

**TRANSLITERATING SINHALA:
A COMPUTATIONAL
APPROACH TO CONVERT
ROMANIZED SINHALA AND
EMBEDDED ENGLISH INTO
SINHALA SCRIPT**

**M.L.A.S. Yapa
2024**

Transliterating Sinhala: A Computational Approach to Convert Romanized Sinhala and Embedded English into Sinhala Script

**M.L.A.S. Yapa
2024**



Transliterating Sinhala: A Computational Approach to Convert Romanized Sinhala and Embedded English into Sinhala Script


**A dissertation submitted for the Degree of Master of
Computer Science**

**M.L.A.S. Yapa
University of Colombo School of Computing
2024**

Declaration


Name of the student: M.L.A.S. Yapa
Registration number: 2020/mcs/100
Name of the Degree Programme: Master of Computer Science
Project/Thesis title: Transliterating Sinhala: A Computational Approach to Convert Romanized Sinhala and Embedded English into Sinhala Script

1. The project/thesis is my original work and has not been submitted previously for a degree at this or any other University/Institute. To the best of my knowledge, it does not contain any material published or written by another person, except as acknowledged in the text.
2. I understand what plagiarism is, the various types of plagiarism, how to avoid it, what my resources are, who can help me if I am unsure about a research or plagiarism issue, as well as what the consequences are at University of Colombo School of Computing (UCSC) for plagiarism.
3. I understand that ignorance is not an excuse for plagiarism and that I am responsible for clarifying, asking questions and utilizing all available resources in order to educate myself and prevent myself from plagiarizing.
4. I am also aware of the dangers of using online plagiarism checkers and sites that offer essays for sale. I understand that if I use these resources, I am solely responsible for the consequences of my actions.
5. I assure that any work I submit with my name on it will reflect my own ideas and effort. I will properly cite all material that is not my own.
6. I understand that there is no acceptable excuse for committing plagiarism and that doing so is a violation of the Student Code of Conduct.

Signature of the Student	Date (DD/MM/YYYY)
	19.09.2024

Certified by Supervisor(s)

This is to certify that this project/thesis is based on the work of the above-mentioned student under my/our supervision. The thesis has been prepared according to the format stipulated and is of an acceptable standard.

	Supervisor 1	Supervisor 2	Supervisor 3
Name	H N D Thilini		
Signature			
Date	29/09/2024		

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisor Dr. H.N.D. Thilini for her invaluable guidance and for motivating me throughout my project. I am also grateful for Dr. B.H.Randil Pushpananda for his guidance and advices.

Furthermore, I am deeply thankful to the Language Technology Research Laboratory at the University of Colombo School of Computing for the valuable inputs in data collection phase.

My special thanks go to my family members who have always backed me to achieve the best and support provided throughout the period. Finally, my special appreciation goes to my friends who have helped me in each step of the project with their insights, support, and motivation.

ABSTRACT

This thesis presents a comprehensive study on the transliteration of Sinhala focusing on the conversion of Romanized Sinhala and embedded English using Neural Machine Translation (NMT) into Sinhala. The research revolves around developing a robust NMT system that leverages a Gated repeated Unit model (GRU) with Bahdanau attention optimized for the nuances of The Sinhala language.

The study begins with an exploration of current methodologies in NMT identifying gaps in The Sinhala transliteration research particularly in the context of handling mixed-language texts commonly found in social media. A novel transliteration system is proposed to address these gaps; the thesis proposes a user-friendly web application developed using Angular 17 for the frontend and Python FastAPI for the backend. This system not only translates embedded English phrases via the Google Translator API but also includes an n-gram model to improve the accuracy of the final transliteration output.

In order to evaluate the effectiveness of the Singlish systems Several evaluation approaches including different scenarios of English transliteration with and without vowels and original words were used. The performance of the model was rigorously tested using BLEU and METEOR scores revealing a high degree of accuracy in transliteration across different testing scenarios.

However, the research acknowledges some limitations particularly in computational resources that limited the size of the training dataset. The project concludes by highlighting potential future enhancements such as exploring advanced neural network architectures such as transformer models and extending the system's capabilities to other regional languages.

This thesis contributes to computational linguistics by providing a practical solution to Sinhala transliteration challenges paving the way for further research in less-studied languages and script conversion technologies.

Keywords - *Natural Language Processing, Transliterating Sinhala, Encoder-Decoder, GRU Model, N-gram model, translating embedded English.*

TABLE OF CONTENTS

Acknowledgement	ii
Abstract.....	iii
Table of Contents	iv
List of Figures.....	vi
List of Tables.....	vii
Abbreviations	viii
Chapter 1 - Introduction	1
1.1 Motivation	1
1.2 Statement of the problem.....	2
1.3 Research Aims and Objectives	2
1.3.1 Aim	2
1.3.2 Objectives.....	3
1.4 Scope.....	4
1.5 Structure of the Thesis	4
Chapter 2 - Literature Review	6
2.1 Neural Machine Translation	6
2.2 Encoder-Decoder Model in Neural Machine Translation.....	6
2.3 GRU Model and Bahdanau Attention.....	7
2.4 Literature.....	7
2.4.1 Literature related to NMT	10
2.5 Research Gaps and Justification for the Project	12
2.5.1 Choice of GRU Over LSTM	14
2.6 Conclusion	14
Chapter 3 - Methodology.....	15
3.1 Design	15
3.1.1 Overview	15

3.1.2 Data Collection.....	16
3.1.3 Data analyzing and cleaning	18
3.1.4 Collecting Sinhala Corpus.....	19
3.1.5 Evaluation and error analysis	20
3.1.6 Implementing a web application	20
3.1.7 Translating Embedded English Words / Phrases	21
3.2 Implementation	22
3.2.1 Overview	22
3.2.2 NMT Model.....	23
3.2.3 Web Application with translation of embedded English phrases.	30
Chapter 4 - Evaluation and Results	32
4.1 Evaluation Approach	32
4.2 Transliteration Unit Breaking Algorithm Evaluation	32
4.3 Transliteration Model Evaluation	33
4.4 N-gram and Embedded English Translation Evaluation.	33
Chapter 05 - Conclusion and Future Work.....	36
Appendix A	I
References	II

LIST OF FIGURES

Figure 3.1	High-level visualization of the Design.....	16
Figure 3.2	Custom web application developed for collecting YouTube comments.....	17
Figure 3.3	Function for Normalizing text	24
Figure 3.4	Function for tokenizing Romanized Sinhala words	25
Figure 3.5	Function for tokenizing Sinhala words.....	26
Figure 3.6	Encoder Class	27
Figure 3.7	Bahdanau Attention Class	27
Figure 3.8	Decoder Class	28
Figure 3.9	User Interface of the developed web application	31

LIST OF TABLES

Table 3.1	Summary of the datasets.....	18
Table 3.2	Example for Sinhala words and their possible Romanized Sinhala variants	18
Table 3.3	Examples for Romanized words embedded with numbers	19
Table 3.4	Examples for possible English conversions	31
Table 4.1	TU Breaking Examples	33
Table 4.2	Evaluation Matrix Scores	33

ABBREVIATIONS

BLSTM - Bidirectional Long-Short Term Memory

BLEU - Bilingual Evaluation Understudy

CER - Coupon Equivalent Rate

DBN - Deep Belief Networks

GRU - Gated Recurrent Unit

LTRL - Language Technology Research Laboratory

LSTM - Long Short-Term Memory

METEOR - Metric for Evaluation of Translation with Explicit Ordering

NLP - Natural Language Processing

NLTK - Natural Language Toolkit

NMT - Neural Machine Translation

RNN - Recurrent Neural Network

SMT - Statistical Machine Translation Models

TU -Transliteration Unit

UCSC - University of Colombo School of Computing

WER - Word Error Rate

CHAPTER 01 - INTRODUCTION

1.1 Motivation

Advancements of digital communication have resulted in the development of powerful technologies which support linguistic diversity. Computational tools that can reliably process and comprehend the distinct linguistic phenomenon of Singlish - the phonetic typing of Sinhala using the English alphabet, are desperately needed in the context of Sri Lanka, where Sinhala is the primary language. There are multiple primary motivators for this study, all of which are aligning with the research objectives specified.

To begin with, there's a substantial gap in the transliteration tools available today, as evidenced by the increasing popularity of Singlish in digital communication. The complexity of Romanized Sinhala and the incorporation of English terms are difficult for many current systems to handle, which might result in errors and misunderstandings in digital text. This research aims to fill this gap by creating an algorithm that can transliterate Singlish phrases successfully, addressing the need for more advanced language processing tools in Sri Lanka.

Secondly, Singlish's non-standard spelling differences and grammatical complexity pose an extra challenge for natural language processing. In addressing this issue, this study collects and analyzes large datasets to train a machine learning model that identifies and translates these deviations into normal Sinhala script. This method advances the study of computational linguistics and may pave the way for solving comparable problems in language processing for other languages.

Moreover, this study has significant practical implications. This study has the potential to improve the digital experience for Sinhala speakers by enhancing the quality and reliability of transliteration tools, hence promoting communication that is more efficient and effective. Additionally, it supports preserving the Sinhala language's linguistic integrity in digital environments, which is essential for maintaining cultural traditions.

Finally, this study aims to improve the process of translating English words into Sinhala in digital text so that the sentences that are produced are accurate and meaningful in their context. this has a significant importance in a bilingual society where code-switching between Sinhala and English is prevalent. To summarize, this study is mainly motivated by the importance to fill the technological gap between processing embedded English and Romanized Sinhala, to advance computational linguistics. Further, to improve the quality of digital communication for

Sinhala speakers.

1.2 Statement of the problem

In Sri Lanka the main language of communication is Sinhala (Englebreksten and Genetti, 2005). Therefore, people use Sinhala to communicate on their social media channels, messaging applications, etc. According to the Department of Census and Statistics Sri Lanka (2012, p.128), 79.7% of the total Sri Lankan population are literate in Sinhala. Even though there are multiple tools that allow people to type Sinhala letters, most of them are more familiar with typing Sinhala in English letters (Romanized Sinhala). This method is known as “Singlish” in Sri Lanka.

It is very common that people use English words when typing, just as when speaking informally. Hence, from this research, a method to transliterate Singlish phrases to Sinhala along with translating English words to Sinhala used in the phrase will be found out. If we take the following Sinhala sentence “මට රෝහලට යන්න ඕනේ” as an example, here the most common word for “රෝහල” will be “hospital”. Therefore, people tend to type such as “Mata hospital ekata yanna one”, “Mt hospital ekt ynn one” and likewise.

