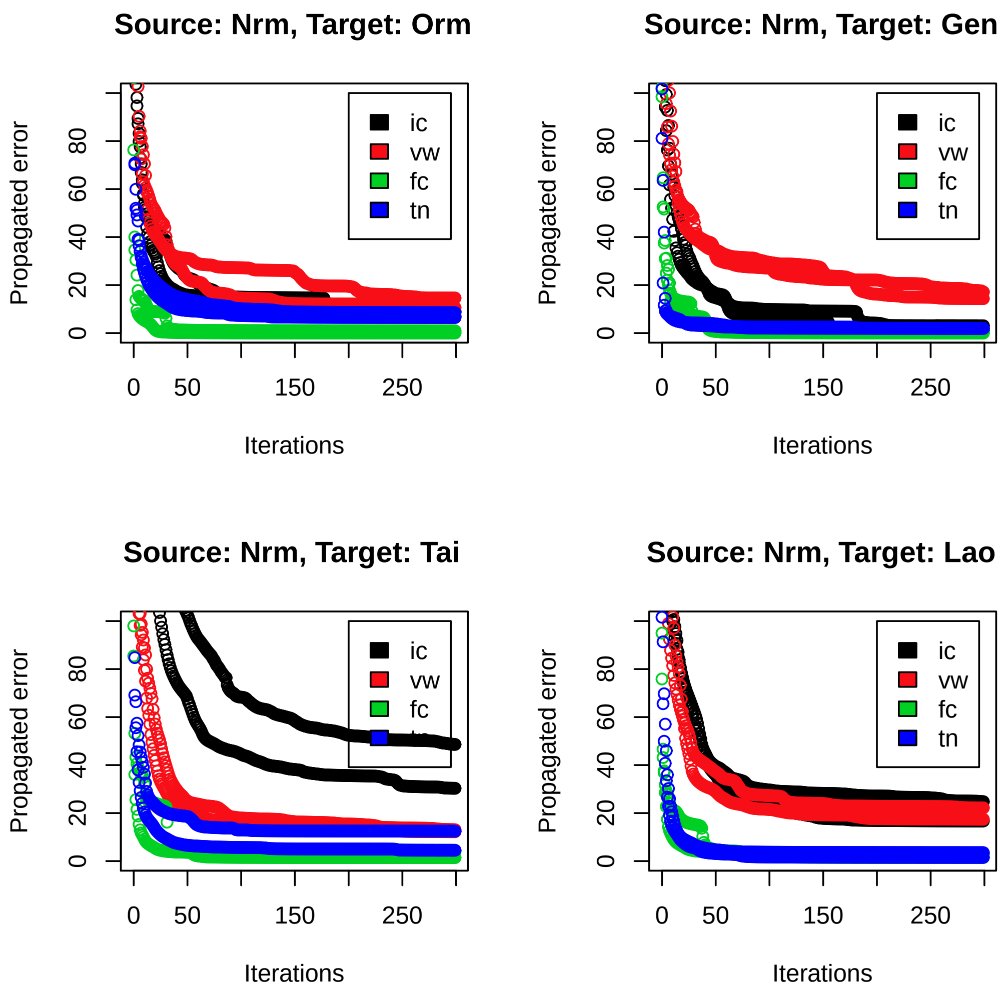


Figure 3.23: Phoneme learning of Nrm syllables

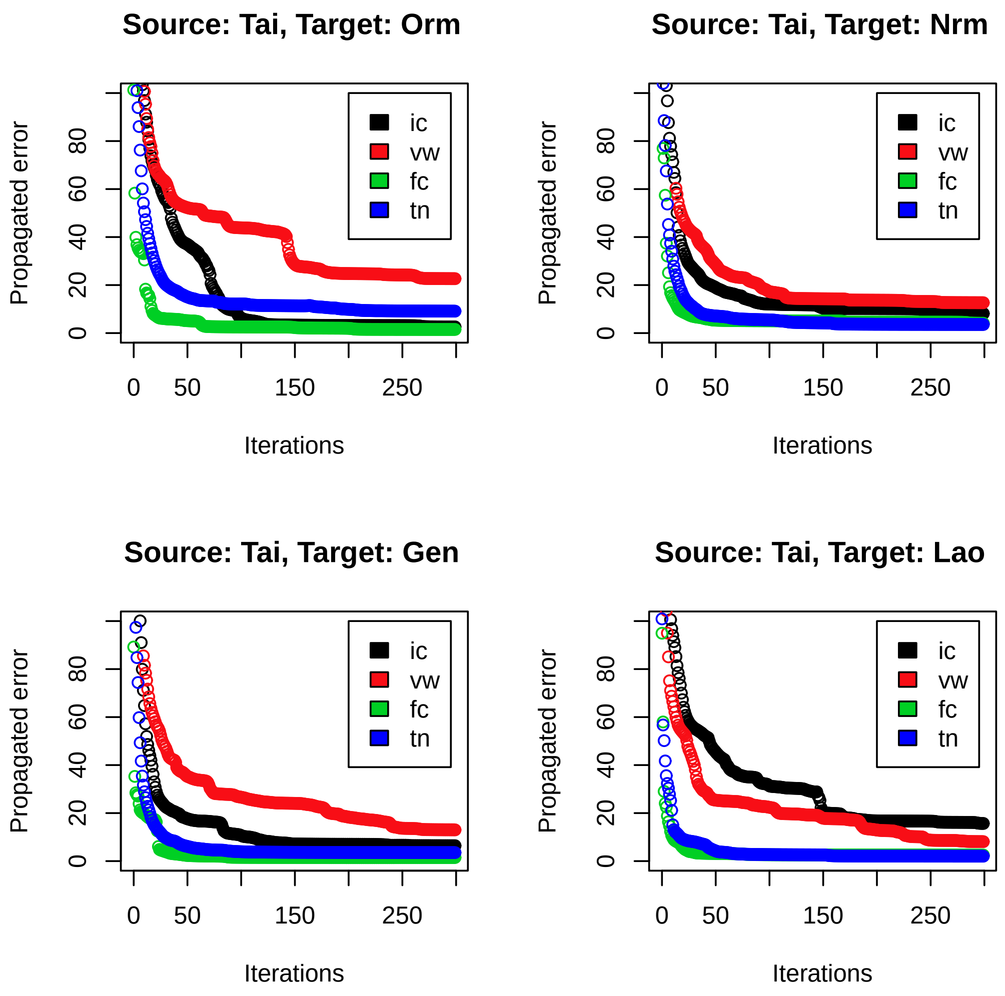


Figure 3.24: Phoneme learning of Thai syllables

target script. Each trial was repeated 3 times. In addition, the syllable catalogue was distributed randomly between training and test sets. Three levels of partitioning were tested, namely, 0.1, 0.5 and 0.9. In short, 1,080 networks were generated and tested in this experiment. Despite the fact that only 300 iterations were used for each learned network, it took 14 days for 10 dual-core iMacs running at 2.13 GHz to complete this processing.

After 300 iterations, the accuracy of the network to transcribe the training and test sets were checked not only for each phoneme but also for the composition word as well. Tables 3.20 and 3.17 show the results for the training set and the test set, respectively. Each value represents the average of 3 trial runs. Standard deviations of 0.02 were typical.

As expected, the predicted outcomes of the training set were more accurate than that of the test set. In fact It was also important to recognize that the error of the composite words were higher than for any single phoneme. It was also interesting to note that adding hidden layers did not improve the accuracy of the predicted phoneme.

To better understand the interaction between the various factors leading to a transcription. Various combinations of fractors were tested to develop a generalized linear model (GLM) of the outcome of the neural network training. The factors tested included the source and target scripts, the fraction of the syllables used in the training set, the number of hidden layers. Insignficant factors were removed from the model on the basis of analysis of variance (ANOVA). The resulting linear model is shown in Equation 3.5. The calculated coefficients and residuals of this GLM are shown in Table 3.21 and the analysis of variance (ANOVA) in Table 3.22.

