DEVELOPING RULE-BASED APPROACHES TO PROCESS CODE-MIXED TEXTUAL DATA AND MAKE WORD EMBEDDINGS MODELS LIGHTWEIGHT USING TOKEN MAPPING

Jayasinghe D.T.

(IT19075754)

B.Sc. (Hons) Degree in Information Technology specializing in Data Science

Department of Computer Systems Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

September 2022

DEVELOPING RULE-BASED APPROACHES TO PROCESS CODE-MIXED TEXTUAL DATA AND MAKE WORD EMBEDDINGS MODELS LIGHTWEIGHT USING TOKEN MAPPING

Jayasinghe D.T.

(IT19075754)

The dissertation was submitted in partial fulfillment of the requirements for the B.Sc. Special Honors degree in Information Technology

Department of Computer Systems Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

September 2022

DECLARATION, COPYRIGHT STATEMENT AND THE STATEMENT OF THE SUPERVISORS

I declare that this is my own work, and this dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. Also, I hereby grant to Sri Lanka Institute of Information Technology the non-exclusive right to reproduce and redistribute my dissertation in whole or part in future works (such as articles or books).

Name	Student ID	Signature
Jayasinghe D.T.	IT19075754	[illegible]

The above candidate is carrying out research for the undergraduate dissertation under my supervision.

Name of the supervisor: Dr. Lakmini Abeywardhana
Signature of the supervisor: Date: 09/09/2022

Name of the co-supervisor: Ms. Dinuka Wijendra
Signature of the co-supervisor: Date: 09/09/2022

ABSTRACT

Chatbots are a trending topic nowadays. Due to the recent advancement in chatbots, many industries and businesses have adopted chatbots to enhance customer reach and provide efficient assistance. Since chatbots have reached a stage where they utilize artificial intelligence and natural language processing, they have become more efficient in understanding human-generated text. However, with the rise of bilingual and multilingual speakers who speak one or more languages in addition to their mother tongue, the terms code-switching and code-mixing have come into existence, where users mix two or more languages while intercommunicating. It has posed a new challenge to natural language processing, especially for chatbots that use artificial intelligence and natural language processing. Researchers have introduced new solutions, such as multilingual machine learning models, to understand patterns in code-switched textual data. However, it has made the models considerably large in size and data-hungry, making them difficult to utilize in production environments since they are resource heavy. Another issue in processing code-switched textual data is generating word vectors for more than one language. Word Embeddings models such as Word2Vec will encounter out-of-vocabulary tokens more often in a code-switched dataset as the equivalent words between languages depend on individual speakers. This research component addresses the issues encountered while processing Sinhala-English code-switched textual data by developing a code-switchable keyboard interface and equivalent token mapping, a technique to map multiple identical words into the same base word, allowing machine learning models to handle out-of-vocabulary words in code-switched data in production.

Keywords: Natural Language Processing, Code-mixing, Code-switching, Word Embeddings, Token Mapping, OOV token handling

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to our team supervisor, Dr. Lakmini Abeywardhana, and our supervisor, Ms. Dinuka Wijendra, for their support, guidance, and feedback. This effort would not have been possible in the absence of their knowledge and expertise. I also commend the Sri Lank Institute of Information Technology for providing a sizable collection of publicly accessible materials, student guides, and course manuals on their official website, which was extremely useful in creating the datasets used in the research.

Secondly, I am grateful to the SLIIT research project team (CDAP), lecturers, and instructors for their tremendous assistance in constructing a well-planned research schedule, document templates, and guidelines. In addition, I would also like to express my gratitude to the panel for the insightful comments and feedback provided throughout the research.

Finally, I would like to praise all those who assisted the research team in conducting the research at any stage that I have not explicitly mentioned.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Description
AI	Artificial Intelligence
API	Application Programming Interface
CaaS	Conversational AI- as-a-Service
CBOW	Continuous Bag of Words
CDD	Conversational Driven Development
DOCX	Word Open XML Format Document
FAQ	Frequently Asked Questions
GUI	Graphical User Interface
GPU	Graphics Processing Unit
IDE	Integrated Development Environment
JS	Java Script
LKR	Sri Lankan Rupee
ML	Machine Learning
MT5	Multilingual Text-to-Text Transfer Transformer
NLP	Natural Language Processing
NLU	Natural Language Understanding
OOV	Out-Of-Vocabulary
PDF	Portable Document Format
RAM	Random Access Memory
SaaS	Software-as-a-Service
SEETM	Sinhala English Equivalent Token Mapper
SKLEARN	Scikit Learn
SLIIT	Sri Lanka Institute of Information Technology
SQL	Structured Query Language
TF-IDF	Term Frequency-Inverse Document Frequency
TPU	Tensor Processing Unit
USCS	University of Colombo School of Computing