Another problem is, people are using different ways to type the same Sinhala word in Romanized form (Sumanathilaka, Weerasinghe and Priyadarshana, 2022, p.108). For example, the word “කවදාද” can be typed in various ways using Romanized English such as “kawadada”, “kwadda”, “kwdd”, “kwadaada”, etc. The objective is to develop a model which will identify the accurate Sinhala words for such Singlish words and similarly for sentences as well.

This study tries to solve these problems via constructing advanced algorithms that can change Singlish phrases into the standard Sinhala script completely while translating English terms which are contained within them correctly. In doing so it will enrich the quality and trustworthiness of digital communication for those who speak Sinhala hence maintaining its language purity in cyber space. Hence, this research will be useful in creating a faster and more efficient way of digital communication.

1.3 Research Aims and Objectives

1.3.1 Aim

This research project's main goal is to improve Sinhala digital communication by creating an advanced computational model that can precisely transliterate Romanized Sinhala (also known as "Singlish") and embedded English into standard Sinhala script. By guaranteeing that the

Sinhala language is faithfully represented and conserved in digital forms, this approach seeks to fill the gaps in current linguistic technology. This will enable more efficient and culturally cohesive communication amongst Sinhala speakers.

1.3.2 Objectives

i. Gathering sufficient data sets to train the machine learning model.

To put together a comprehensive data set that includes a variety of Singlish sentences and their equivalents in standard Sinhala. This data set will be crucial in training and testing the machine learning model developed which will help to ensure its accuracy and resilience in a range of linguistic scenarios.

ii. Analyzing currently available transliteration tools.

To carry out an in-depth analysis of the transliteration tools that are currently available. This analysis helps to identify the current drawbacks and positive aspects offering a fundamental knowledge base that will guide the creation of a better transliteration algorithm.

iii. Algorithm Development:

To develop and refine an algorithm to transliterate Singlish para phrases written in different spellings along with translating English words to Sinhala. Further, in order to ensure that the result is both linguistically accurate and appropriate for the context, this system will also have the ability to translate embedded English terms within these phrases.

iv. To identify the most appropriate Sinhala word:

To develop a process inside the model that determines the most appropriate Sinhala word corresponding to a given Singlish word, considering the different ways a Singlish word can be Romanized. This feature is essential for the transliterated text to retain its semantic integrity.

v. Contextual Integration:

To identify the most suitable Sinhala case/suffix which can be used after translating English words to Sinhala to obtain a meaningful sentence in terms of context and grammar. This is highly important to maintain the natural flow and the true meaning of the sentence.

1.4 Scope

The purpose of this study project is to specifically address the difficulties and complexity associated with transliterating English sentences embedded in normal Sinhala script and Romanized Sinhala, also known as Singlish.

Through the literature review, an in-depth examination of existing research in the field of natural language processing (NLP) in the context of Sinhala. This review will cover previous studies and their findings on the methodology and identified limitations. The aim of this project is to develop a comprehensive understanding of the current state of research and identify gaps that can be addressed. As such this study is concentrated on the application and development of NLP tools and techniques. Accordingly, it will build on machine learning models to develop an algorithm capable of handling the unique challenges presented by Singlish and embedded English. This includes the variability in Romanized spelling and contextual integration of English words in Sinhala sentences.

Further, the practical application of the developed model will also be under the scope of this study. This involves a rigorous testing of its efficiency in accurately transliterating and translating Romanized Sinhala and English into standard Sinhala script. Testing will be extensive using a variety of datasets to ensure the robustness and adaptability of the model.

However, the project has its boundaries and limitations. It will mainly focus on transliteration and translation aspects within the specified linguistic framework without going into broader linguistic studies beyond the scope of the interaction of the English language and the Sinhala.

The study will also outline potential areas for future research based on the findings and limitations of the current study. This will provide useful guidance to future research efforts in this evolving field.

1.5 Structure of the Thesis

Chapter 1: Introduction

This chapter provides the basic background information pertaining to the research topic, outlines the research questions and objectives and provides justification for the study in the field of computational linguistics and digital communication. Further, it specifies the boundaries within which the study will be conducted.

Chapter 2: Literature Review

This chapter consists of an in-depth analysis of the relevant theories and the core concepts based

on which the study is conducted. Further this chapter reviews the existing literature in the field related to back-transliteration and specifies the challenges and the limitations in its application in the current context.

Chapter 3: Methodology

This chapter presents in detail the design and structure of the research project and the plan followed to achieve the objectives. Further, it explains the proposed model's architecture for transliteration and translating texts. Moreover, it explains the code implementation of the identified solution and details about the programming languages and packages used to develop the solution.

Chapter 4: Evaluation and Results

This chapter includes a comprehensive discussion of the experiments conducted and the achieved results. Further it elaborates the extent to which the research goals are achieved and discusses their implications.

Chapter 6: Conclusion

The conclusion based on the research findings will be included in this chapter where it summarizes how the project results address the research questions and it highlights the study's contribution to the field of computer science and linguistics. And also, this chapter will suggest the possible future extensions for future research based on the findings and limitations of the current project.

CHAPTER 2 - LITERATURE REVIEW

2.1 Neural Machine Translation

Neural Machine Translation (NMT) is a modern method for Machine Translation that uses advanced computer algorithms from the field of deep learning. This method helps to convert text from one language to another in different ways than older systems. Instead of relying on fixed rules or statistical analysis NMT uses a complex system known as artificial neural networks particularly the type called the sequence-to-sequence model.

This approach started becoming popular in the early 2010s after the research work done by Sutskever et al. (2014) and Bahdanau et al. (2014). Sutskever et al. introduced the seq2seq model which consists of two parts: the first part encoder which reads and processes the text to be translated and the second part decoder which creates the translation in the new language. Bahdanau et al. improved this model by adding something called an attention mechanism. This addition helps the system to focus more on different parts of the text while translating particularly longer sentences, making translation more accurate.

NMT offers several advantages over traditional statistical machine translation models (SMT). It improves the Quality of Translation as NMT can produce more fluent and contextually accurate translations. In addition, NMT models learn to translate directly from source to target text without needing intermediate steps or alignment models which were necessities in SMT in addition NMT can better handle long-range dependencies in sentences (Murali et al. 2019, pp. 75-81).

Despite its advantages NMT has some issues. The training of NMT models requires large datasets and extensive computational resources. There are also issues with handling rare words and maintaining consistency of translation in terms of terms and style across large documents.

2.2 Encoder-Decoder Model in Neural Machine Translation

The encoder-decoder model serves as the foundation of modern machines (NMT) systems of Neural Machine translation. This architecture is often implemented by using recurrent neural networks (RNNs) or transformers specifically designed to handle the complexities of translating text in an end-to-end manner.

Encoder: The encoder part of the model efficiently processes the source text capturing its semantic and syntactic information. It transforms input into a context vector, a compressed representation of the source sentence capturing its essence.

Decoder: The decoder then takes this vector and generates the translated text in the target

language. It operates sequentially predicting one word at a time while being conditioned on context vectors and previously generated words.

2.3 GRU Model and Bahdanau Attention

The implementation of the GRU (Gated Recurrent Unit) model coupled with Bahdanau attention forms a pivotal part of our system's architecture and enhances its capability in the tasks in.

The GRU model is a type of recurrent neural network (RNN) known for its efficiency and effectiveness in sequence modeling tasks. Key features of the GRU model include:

Simplified Architecture: Unlike traditional RNNs and its counterpart LSTM (Long Short-Term Memory) GRU has a less complex structure with fewer parameters making it computationally more efficient while retaining the ability to capture.

Gating Mechanisms: To control the flow of information and data the GRUs have update and reset gates. These gates determine how much information must be passed through to the future aiding in solving the vanish gradient problem common in standard RNNs.

The Bahdanau attention or additive attention is integrated into the GRU model to enhance its performance in translation tasks:

Context-based Translation: This attention mechanism allows the model to focus on different parts of the input sequence while generating each word in the output, leading to more accurate and contextually relevant translations.

Dynamic Content Weighing: Bahdanau Attention dynamically assigns weights to different input tokens, making the model's output sensitive to the most relevant parts of the input, a crucial feature for long sequences.

2.4 Literature

The existing systems for transliteration and suggestion generation in Sinhala, being rule-based, lack the ability to achieve word predictions based on different shorthand typing patterns. To address this limitation, a study has been proposed by Sumanathilaka, Weerasinghe and Priyadarshana (2023, p.53) by a novel transliteration system using an enhanced Trie data structure for efficient word prediction. The system utilizes a survey to identify various typing patterns and formulates rules accordingly. These rules are then used to annotate a Sinhala dictionary and train the Trie model. The trained model predicts possible Sinhala words based on Romanized Sinhala input and compares them with a Romanized Sinhala to Sinhala knowledge base to generate unique Sinhala word suggestions. Experimental testing of the model using 200 unique Romanized Sinhala test data yielded a word-level prediction accuracy

of 84%. This novel transliterator aims to address the ambiguity issue in Romanized Sinhala to Sinhala transliterations and enhance the typing experience for users of Romanized Sinhala.

The research done by Hettiarachchi, Weerasinghe and Pushpananda (2020, pp. 250-255) aims to automatically detect hate content in social media comments and documents written in Romanized Sinhala. While previous studies focused on hate speech recognition in English or other languages, this research focuses on identifying Sinhala words written in English letters. The growing issue of hate words and hated texts prompts the application of machine learning and computer science to combat this problem. The study compares various feature extraction methods, four machine learning algorithms, and different N-gram values (unigram, bigram, and trigram), with a minimum document frequency (Min-Df) value of 3. The investigation involves comparing different features for classifying hate speech comments on Facebook using a dataset of approximately 2500 comments, including some containing hate speech. The classifier is trained and tested with these diverse features.

The paper done by Salaev, Kuriyozov and Gomez-Rogriguez (2022, p.7) introduces a Python code, web tool, and API developed for transliterating the low-resource Uzbek language between Cyrillic and Latin alphabets. It also includes the reformed version of the Latin alphabet mandated for legal texts by 2023. The authors address the challenges specific to the transliteration of these three scripts, which cannot be solved through simple character mapping. They discuss ongoing efforts to handle user-input related issues and outline future work to improve output quality by implementing more mapping rules, user input cleaning techniques, and incorporating a pretrained neural language model to handle unseen cases. Additionally, the authors express their intention to develop an Uzbek language pipeline for NLP tasks like tokenization, POS tagging, morphological analysis, and parsing in the future.

Asian language transliteration systems have been developed in part thanks to the direct and indirect contributions of various research papers. Given that Hindi and Sinhala are both Indo-Aryan languages with rich morphologies (Pushpananda, Weerasinghe and Niranjana, 2014, pp. 129-133), similar methods for both languages' transliteration systems were used. The important studies that have directly influenced the current research are discussed below.