$$Correctness_{Testset} = Src + Target + Frag \tag{3.5}$$

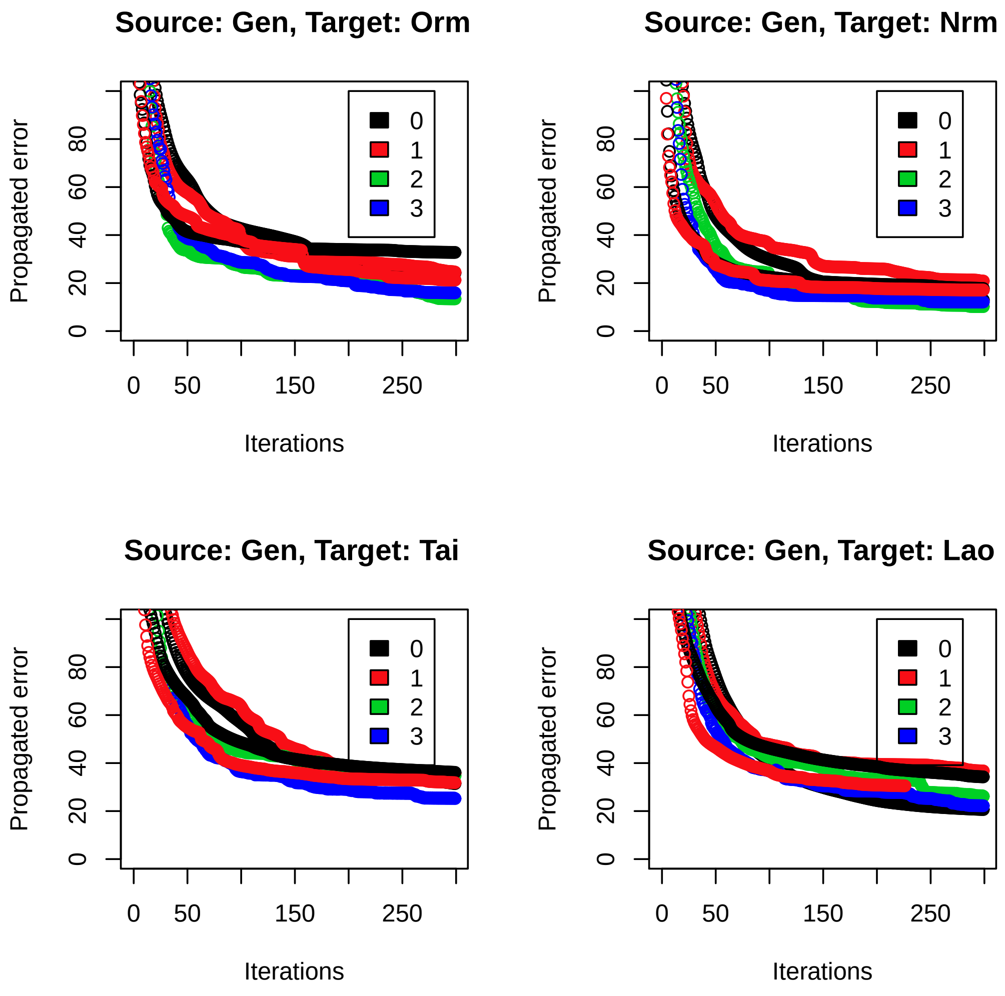


Figure 3.25: Hidden layer of Gen syllables

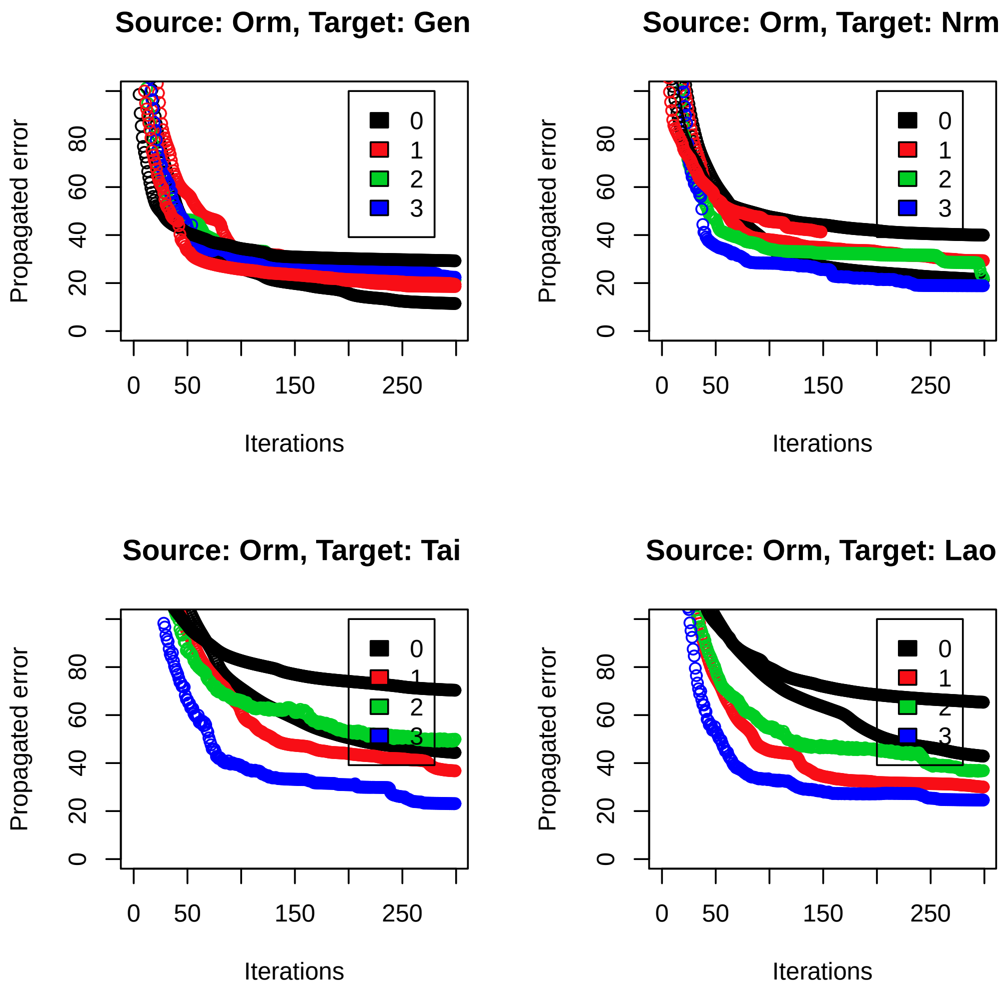


Figure 3.26: Hidden layer of Orm syllables

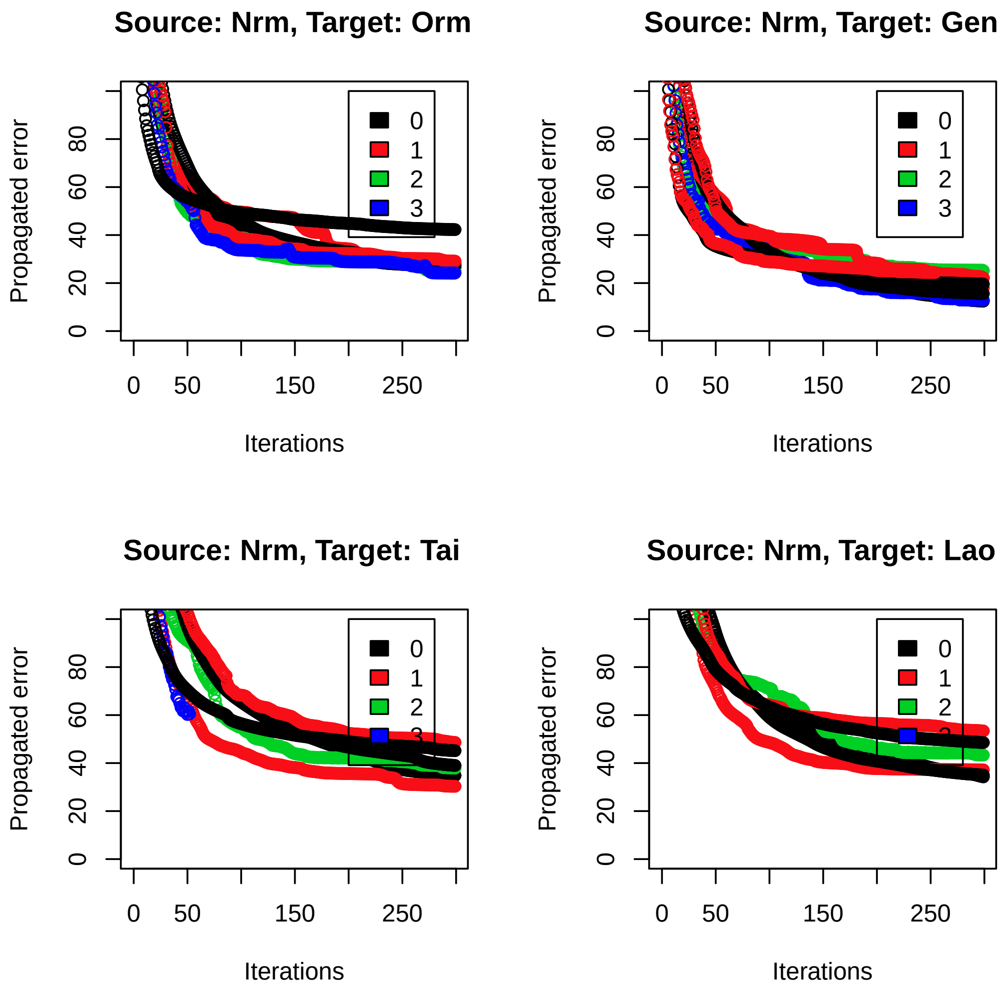


Figure 3.27: Hidden layer of Nrm syllables

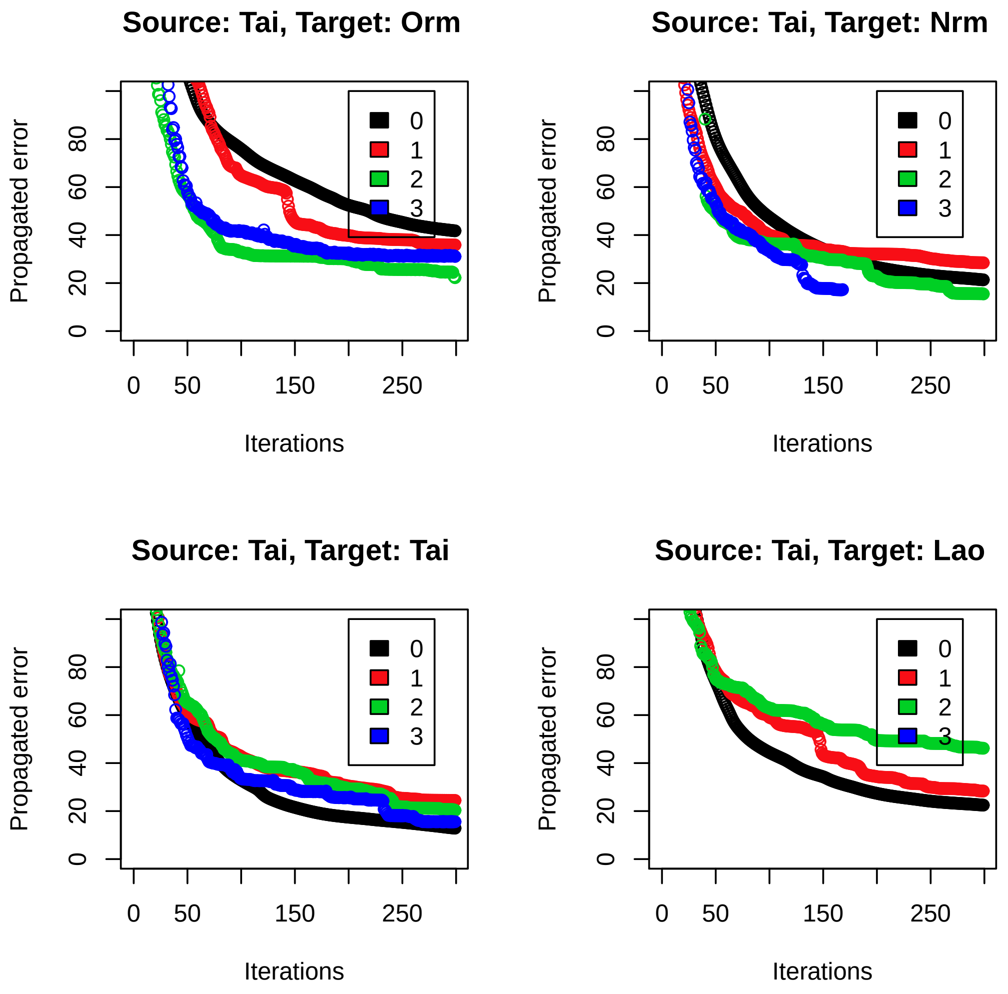


Figure 3.28: Hidden layer of Thai syllables

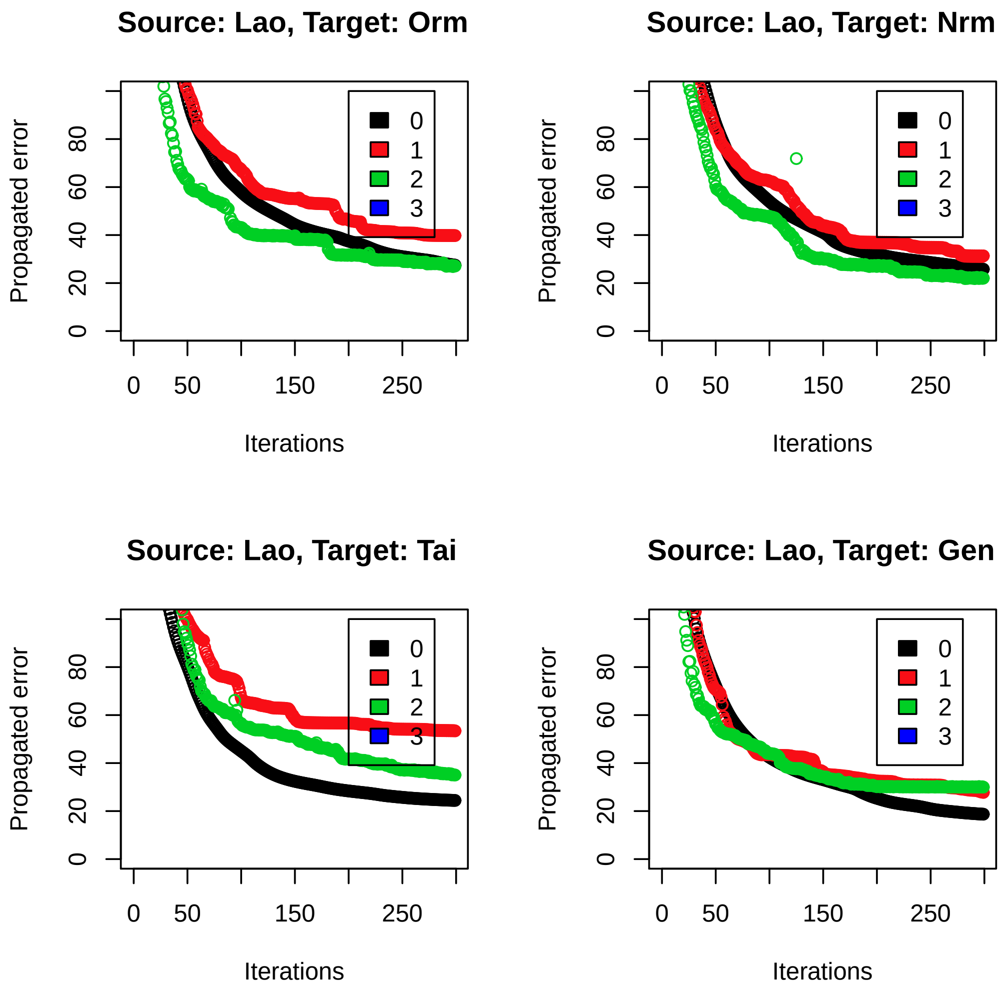


Figure 3.29: Hidden layer of Lao syllables

Table 3.17: Accuracy of transcripting the training set from Gen

Figures indicate the fraction of correctly predicted outcomes. Each number represents an average of 3 trial.