1. INTRODUCTION

1.1 Background & Literature survey

Natural Language Processing (NLP) or computational linguistics is a broader topic and one of the subfields of Machine Learning (ML) and linguistics, and the objective of NLP is to provide computing devices and applications with the ability to process and understand natural human languages [1]. Recent advances in Deep Learning (DL) have resulted in Artificial Neural Networks (ANNs) that can perform NLP with models trained on trillions of features [2]. However, these models require a vast collection of text data to acquire a higher performance level. Even though resource-rich languages such as English contain a lot of production-level NLP tools and are always in the realm of NLP research interests, low-resource languages such as Sinhala often struggle due to their less digital presence and low research interest. Another fact is that bilingual or multilingual speakers tend to use a variety of language patterns by mixing distinct languages, the vocabulary and lexicon of languages, and the characters of multiple languages, which has made building NLP tools more challenging [3]. Even for DL-based solutions, a vast amount of data is not often present, especially in low-resource languages to train models that can recognize these patterns [2], [4].

A wide range of businesses and industries utilize chatbots to provide consumer support. The popularity of chatbots has increased since their inception due to their ability to provide user interaction and entertainment while somehow gaining the user's trust for the outcomes of chatbot [5]. As a result, it is frequent to find chatbot-centric self-service products in various industries, including e-commerce, health, telecommunication, and many others. At the dawn of the chatbot solutions, chatbot query patterns and responses were hard-coded, widely known as the rule-based and pattern-matching approach, where information retrieval was a challenging task, and the end-users had to adapt to the rules to use the chatbots productively [5]. However, the architecture of chatbot solutions has rapidly advanced and have reached a state where they utilize and rely on novel approaches, including Artificial intelligence (AI), Machine Learning (ML), and Natural Language Processing (NLP) chatbots, that have enabled chatbots to process natural language queries and have almost human-like conversations [5], [1].

Languages such as English contain a lot of NLP tools or are always in the realm of NLP research interests. Even though, Sinhala is the national language of Sri Lanka, over 90% of the population is literate. More than 15 million people in the country speak Sinhala, and computers are used quite frequently, particularly at work. However, even now, there are not many more databases, websites, or other online materials in Sinhala, thus texting and e-mail is hardly ever used. In that case, Sinhala contain a considerably lower research interest. There is a need for multilingual and monolingual machine learning models that can handle more than single language, including Sinhala, because there are more bilingual and monolingual speakers nowadays. Despite the fact that they have multiple interpretations, the terms "code-mixing" and "code-switching" are frequently misunderstood. While mixed language usage of more than one language is referred to as code-switching and "code-mixing" is the practice of typing words of one language using the alphabet of another language. Both concepts are currently being researched [11]. For a variety of tasks that require handling code-switched text input, researchers have created monolingual models, including multilingual BERT [2], rapid text word embeddings [3], and MT5. A couple of drawbacks of these multilingual models are the vast increment in vocabulary size and training time. Only a limited fraction of monolingual, bilingual, and multilingual machine learning models for NLP exist for low-resource languages such as Sinhala [4].

With the rise of the Internet, social media, and online presence in general, people often use text-based approaches to communicate with others or retrieve information, and NLP is more required than ever to process these textual data. Online platforms have extended their language support beyond English, creating an immense vacancy for non-English NLP tools. Increment in mobile devices and apps such as Helakuru has caused the digital presence of Sinhala to emerge **Error! Reference source not found.**. The ability to switch between keyboard interfaces quickly has become one of the reasons for code-switched language patterns to exist. In **Error! Reference source not found.**, it is described how social media has influenced people to use code-mixed language, and it discusses how people are using words such as කෑම using different representations like *Kema* or *Kama* using English alphabet characters. The vice-versa of the above fact is also true, as people use Sinhala characters to represent English words such as *Degree* as ඩිග්‍රී while typing.