In the study of Wijerathna et al (2012, pp. 14-18) a mechanism for translating from English to Sinhala and from Sinhala to English was implemented. To make the system easier to use, Sinhala words have been submitted in Singlish format, and a rule-based method was used to

transliterate Singlish to Sinhalese. As translating from English to Sinhalese is the main objective of this study, the translation method also incorporates morphological and grammar principles from both languages. The total system was tested with 500 Sinhala sentences and translated those words with an accuracy of 87% (Wijerathna et al, 2012, p. 14). Since the system relies on predefined rules and some words require human interactions to translate, word ambiguity is one of its key drawbacks.

A transducer for English to Sinhalese transliteration was created by Hettige and Karunananda (2007, pp. 209-214) using finite-state automaton. In this work, two different transliteration methods from English to Sinhalese have been implemented. The first model was developed for words with original English letters, and the second model was developed for terms using Sinhala letters. A special Sinhala to English mapping table has been proposed because producing Sinhala letters using English letters deviates significantly from the original model.

Liwera and Ranathunga (2020, pp. 1-5), has presented a trigram model for transliteration of Romanized-Sinhala texts in 2020. In their work, they have implemented a system in which a bigram model and a unigram model serve as backup plans in the event that trigrams are not discovered. They employed a rule-based method to execute transliteration if no n-grams were discovered. For their model, they were able to attain 77%-character level accuracy.

A grapheme and phoneme-based transliteration model has been implemented in the research done by Oh and Choi (2005, pp. 1737-1748). The model's development took both the process of orthographic and phonetic conversion into account. It has been compared using a variety of machine learning techniques to the prior grapheme-based and phoneme-based models. This model's performance has improved by 13% to 78% compared to the previous models. This research will be useful to get an idea about the grapheme and phoneme-based models which will be useful to implement a model.

Another research that has used grapheme and phoneme is “Sinhala Grapheme-to-Phoneme Conversion and Rules for Schwa Epenthesis” by Wasala, Weerasinghe and Gamage (2006, pp, 890-897). A way to translate Sinhala Unicode text into phonemic pronunciation specifications is illustrated in this paper. The primary focus of this study was on consonant schwa and /a/ vowel epenthesis confusions. The proposed rules were tested using 30000 different terms to overcome the disambiguates. The grapheme-to-phoneme conversion model has a 98% accuracy rate.

Hailu and Josan (2017, p. 205) have proposed a rule-based approach to transliterate English to Tigrinya. It has defined a set of grammatical and lexical rules. The drawbacks with the rule-based approach have also been covered. The developed system performed at 90.9% accuracy when measured using the word accuracy rate.

A word level Singlish to Sinhala translator called “Swa-Bhasha” has been proposed by Athukorala and Sumanathilaka (2022, p. 29) using a rules-based approach. The system's word-level prediction accuracy was 84%, while its suggestion-level prediction accuracy was 92%. However, the current approach has been implemented without putting much emphasis on grammar and phrase transliteration. As a result, they have mentioned that the technology can be improved in the future to transliterate sentences and paragraphs using Sinhala grammar rules.

“A Deep Learning Approach to Machine Transliteration” by Deselaers et al (2009, pp. 233-241) has presented a novel method based on deep belief networks (DBN). Several characteristics that can be used for both translation and transliteration have been demonstrated. It contains intriguing aspects relating to sequence reordering. The results are said to be 1% better overall when DBN-based transliterations are added, which is well behind the other approaches.

A Pali to Sinhala translator has been proposed by Shalini and Hettige (2017, p.23) which consists of three models namely Pali morphology analyzer, dictionary-based translator, and the Sinhala morphology generator. In the proposed system, they are only able to deal with a limited amount of data because of lack of availability.

2.4.1 Literature related to NMT

Shazal Usman and Habash developed in 2020 using GRU an Encoder-Decoder model focusing on a novel approach that integrates language detection and transliteration. This method specifically targets Arabizi text leaving unmodified foreign language texts while transliterating Arabizi into Arabic. They introduced a comprehensive model that not only detects Arabizi but also transliterates it into a code-mixed output conforming to consistent Arabic spelling norms especially in line with the CODA convention. The model addresses the non-linear alignment between Arabizi and the CODA convention by using special characters [+] and [-] respectively to indicate. Moreover, the symbol is used to denote foreign words in the source text which remain unchanged during training and are replaced with their original forms in a subsequent

post-processing step. This procedure also applies to words tagged with [+] and [-] which are in the post-processing phase merged or split accordingly. Shazal and friends. The latter is facilitated by inserting <sow> and <eow> tags for the first and the end of words in sentences. Additionally, their research revealed that factors such as synthetic data usage foreign language embeddings and multi-word sequences can negatively affect the model's accuracy. The model underwent testing on the BOLT Egyptian Arabic SMS/Chat and Transliteration corpus, which contains around 275,000 words, achieving a word accuracy of 80.6% and a BLEU (Bilingual Evaluation Understudy) score of 58.7% (Bies et al., 2014).

Ameur, Meziane, and Guessoum (2017) have made a significant contribution to the subject of machine transliteration with their research, which primarily focuses on Arabic and English. The Arabic language has received relatively less attention in transliteration tasks than languages like English, French, and Chinese. This study fills a significant research gap in the field by focusing on specific segments of the input sequence to improve phonetic and orthographic representation. According to the study, their suggested method performs more efficiently when transliterating Arabic to English than some of the earlier studies in the field. This work highlights the necessity for further targeted investigations on less studied languages like Arabic and offers insightful information about the usage of attention-based encoder-decoder models in transliteration tasks. Future studies in the field of machine transliteration can benefit greatly from the conclusions and methods that Ameur et al. provided.

Grundkiewicz and Heafield (2018) conducted an extensive study on the transliteration of named entities across various languages using a range of techniques. Their approach included strategies such as the dropout regularization model involving rescoring and right-to-left models and back-translation. The study involved training and testing models for named entity transliteration on five datasets in multiple languages including english thai persian chinese vietnamese hindi tamil kannada bangla and Hebrew. Researchers adopted the bi-deep model architecture characterized by stacked decoders and four bi-directional alternating encoders with two-layer transition cells. Their research concluded that the NMT approach is highly effective for the development of machine translation systems.

The study conducted by Ansari et al. (2022) marks a significant advance in the field of transliteration especially in the context of Devanagari to Roman Hindi transliteration and its reverse. This relatively underexplored research area presents unique challenges and opportunities for neural machine translation techniques (NMT). The project aims to determine

transliteration of Roman Hindi script and Devanagari script. In a single language with different scripts the requirement of Maintaining a language presents specific linguistic and technical challenges which make this study special. The model employs a sequence-to-sequence neural network structure comprising two main components: an encoder for transforming a source language word into a contextual representation and a decoder for generating. This architecture is well-established in the NMT for its effectiveness in handling such translation tasks. A distinctive feature of this model is the use of GRU for the encoder and decoder networks. GRUs are known for their efficiency in sequence modeling and for their ability to handle long-term dependencies crucial in transliteration tasks. The choice of a multilayer GRU network highlights the model's capacity to capture complex linguistic nuances between these scripts. The model's performance was evaluated in the measurement of transliteration accuracy using the standard metrics coupon equivalent rate (CER) and word error rate (WER). The study reports a WER of 64.8% and a CER of 20.1%, indicating a substantial improvement over existing methods for Hindi-English bilingual machine transliteration. The success of this model in effective handling of Hindi to English translation demonstrates the potential of sequence-to-sequence networks based on gru in transliterating between complex scripts. It contributes to the broader understanding of the NMT in less commonly studied language pairs and script combinations.

Nanayakkara, Nadungodage & Pushpananda (2023) introduced a sequence-to-sequence model leveraging an encoder-decoder framework for the back-transliteration from Romanized text to Sinhala. This study represents a recent integration of Neural Machine Translation (NMT) in the realm of english-sinhala back-transliteration. The model uses a bidirectional long short-term Memory (BLSTM) for the encoder and a standard long-short term Memory (LSTM) for the decoder. Transliteration Units in this model form the core element of the Transliteration process. The Data collected for training testing and evaluation was sourced from two different sources. The model's effectiveness was evaluated using BLEU and METEOR (Metric for Evaluation of Translation with Explicit Ordering) scores with results on The Dakshina dataset showing an average BLEU score of 0.57 and a METEOR score of 0.47 while The LTRL dataset achieved higher scores of 0.81 and 0.78 respectively.

2.5 Research Gaps and Justification for the Project

This research project is undertaken as a response to several identified gaps in the field of neural machine translation (NMT) focusing specifically on the back-transliteration of Sinhala. The following points highlight areas where this research contributes significantly.

Limited Research in Back-Transliterating Sinhala Using NMT

There is a notable scarcity of research investigating the use of NMT in the back-transliterating of Sinhala. Most existing studies have focused on the more common spoken languages leaving a gap in knowledge and technology for Sinhala. This project aims to bridge this gap by developing an NMT model specifically tailored to the Sinhala language. This contribution is intended to contribute to the broader applicability of NMT in less-research.

Challenges in identifying native English Words in Romanized Sinhala Code-Mixed Texts

In the world of digital communication and social media where code-mixing is prevalent, accurately identifying and transliterating native English words embedded In Romanized Sinhala texts poses a significant challenge. This study aims to address This by studying and implementing methods for language identification in code-mixed texts. The ability to differentiate and appropriately handle different languages in the same text is crucial for the accuracy and usability of back-transliteration systems in real-world applications.

Gap in Comprehensive Systems with User Interaction

While there has been research focusing on statistical machine translation (SMT) and rule-based approaches for the Sinhala transliteration there is a notable lack of comprehensive systems that include user interaction. Most existing studies end at the development stage of the model without any extension into practical applications. This research fills this void by not only developing a robust NMT model but also integrating This model into a user-friendly web application. This holistic approach ensures that the research benefits are directly accessible to the users making it a complete system that extends beyond theoretical modeling to practical utility.

This research project is positioned to make a significant contribution by addressing these critical gaps in the field of NMT for Sinhala. This project aims to improve the application and accessibility of NMT in Sinhala and possibly in other less-studied languages By focusing on under-researched areas implementing language identification in code-mixed contexts and developing a complete system with user interfaces.

2.5.1 Choice of GRU Over LSTM

Gated Recurrent Units (GRU) was chosen as opposed to Long Short-Term Memory (LSTM) Units used by Nanayakkara et al (2023) In developing our neural machine translation model for Sinhala. This decision was grounded in several factors contrasting our approach and their methodology.

GRUs offer a more efficient architecture compared to LSTMs making them computationally efficient (Chung et al., 2014). With fewer gates GRUs provide a balance between performance and computation resources.

When considering the dataset size and model performance, GRUs are a more suitable choice based on the size of our dataset as they have been shown to perform well with smaller datasets (Jozefowicz et al., 2015). This efficiency is especially valuable in context of specific language tasks such as the Sinhala transliteration.