Src	**Target**	**Hidden**	icns	vow	fcns	ton	syl
Gen	Tai	0	0.90	0.95	1.00	0.99	0.84
Gen	Tai	1	0.85	0.95	1.00	0.99	0.80
Gen	Tai	2	0.82	0.96	1.00	1.00	0.79
Gen	Tai	3	0.94	0.93	1.00	0.99	0.87
Gen	Lao	0	0.95	0.96	1.00	0.99	0.90
Gen	Lao	1	0.90	0.91	1.00	0.99	0.82
Gen	Lao	2	0.92	0.97	1.00	0.99	0.89
Gen	Lao	3	0.92	0.97	1.00	0.99	0.89
Gen	Orm	0	1.00	0.95	0.98	0.95	0.89
Gen	Orm	1	1.00	0.88	0.99	0.98	0.86
Gen	Orm	2	0.99	0.93	1.00	0.99	0.92
Gen	Orm	3	0.98	0.96	1.00	0.97	0.91
Gen	Nrm	0	0.98	0.96	1.00	1.00	0.89
Gen	Nrm	1	0.94	0.94	0.99	1.00	0.92
Gen	Nrm	2	0.99	0.95	1.00	1.00	0.94
Gen	Nrm	3	0.96	0.96	1.00	1.00	0.93

Table 3.18: Correctness of predicted outcomes using the test sets as input

Numbers represent the average fraction of correct renderings based on 3 separate runs

Source	Target script				
script	Gen	Orm	Nrm	Tai	Lao
Gen	–	0.705	0.763	0.586	0.560
Orm	0.714	–	0.656	0.416	0.445
Nrm	0.783	0.622	–	0.554	0.515
Tai	0.611	0.503	0.611	–	0.619
Lao	0.641	0.465	0.590	0.603	–

Table 3.19: Correctness of predicted outcomes using the training sets as input
Numbers represent the average fraction of correct renderings based on 3 separate runs

Source	Target script				
script	Gen	Orm	Nrm	Tai	Lao
Gen	–	0.998	0.997	0.997	0.997
Orm	0.918	–	0.896	0.808	0.858
Nrm	0.909	0.871	–	0.872	0.831
Tai	0.921	0.852	0.894	–	0.816
Lao	0.997	0.997	0.998	0.997	–

Table 3.20: Accuracy of transcripting the test set from Gen

Figures indicate the fraction of correctly predicted outcomes. Each number represents an average of 3 trials.

Src	**Target**	**Hidden**	**icns**	**vow**	**fcns**	**ton**	**syl**
Gen	Tai	0	0.69	0.75	0.99	0.85	0.57
Gen	Tai	1	0.73	0.79	0.99	0.88	0.56
Gen	Tai	2	0.77	0.82	0.99	0.85	0.52
Gen	Tai	3	0.77	0.80	0.99	0.85	0.48
Gen	Lao	0	0.73	0.75	0.87	0.87	0.51
Gen	Lao	1	0.75	0.79	0.98	0.90	0.56
Gen	Lao	2	0.76	0.79	0.97	0.92	0.58
Gen	Lao	3	0.76	0.75	0.90	0.88	0.58
Gen	Orm	0	0.98	0.92	1.00	0.79	0.71
Gen	Orm	1	0.97	0.77	0.97	0.81	0.58
Gen	Orm	2	0.96	0.90	1.00	0.80	0.69
Gen	Orm	3	0.97	0.92	1.00	0.83	0.74
Gen	Nrm	0	0.91	0.85	0.96	0.98	0.79
Gen	Nrm	1	0.88	0.85	0.96	0.97	0.75
Gen	Nrm	2	0.94	0.87	0.95	0.97	0.82
Gen	Nrm	3	0.91	0.87	0.96	0.97	0.78

Table 3.21: Residuals and coefficients of the GLM

```
Residuals:
          Min          1Q      Median       3Q       Max
-0.119999       -0.039864   -0.001839  0.032364  0.166366

Coefficients:
                 Estimate  Std. Error   t value  Pr(>|t|)
(Intercept)     0.7764403   0.0240220    32.322   < 2e-16  ***
SrcLao         -0.0976398   0.0252682    -3.864  0.000242  ***
SrcNrm         -0.0225185   0.0233400    -0.965  0.337873
SrcOrm         -0.1092955   0.0213578    -5.117  2.48e-06  ***
SrcTai         -0.0883889   0.0233327    -3.788  0.000312  ***
TargetLao      -0.1622669   0.0264517    -6.134  4.17e-08  ***
TargetNrm      -0.0290091   0.0210838    -1.376  0.173120
TargetOrm      -0.1264250   0.0236027    -5.356  9.71e-07  ***
TargetTai      -0.1356896   0.0276680    -4.904  5.63e-06  ***
Frag            0.4027506   0.0550500     7.316  2.88e-10  ***
PropErr        -0.0023630   0.0009049    -2.611  0.010965  *
---
      Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.

Residual standard error: 0.0577 on 72 degrees of freedom
Multiple R-squared: 0.7949, Adjusted R-squared: 0.7664
F-statistic: 27.9 on 10 and 72 DF, p-value: < 2.2e-16
```

The ANOVA analysis was consistent with the observation that correct transcriptions involving Thai, Lao or Old Roman script were harder to achieve. At the same time, increasing the size of the training set relative to the full number of possibilities significantly helped to improve accuracy. It also showed that the propagated error of a trained system had less influence on the accuracy of outcome than the other factors.

Comparison of word level performance is non-trival especially when attempting to correct for the word frequency distribution and the differences between the phonetics

Table 3.22: ANOVA of GLM

```
            Df  Sum Sq  Mean Sq F value    Pr(>F)    
Src          4 0.19141 0.047853 14.3717 1.132e-08 ***
Target       4 0.55727 0.139319 41.8416 < 2.2e-16 ***
Frag         1 0.15751 0.157506 47.3038 1.856e-09 ***
PropErr      1 0.02271 0.022707  6.8197   0.01096 *  
Residuals   72 0.23974 0.003330                      
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’
```

of Iu Mien words and Biblical proper names. Nevertheless, simple experiment was attempted to gain some insight as to whether direct transcription was better than a 2-step transcription via a generic script. In this experiment 50 words were chosen at random from the full word list. The words were subjected to the neural net transcription rules for each of the phonemes. The neural net transcription rules used were derived by using a training set of 10% of the syllable list without hidden layers. Accuracy of the computer-derived transcripts is shown in Table 3.23.

Table 3.23: Transcription of 50 random words via neural networks
Numbers indicate the number of words correctly rendered into the target script

Source	Target script				
script	Gen	Orm	Nrm	Tai	Lao
Gen	-	40	44	32	26
Orm	43	-	39	29	25
Nrm	46	41	-	31	32
Tai	18	22	23	-	37
Lao	23	25	27	45	-

The results of the transcription of the 50 random words were consistent with those obtained in the transcription of syllables. The results suggest that difficulties in transcription are symmetrical. For example, the lower accuracy of Lao and Thai transcriptions of words in Orm script was also seen when Lao and Thai words were transcribed to Orm.

The second part of the experiment was to take output of transcription into the generic script and transcribe it into the other scripts. The results of this experiment is given in Table 3.24.

Table 3.24: Secondary transcriptions after transcribing 50 random words to the Generic Script
Numbers indicate the number of words correctly rendered into the target script

Source	Target source			
script	Orm	Nrm	Tai	Lao
Orm	35	37	27	21
Nrm	39	41	31	25
Tai	17	14	11	12
Lao	19	22	17	15

This experiment would suggest that the second transcription introduces additional errors into the transcription process. Thus, it would appear that a two-step transcription imposes additional expense in both processing time and errors of transcription. However, more experiments would be required to identify the extent of this cost.

3.6 Developing the web application

Once the databases for the the web application were established, Ruby on Rails was invoked to create the framework for the web application that was described in the Section 1.8. This automated step produced the components described in Table 3.25. The learned neural networks were stored as YAML formatted objects that can be easily loaded on demand. The links were added to the controller for the Rails application for both the text decomposition class definitions and the neural network loader. In this way, only the essential neural networks are loaded.

Table 3.25: Products of Rails application setup

Resources created	**Number of files**
Configuration files	5
Directories	31
How-To Documentation files	3
Javascript files	6
Log files	4
Session scripts	15
Web pages	6

Ruby on Rails was implemented as a framework that depends on a number of of Ruby modules. Throughout the duration of this project, there were numerous updates to the modules. As new features were added to the website, new modules and updates would need to be installed. The list of modules currently used by the website is given in Table 3.26. The Rails utility Bundler was used to manage the dependencies of these 35 modules.

While the transcription results were promising, online users did not feel that output was reliable enough across all scripts to be practical at this stage. Nevertheless, the web environment was used as a forum to gain feedback. However, the original Rails application proved to be to be too elaborate for this stage of development. The site attracted insufficient volunteers to provide useful feedback. It would appear that the formality of signing up for a website of limited use greatly reduced the participation.

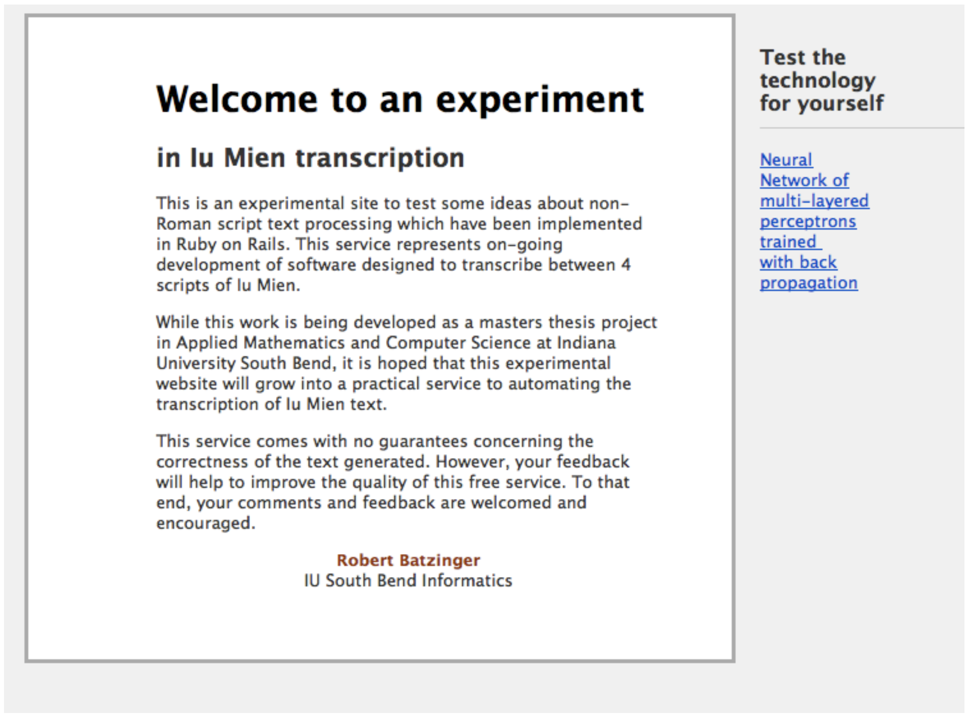
After 12 months with disappointing lack of growth in the use of the site, a simpler

Table 3.26: Modules used in the online application

Module	Version	Module	Version	Module	Version
abstract	1.0.0	diff-lcs	1.1.3	rake	0.9.2
actionmailer	3.0.9	erubis	2.6.6	rails	3.0.9
actionpack	3.0.9	gherkin	2.5.1	railties	3.0.9
activemodel	3.0.9	i18n	0.5.0	rdoc	3.9.4
activerecord	3.0.9	json	1.6.1	sqlite3	1.3.4
activeresource	3.0.9	mail	2.2.19	sqlite3-ruby	1.3.3
activesupport	3.0.9	mime-types	1.16	term-ansicolor	1.0.6
ai4r	1.9	nokogiri	1.5.0	thor	0.14.6
arel	2.0.10	polyglot	0.3.2	treetop	1.4.10
builder	2.1.2	rack	1.2.4	tzinfo	0.3.30
bundler	1.0.15	rack-mount	0.6.14	webrat	0.7.3
cucumber	1.1.0	rack-test	0.5.7		

web application was developed. Screen shots of the new website have been included in the following figures. The home page provides a basic introduction to the site and related disclaimers. (Figure 3.30). After clicking on the link for the transcription service, the users would recieve a form to submit Iu Mien text in the script of their choosing. (Figure 3.31) The system would then develop a multi-script response in HTML which is then sent back to the user. (Figure 3.32).

This new change in design appeared to be a step in the right direction as in the past month, nearly 260 transcriptions were attempted. In the 2 previous years, less than 10 transcriptions were attempted. More work is required to improve both the quality of the transcriptions as well as the use of the online service.



Welcome to an experiment

in Iu Mien transcription

This is an experimental site to test some ideas about non-Roman script text processing which have been implemented in Ruby on Rails. This service represents on-going development of software designed to transcribe between 4 scripts of Iu Mien.

While this work is being developed as a masters thesis project in Applied Mathematics and Computer Science at Indiana University South Bend, it is hoped that this experimental website will grow into a practical service to automating the transcription of Iu Mien text.