In modern chatbot building frameworks such as Dialog Flow and Rasa **Error! Reference source not found.**, pre-trained machine-learning models such as language models and word embeddings can be attached to the natural language understanding (NLU) pipeline to generate dense features to train other machine learning components such as text classifiers and entity recognition models. These frameworks support languages beyond English. However, these frameworks usually prefer lightweight

Table 1.1.1. The difference between code-mixing and code-switching for Sinhala and English

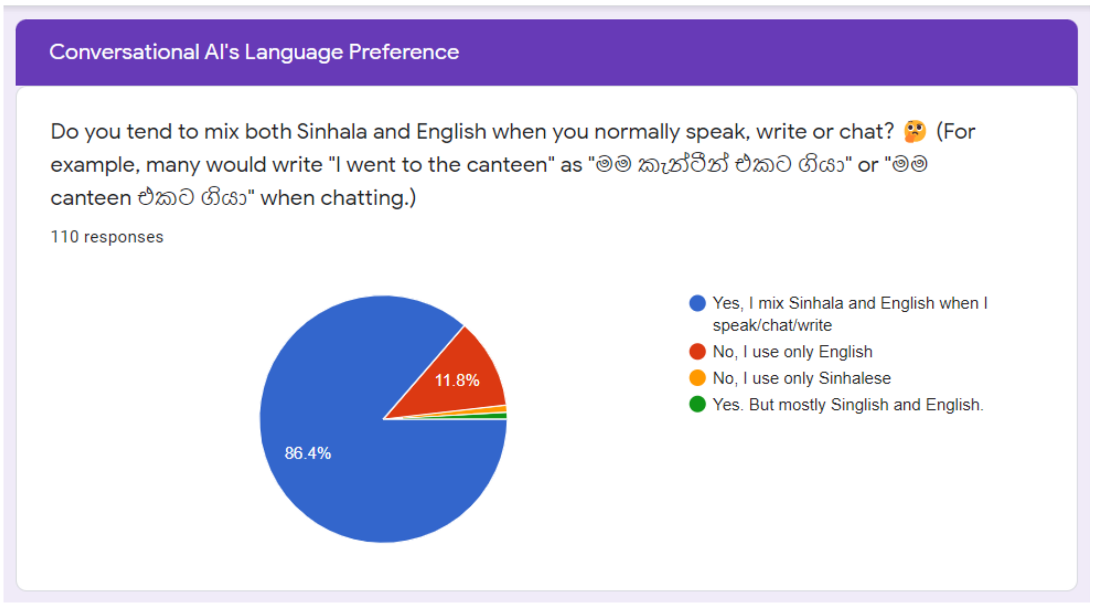
English version	**Sinhala version**	**Code-mixed version**	**Code-switched version**
What are the requirements needed to do an IT degree?	තොරතුරු තාක්ෂණ උපාධියක් කිරීමට තිබිය යුතු අවශ්‍යතා මොනවාද?	IT degree ekak karanna ona requirements monawada?	IT degree එකක් කරන්න ඕන requirements මොනවාද?
How to apply for Library Membership?	පුස්තකාල සාමාජිකත්වය සඳහා අයදුම් කරන්නේ කොහොමද?	Library Membership ekata apply karanne kohomada?	Library Membership එකට apply කරන්නේ කොහොමද?

models since large models might cause high latency and are usually resource intensive. **Error! Reference source not found.**

Word embeddings can be used to generate dense features by representing words in the vector space. Binary Encoding, TF Encoding, TF-IDF Encoding, Latent Semantic Analysis Encoding, and Word2Vec Embedding can be listed as a few out of the many word embeddings techniques. Word2Vec Embeddings **Error! Reference source not found.** is still a widely used word embeddings technique, and there are two main approaches to train word embeddings models known as CBOW (Continuous Bag of Words) and Skip-Gram. Both are neural network architectures and can essentially learn to represent words in the vector space. By default, neural networks work best when there is more training data. **Error! Reference source not found.**Thus, having a large enough text corpus is required to train a good word embeddings model. Not having a specific token can cause out-of-vocabulary (OOV) token issues. Researchers have introduced n-gram based word embeddings models such as fastText and byte-pair embeddings to address the OOV token issue. Modern NLP-based solutions such as artificial intelligence chatbots can rely on these word embeddings models to generate dense features to train their integrated deep learning models for better accuracy **Error! Reference source not found.Error! Reference source not found.**.

To get a clear idea about the language background, a recent survey was conducted for the overall research project including questions covering all four research components. The survey studied the current trends in text-based communication, machine learning model usage preference, and code-switching trends. The population of interest of the survey was 110 Undergraduates of the Faculty of Computing of Sri Lanka Institute of Information Technology, and it was decided based on the technical nature of the survey questions. Refer to **Error! Reference source not found.** to observe the complete list of survey questions and responses.