Further, GRU has flexibility in Learning Long-Term Dependencies. Although LSTMs are renowned for their ability to learn long term dependency GRUs can achieve similar levels of performance (Chung et al., 2014). It makes them a robust alternative for the task of back-transliteration.

2.6 Conclusion

The field of neural machine translation is shown in the conclusion of this literature review particularly in the context of less-researched languages like Sinhala ripe with opportunities for innovation and growth. The review highlighted significant advancements and methodologies in NMT from foundational encoder-decoder models to recent developments in GRU and LSTM architectures. It also highlighted the importance of addressing language specific challenges such as code mixing in social media texts and the issues involved in back-transliteration processes.

The identified research gaps point to a need for more focused studies in Sinhala transliteration using NMT that emphasize not just model development but also practical applications including user interaction. This project aims to contribute to this area by not only developing a robust NMT model for Sinhala but also by presenting a comprehensive system that incorporates a user-friendly interface.

CHAPTER 3 - METHODOLOGY

3.1 Design

3.1.1 Overview

This chapter provides an overview of the overall design and the methodology of the research. The research journey is characterized by an experimental research design, a strategic choice which supports the iterative process of testing refinement and optimizing the transliteration system.

Using this methodology a multi-stage process is followed each stage building on the previous one to achieve the overarching goal of accurate and efficient transliteration. The research begins with a thorough process of collecting the data gathering a large amount of Romanized Sinhala phrases and embedded English phrases. It then is subjected to a phase of analysis and pre-processing laying the foundation for the subsequent steps.

The research continues a few months after the production of the dataset to the critical task of selecting and tailoring an NLP model to accommodate the complexities of Sinhala transliteration. The decision of the GRU model enhanced by an Attention mechanism is a pivotal decision reflecting the innovative edge of the project. The model was chosen not only for its technical suitability but also for its potential to push the boundaries of current transliteration practices.

The following sections of this chapter will elaborate on the complex processes involved in modeling training creation of a comprehensive dictionary from the dataset and the integration of user interface elements with the Google Translator API. This integration is a key feature which allows the seamless incorporation of English words and phrases into Romanized Sinhala sentences.

The methodology provides a detailed description of the process of transliteration itself, specifically in the handling of words not found in the initial word dictionary. This includes an explanation of how the GRU model is used with the Attention mechanism to generate accurate transliterations of previously unseen words.

Figure 01 offers a high-level visualization of these key stages, providing a roadmap for the journey undertaken in this research.

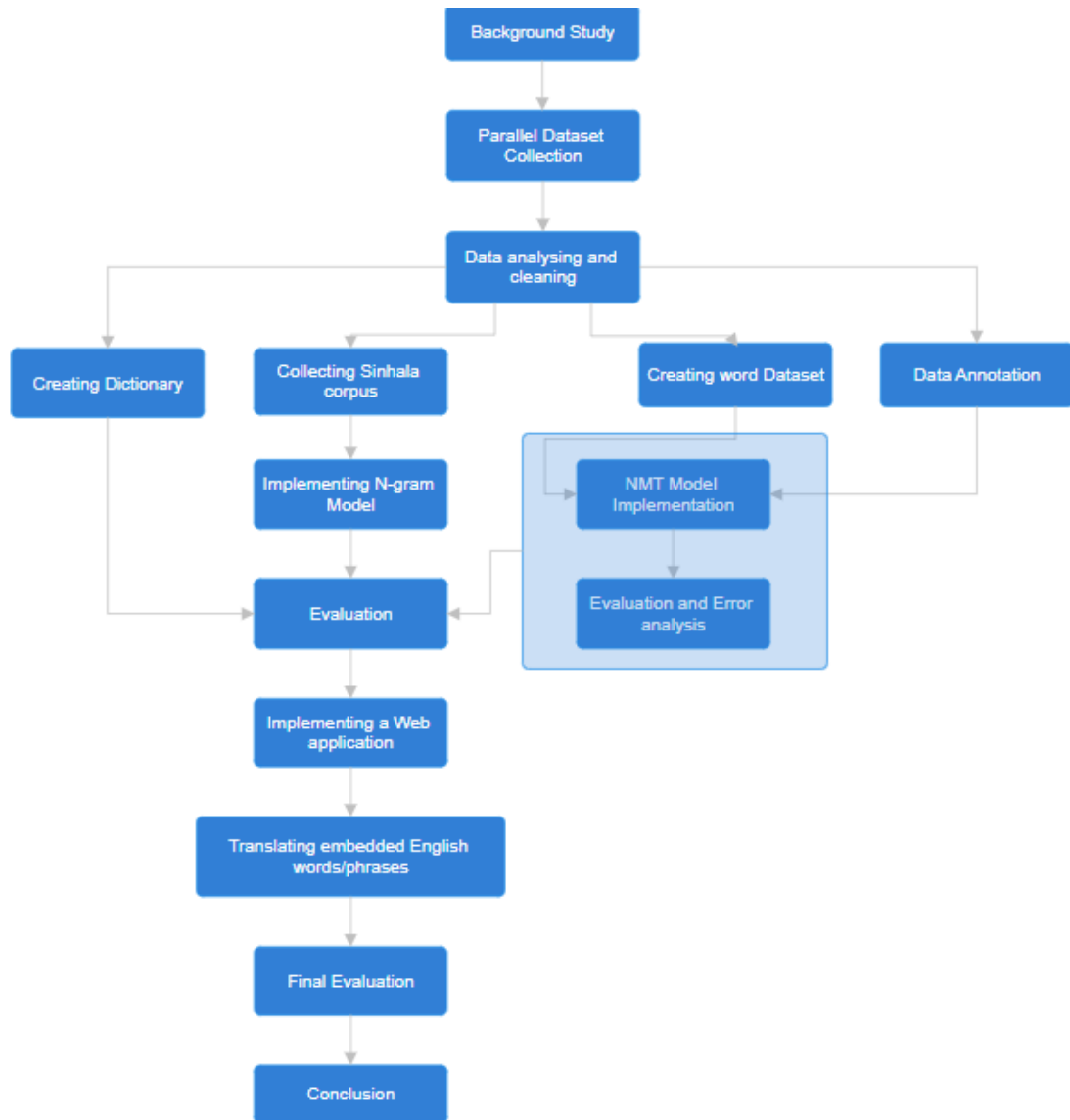


Figure 3.1 High-level visualization of the Design

3.1.2 Data Collection

Parallel data set of Romanized Sinhala and native Sinhala sentences was collected from 3 main sources.

LTRL Data Set

The dataset for this study was developed and owned by The Language Technology Research Laboratory (LTRL) at The School of Computing of The University of Colombo (UCSC). This valuable collection was the result of a collective effort that involved volunteers who added WhatsApp messages. These messages consisting of parallel sentences were then manually annotated in both Romanized Sinhala and native Sinhala.

Dakshina Data Set [1]

The dataset Dakshina developed by Google is a special resource for South Asian languages. It is a comprehensive collection based on Wikipedia texts which is tailored for 12 South Asian languages including both native corpora and Romanized including Sinhala. It provides the dataset to open-source standards and is licensed under CC BY-SA 4.0 to facilitate its accessibility and use in various research projects.

YouTube comments

Another key aspect of our data collection was to source data from YouTube, a platform rich in user-generated content. A custom web application specifically designed to fetch comments from selected YouTube videos was created to achieve this. The process starts by inputting The URL of the video selected in the application. Once this information is entered the application retrieves all comments associated with the video in the best possible way. These comments often cover a mix of native Sinhala and Singlish (Romanized Sinhala) providing a diverse range of linguistic data.

The comments are automatically transferred to Google sheets upon selection. In addition to individual videos the application can aggregate comments from whole YouTube channels further expanding the dataset's scope. The collected data comprising both Singlish and Sinhala comments is processed to generate parallel sentences in both scripts. Figure given below is a screenshot of the custom web application developed for collecting YouTube comments, showcasing its user interface and functionality.



Figure 3.2 Custom web application developed for collecting YouTube comments

Table 3.1 Summary of the datasets

Dataset	Unique Sentences	Unique Romanized Words	Unique Sinhala Words
LTRL	51,326	45,372	35,176
Dakshina	8,446	38,737	32,912
YouTube Comments	7314	29,576	20,821

3.1.3 Data analyzing and cleaning

A significant observation about the transliteration patterns from Sinhala script to Roman script was made by analyzing the dataset. Several people tend to follow different patterns based on their own personal convenience or understanding. Due to this, it was observed in certain situations was the presence of multiple Romanized Sinhala variants corresponding to a single native Sinhala. To illustrate this the following table provides examples in which a native Sinhala word is associated with several Romanized counterparts.

Table 3.2 Example for Sinhala words and their possible Romanized Sinhala variants

Sinhala Word	Possible Romanized Sinhala Words
අහරුචාදා	agaharuwda, angaharuwada, anghruwada, agaharuwada
ලියන්න	lynn, liynn, lianna, liyanna, lynna, liynna, liyann
වෙලාවට	welawata, welawat, wlwt, wlwta, welwt, welwta, welawta, welawt, velavata, welwat

In the data cleaning phase specific steps were taken to ensure the integrity and usability of the collected data. The initial cleaning For the YouTube comment list application included the removal of emojis and special characters from comments.

Similarly, a thorough cleaning process was executed in the datasets obtained from the Language Technology Research Laboratory (LTRL) and the Dakshina dataset. This process not only mirrors the removal of emojis and special characters as in the comments on YouTube but also

extends to the elimination of numbers. Numbers were found either embedded within words or represented by the words themselves. Removing these numerical elements was crucial as they could potentially create a lot of ambiguities to the learning process of the transliteration model. The following table describes some examples for the above scenario.

Table 3.3 Examples for Romanized words embedded with numbers

Romanized Sinhala word	Native Sinhala word
4n	ෆෆෆෆ
4th	ෆෆෆෆ
4to	ෆෆෆෆ
2nd	ෆෆෆෆ

3.1.4 Collecting Sinhala Corpus

The training of the n-gram model was supported by a corpus of Sinhala sentences sourced from several distinct data sources as follows.

- FLORES-200 Dataset [2] - Developed by Facebook AI research this data is optimized as a benchmarking tool for evaluating NLP models in less-resourced languages. It is the result of extensive research aimed at improving model performance in diverse linguistic settings.
- SiClaEn Dataset [3] - Comprising two segments this Dataset includes a Sinhala News Dataset along with the English News Dataset. The latter, which comes from local news outlets like AdaDerana and newsfirst encapsulates a collection of 5221 Sinhala sentences. These sentences cover a range of topics such as politics, science and technology and sport supplying a broad spectrum of formal written Sinhala.
- Short Sentence Dataset [4] - this compilation contains Short Sinhala sentences from a Flickr image dataset. These sentences were formed based on image captions provided by participants tasked with describing 500 different images thus creating a dataset rich in diversity and creativity.