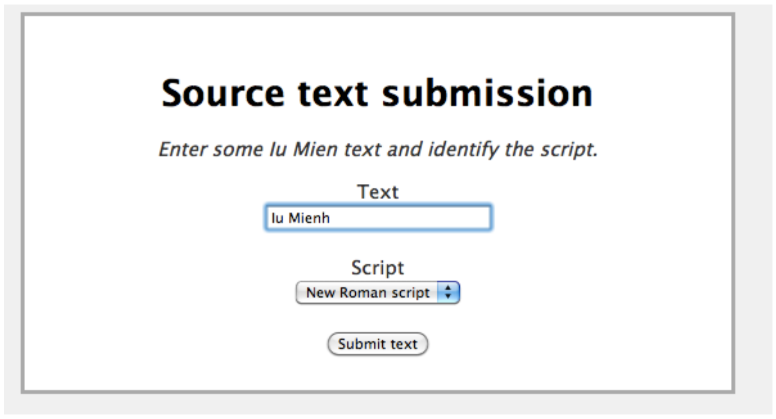
This service comes with no guarantees concerning the correctness of the text generated. However, your feedback will help to improve the quality of this free service. To that end, your comments and feedback are welcomed and encouraged.

Robert Batzinger
IU South Bend Informatics

Test the technology for yourself

Neural Network of multi-layered perceptrons trained with back propagation

Figure 3.30: Login page



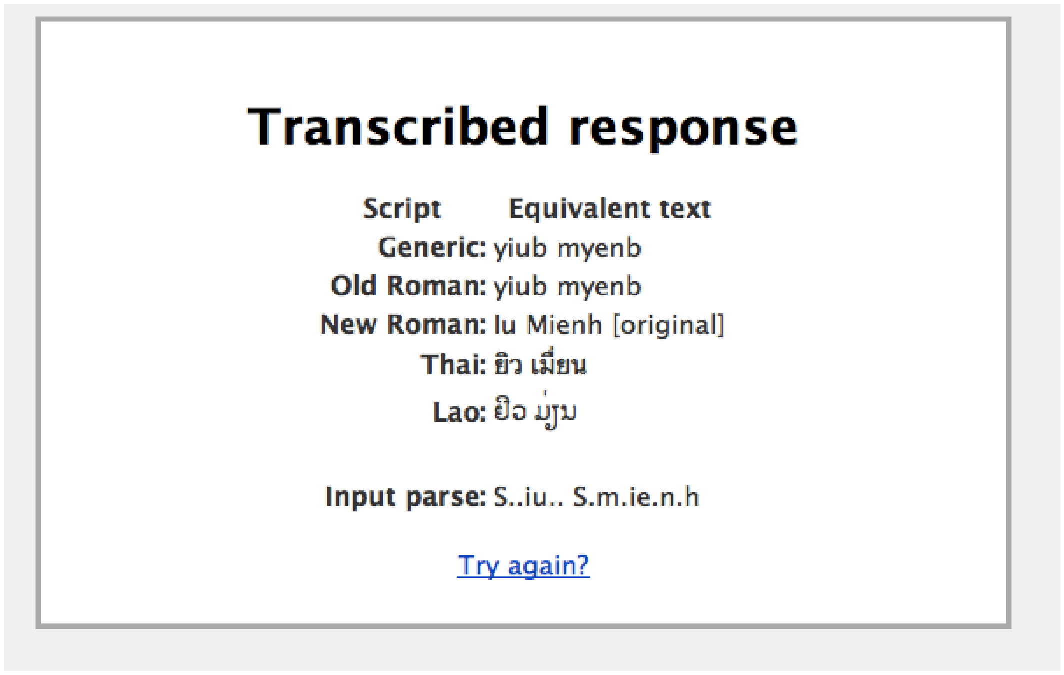
Source text submission

Enter some Iu Mien text and identify the script.

Text: Iu Mienh

Script: New Roman script

Figure 3.31: Submitting the text



Transcribed response

Script	Equivalent text
Generic:	yiub myenb
Old Roman:	yiub myenb
New Roman:	Iu Mienh [original]
Thai:	ยิว เมี่ยน
Lao:	ຢິວ ມ່ຽນ

Input parse: S..iu.. S.m.ie.n.h

Figure 3.32: Viewing the transcribed text

CHAPTER 4

Conclusion

4.1 Key paradigms for this project

This project was greatly assisted by two important paradigms:

- **The integration of unit test methods into class definition libraries:** Unit testing of software played an important and integrated part of the entire development. During the early days of learning Ruby, test units were used to verify the correctness of the code produced and the behavior of the software developed. However, later in the project, the test units also became important for identifying unexpected outcomes and interactions between modules. In fact on several occasions, upgrading to radically new versions of modules was greatly simplified because unit test methods had become a standard part of each class definition.

- **The use of statistical techniques to verify the relevance of observed outcomes:** A number of statistical approaches were used throughout this study to better understand the nature of transcription. Correlation of frequency rank order helped to distinguish between one-to-one and one-to-many relationships. Smooth scatterplots were found to be more informative than standard scatter-plots because color intensity provided a better hint of the underlying frequency. Multiple sampling was used to block for random effects. ANOVA was helpful to identify the relative impact of various factors and to rule out synergistic effects. This study did not depend solely on any one statistical method but it was interesting to note the consistency between the different methods in identifying what combination of scripts pose the greatest challenge in transcription.

However, important lessons were also learned through failure. The design of the web application is a good example. The initial design was far too complex for the end user especially when the transcriptions produced were not publishable quality. At this stage of the development, it would appear that a clean and simple interface is more attractive to end users than layers of security.

4.2 Reliability of derived transcriptions

Acceptable results were obtained for transcriptions between a Roman script to another Roman script as well as between an non-Roman script and another non-Roman script especially if the neural net or the ID3 was based on a training set that represented 90% of all syllables. However, transcribing between a Roman and an non-Roman script were disappointing with composite word error rates of 50%.

Having explored automated transcription of the Iu Mien, it is apparent that there are a couple features that helped to contribute to these outcomes. Future research in these areas could potentially reduce the error of transcription.

- **Typographic errors:** In the process of reducing words to syllables, the effective error rate was increased. A novice typist typically produce errors at a rate of 5%, within an edited publication the error rate drops to the order of 1 per 10 thousand. At this rate, several hundred typos could be expected among the million words published in a typical Bible translation. However, reducing a

Bible to a list of unique syllables effectively concentrates the errors. What was a few hundred per million words becomes a few hundred per thousand syllables making the error within the training set of the order 10%. Such errors can potentially create ambiguities in the textbase. Transcription rule development would be hindered by such contradictions to good information. Removing errors from the syllables list will increase the efficacy of the syllable list. It would also have been better to handle words like **ฯฒฯ** as an exception or abbreviation.

- **The phonetic model:** It is clear that the simple phonetic model did not provide enough information about the class of the character and the vowel length of the syllable. These features are important considerations for the selection of tone marks. Increasing the number of parsed features would not only provide more effective clues but also reduce the bitwidth of the input. In theory, this would also allow for better mapping of initial consonants and vowel features, and increase the informational and statistical power of the values provided.

  Based on the token patterns discovered in this study, it is possible to create more refined rules that would further reduce the number of tokens needed to describe a syllable. An example of such a parsing is given in Figures 4.1 to 4.2. Table 4.1 contrasts the effective input size compared to that used by this study showing the informational gains to be expected. This would increase the number of orthogonal features being parsed which would lead to better neural networks.

Table 4.1 shows that the input tokens could be reduced in size for most scripts. However, in the case of the New Roman script, it was not possible to determine whether `n` or `m` were meant to be part of digraph like `ng` or a consonant modifier in the following sequences: (`mv, mx, nc, nd, nh, nj, nq, nv, nx, nz`) Likewise, it

Table 4.1: Input token size

	Current	Proposed
Old Roman	122	109
New Roman	123	125
Tai	179	125
Lao	220	104

is suspected that the following diphthong are actually a contraction between an open pre-syllable and an open syllable. (`a'ei, a'i, a'o, a'waa, a'yie`) If either of these observations are correct, even the input count of the New Roman script could be shortened.

$$\left(\begin{array}{cccc} \textit{picon} & \textit{pdip} & \textit{pvow} & \textit{pstp} \\ \begin{pmatrix} \text{B}|\text{D}| \\ \text{G}|\text{K}| \\ \text{P}|\text{Q}| \\ \text{R}|\text{Z}| \\ \text{f}|\text{h}| \\ \text{k}|\text{l}| \\ \text{m}|\text{p}| \\ \text{s}|\text{t}| \\ \text{z} \end{pmatrix} & \begin{bmatrix} \text{w} \\ \text{y} \\ \text{yw} \end{bmatrix} & \begin{bmatrix} \text{a} \\ \text{aa} \\ \text{c} \\ \text{e} \\ \text{i} \\ \text{o} \\ \text{u} \\ \text{ua} \\ \text{r} \\ \text{x} \end{bmatrix} & [\text{'}] \end{array}\right) \begin{array}{cccccc} \textit{icons} & \textit{dip} & \textit{vow} & \textit{stp} & \textit{fcons} & \textit{ton} \\ \begin{pmatrix} \text{B}|\text{D}| \\ \text{E}|\text{F}| \\ \text{G}|\text{H}| \\ \text{J}|\text{K}| \\ \text{L}|\text{M}| \\ \text{N}|\text{P}| \\ \text{Q}|\text{R}| \\ \text{T}|\text{V}| \\ \text{W}|\text{Y}| \\ \text{Z}|\text{f}|\text{h}| \\ \text{k}|\text{l}|\text{m}| \\ \text{n}|\text{p}|\text{s}| \\ \text{t}|\text{v}|\text{z} \end{pmatrix} & \begin{bmatrix} \text{w} \\ \text{y} \\ \text{wy} \end{bmatrix} & \begin{pmatrix} \text{a}|\text{aa}| \\ \text{ai}|\text{aai}| \\ \text{au}|\text{aau}| \\ \text{aye}|\text{c}| \\ \text{e}|\text{ei}|\text{eu}| \\ \text{i}|\text{ia}|\text{iu}| \\ \text{o}|\text{oi}|\text{r}| \\ \text{ru}|\text{u}| \\ \text{ua}|\text{x}|\text{xi} \end{pmatrix} & \begin{bmatrix} \text{k} \\ \text{m} \\ \text{n} \\ \text{p} \\ \text{t} \\ \text{v} \end{bmatrix} & [\text{'}] & \begin{bmatrix} \text{b} \\ \text{d} \\ \text{g} \\ \text{j} \\ \text{q} \end{bmatrix} \end{array}$$

Figure 4.1: Revised Old Roman syllable parsing

$$\left(\overset{pcmod}{\begin{bmatrix} \text{m} \\ \text{n} \end{bmatrix}} \overset{picon}{\begin{bmatrix} \text{b} \\ \text{c} \\ \text{d} \\ \text{f} \\ \text{g} \\ \text{h} \\ \text{k} \\ \text{l} \\ \text{m} \\ \text{n} \\ \text{ng} \\ \text{p} \\ \text{q} \\ \text{s} \\ \text{z} \end{bmatrix}} \overset{pdip}{\begin{bmatrix} \text{y} \end{bmatrix}} \overset{pvow}{\begin{bmatrix} \text{a} \\ \text{aa} \\ \text{aau} \\ \text{ae} \\ \text{ai} \\ \text{au} \\ \text{e} \\ \text{ei} \\ \text{i} \\ \text{ie} \\ \text{o} \\ \text{oei} \\ \text{oi} \\ \text{or} \\ \text{ou} \\ \text{u} \\ \text{ua} \\ \text{ui} \\ \text{uo} \end{bmatrix}} \overset{pstp}{\begin{bmatrix} \text{'} \end{bmatrix}} \right) \overset{cmod}{\begin{bmatrix} \text{h} \\ \text{hm} \\ \text{m} \\ \text{n} \end{bmatrix}} \overset{icon}{\begin{bmatrix} \text{b} \\ \text{c} \\ \text{d} \\ \text{f} \\ \text{g} \\ \text{h} \\ \text{j} \\ \text{k} \\ \text{l} \\ \text{m} \\ \text{n} \\ \text{ng} \\ \text{p} \\ \text{q} \\ \text{s} \\ \text{t} \\ \text{v} \\ \text{w} \\ \text{x} \\ \text{z} \end{bmatrix}} \overset{dip}{\begin{bmatrix} \text{w} \\ \text{y} \end{bmatrix}} \overset{vow}{\begin{pmatrix} \text{a|aa|} \\ \text{aai|aau|} \\ \text{ae|ai|} \\ \text{au|e|ei|} \\ \text{er|eu|i|} \\ \text{ia|iaa|} \\ \text{iaau|iae|} \\ \text{iau|ie|} \\ \text{iei|io|} \\ \text{ior|iou|} \\ \text{iu|iua|} \\ \text{iui|o|oi|} \\ \text{or|ou|u|} \\ \text{ua|uaa|} \\ \text{uaai|uae|} \\ \text{uai|ue|} \\ \text{uei|ui|} \\ \text{uo} \end{pmatrix}} \overset{stp}{\begin{bmatrix} \text{'} \end{bmatrix}} \overset{fcons}{\begin{bmatrix} \text{k} \\ \text{m} \\ \text{n} \\ \text{ng} \\ \text{p} \\ \text{q} \\ \text{t} \end{bmatrix}} \overset{ton}{\begin{bmatrix} \text{c} \\ \text{h} \\ \text{v} \\ \text{x} \\ \text{z} \end{bmatrix}}$$

Figure 4.2: Revised New Roman syllable parsing

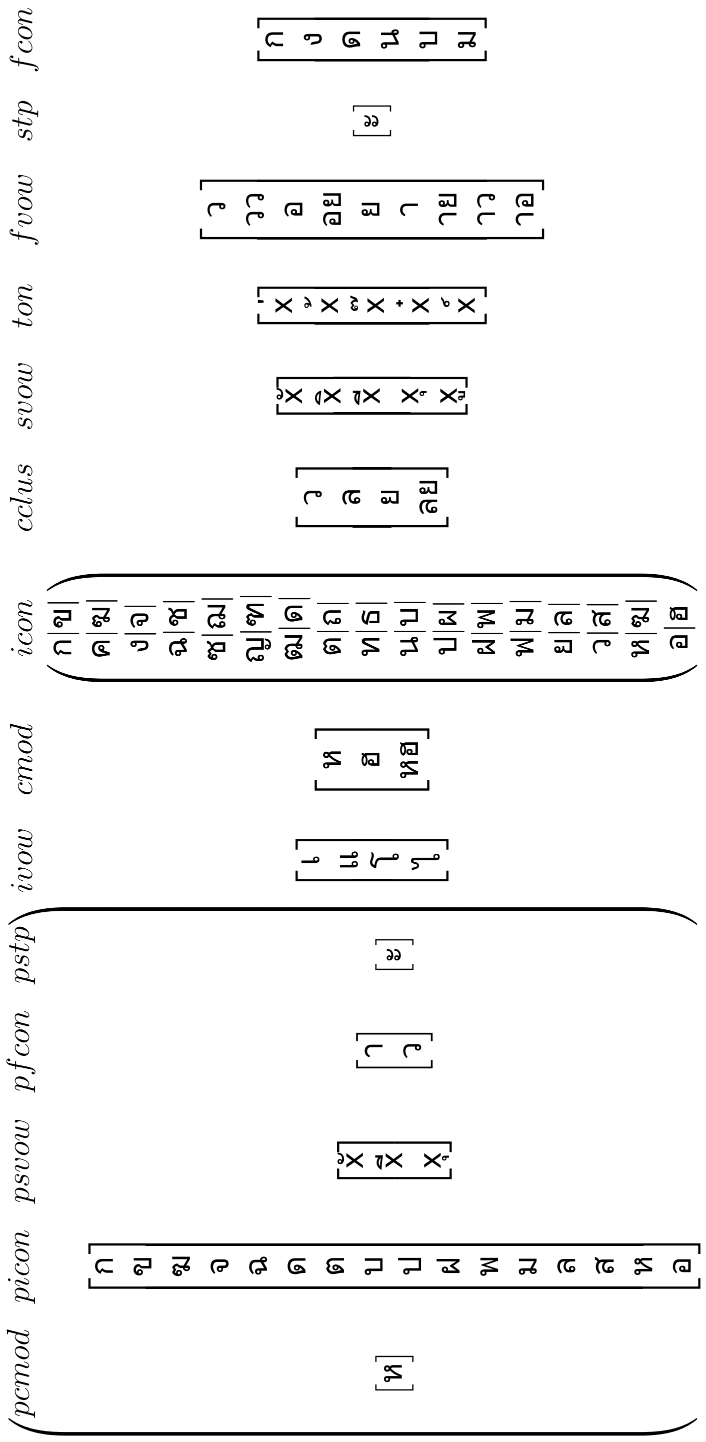


Figure 4.3: Revised the Thai syllable parsing

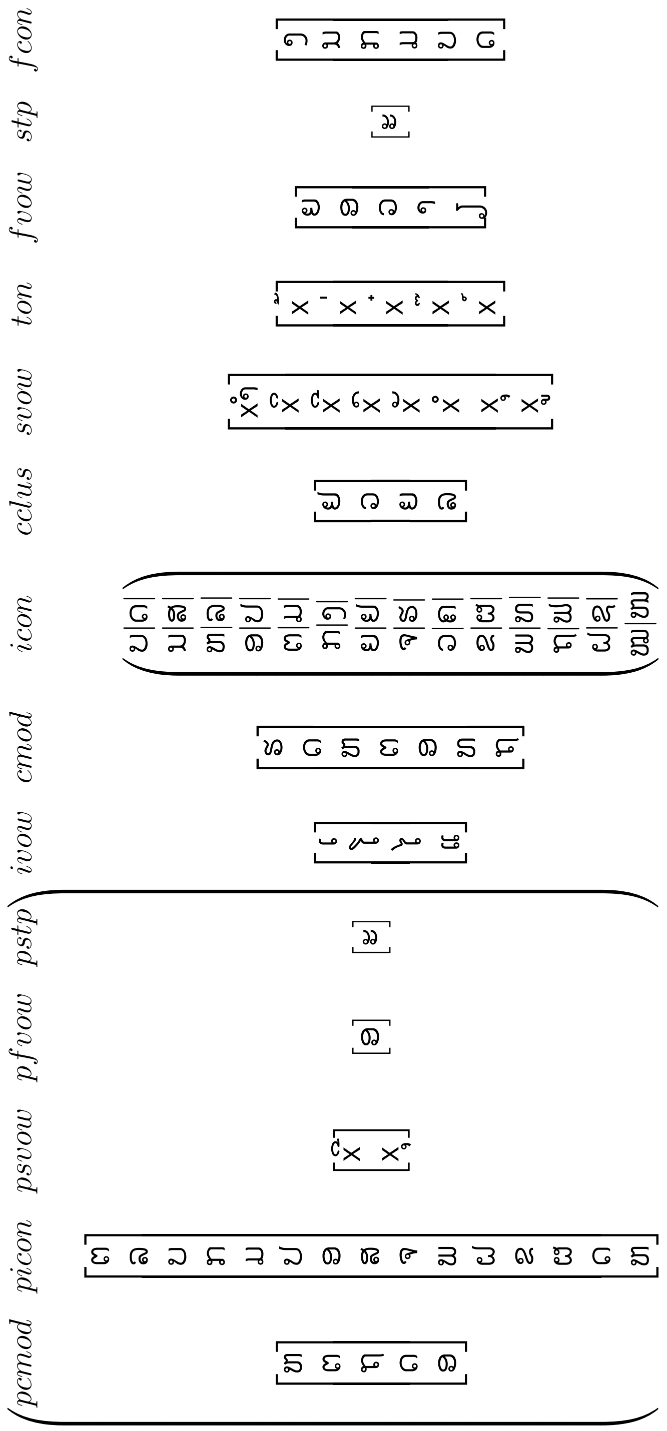


Figure 4.4: Revised the Lao syllable parsing

4.3 Improving the speed and performance

A neural network represents an computationally expensive operation. Given that the word distribution conformed to Zapf's Law, storing the equivalent syllables for the 200 most used Iu Mien words in a hashed array should effectively reduce the error rate for 2 main reasons. A hash table of the 200 most used words represents 90% of words communicated and would generally ensure that at least 90% of the transcribed words would be correct. Secondly, studies in English, French and German have shown that the most infrequent words tend to follow regular rules more closely than frequently used words, and there appears to be a regularization process across these languages that convert ancestral forms to gradually yield to emerging linguistic rules.[48, 49] These authors have suggested that such rules may also apply to the evolution of other languages as well. If this is true of Iu Mien, removing the most frequent words would have the potential of also removing a good portion of the irregular words which would make it easier to generate a more accurate neural network. At the same time, it would greatly improve performance as a hashed lookup is much faster than decomposing the word into syllables and phonemes, calculating the equivalent token via neural network and recomposing the output.

4.4 Choice of programming environment

The selection of Ruby as the scripting language for this project was for the most part an excellent choice. The built-in support for cross-indexed documentation as well as both unit and integrity checking were features of the language that were exploited throughout this project. As a programming language, Ruby truly lives up to its author's intent that Ruby syntax and design is governed by the law of minimum surprise.[50]

In retrospect, Ruby was a easy language to learn and master as it exhibited the flexibility and power of the C programming language within the object-oriented syntax of Java. However, throughout the duration of this project, Ruby was a language with cutting edge features in great demand within the highly competitive arena of web application development. The language has undergone multiple and significant changes throughout the 5 year history of this project. (The first programs were developed on Ruby, version 1.8.1 and current development is done on version 1.9.3.) For the most part, upgrades could be ignored until a new module was required for the next stage of development. Gem, the Ruby installed package manager, did an excellent job of handling dependencies between modules. However, installing a new module would often require updating others and some of those updates represented radical changes to early versions. Fortunately, the unit and integrity tests automated the process of checking for breaks in the software as it would not be uncommon for an update to break several routines. While most of the fixes were simple matters of configuration attribute settings or adopting new syntax, it nevertheless added to the amount of effort needed to maintain the development environment. Such is the cost of using cutting-edge technology on a project.

Within the industry, the selection of a programming environment still continues to be a very personal decision. I do feel that Ruby was a good choice of programming language for me as it was a good fit to who I am and how I work as a programmer. I found using Ruby easy and enjoyed using it because it modeled the way I work and think, especially in the following ways.

- I could analyze problems by building class libraries which were relatively easy to refactor.

- I could address my paranoia about software failures with copious unit and integrity tests.
- My comments and documentation within the source code could be automatically cross-indexed and reformatted as a collection of webpages.
- It allowed precise control for switching between UTF-8 and 8-bit ASCII in the strings and regular expressions of my software.

With recent developments and releases of new versions for other programming languages and web application, the search for a reliable development environment would no doubt reveal more options than what was available 5 years ago. Each language has been tailored to the needs and mindset of its users. The flame wars between programming language user groups have greatly died down and have been largely replaced by rapid porting of the best features from one languages to another. Both Ruby and Ruby on Rails have raised programmer expectations of the languages they work with and other open source languages have been borrowing Ruby technology and paradigms. JRuby, Groovy, Django, and CakePHP are popular attempts to port Rails frameworks to other programming languages. Even new languages like Lua and Erlang have built heavily on the lessons learned with Ruby.

The competitive development of open-source languages like Ruby will mean that programmers can continue to expect new programming tools to facilitate their work and better model the way they work and think. Even current languages are undergoing immense development to better support Unicode characters, application frameworks, and unit testing. Given the changes seen in the past 5 years, there is no doubt that the programming languages that will arise in the next decade promise to provide significant changes on how we conceptualize and develop computer solutions to problems.

APPENDIX A

The Thai and Lao Scripts

Thai and Lao are related languages which share common approaches to phonetics and writing systems. Historically both writing systems date back to 1283 when King Rankhamhaeng of Sukhothai adapted the Khmer script for use with the Thai-Lao languages. This worked well as both languages have 5 tones: mid, low, high, falling and rising. From these origins in north-central Thailand, the use of the script proliferated and underwent further modification and adaptation as it migrated along the major trade routes of the Chao Phraya River to the south and the Mekong River to the north. Within the cultural centers of the Lao and Siamese kingdoms, the Lao and Thai script diverged.

In Lao the emphasis was on simplifying the script. For the most part this was achieved by reducing the number of consonant and vowel markers. However, several regional variants emerged and were used until a single national script was established by the 1960 Official Order of the Lao government.[51]

For Thai, non-phonetic elements were introduced into the writing system to provide information about the origins of words. This is achieved through use of homophones and the use of a garun to include unspoken characters that occur in the original spelling of borrowed works in their language of origin. Foreign learners of Thai are often confused by the fact that **[ขฃคฅฆ]**, **[ฐฑฒถทธ]**, and **[ซศษส]** are sets of characters that have been used to represent ’k’, ’t’ and ’s’ consonants, respectively. The actual character used in the official spelling of a word depends on the tone and

Table A.1: Tone marking rules for Thai and Lao scripts

Consonant class	**Vowel length**	**Spoken tone** mid	low	high	falling	rising
Low	long	x คา		x้ ค้า	x่ ค่า	
Mid	long	x กา	x่ ก่า	x้ ก้า	x๊ ก๊า	x๋ ก๋า
Mid	short		x กะ	x๊ ก๊ะ	x้ ก้ะ	
High	long		x่ ข่า		x้ ข้า	x ขา

origin of the word.