- iv. OSCAR [5] - An acronym for Open Super-large compressed ALMAnaCH corpus OSCAR represents a significant multilingual corpus. It was formulated using the goclassy architecture and was formulated using a combination of sorting and filtering of the Common Crawl corpus in 166 languages including Sinhala.

In addition to this, Sinhala sentences which were used to train the model were also gathered.

3.1.5 Evaluation and error analysis

This evaluation of the transliteration model focused on primarily two aspects, the general accuracy of the back-transliteration process and its context-sensitive nature.

Evaluation Metrics

The performance of the transliteration model was quantitatively assessed using two well-known metrics The score of BLEU and the score of METEOR.

The score of The BLEU is important for machine-translated texts including back-transliteration. It involves comparing the target sequence (ideal output) to the candidate sequence (model output) to determine the degree of similarity. The BLEU score is based on a scale of 0 to 1 where a higher score indicates a high similarity to the target sequence. This metric is focused specifically on precision values providing a quantitative measure of the accuracy of the transliteration model.

The METEOR score complements The BLEU score by ensuring both precision and recall in its calculations therefore enabling a more balanced assessment. METEOR provides an additional evaluation layer particularly useful for assessing the contextual accuracy of the model.

3.1.6 Implementing a web application

A user-friendly web application was developed, serving as the interface for the transliteration system. This application was developed with Angular for the front end and Fast API for the back end ensuring a responsive and efficient user experience.

Functionality and Integration

The primary functionality of the web application lies in its ability to predict native Sinhala sentences from user-typed Romanized Sinhala input (Singlish). Upon completion of the research the aforementioned advanced functionality is achieved through the seamless

integration of various models and components.

The Transliteration model, the core of the application, converts the Romanized Sinhala input into its native Sinhala script. This model based on a GRU network with an Attention mechanism is crucial for the first stage of the prediction process.

To improve the accuracy and fluency of transliterated sentences a N-Gram model is incorporated. The model assists in predicting the most likely sequence of words adding to the outputs a layer of linguistic coherence and context-awareness.

Recognizing the frequently incorporated English words in Singlish the application includes a feature for translating Embedded English words. This translation is facilitated by an integration with the Google Translator API ensuring that the English words in the context of the sentence are appropriately translated into Sinhala.

Also, a comprehensive Dictionary mapping is used. This dictionary generated from the datasets provides a reference for standard transliterations and assists in the ensure consistent and accurate transliteration process.

User Interaction

The app is designed with a focus on user-friendly interface. Users can simply type or paste a Singlish sentence into the application which processes the input through these integrated systems to display the most accurate native Sinhala sentence. This interface demonstrates not only the practical application of research but also provides an accessible tool for users to interact with the transliteration system.

The development of this web application represents a significant step in making research results tangible and user accessible. It illustrates the practical application of complex NLP models in a real-world scenario offering an intuitive platform for users to experience the sophistication of modern language processing technologies.

3.1.7 Translating Embedded English Words / Phrases

A unique challenge in transliterating Singlish sentences which often contain a mix of English and Sinhala is the correct translation of embedded English words or phrases. For this a specialized approach was developed and integrated into the web application.

Handling Suffixes

In this translation process an additional complexity is the presence of specific suffixes that often follow English phrases in Singlish sentences. While part of the colloquial Singlish structure These suffixes can create difficulties in translation. This is intended to identify and omit certain suffixes from English phrases before they are sent to the Google Translator API. This omission is crucial for the achieving accurate and contextually appropriate translations as it prevents the suffixes from altering the intended meaning of English phrases translated into Sinhala.

Integration with the Transliteration System

The translation of these English words or phrases is integrated seamlessly into the overall transliteration process. Once the English segments have been translated into Sinhala, they are reinserted into the original sentence structure. This integration ensures that the final transliterated output is a coherent Sinhala sentence that accurately reflects the mixed language nature of the original Singlish input.

3.2 Implementation

3.2.1 Overview

This Chapter delves into the specifics of the implementation of the research project effectively translating the conceptual designs detailed in Chapter 3 into a fully operational system. The implementation is categorized into several key components, each of which is explained in the following sections.

At the core of the implementation is python selected for its flexibility and powerful ecosystem of libraries instrumental to handling the complexities of natural language processing and web application development. The project strongly relies on the following libraries.

A cornerstone for building and training advanced machine learning models PyTorch provides an intuitive framework for deep learning tasks particularly in the development of transliteration and N-gram models.

- This library is used for its efficient preprocessing of text and its seamless integration with PyTorch facilitating the handling of textual data.
- NumPy Essential for high-performance operations on multi-dimensional arrays and

matrices NumPy is a fundamental tool in the processing and manipulation of data.

- The award-winning Pandas based database is used for analyzing and managing the data.
- Natural Language Toolkit (NLTK) and spacy These libraries are essential for Natural Language processing tasks including text manipulation and linguistic analysis.
- Googletrans (4.0.0-rc1) led by Google Translate API for its ability to Translate English phrases within Singlish sentences.
- Matplotlib Used for the visualization of data aiding in the analysis and presentation of results.
- This forms the backbone of the web application with FastAPI providing a robust framework for building APIs and Uvicorn acting as the ASGI server.
- Pydantic is used for the validation of data and settings especially in the context of FastAPI.

3.2.2 NMT Model

Data Preprocessing

A crucial step in the preparation of the dataset for the transliteration process involved normalizing both Singlish and Sinhala sentences. To achieve this a unified function was developed to normalize text. This function standardizes the input text by several stages enhancing consistency and reducing variability in the dataset. Following is a breakdown of the function.

- i. The function starts by converting all text into lowercase and stripping leading or trailing whitespace. This ensures uniformity in the letter case over all texts.
- ii. punctuation and Symbol replacement dot colons and similar punctuations are replaced by spaces. This step is crucial to avoiding interrupted sentences caused by punctuation marks.
- iii. removing other punctuation marks the function then removes all remaining punctuation marks defined by python string punctuation further cleaning the text.
- iv. The next step involves the storage of Roman alphabet characters (a-z, A-Z) and Sinhala script characters (Unicode range 0D80 to 0DFF). This filter excludes numbers and other non-relevant characters focusing the dataset on linguistically relevant elements.
- v. Any instance of multiple consecutive spaces is reduced to a single space ensuring consistency in word separation throughout the text.

By applying this normalized text function to the dataset we ensured that the input of our models was clean, standardized, and free of common textual anomalies that could hinder the accuracy of the transliteration.

```
def normalize_text(text):
    text = text.lower().strip()

    # Replace dots and commas with spaces
    text = re.sub(pattern: r'[.,:]+', repl: ' ', text)

    # Remove other punctuations
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Remove numbers and retain only Roman and Sinhala letters
    # Sinhala Unicode block ranges from 0D80 to 0DFF
    text = re.sub(pattern: r'^a-zA-Z\u0D80-\u0DFF\s', repl: '', text)

    # Replace multiple spaces with a single space
    text = re.sub(pattern: r'\s+', repl: ' ', text).strip()
    return text
```

Figure 3.3 Function for Normalizing text

Transliteration Unit Algorithm

A pivotal element in the transliteration process is the Transliteration Unit (TU) algorithm, which plays a crucial role in breaking down words into syllables. This algorithm is a key component in converting Romanized Sinhala into native script, ensuring each syllable is accurately and efficiently processed.

○ **Algorithm Development and Influence:**

The development of our TU algorithm is heavily influenced by the model proposed by Nanayakkara et al (2023). This model forms the basis for our approach, particularly in handling the complexities of the Sinhala script and its transliteration from Romanized form. The algorithm focuses on the intricate patterns of consonants and vowels in a word, a critical aspect for accurate transliteration.

○ **Improvements and Specifics:**

Building on the foundational model, significant improvements were made, particularly in the handling of the consonant "h." The original algorithm had limitations in adequately addressing the diverse roles of "h" in Sinhala, especially when used as a suffix to create specific sounds. Our algorithm introduces new logic to overcome these challenges, ensuring a more nuanced and accurate transliteration.

```

def tokenize_singlish_word(singlish_word):
    singlish_word = singlish_word.lower()

    vowels = ['a', 'e', 'i', 'o', 'u']
    # considering ch, sh, th, dh
    special_consonant_suffix = ['c', 's', 't', 'd']

    word = remove_unnecessary_repeats(singlish_word)
    buf = word[0]
    final_list = []
    for i in range(1, len(word)):
        char = word[i]
        if char in vowels:
            if word[i - 1] not in vowels or word[i - 1] == char:
                buf += char
            else:
                final_list.append(buf)
                buf = char

        elif char == 'h' and word[i - 1] in special_consonant_suffix:
            buf += char
        else:
            final_list.append(buf)
            buf = char

    final_list.append(buf)
    final_list.append("$")

    return " ".join(final_list)

```

Figure 3.4 Function for tokenizing Romanized Sinhala words

Additionally, the algorithm was further refined to enhance its performance in recognizing specific Sinhala sounds, such as "ඔ", "ඹ", and "ඵ", which had shown poor recognition in the original model. The inclusion of the Zero-width-joiner and “Anuswargaya” ("ෆ") was another significant enhancement, addressing previous gaps in the algorithm.

In implementation, the TU algorithm utilizes defined patterns of consonant, vowel, and special consonant suffix combinations. These combinations are key to identifying TUs accurately. Special consonant suffixes refer to those consonants that can generate new sounds when combined with other consonants. The algorithm uses a set of rules, optimized through testing, to select the best-performing combinations for transliteration.

The revised and improved TU algorithm demonstrates enhanced efficiency in breaking down Romanized Sinhala texts into TUs.


```

import torch.nn as nn

11 usages new *
class EncoderRNN(nn.Module):
    new *
    def __init__(self, input_size, hidden_size, dropout_p=0.1):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)
        self.dropout = nn.Dropout(dropout_p)

    new *
    def forward(self, _input):
        embedded = self.dropout(self.embedding(_input))
        output, hidden = self.gru(embedded)
        return output, hidden

```

Figure 3.6 Encoder Class

○ Bahdanau Attention

The Bahdanau Attention module is a key component of the decoder. It calculates attention weights and produces a context vector by focusing on specific parts of the input sequence:

- i. Attention Mechanism - Implements the attention mechanism using linear layers (W_a , U_a , and V_a) and a tanh activation function followed by a softmax to derive the weights.
- ii. Context Vector calculation - The attention weights are used to create a context vector which represents the focal part of the input sequence.

```

class BahdanauAttention(nn.Module):
    new *
    def __init__(self, hidden_size):
        super(BahdanauAttention, self).__init__()
        self.Wa = nn.Linear(hidden_size, hidden_size)
        self.Ua = nn.Linear(hidden_size, hidden_size)
        self.Va = nn.Linear(hidden_size, out_features=1)

    new *
    def forward(self, query, keys):
        scores = self.Va(torch.tanh(self.Wa(query) + self.Ua(keys)))
        scores = scores.squeeze(2).unsqueeze(1)

        weights = F.softmax(scores, dim=-1)
        context = torch.bmm(weights, keys)

        return context, weights

```

Figure 3.7 Bahdanau Attention Class

- **AttnDecoderRNN**

The AttnDecoderRNN class which is the heart of the decoder uses attention mechanism to improve the transliteration process:

- Embedding and Dropout Layers - Similar to the encoder the embedding layer is followed by a Dropout in an embedding layer.
- GRU Layer - This layer takes a combination of the embedded input and the context vector from the attention mechanism, so the model focuses on relevant parts of the input while generating each token in the output sequence.
- Output layer - A linear layer combines the output of the gru with the output vocabulary to determine the size of the output.

The forward method is able to manage the decoding process step by step either by using teacher force (using the actual target tokens as the next input) or by re-feeding the predicted outputs back.

```
class AttnDecoderRNN(nn.Module):
    new *
    def __init__(self, hidden_size, output_size, dropout_p=0.1):
        super(AttnDecoderRNN, self).__init__()
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.attention = BahdanauAttention(hidden_size)
        self.gru = nn.GRU(2 * hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, encoder_outputs, encoder_hidden, target_tensor=None, max_length=None):
        batch_size = encoder_outputs.size(0)
        decoder_input = torch.empty(batch_size, 1, dtype=torch.long, device=device).fill_($SOS_token)
        decoder_hidden = encoder_hidden
        decoder_outputs = []
        attentions = []

        for i in range(max_length):
            decoder_output, decoder_hidden, attn_weights = self.forward_step(
                decoder_input, decoder_hidden, encoder_outputs
            )
            decoder_outputs.append(decoder_output)
            attentions.append(attn_weights)

            if target_tensor is not None:
                # Teacher forcing: Feed the target as the next input
                decoder_input = target_tensor[:, i].unsqueeze(1) # Teacher forcing
            else:
                # Without teacher forcing: use its own predictions as the next input
                _, top1 = decoder_output.topk(1)
                decoder_input = top1.squeeze(-1).detach() # detach from history as input

        decoder_outputs = torch.cat(decoder_outputs, dim=1)
        decoder_outputs = F.log_softmax(decoder_outputs, dim=-1)
        attentions = torch.cat(attentions, dim=1)

        return decoder_outputs, decoder_hidden, attentions
```

Figure 3.8 Decoder Class

N-gram Model

The n-gram model is another key component of the transliteration system which is specifically designed to predict the most probable sentence construction in Sinhala from various potential mutations. The model combines the trigram (three-word sequences) and the bigram (two-word sequences) approaches to improve the prediction accuracy.

○ Model Generation Process

The generation of these n-gram models follows a structured process.

i. Preparing the Corpus:

- a. The corpus is sourced from an Excel sheet ensuring a substantial and diverse dataset.
- b. Each sentence is normalized using the function `normalize text` which provides consistency and standardization in the dataset.

ii. Trigram Model:

- a. The tokenized corpus is processed to generate trigrams with each sentence tokenized into words and padded with the starting and ending `s` tokens.
- b. Trigrams are counted using the python `Counter` class resulting in a frequency distribution of all possible trigrams in the corpus. These trigram counts are serialized then into a JSON file `trigram model` making the model easily accessible and reusable.

iii. Bigram Model

- a. Similar bigrams are generated and counted for the tokenized corpus and are likewise. The bigram counts are also serialized and saved in a JSON file `bigram_model.json`.

○ Application of the Models

In the system Both the trigram and bigram are used to predict the most contextually appropriate sequence of words in a given sentence. These models consider the frequency of the word sequences in the training corpus enabling the system to choose sentences linguistically coherent and closer to natural Sinhala syntax.

○ Advantages of n-grams

The n-gram models add a layer of context sensitivity to the predictions by analyzing word sequences which is particularly crucial in a language with complex syntactic structures such as

Sinhala.

The dual approach of using both trigrams and bigrams allows the system to balance between more contextually rich predictions (trigrams) and flexible less constrained predictions (bigrams).

The integration of these n-gram models into the transliteration system marks a significant enhancement in its ability to generate accurate and contextually appropriate Sinhala sentences thus improving the overall effectiveness of them.

3.2.3 Web Application with translation of embedded English phrases.

Web Application Design

The web application for our transliteration system is built using Angular 17, offering a user-friendly and interactive interface. Key features of the application include:

Text Area - Users can input sentences containing English phrases embedded in Sinhala directly into a designated text area. This allows for easy and convenient entry of the text that needs to be transliterated.

File Upload Option - A file picker feature enables users to upload a .txt file containing a list of English words that require translation. This functionality caters to users who prefer batch processing of words, enhancing the application's usability.

Translate Button - Upon clicking the 'Translate' button, the application processes the input and displays the best possible transliterated sentence below the input area.

Backend Processing with Python FastAPI

The backend of the application, powered by Python FastAPI, handles the processing of the user input with several critical steps:

- 1. Preprocessing** - The submitted sentence and the English word list undergo preprocessing tasks. This includes checking for the presence of English phrases within the Sinhala sentences.
- 2. Translation of English Phrases** - If any English phrases are detected, they are translated using the Google Translator API. This step is crucial for maintaining the semantic integrity of the sentence. Common suffixes following English words are recognized and translated

appropriately into their English counterparts like 'a', 'the', 'in', 'from', etc. Following table shows some examples on how the known Sinhala suffixes are replaced by English prefixes.

Table 3.4 Examples for possible English conversions

Romanized text with English	Possible English Phrase
Doctor kenek	A Doctor
Class eka	The class
Universities walata	To universities
Play karanakan	Until play

3. **Dictionary Mapping and Transliteration Model** - The system then generates all possible sentences based on the dictionary mapping and the transliteration model. This step involves converting the preprocessed and translated input into potential Sinhala sentences.
4. **Selection of the Best Sentence using N-Gram Model** - Finally, these potential sentences are fed into an n-gram model. The n-gram model evaluates the sentences based on linguistic correctness and context, selecting the best possible sentence. This sentence is then presented to the user as the final output.

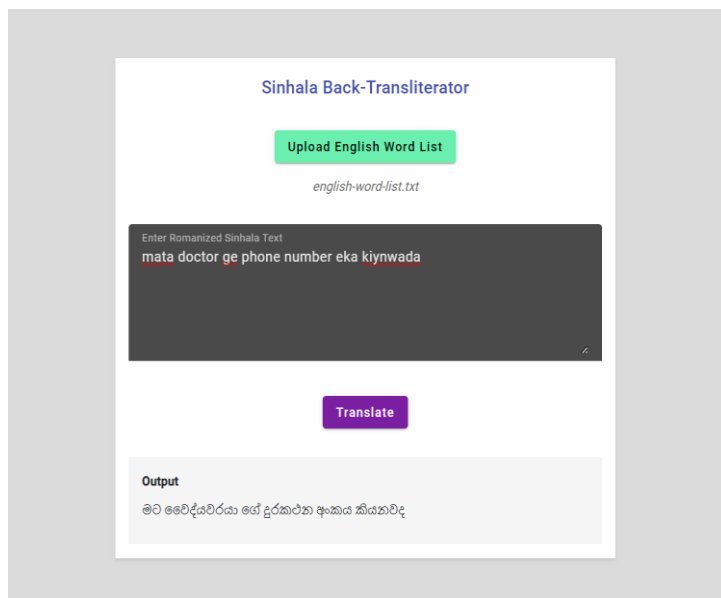


Figure 3.9. User Interface of the developed web application

CHAPTER 4 - EVALUATION AND RESULTS

4.1 Evaluation Approach

The evaluation plan for this research project was designed to employ a combination of experimental approaches and modern language evaluation methods in order to ensure a comprehensive assessment of the model's performance. The measurables selected were BLEU and METEOR.

The evaluation process was to cover various transliteration and translation scenarios focusing on Singlish transliteration under different conditions. This diverse approach was intended to test the adaptability and accuracy of the model across different linguistic contexts.

4.2 Transliteration Unit Breaking Algorithm Evaluation

In the context of improving the transliteration of Romanized Sinhala, this study has further refined an algorithm originally proposed by Hettige et al. (2007), which was also utilized in recent work by Nanayakkara et al. This modification has significantly enhanced the accuracy of tokenizing Romanized Sinhala words.

As part of this research, an extension of the work conducted by Nanayakkara et al. was undertaken. In their study, they reported an accuracy of approximately 90% in tokenizing a set of 1000 words. In the implementation phase of our research, as discussed in the previous sections, this algorithm was meticulously developed and tested. The evaluation process involved selecting random samples of 1000 words, examined in batches of 10, which included complex words from the Dakshina Dataset. The accuracy observed in this evaluation was approximately 95%. However, it was noted that the algorithm faced challenges in accurately tokenizing some complex words.

One notable issue pertains to the Singlish transliteration unit (syllable) breaking algorithm, particularly with words that include characters such as "෧෧෧", "෧෧", and "෧a" which leads to a mismatch in the count of Sinhala and Singlish syllables for each word. The following table will describe how the syllable count mismatches.

Table 4.1 TU Breaking Examples

Singlish Word	Words after breaking into TUs
waidya (වෛද්‍ය)	wa i d ya (වෛ ද් ය)
Maithree (මෛත්‍රී)	ma i th ree (මෛ ත් රී)
irthuwa (ඔත්තුව)	i r thu wa (ඔ තු ව)
krura (කරු)	k ru ra (ක රු)
oushadha (ඖෂධ)	o u sha dha (ඖ ෂ ධ)

4.3 Transliteration Model Evaluation

The GRU model was rigorously tested at different linguistic levels: word level, TU (Transliteration Unit) level, and character level. The training and evaluation of the model utilized a comprehensive word dataset, which was split in a 9:1 ratio for the purposes of training and testing.

To gauge the effectiveness of the GRU model, we employed the BLEU (Bilingual Evaluation Understudy) and METEOR (Metric for Evaluation of Translation with Explicit Ordering) scores. It underwent training for a specified number of epochs, achieving significant categorical accuracy and a low loss rate in both the training data and the validation split.

Table 4.2 Evaluation Matrix Scores

Evaluation Matrix	Word Level score	TU level score
BLEU	0.1643	0.1618
METEOR	0.1225	0.2661

4.4 N-gram and Embedded English Translation Evaluation.

The evaluation of the n-gram model and the English Translation Evaluation was conducted across six distinct approaches, each with varying configurations of vowel presence and English word inclusion in Singlish transliteration. The findings are as follows:

- **Singlish Transliteration with Vowels between Words (Without Original English Words)**

Higher accuracy was observed in this approach, suggesting that including vowels between Singlish words improves the model's ability to accurately transliterate the text.