In both the Lao and Thai scripts, there are three distinct classes of consonants: low, mid and high. Each class of consonant has its inherent tone level. Both scripts use tone markers to alter the inherit tone of the consonants class. Many of the low class consonants have corresponding high class consonants of the same sound. For example, in Thai an initial 's' sound can be rendered as either the low class ซ or the high class ส, depending on the tone required. In both scripts, unmatched low class consonants a silenced high class character marker to create the high class equivalent. Thus, an initial 'm' sound is rendered in Thai as either the low class ม or the high class digraph หม. The rules for tone marking in Thai and Lao are based on the class of the consonant, the length of the vowel and the tone change marker given. These rules have been summarized in Table A.1. The phonetic names, class and role of Unicode codes points for Thai and Lao characters are given in Tables A.2 and A.3, respectively.

Because the Thai and Lao national scripts have both evolved with complicated reading rules, there are number of changes made to simplify literacy when these scripts were adapted for use with Iu Mien. For example, Iu Mien vowel lengths are not as prominent a feature as they are in Thai and Lao. So short vowels have been used in presyllables and long vowels in syllables. Some consonants and vowel combinations were added to render phonetic features unique to Iu Mien.[52]

Table A.2: The Thai Character Set

Unicode		Description		Unicode		Description		Unicode		Description		Unicode		Description	
E01	ก	M	Ko Kai	E17	ท	L	Tho Thahan	E2D	อ	M	O Ang	E47	็	V	Maitaikhu
E02	ข	H	Kho Khai	E18	ธ	L	Tho Thong	E2E	ฮ	L	Ho Nokhuk	E48	่	T	Maiek
E03	ฃ	H	Kho Khuat	E19	น	L	No Nu	E2F	ฯ	L	Paiyannoi	E49	้	T	Maitho
E04	ค	L	Kho Khwai	E1A	บ	M	Bo Baimai	E30	ะ	v	Sara A	E4A	๊	T	Maitri
E05	ฅ	L	Kho Khon	E1B	ป	M	Po Pla	E31	ั	V	Maihanakat	E4B	๋	T	Maichattawa
E06	ฆ	L	Kho Rakhang	E1C	ผ	H	Pho Phung	E32	า	V	Sara Aa	E4C	์	S	Thanthakhat
E07	ง	L	Ngo Ngu	E1D	ฝ	H	Fo Fa	E33	ำ	V	Sara Am	E4D	ํ	V	Nikhahit
E08	จ	M	Cho Chan	E1E	พ	L	Pho Phan	E34	ิ	v	Sara I	E4E	๎	V	Yamakkan
E09	ฉ	H	Cho Ching	E1F	ฟ	L	Fo Fan	E35	ี	V	Sara Ii	E4F	๏	S	Fongman
E0A	ช	L	Cho Chang	E20	ภ	L	Pho Samphao	E36	ึ	v	Sara Ue	E50	๐	D	Zero
E0B	ซ	L	So So	E21	ม	L	Mo Ma	E37	ื	V	Sara Uee	E51	๑	D	One
E0C	ฌ	L	Cho Choe	E22	ย	L	Yo Yak	E38	ุ	v	Sara U	E52	๒	D	Two
E0D	ญ	L	Yo Ying	E23	ร	L	Ro Rua	E39	ู	V	Sara Uu	E53	๓	D	Three
E0E	ฎ	M	Do Chada	E24	ฤ	L	Ru	E3A	ฺ	V	Phinthu	E54	๔	D	Four
E0F	ฏ	M	To Patak	E25	ล	L	Lo Ling	E3F	฿	S	Baht	E55	๕	D	Five
E10	ฐ	H	Tho Than	E26	ฦ	L	Lu	E40	เ	v	E	E56	๖	D	Six
E11	ฑ	L	Tho Nangmontho	E27	ว	L	Wo Waen	E41	แ	V	Ae	E57	๗	D	Seven
E12	ฒ	L	Tho Phuthao	E28	ศ	L	So Sala	E42	โ	V	O	E58	๘	D	Eight
E13	ณ	L	No Nen	E29	ษ	H	So Rusi	E43	ใ	V	Ai Maimuan	E59	๙	D	Nine
E14	ด	M	Do Dek	E2A	ส	H	So Sua	E44	ไ	V	Ai Maimalai	E5A	๚	L	Angkhankhu
E15	ต	M	To Tao	E2B	ห	H	Ho Hip	E45	ๅ	V	Lakkhangyao	E5B	๛	L	Khomut
E16	ถ	H	Tho Thung	E2C	ฬ	L	Lo Chula	E46	ๆ	V	Maiyamok				

Key: L: low class letter; M: mid class letter; H: high class letter; V: vowel; T: tone; S: symbol; D: digit

Table A.3: The Lao Character Set

Unicode		Description		Unicode		Description		Unicode		Description		Unicode		Description	
E81	ກ	L	Ko	E99	ນ	L	No	EB1	◌ັ	V	Mai Kan	EC8	◌່	T	Mai Ek
E82	ຂ	L	Kho Sung	E9A	ບ	L	Bo	EB2	າ	V	Aa	EC9	◌້	T	Mai Tho
E84	ຄ	L	Kho Tam	E9B	ປ	L	Po	EB3	◌ຳ	V	Am	ECA	◌໊	T	Mai Ti
E87	ງ	L	Ngo	E9C	ຜ	L	Pho Sung	EB4	◌ິ	V	I	ECB	◌໋	T	Mai Catawa
E88	ຈ	L	Co	E9D	ຝ	L	Fo Tam	EB5	◌ີ	V	Ii	ECC	◌໌	S	Cancellation
E8A	ຊ	L	So Tam	E9E	ພ	L	Pho Tam	EB6	◌ຶ	V	Y	ECD	◌ໍ	S	Niggahita
E8D	ຍ	L	Nyo	E9F	ຟ	L	Fo Sung	EB7	◌ື	V	Yy	ED0	໐	D	Zero
E94	ດ	L	Do	EA1	ມ	L	Mo	EB8	◌ຸ	V	U	ED1	໑	D	One
E95	ຕ	L	To	EA2	ຢ	L	Yo	EB9	◌ູ	V	Uu	ED2	໒	D	Two
E96	ຖ	L	Tho Sung	EA3	ຣ	L	Lo Ling	EBB	◌ົ	V	Mai Kon	ED3	໓	D	Three
E97	ທ	L	Tho Tam	EA5	ລ	L	Lo Loot	EBC	◌ຼ	V	Lo	ED4	໔	D	Four
E88	ຈ	L	Co	EA7	ວ	L	Wo	EBD	ຽ	V	Nyo	ED5	໕	D	Five
E8A	ຊ	L	So Tam	EAA	ສ	L	So Sung	EC0	ເ	V	E	ED6	໖	D	Six
E8D	ຍ	L	Nyo	EAB	ຫ	L	Ho Sung	EC1	ແ	V	Ei	ED7	໗	D	Seven
E94	ດ	L	Do	EAD	ອ	L	O	EC2	ໂ	V	O	ED8	໘	D	Eight
E95	ຕ	L	To	EAE	ຮ	L	Ho Tam	EC3	ໃ	V	Ay	ED9	໙	D	Nine
E96	ຖ	L	Tho Sung	EAF	ຯ	S	Ellipsis	EC4	ໄ	V	Ai	EDC	ໜ	L	Ho No
E97	ທ	L	Tho Tam	EB0	ະ	V	A	EC6	ໆ	S	Ko La	EDD	ໝ	L	Ho Mo

Key: L: low class letter; M: mid class letter; H: high class letter; V: vowel; T: tone; S: symbol; D: digit

APPENDIX B
Source file samples

Figures B.1 to B.5 contain text samples taken from the corresponding source files used to publish the Iu Mien translation of the Bible. The content of this text represents the introduction and first 5 verses of the Iu Mien translation of the Bible portion known as Third Letter of John and referenced as 3Jn 1-5. As seen in the following pages, the source text files contained both the Iu Mien text and the corresponding text object markers as per the United Bible Societies Asia-Pacific Regional Standard Format Code.[46]

In the standard formating used, each marker was assigned symbols mnemonically to identify the role of subsequent text. In this way, text elements are marked for automated processing and typesetting under TEX. The text of the main title and sub titles follow the format markers `\mt` and `\st`. In these examples, there is also support for section and sub-section headings (`\s` and `\ss`), chapter and verse milestones (`\c` and `\v`), start of paragraph and levels of stanzas (`\p` and `\q1`, `\q2`, `\q3`, etc). Special characters style changes to identify a book title (`\bt`) or the resumption of normal text (`\tx`). The contents of introductory and tabular environments were marked by their respective sets of delimiters, (ie `\ib ... \ie` and `\tb ... \te`).

```
\id 3JN 3jn01 YAO, 9-2-89, Aspray, Thailand
\h 3 [yo-han]
\ib
\mt 3 [yo-han]
\st [yo-han fiaq Eei ta'faam zeiq fyenj]
\s [piuj mevb waag]
\p
\bt [yo-han fiaq Eei ta'faam zeiq fyenj]
\tx [yo-han] fiaq pun taub txb Jiu= paav Eei myenb, Buaj heug [kaai-atg].
[yo-han] fiaq Zev [kaai-atg] weig zu'g [kaai-atg] Hruq lovj, a'Neiq zipq
syenj [ye-su] Eei myenb. fyenj yaag Buaj [kaai-atg] xij zu'g faij fim
Buvb Jyenq taub myenb, Buaj heug [Di-o-te-fetq].
\tb2
\th [fyenj kxvq] &\bl             1-4 & [Jiad] kxn waag\bl
5-8 & [Zev kaai-atg] Eei waag\bl  9-10 & [Gemb Di-o-te-fetq]\bl
11-12 & [Zev De-metq-li-atg]\bl   13-15 & [setq] mweid waag\bl
\te
\ie
\c 0
\p
\v 1 [yia], Jiu= paav Eei myenb koj, fiaq fyenj pun Zyen Namq Eei
[kaai-atg].\x 1 *[kov= zob] 19:29; [lo-maa] 16:23; 1 [Ko-lin-To] 1:14*
yia zyen=~zyen Namq meib.
\p
\v 2 Namq Eei lod= kcv aa'b! yia Toq [Tin= huvb] pun meib zruj haiq
Euvg yaag hcvb wavg cvj pun meib wavg syavj, Navq yia hiuq
tu'q meib Eei livb wrnb wavg syavj nx.
\v 3 yia a'Neiq haig maaib teij kxj= yrud taaib naaiq Buaj yia,
meib zyepg zua'q Eei kan zyen leid, haiq zang yaag ei Jyenq
zyen leid zruj.
\v 4 maiq maaib haiq Euvg pun yia kaub Fyen= yrub Jiaj naaiq
a'q! se yia haid myenb kxvq yia Eei fu'Jweiq zang=~zang kan
lovj zyen leid.
\s [Zev kaai-atg zipq Kc'q lovj]
\p
\v 5 [Namq] Eei lod= kcv aag, meib za'kevb Tevj tu'q syenj [ye-su]
Eei kxj= yrud lovj haig. maiq kunq maiq pwatg Jiaj Eei myenb,
meib yaag lovg Hruq Tevj ninb Bua.
```

Figure B.1: Source text in the Generic script: (3Jn 1-5)

```
\id 3JN 3jn01 YAO, 9-2-89, Aspray, Thailand
\h 3 yo-han
\ib
\mt 3 yo-han
\st yo-han fiaq Eei ta'faam zeiq fyenj
\s piuj mevb waag
\p
\bt yo-han fiaq Eei ta'faam zeiq fyenj
\tx yo-han fiaq pun taub txb Jiub paav Eei myenb, Buaj heug kaai-atg.
yo-han fiaq Zev kaai-atg weig zu'g kaai-atg Hruq lovj, a'Neiq zipq
syenj ye-su Eei myenb. fyenj yaag Buaj kaai-atg xij zu'g faij fim
Buvb Jyenq taub myenb, Buaj heug Di-o-te-fetq.
\tb2
\th fyenj kxvq &\bl             1-4 & Jiad kxn waag\bl
5-8 & Zev kaai-atg Eei waag\bl  9-10 & Gemb Di-o-te-fetq\bl
11-12 & Zev De-metq-li-atg\bl   13-15 & setq mweid waag\bl
\te
\ie
\c 1
\p
\v 1 yia, Jiub paav Eei myenb koj, fiaq fyenj pun Zyen Namq Eei
kaai-atg.\x 1 *kovb zob 19:29; lo-maa 16:23; 1 Ko-lin-To 1:14*
yia zyenb zyen Namq meib.
\p
\v 2 Namq Eei lob kcv aa'b! yia Toq Tinb huvb pun meib zruj haiq
Euvg yaag hcvb wavg cvj pun meib wavg syavj, Navq yia hiuq
tu'q meib Eei livb wrnb wavg syavj nx.
\v 3 yia a'Neiq haig maaib teij kxb yrud taaib naaiq Buaj yia,
meib zyepg zua'q Eei kan zyen leid, haiq zang yaag ei Jyenq
zyen leid zruj.
\v 4 maiq maaib haiq Euvg pun yia kaub Fyenb yrub Jiaj naaiq
a'q! se yia haid myenb kxvq yia Eei fu'Jweiq zanb zang kan
lovj zyen leid.
\s Zev kaai-atg zipq Kc'q lovj
\p
\v 5 Namq Eei lob kcv aag, meib za'kevb Tevj tu'q syenj ye-su
Eei kxb yrud lovj haig. maiq kunq maiq pwatg Jiaj Eei myenb,
meib yaag lovg Hruq Tevj ninb Bua.
```