- **Singlish Transliteration without Vowels between Words (Without Original English Words)**

A slight reduction in accuracy compared to the approach with vowels, indicating the model's dependency on vowel presence for optimal transliteration.

- **Singlish Transliteration with Reduced Vowels between Words (Without Original English Words)**

Similar to the approach without vowels, reduced vowels also led to a marginal decrease in accuracy, further affirming the importance of full vowel inclusion for better transliteration results.

- **Singlish Transliteration with Vowels between Words (With Original English Words)**

This approach also yielded high accuracy, comparable to the scenario without original English words, demonstrating the model's proficiency in handling Singlish transliteration with complete vowel presence, irrespective of the inclusion of English words.

- **Singlish Transliteration without Vowels between Words (With Original English Words)**

Similar to its counterpart without English words, this approach saw a decrease in accuracy due to the absence of vowels, reinforcing the trend observed in the earlier scenarios.

- **Singlish Transliteration with Reduced Vowels between Words (With Original English Words)**

Consistent with the pattern, this method also resulted in lower accuracy than the approaches with full vowel presence, highlighting the same challenges posed by reduced vowel inclusion.

- **Impact of Known English Words:**

It was noted that the presence of known English words in the transliterated sentences did not significantly impact the overall accuracy. This is attributed to the fact that the translation of these English words was handled independently from the transliteration process and the language model. The model's ability to effectively separate and process English translations alongside Sinhala transliterations illustrates its robustness in handling mixed-language contexts.

CHAPTER 05 - CONCLUSION AND FUTURE WORK

This research project was aimed at developing and implementing a neural machine translation system for Sinhala transliteration while addressing the challenges in transliterating Romanized Sinhala and embedded English into native Sinhala script.

A user-friendly web application was developed through which the advanced NLP techniques such as GRU models with Bahdanau Attention and n-gram modeling can be used to predict the most probable Sinhala sentence corresponding to the user's Singlish input.

The study was able to evaluate the effectiveness of GRU models in transliteration tasks, and ability to handle the complexities of Sinhala language processing and Integration of Google Translator API demonstrated effective translation of embedded English phrases enhancing the accuracy of transliteration. Moreover, Real-time user interactions and transliteration duties were made possible by the web application's implementation, which confirmed the research's practical relevance.

Furthermore, this study was able to contribute to the existing literature by addressing the lack of comprehensive research in Sinhala transliteration using NMT. The system's ability to recognize and translate English phrases in texts with mixed codes is an essential component of precise transliteration of this model. Beyond just developing a model, the research produced a whole system, demonstrating an important advancement over previous studies.

Despite the above, the research encountered some limitations as well. Large dataset training imposed a significant computing burden, which was one of the project's main challenges. Neural machine translation models that use sophisticated training methods, such as transformer models and GRUs, frequently need a large amount of computing resources, such as processing power, RAM, and expensive GPUs.

The amount of data that could be used for training was limited in this study due to these computational limitations. The amount of data that could be trained was limited by the computer resources available, even though larger datasets might potentially increase the model's robustness and performance, particularly when handling a variety of linguistic scenarios. The model's power to generalize over less common phrases or more complicated sentence patterns may be impacted by this constraint.

The findings of this project pave the way for several future research directions. Future research could incorporate larger and more diverse datasets. This could increase the model's credibility and accuracy by enabling it to handle a larger variety of phrases and linguistic subtleties.

The introduction of transformer architectures like BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pretrained Transformer), and T5 (Text-To-Text Transfer Transformer) has revolutionized various NLP tasks. Applying these models to the domain of transliteration could yield substantial improvements in accuracy and efficiency. Their ability to process entire sequences of text simultaneously, as opposed to the sequential processing by RNNs and GRUs, makes them particularly well-suited for complex transliteration tasks involving code-mixed texts.

At the moment, the system uses user-selected words and a rule-based methodology to translate word phrases. Future studies may focus on developing more sophisticated techniques, which could incorporate machine learning algorithms for the automatic identification and translation of a wider array of word phrases. With this enhancement, the rule-based method's drawbacks would be eliminated and a more dynamic and thorough translation of texts in mixed languages would be possible.

APPENDIX A

CODE LISTINGS

A.1 required packages

```
--index-url https://download.pytorch.org/whl/cu121
torch==2.2.0+cu121
torchttext==0.17.0+cpu

numpy
matplotlib
pandas
google-api-python-client~=2.114.0
google-auth-httpplib2
google-auth-oauthlib
scikit-learn
nltk
spacy
googletrans==4.0.0-rc1

fastapi
uvicorn[standard]
pydantic
```

A.2 Data Loader

```
from apiclient import discovery
from google.oauth2 import service_account

from utils.Lang import Lang, Lang2
from utils.tokenizer import normalize_text, tokenize_singlish_word, tokenize_sinhala_word

4 usages new*
def read_excel_file_data():
    try:
        google_sheet_scopes = ["https://www.googleapis.com/auth/drive", "https://www.googleapis.com/auth/drive.file",
                               "https://www.googleapis.com/auth/spreadsheets"]
        secret_file = os.path.join(os.getcwd(), '../utils/service_account.json')

        spreadsheet_id = '1GvWUaWuXLZm42ftcp7LUPCF3AEoNSMn-eYTNCJHeYM'
        range_name = 'sample data!A1:A800000'

        credentials = service_account.Credentials.from_service_account_file(secret_file, scopes=google_sheet_scopes)
        service = discovery.build(serviceName='sheets', version='v4', credentials=credentials)

        result = service.spreadsheets().values().get(
            spreadsheetId=spreadsheet_id, range=range_name).execute()
        rows = result.get('values', [])
        return rows

    except OSError as e:
        print(e)
```

```
def read_langs():
    # sentence_pairs = read_excel_file_data()
    sentence_pairs = load_tsv('data/data-set.tsv')
    print("Read %s sentence pairs" % len(sentence_pairs))

    # Filter out sentences with more than 50 words
    sentence_pairs = [pair for pair in sentence_pairs if
                      len(pair[0].split()) <= 10 and len(pair[1].split()) <= 10]

    train_data, eval_data = split_data(sentence_pairs)

    # Save eval_data to a file
    with open('eval_data.tsv', 'w', newline='', encoding='utf-8') as f:
        writer = csv.writer(f, delimiter='\t')
        writer.writerows(eval_data)

    random_count = 20000
    if len(train_data) > random_count:
        train_data = random.sample(train_data, random_count) # take random sentences

    print("Trimmed to %s train sentence pairs" % len(train_data))
    print("Counting words...")

    input_sentences = Lang("singlish")
    output_sentences = Lang("sinhala")

    return input_sentences, output_sentences, train_data
```

A.3 Data Preprocessing Codes

```
def remove_unnecessary_repeats(word):
    prev = word[0]
    repeat_count = 1
    ans = word[0]
    for i in range(1, len(word)):
        if word[i].isdigit():
            ans += word[i]
            prev = word[i]
            continue

        if word[i] == prev:
            repeat_count += 1
        else:
            repeat_count = 1

        if repeat_count < 3:
            ans += word[i]
            prev = word[i]

    return ans
```

```
def normalize_text(text):
    text = text.lower().strip()

    # Replace dots and commas with spaces
    text = re.sub(pattern: r'[.,:]+', repl: ' ', text)

    # Remove other punctuations
    text = text.translate(str.maketrans('', '', string.punctuation))

    # Remove numbers and retain only Roman and Sinhala letters
    # Sinhala Unicode block ranges from 0D80 to 0DFF
    text = re.sub(pattern: r'^a-zA-Z\u0D80-\u0DFF\s', repl: '', text)

    # Replace multiple spaces with a single space
    text = re.sub(pattern: r'\s+', repl: ' ', text).strip()
    text = remove_unnecessary_repeats(text)
    return text
```


A.4 Encoder Decoder Model

```
import torch.nn as nn

11 usages new *
class EncoderRNN(nn.Module):
    new *
    def __init__(self, input_size, hidden_size, dropout_p=0.1):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)
        self.dropout = nn.Dropout(dropout_p)

    new *
    def forward(self, _input):
        embedded = self.dropout(self.embedding(_input))
        output, hidden = self.gru(embedded)
        return output, hidden
```

```
class BahdanauAttention(nn.Module):
    new *
    def __init__(self, hidden_size):
        super(BahdanauAttention, self).__init__()
        self.Wa = nn.Linear(hidden_size, hidden_size)
        self.Ua = nn.Linear(hidden_size, hidden_size)
        self.Va = nn.Linear(hidden_size, out_features=1)

    new *
    def forward(self, query, keys):
        scores = self.Va(torch.tanh(self.Wa(query) + self.Ua(keys)))
        scores = scores.squeeze(2).unsqueeze(1)

        weights = F.softmax(scores, dim=-1)
        context = torch.bmm(weights, keys)

        return context, weights
```

```
class AttnDecoderRNN(nn.Module):
    new *
    def __init__(self, hidden_size, output_size, dropout_p=0.1):
        super(AttnDecoderRNN, self).__init__()
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.attention = BahdanauAttention(hidden_size)
        self.gru = nn.GRU(2 * hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)
        self.dropout = nn.Dropout(dropout_p)
```

```

def forward(self, encoder_outputs, encoder_hidden, target_tensor=None, max_length=None):
    batch_size = encoder_outputs.size(0)
    decoder_input = torch.empty(batch_size, 1, dtype=torch.long, device=device).fill_(SOS_token)
    decoder_hidden = encoder_hidden
    decoder_outputs = []
    attentions = []

    for i in range(max_length):
        decoder_output, decoder_hidden, attn_weights = self.forward_step(
            decoder_input, decoder_hidden, encoder_outputs
        )
        decoder_outputs.append(decoder_output)
        attentions.append(attn_weights)

        if target_tensor is not None:
            # Teacher forcing: Feed the target as the next input
            decoder_input = target_tensor[:, i].unsqueeze(1) # Teacher forcing
        else:
            # Without teacher forcing: use its own predictions as the next input
            _, top1 = decoder_output.topk(1)
            decoder_input = top1.squeeze(-1).detach() # detach from history as input

    decoder_outputs = torch.cat(decoder_outputs, dim=1)
    decoder_outputs = F.log_softmax(decoder_outputs, dim=-1)
    attentions = torch.cat(attentions, dim=1)

    return decoder_outputs, decoder_hidden, attentions