Figure B.2: Source text in Old Roman script: (3n 1-5)

```
\id 3JN 3jn01 YAO, 9-2-89, Aspray, Thailand

\h 3 Yo^han
\ib
\mt 3 Yo^han
\st Yo^han Fiev Nyei Da'faam Zeiv Fienx
\s Biux Mengh Waac
\p
\bt Yo^han Fiev Nyei Da'faam Zeiv Fienx
\tx Yo^han fiev bun dauh dorh jiu-baang nyei mienh, mbuox heuc Gaai^atc.
Yo^han fiev ceng Gaai^atc weic zuqc Gaai^atc hnyouv longx, a'hneiv zipv
sienx Yesu nyei mienh. Fienx yaac mbuox Gaai^atc oix zuqc faix fim
mbungh jienv dauh mienh, mbuox heuc Ndi^o^de^fetv.
\tb2
\th Fienx Gorngv &\bl             1-4 & Jiez gorn waac\bl
5-8 & Ceng Gaai^atc nyei waac\bl  9-10 & Nqemh Ndi^o^de^fetv\bl
11-12 & Ceng Nde^metv^li^atc\bl   13-15 & Setv mueiz waac\bl
\te
\ie
\c 1
\p
\v 1 Yie, jiu-baang nyei mienh gox, fiev fienx
bun cien hnamv nyei
Gaai^atc.\x 1 *Gong-Zoh 19:29; Lomaa 16:23; 1 Ko^lin^to 1:14*
Yie zien-zien hnamv meih.
\p
\v 2 Hnamv nyei loz-gaeng aah! Yie tov Tin-Hungh bun meih zoux haaix
nyungc yaac haengh wangc aengx bun meih wangc siangx, hnangv yie hiuv
duqv meih nyei lingh wuonh wangc siangx nor.
\v 3 Yie a'hneiv haic maaih deix gorx-youz daaih naaiv mbuox yie,
meih ziepc zuoqv nyei gan zien leiz, haaix zanc yaac ei jienv
zien leiz zoux.
\v 4 Maiv maaih haaix nyungc bun yie gauh njien-youh jiex naaiv
aqv! Se yie haiz mienh gorngv yie nyei fu'jueiv zanc-zanc gan
longx zien leiz.
\s Ceng Gaai^atc Zipv Kaeqv Longx
\p
\v 5 Hnamv nyei loz-gaeng aac, meih za'gengh tengx duqv sienx Yesu
nyei gorx-youz longx haic. Maiv gunv maiv buatc jiex nyei mienh,
meih yaac longc hnyouv tengx ninh mbuo.
```

Figure B.3: Source text in New Roman script: (3Jn 1-5)

```
\id 3JN 3jn01 YAO, 9-2-89, Aspray, Thailand
\h 3 โย^ฮัน
\ib
\mt 3 โย^ฮัน
\st โย^ฮัน เฟี้ย เญย ตะฟาม เฒ้ย เฝียน
\s ปิ๋ว เม่ง หว่า
\p
\bt โย^ฮัน เฟี้ย เญย ตะฟาม เฒ้ย เฝียน
\tx โย^ฮัน เฟี้ย ปุน เต้า ต้อ จิว-ปาง เญย เมี่ยน, บั๋ว เห่ว กาย^อัด.
โย^ฮัน เฟี้ย เธง กาย^อัด เหว่ย หฒุ กาย^อัด เฮญี้ยว หลง, อะเฮน้ย ฒิบ
เสียน เย^ซู\gb เญย เมี่ยน. เฝียน หย่า บั๋ว กาย^อัด อ๋อย หฒุ ไฝ พิม
บู้ง เจี๊ยน เต้า เมี่ยน, บั๋ว เห่ว ดี^โอ^เต^เฟ้ด.
\tb2
\th เฝียน ก๊อง &\bl 1-4  & เจี๋ย กอน หว่า\bl
5-8  & เธง กาย^อัด เญย หว่า\bl 9-10  & เฆ่ม ดี^โอ^เต^เฟ้ด\bl
11-12  & เธง เด^เม้ด^ลิ^อัด\bl 13-15  & เซ้ด เมว์ย หว่า\bl
\te
\ie
\c 1
\p
\v 1 เยีย, จิว-ปาง เญย เมี่ยน โก๋, เฟี้ย เฝียน
ปุน เธียน ฮนั้ม เญย
กาย^อัด.\x 1  *กง-โฒ่ 19:29; โล^มา 16:23; 1 โค^ลิน^โท 1:14*
เยีย เฒียนๆ ฮนั้ม เม่ย.
\p
\v 2 ฮนั้ม เญย โล์-แกง อ้า! เยีย โท้ ทิน-ฮู่ง ปุน เม่ย โหฒว หาย
หญู่ง หย่า แฮ่ง หวั่ง แอ๋ง ปุน เม่ย หวั่ง เสียง,\ths ฮนั้ง เยีย ฮิ้ว
ตุ๊ เม่ย เญย ลี่ง ว่วน หวั่ง เสียง นอ.
\v 3 เยีย อะเฮน้ย ไห่\gb ม่าย เต๋ย ก๋อ-โย์ว ต้าย น้าย บั๋ว เยีย,
เม่ย เหฒียบ ฒัวะ เญย กัน เฒียน เล์ย, หาย หฒั่น หย่า เอย เจี๊ยน
เฒียน เล์ย โหฒว.
\v 4 ไม้ ม่าย หาย หญู่ง ปุน เยีย เก้า เฌียน-โย่ว เจี๋ย น้าย
อ๊ะ! เซ เยีย ไฮ์ เมี่ยน ก๊อง เยีย เญย ฝุเจว๊ย หฒั่นๆ กัน
หลง เฒียน เล์ย.
\s เธง กาย^อัด ฒิบ แคะ หลง
\p
\v 5 ฮนั้ม เญย โล์-แกง อ่า, เม่ย หฒะเก้ง เถง ตุ๊ เสียน เย^ซู
เญย ก๋อ-โย์ว หลง ไห่. ไม้ กุ๊น ไม้ ปวัด เจี๋ย เญย เมี่ยน,
เม่ย หย่า หล่ง เฮญี้ยว เถง นิ่น บัว.
```

Figure B.4: Source text in Thai script: (3Jn 1-5)

```
\id 3JN 3jn01 YAO, 9-2-89, Aspray, Thailand
\h ໂຢˆຮັນ
\ib
\mt 3 ໂຢˆຮັນ
\st ໂຢˆຮັນ ເພ້ຍ ເຍີຍ ຕະຟາມ ເຕສື໊ຍ ຝຽນ
\s ປື໋ອ ເມ່ງ ຫວ່າ
\p
\bt ໂຢˆຮັນ ເພ້ຍ ເຍີຍ ຕະຟາມ ເຕສື໊ຍ ຝຽນ
\tx ໂຢˆຮັນ ເພ້ຍ ປຸນ ເຕົ໊າ ຕົ໋ ຈິວ-ປາງ ເຍີຍ ມ່ຽນ, ບົ໋ວ ເຫ່ວ ກາຍˆອັດ.
ໂຢˆຮັນ ເພ້ຍ ເທສງ ກາຍˆອັດ ເຫວີ່ຍ ຕສຸ ກາຍˆອັດ ໂຮຍ້ວ ຫລີງ, ອະເຣນີ້ຍ ຕສຶ໊ບ
ສຽນ ເຢˆຊູ ເຍີຍ ມ່ຽນ. ຝຽນ ຢ່າ ບົ໋ວ ກາຍˆອັດ ອ໋ອຍ ຕສຸ ໄຜ ພິມ
ບູ້ງ ຈ໊ຽນ ເຕົ໊າ ມ່ຽນ, ບົ໋ວ ເຫ່ວ ດີˆໂອˆເຕˆເຟດ.
\tb2
\th ຝຽນ ກ໊ອງ &  \bl           1-4  & ເຈ໌ຍ ກອນ ຫວ່າ \bl
5-8  & ເທສງ ກາຍˆອັດ ເຍີຍ ຫວ່າ \bl       9-10  & ເອກ້ມ ດີˆໂອˆເຕˆເຟດ \bl
11-12  & ເທສງ ເດˆເມດˆລີˆອັດ\bl       13-15  & ເຊດ ເມວີ໊ຍ ຫວ່າ\bl
\te
\ie
\c 1
\p
\v 1 ເຢຍ, ຈິວ-ປາງ ເຍີຍ ມ່ຽນ ໂກ໋, ເພ້ຍ ຝຽນ ປຸນ ທສຽນ ຮນ້ຳ ເຍີຍ ກາຍˆອັດ.
\x 1 *ກົງ-ໂຕສ໌ 19:29; ໂລˆມາ 16:23; 1 ໂຄˆລິນˆໂທ 1:14*
ເຢຍ ຕສຽນໆ ຮນ້ຳ ເມີ່ຍ.
\p
\v 2 ຮນ້ຳ ເຍີຍ ໂລ໌-ແກງ ອ້າ! ເຢຍ ໂຫ້ ທິນ-ຮູ່ງ ປຸນ ເມີ່ຍ ໂຕສ໋ວ ຫາຍ
ຫຍູ່ງ ຢ່າ ແຮ່ງ ຫວັ່ງ ແອ໋ງ ປຸນ ເມີ່ຍ ຫວັ່ງ ສຢັງ, ຮນັ້ງ ເຢຍ ຮີ້ວ
ຕຸ໊ ເມີ່ຍ ເຍີຍ ລີ່ງ ວ່ວນ ຫວັ່ງ ສຢັງ ນໍ.
\v 3 ເຢຍ ອະເຣນີ້ຍ ໄຫ່ ມ່າຍ ເຕີ໋ຍ ກໍ໋-ໂຢ໌ວ ຕ້າຍ ນ້າຍ ບົ໋ວ ເຢຍ,
ເມີ່ຍ ຕສຽບ ຕສົ໊ວະ ເຍີຍ ກັນ ຕສຽນ ເລີ໊ຍ, ຫາຍ ຕສັ່ນ ຢ່າ ເອີຍ ຈ໊ຽນ
ຕສຽນ ເລີ໊ຍ ໂຕສ໋ວ.
\v 4 ໄມ້ ມ່າຍ ຫາຍ ຫຍູ່ງ ປຸນ ເຢຍ ເກົ້າ ອຈຽນ-ໂຢ້ວ ເຈ໋ຍ ນ້າຍ
ອ໊ະ! ເຊ ເຢຍ ໄຮ໌ ມ່ຽນ ກ໊ອງ ເຢຍ ເຍີຍ ຝຸຈ໊ວຍ ຕສັ່ນໆ ກັນ ຫລີງ ຕສຽນ ເລີ໊ຍ.
\s ເທສງ ກາຍˆອັດ ຕສຶ໊ບ ແຄະ ຫລີງ
\p
\v 5 ຮນ້ຳ ເຍີຍ ໂລ໌-ແກງ ອ່າ, ເມີ່ຍ ຕສະເກ້ງ ເຖງ ຕຸ໊ ສຽນ ເຢˆຊູ
ເຍີຍ ກໍ໋-ໂຢ໌ວ ຫລີງ ໄຫ່. ໄມ້ ກຸ໊ນ ໄມ້ ປວັດ ເຈ໋ຍ ເຍີຍ ມ່ຽນ,
ເມີ່ຍ ຢ່າ ຫລີ່ງ ໂຮຍ້ວ ເຖງ ນິ່ນ ບົວ.
```

Figure B.5: Source text in Lao script: (3Jn 1-5)

APPENDIX C

Website behavoir specifications

The following sections contain the behavorial specifications for the various features of the simplified online transcription service hosted on Heroku in August 2011.[1]

C.1 Splash page

```
Feature: a slash screen
As a user I want to be assured that
the site is an open service that I am
authorized to use.

Scenario: Link on splash page
When I have requested the home page
Then I will see the splash page
And I will see a link to the text submission page
```

Code Frag. C.1: Behavior of the splash page

[1]This is the third revision of the website hosted at `http://mien.heroku.com`.