```

A.5 Training

```

def train(train_data_loader, encoder, decoder, n_epochs, learning_rate=0.0008,
          print_every=100, plot_every=100, save_every=10):
    start = time.time()
    plot_losses = []
    print_loss_total = 0 # Reset every print_every
    plot_loss_total = 0 # Reset every plot_every

    encoder_optimizer = optim.Adam(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.Adam(decoder.parameters(), lr=learning_rate)
    criterion = nn.NLLLoss()

    for epoch in range(1, n_epochs + 1):
        loss = train_epoch(train_data_loader, encoder, decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss

        if epoch % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0
            print('%s (%d %d%%) %.4f' % (timeSince(start, epoch / n_epochs),
                                         epoch, epoch / n_epochs * 100, print_loss_avg))

        if epoch % plot_every == 0:
            plot_loss_avg = plot_loss_total / plot_every
            plot_losses.append(plot_loss_avg)
            plot_loss_total = 0

        if epoch % save_every == 0:
            checkpoint = {
                'epoch': epoch,
                'encoder_state_dict': encoder.state_dict()
            }

```

```

def train_epoch(dataloader, encoder, decoder, encoder_optimizer,
                decoder_optimizer, criterion):
    total_loss = 0
    for data in dataloader:
        input_tensor, target_tensor = data

        max_length = target_tensor.size(1) # Determine the sequence length for the current batch

        encoder_optimizer.zero_grad()
        decoder_optimizer.zero_grad()

        encoder_outputs, encoder_hidden = encoder(input_tensor)
        decoder_outputs, _, _ = decoder(encoder_outputs, encoder_hidden, target_tensor, max_length)

        loss = criterion(
            decoder_outputs.view(-1, decoder_outputs.size(-1)),
            target_tensor.view(-1)
        )
        loss.backward()

        encoder_optimizer.step()
        decoder_optimizer.step()

        total_loss += loss.item()

    return total_loss / len(dataloader)

```

A.6 Evaluate

```

def evaluate(encoder, decoder, sentence, input_lang, output_lang):
    with torch.no_grad():
        input_tensor = tensorFromSentence(input_lang, sentence)
        max_length = input_tensor.size(1) # Determine the sequence length for the current sentence

        encoder_outputs, encoder_hidden = encoder(input_tensor)
        decoder_outputs, decoder_hidden, decoder_attn = decoder(encoder_outputs, encoder_hidden, max_length=max_length)

        _, top1 = decoder_outputs.topk(1)
        decoded_ids = top1.squeeze()

        decoded_words = []
        for idx in decoded_ids:
            if idx.item() == EOS_token:
                decoded_words.append('<eos>')
                break
            decoded_words.append(output_lang.index2letter[idx.item()])
        return decoded_words, decoder_attn

```


A.7 Web API

```
@app.post("/translate")
def translate(request: TranslationRequest):
    input_sentence = request.text
    english_word_list = set(request.word_list) # Convert to set for efficiency

    # Find and translate English phrases
    english_phrases_translations = find_english_words_and_phrases(input_sentence, english_word_list)

    translated_parts = {original: translator.translate(translation, dest='si').text
                        for original, translation in english_phrases_translations}

    # Process each word in the sentence
    input_words = input_sentence.split()
    sentence_translations = []
    i = 0
    while i < len(input_words):
        # Extract the current sequence of words for comparison
        current_sequence = ' '.join(input_words[i:])

        # Check if the current sequence starts with an identified English phrase
        phrase_translated = False
        for original_phrase, translated_phrase in translated_parts.items():
            if current_sequence.startswith(original_phrase):
                sentence_translations.append([translated_phrase])
                i += len(original_phrase.split()) # Move index past this phrase
                phrase_translated = True
                break
```

```
        # If not part of a translated phrase, process normally
        if not phrase_translated:
            word = input_words[i]
            if word in word_mapping:
                translations = word_mapping[word]
            else:
                translations = [predict_with_model(word)]
            sentence_translations.append(translations)
            i += 1

    # Generate all possible sentence combinations
    all_combinations = list(product(*sentence_translations))
    translated_sentences = [' '.join(combo) for combo in all_combinations]

    # Calculate probabilities using n-gram models
    best_sentence = None
    max_probability = 0
    for sentence in translated_sentences:
        probability = calculate_ngram_probability(sentence, trigram_model) # You can switch to bigram_model
        if probability > max_probability:
            max_probability = probability
            best_sentence = sentence

    return {"translated": best_sentence}
```

REFERENCES

- Ameur, M.S.H., Meziane, F., and Guessoum, A. (2017). 'Arabic Machine Transliteration Using an Attention-Based Encoder-Decoder Model'. *Procedia Computer Science*, 117, pp.287-297.
- Ansari, M.Z., Ahmad, T., Beg, M.M.S., and Ahmad, F. (2022). 'Hindi to English Transliteration Using Multilayer Gated Recurrent Units'. *Indonesian Journal of Electrical Engineering and Computer Science*, 27(2), pp.1083-1090.
- Athukorala, M.U. and Sumanathilaka, D.K., 2022, February. "Swa Bhasha: Message-Based Singlish to Sinhala Transliteration," In *Proceedings of the International Conference on Innovations in Info-business and Technology* (Vol. 9).
- Banerjee, S. and Lavie, A., 2005, June. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization* (pp. 65-72).
- Bies, A., Song, Z., Maamouri, M., Grimes, S., Lee, H., Wright, J., Strassel, S., Habash, N., Eskander, R. and Rambow, O., 2014, October. Transliteration of arabizi into arabic orthography: Developing a parallel annotated arabizi-arabic script sms/chat corpus. In *Proceedings of the EMNLP 2014 workshop on Arabic natural language processing (ANLP)* (pp. 93-103)
- Cho, K., Van Merriënboer, B., Bahdanau, D. and Bengio, Y., 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). 'Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation', *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
- De Silva, A.D., 2021. Singlish to sinhala converter using machine learning (Doctoral dissertation).
- Department of Census and Statistics Sri Lanka, 2012. Percentage of population aged 10 years and over in major ethnic groups by district and ability to speak Sinhala, Tamil and English languages[online]. Available at: <http://www.statistics.gov.lk/Population/StaticalInformation/CPH2011/CensusPopulationHousing2012-FinalReport> (p.128)
- Deselaers, T., Hasan, S., Bender, O. and Ney, H., 2009, March. A deep learning approach to machine transliteration. In *Proceedings of the Fourth Workshop on Statistical Machine Translation* (pp. 233-241).

- Englebretson, R. and Genetti, C., 2005. Santa Barbara papers in linguistics: Proceeding from the workshop on Sinhala linguistics. Santa Barbara, CA: Department of Linguistics at the University of California, Santa Barbara, 1, p.3.
- Grundkiewicz, R. & Heafield, K. (2018). 'Neural Machine Translation Techniques for Named Entity Transliteration'. Available at: <http://workshop.colips.org/news2018>, pp. 89–94.
- Hailu, G. and Josan, G.S., 2017. RULE BASED APPROACH FOR TRANSLITERATION OF ENGLISH TO TIGRIGNA. *International Journal of Multidisciplinary Educational Research*, 6(7), p.205.
- Hettiarachchi, N., Weerasinghe, R. and Pushpanda, R., 2020, November. Detecting hate speech in social media articles in romanized sinhala. In 2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer) (pp. 250-255). IEEE.
- Hettige, B. and Karunananda, A.S., 2007, August. Transliteration system for English to Sinhala machine translation. In 2007 International Conference on Industrial and Information Systems (pp. 209-214). IEEE.
- Liwera, W.M.P. and Ranathunga, L., 2020, December. Combination of trigram and rule-based model for singlish to sinhala transliteration by focusing social media text. In 2020 From Innovation to Impact (FITI) (Vol. 1, pp. 1-5). IEEE.
- Jozefowicz, R., Zaremba, W. and Sutskever, I., 2015, June. An empirical exploration of recurrent network architectures. In International conference on machine learning (pp. 2342-2350). PMLR.
- Mutal, J., Volkart, L., Bouillon, P., Girletti, S. and Estrella, P., 2019, September. Differences between SMT and NMT output-a translators' point of view. In Proceedings of the Human-Informed Translation and Interpreting Technology Workshop (HiT-IT 2019) (pp. 75-81).
- Nanayakkara, R., Nadungodage, T. & Pushpananda, R. (2023). 'Context Aware Back-Transliteration from English to Sinhala', pp. 051–056. doi: 10.1109/ictcr58063.2022.10024072.
- Oh, J.H. and Choi, K.S., 2005. Machine learning based English-to-Korean transliteration using grapheme and phoneme information. *IEICE transactions on information and systems*, 88(7), pp.1737-1748.
- Papineni, K., Roukos, S., Ward, T. and Zhu, W.J., 2002, July. Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting of the Association for Computational Linguistics (pp. 311-318).
- Pushpananda, R., Weerasinghe, R. and Niranjana, M., 2014, November. Sinhala-tamil machine translation: Towards better translation quality. In Proceedings of the Australasian Language Technology Association Workshop 2014 (pp. 129-133).
- Roark, B., Wolf-Sonkin, L., Kirov, C., Mielke, S.J., Johnny, C., Demirsahin, I. and Hall, K., 2020. Processing South Asian languages written in the Latin script: the Dakshina dataset. arXiv preprint arXiv:2007.01176.

- Salaev, U., Kuriyozov, E. and Gómez-Rodríguez, C., 2022. A machine transliteration tool between Uzbek alphabets. arXiv preprint arXiv:2205.09578.
- Shalini, R.M.M. and Hettige, B., 2017, October. Dictionary based machine translation system for pali to sinhala. In SLAAI-International Conference on Artificial Intelligence (p. 23).
- Sumanathilaka, T.G.D.K., Weerasinghe, R. and Priyadarshana, H.Y.P.P., 2022, February. Romanized Sinhala to Sinhala Transliteration using a Hybrid Approach. In Proceedings of the International Conference on Innovations in Info-business and Technology (Vol. 108).
- Sumanathilaka, T.G.D.K., Weerasinghe, R. and Priyadarshana, H.Y.P.P., 2023. Sinhala word suggestion algorithm for ad hoc Romanized Sinhala transliterations using a Trie (p. 53).
- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). 'Sequence to Sequence Learning with Neural Networks', Advances in Neural Information Processing Systems.
- Wasala, A., Weerasinghe, R. and Gamage, K., 2006, July. Sinhala grapheme-to-phoneme conversion and rules for schwa epenthesis. In Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions (pp. 890-897).
- Wijerathna, L., Somaweera, W.L.S.L., Kaduruwana, S.L., Wijesinghe, Y.V., De Silva, D.I., Pulasinghe, K. and Thellijjagoda, S., 2012, December. A translator from sinhala to english and english to sinhala (sees). In International Conference on Advances in ICT for Emerging Regions (ICTer2012) (pp. 14-18). IEEE.