C.2 Text submission

```
Feature: Text submission
As a user I want to be able to submit Iu Mien text
for transcription into 4 scripts.

Scenario: Forgotten text
Given I have a copy of the submit text form
When I have clicked on the submit text button
And I am missing the text sample
Then I will see an error message
And I will see the submit text form

Scenario: Forgotten script id
Given I have a copy of the submit text form
When I have clicked on the submit text button
And I am missing a valid script id
Then I will see an error message
And I will see the submit text form

Scenario: Completed text submission form
Given I have a copy of the submit text form
When I have supplied the script id
And I have supplied the text sample
And I have clicked on the submit text button
Then I will see the results page
```

Code Frag. C.2: Behavior of the text submission page

C.3 Results page

```
Feature: User results
As a user I want to be able to view the
results of autotranscription

Scenario: The text cannot be parsed
Given I have a copy of the submit text form
And I have supplied the script id
And I have supplied the wierd text
When I have clicked on the submit text button
Then I will see the results page
And I will see an error message in the parsed text

Scenario: The text cannot be parsed
Given I have a copy of the submit text form
And I have supplied the script id
And I have supplied the wierd text
When I have clicked on the submit text button
Then I will see the results page
And I will see transcribed versions
```

Code Frag. C.3: Behavior of the text results

Bibliography

[1] Eugene Peterson. *The Message: The Bible in Contemporary Language.* NavPress, Colorado Springs, CO, 2002.

[2] Thailand Bible Society. *Iu Mien Bible in Old Roman Script.* Thailand Bible Society, Bangkok, Thailand, 2007.

[3] Thailand Bible Society. *Iu Mien Bible in New Roman Script.* Thailand Bible Society, Bangkok, Thailand, 2007.

[4] Thailand Bible Society. *Iu Mien Bible in Thai Script.* Thailand Bible Society, Bangkok, Thailand, 2007.

[5] Thailand Bible Society. *Iu Mien Bible in Lao Script.* Thailand Bible Society, Bangkok, Thailand, 2007.

[6] Richard Sproat. *A Computational Theory of Writing Systems.* Cambridge University Press, 2000.

[7] Krisana Charoenwong. *The Nationalist Chinese (Kuomintang) Troops in Northern Thailand: a study on the political, economic and social effects of their resettlement, 1945-1980's.* PhD thesis, Faculty of Social Sciences and Humanities, National University of Malaysia, Bangi, 1999.

[8] Robert N. Kearney and Clark D. Neher. *Politics and Moderization in South and Southeast Asia.* John Wiley and Sons, New York, 1975.

[9] John L. S. Girling. *Thailand: Society and Politics.* Cornell University Press, Ithaca, NY, 1981.

[10] Robert P. Batzinger. The Computer-Assisted Text Processing Needs of Asia-Pacific. Technical report, United Bible Societies CATP Center, Chiang Mai, Thailand, 1988.

[11] Helen Abadzi. Strategies and policies for literacy. Report of the World Bank, Operations Evaluation Department. The electronic copy available at `http://portal.unesco.org/education`, March 2006.

[12] L. Ehri. Learning to read words: Theory, findings, and issues. *Scientific Studies of Reading*, 9:167–188, 2005.

[13] A. Holm and B. Dodd. The effect of first written language on the acquisition of english literacy. *Cognition*, 59:119–147, 1996.

[14] N. Akamatsu. The effects of first language orthographic features on second language reading in text. *Language Learning*, 2003.

[15] M. O'Connor. The alphabet as a technology. In Peter T. Daniels and William Bright, editors, *The world's writing systems*, pages 141–159. Oxford University Press, 1996.

[16] Raymond G. Gordon, Jr., editor. *Ethnologue: Languages of the World.* SIL International, Dallas, TX, fifteenth edition, 2005.

[17] Peter T. Daniels. Methods of decipherment. In Peter T. Daniels and William Bright, editors, *The world's writing systems*, pages 141–159. Oxford University Press, New York, 1996.

[18] I. Dan Melamed. *Empirical Methods for Exploiting Parallel Texts.* MIT Press, Cambridge, MA, 2001.

[19] Stefan Wermter, Ellen Riloff, and Gabriele Scheler. Learning approaches for natural language processing. In *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, pages 1–16, London, UK, 1996. Springer-Verlag.

[20] Vasileios Hatzivassiloglou. Do we need linguistics when we have statistics? a comparative analysis of the contributions of linguistic cues to a statistical word grouping system. In Judith L. Klavans and Philip S. Resnik, editors, *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, pages 67–94. MIT Press, Cambridge, Massachusetts, 1996.

[21] Steven A. Jacobson. *Yup'ik Eskimo Dictionary.* University of Alaska Press, Fairbanks, AK, 1984.

[22] Carolyn Penstein Rosé and Alex H. Waibel. Recovering from parser failures: A hybrid statistical and symbolic approach. In Judith L. Klavans and Philip S. Resnik, editors, *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*, pages 157–179. MIT Press, Cambridge, Massachusetts, 1996.

[23] Random House, editor. *Random House Webster's Unabridged Dictionary.* Random House, second edition, 2005.

[24] Andreea Cervatiuc. Esl vocabulary acquisition: Target and approach. *The Internet TESL Journal (`http://iteslj.org/`)*, XIV(1), Jan 2008.

[25] Mark Davies. The corpus of contemporary american english (coca): 425 million words, 1990-present. Available online at `http://corpus.byu.edu/`, 2008.

[26] M. Paul Lewis, editor. *Ethnologue: Languages of the World.* SIL International, Dallas, TX, sixteenth edition, 2009.

[27] Herbert C. Purnell. *Yao-English Dictionary.* Department of Asian Studies, Cornell University, Ithaca, NY, 1968.

[28] Mary R. Haas. Book review of *yao-english dictionary* compiled by sylvia j lombard and edited by herbert c. purnell, jr. *American Anthropologist*, 71:367–368, 1969.

[29] Donald E. Knuth. Backus normal form vs. backus naur form. *Communications of the ACM*, 7(12):735–736, 1964.

[30] Ann Burgess, editor. *Mien Hymnbook: Old Roman Script Edition.* O.M.F., Bangkok, Thailand, 1989.

[31] Ann Burgess, editor. *Mien Hymnbook: New Roman Script Edition.* O.M.F., Bangkok, Thailand, 1989.

[32] Ann Burgess, editor. *Mien Hymnbook: Thai Script Edition.* O.M.F., Bangkok, Thailand, 1989.

[33] John Ross Quinlan. Induction of decision trees. *Machine Learning*, pages 81–106, Mar 1986.

[34] Paul Werbos. *Beyond regression: New tools for prediction and analysis in the behaviorial sciences.* PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA, Nov 1974.

[35] Dave Thomas, David Heinemeier Hansson, Leon Breedt, Mike Clark, Thomas Fuchs, and Andeas Schwarz. *Agile Web Deveopment with Rails.* The Pragmatic Bookshelf, Raleigh, NC, 2005.

[36] Michael Swaine. Ruby on rails: Java's successor. *Dr Dobb's Journal*, 32(385):20–28, June 2006.

[37] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[38] Wayne Sewell. *Weaving a Program: Literate Programming in WEB*. Van Nostrand Reinhold, New York, NY 10003, 1989.

[39] Dave Thomas. *Programming Ruby: The Pragmatic Programmers' Guide*. The Pragmatic Bookshelf, Raleigh, NC, 2005.

[40] Steve Pugh. *Wicked cool Ruby scripts : useful scripts that solve difficult problems*. No Starch Press, San Francisco, CA, 2009.

[41] Matt Wynne and Aslak Hellesøy. *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. The Pragmatic Bookshelf, Dallas, TX, 2011.

[42] David Chelimsky, David Astels, Zach Dennis, Aslak Hellesoy, Bryan Helmkamp, and Dan North. *The RSpec Book: Behavior-driven development with RSpec, Cucumber and Friends*. Facets of Ruby. Pragmatic Bookshelf, Raleigh, NC, 2010.

[43] Sergio Fierns and Thomas Kern. Ai4r :: Artificial intelligence for ruby. available online at `http://ai4r.rubyforge.org`, 2007.

[44] The Unicode Consortium. *The Unicode 4.0 Standard*. Addison-Wesley Publishing Company, Reading, MA, fourth edition, 2004.

[45] The Unicode Consortium. *The Unicode 5.0 Standard*. Addison-Wesley Publishing Company, Reading, MA, fifth edition, 2007.

[46] Robert P. Batzinger. *Standard Format Marking of Scripture*. United Bible Societies Asia-Pacific Technical Support Office for Computer-Assisted Text Processing (UBS-APTSOCAP), Bible House, Singapore, 1992.

[47] George Kingsley Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Cambridge, MA, 1949.

[48] Erez Lieberman, Jean-Baptiste Michel, Joe Jackson, Tina Tang, and Martin A. Nowak1. Quantifying the evolutionary dynamics of language. *Nature*, 449:713–716, July 2007.

[49] Eva Grabowski and Dieter Mindt. Die unregelmäßigen verben des englischen: eine lernliste auf empirischer grundlage. *Die Neueren Sprachen*, 93(4):334–353, 1994.

[50] Yukihiro Matsumoto. *Ruby in a Nutshell.* O'Reilly & Associates, Sebastopol, CA, 2002.

[51] Anthony Diller. Thai and lao writing. In Peter T. Daniels and William Bright, editors, *The world's writing systems*, pages 457–466. Oxford University Press, 1996.

[52] William A. Smalley. The use of non-roman script for new languages. In William A. Smalley, editor, *Orthography studies: articles on new writing systems*, volume IV of *Helps for Translators*, pages 71–107. United Bible Societies, London, UK, 1964.

Vita

Robert Batzinger was born in 1953 in Schenectady, NY and has been involved in research from a young age. Upon graduation from High School in 1971, he assisted on the pioneering work in immuno-fluorence in the New York State Rabies Laboratory as a summer lab assistant. That fall, he started his studies in analytical organic chemistry at Massachusetts Institute of Technology and as an undergraduate research assistant, he participated in the research leading up to the isolation and identification of aflatoxin. He graduated with a SB in Chemistry after 3 years of study and then studied parasite pharmacology as a research fellow under Dr. Ernest Bueding at Johns Hopkins University School of Public Health, graduated in 1978 with a PhD in Pathobiology. From there, he entered two years of post-doctoral studies in chemical carcinogenesis under Drs. Elizabeth and James Miller at the McArdle Laboratories for Cancer Research at Wisconsin University in Madison. In 1981 he joined Payap University in Chiang Mai, Thailand as the Acting Dean of the Faculty of Science and Head of the Faculty of Pharmacy Development Project. During this time, he started the Department of Computer Science and Office of Information Technology Services as well as reversed engineered the CP/M operating system to handle Thai data.

In 1985, he became the Director of the Non-Roman Development Project for the United Bible Societies (UBS) in Chiang Mai, Thailand where he developed text processing software to facilitate keyboarding, editing, and typesetting of Asian language text in non-Roman script. In 1990, the project was moved to a regional facility in Singapore where Dr. Batzinger became the Director of the UBS Asia-Pacific Technical Support Office for Computer-Assisted Text Processing that provided technical assistance and training for over 600 translation and publishing projects in 23 countries in the region.

In 2003, he moved to the States and joined the IU South Bend Informatics as Lab manager in 2004. He also began studies in the Masters program in Computer Science and Applied Mathematics in 2005. In the years that followed, Dr. Batzinger was working and studying at IU South Bend, he also taught Introduction to Web Programming (CSCI A-340) and Introduction to Object-oriented programming in Ruby (CSCI-A201). He participated in a department project to introduce Visual Basic to the Technology Magnet Program at Riley High School. He also provided instruction and consultation on both LaTeX and XeLaTeX while also assisting in various research projects within the Department. Dr. Batzinger has also promoted the inclusion of open source software not only on student lab builds both in the Computer Science and Informatics Department Labs but also in the IU-ITS workstations in the open lab workstations and lecture rooms across campus.

Dr. Batzinger's professional interests include the use of artificial intelligence in data mining, natural language processing techniques in publishing support, and applications of web technologies in software engineering.