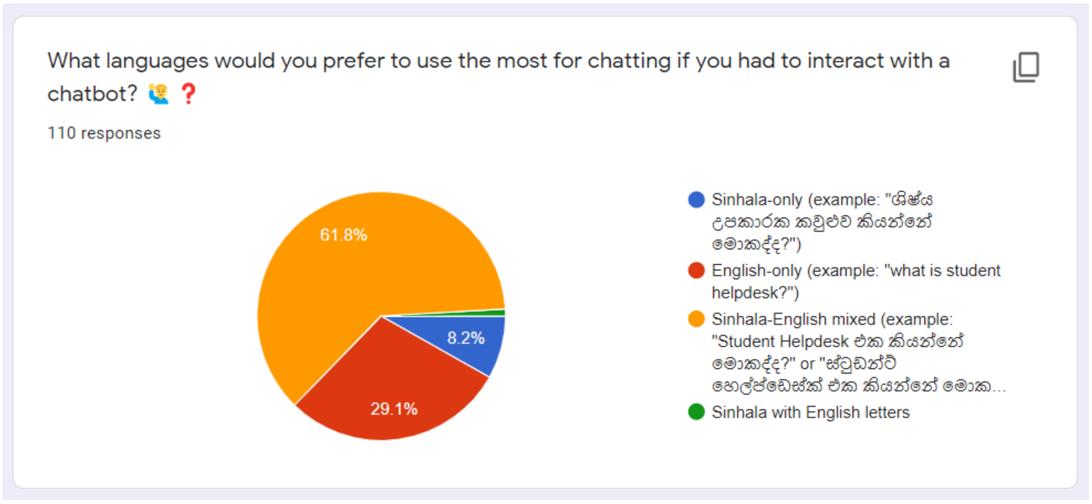
Figure 1.1.1 and Figure 1.1.2 illustrates how participants of the survey prefer using a code-switched sentence structure rather than using a single language out of the given languages, Sinhala, and English. **Error! Reference source not found.** illustrates how the survey results confirm that there is a higher probability to download existing machine learning models in which case a lower file size is preferred.


Conversational AI's Language Preference

Do you tend to mix both Sinhala and English when you normally speak, write or chat? 🤔 (For example, many would write "I went to the canteen" as "මම කැන්ටීන් එකට ගියා" or "මම canteen එකට ගියා" when chatting.)

110 responses

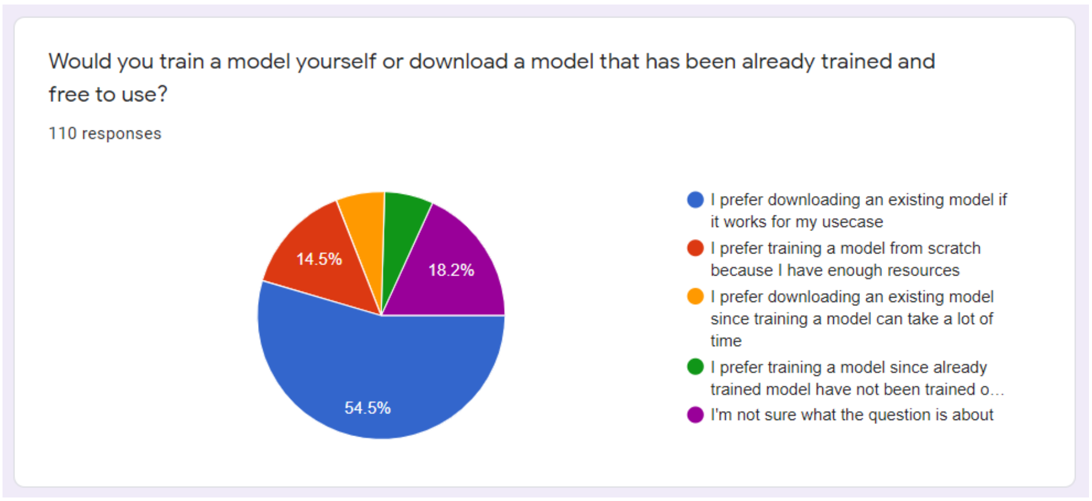
Figure 1.1.1. Summary of survey responses received for the question "Do you tend to mix both Sinhala and English when you normally speak, write, or chat?”


What languages would you prefer to use the most for chatting if you had to interact with a chatbot? 🙋 ❓

110 responses

● Sinhala-only (example: "ශිෂ්‍ය උපකාරක කවුළුව කියන්නේ මොකද්ද?")
● English-only (example: "what is student helpdesk?")
● Sinhala-English mixed (example: "Student Helpdesk එක කියන්නේ මොකද්ද?" or "ස්ටුඩන්ට් හෙල්ප්ඩෙස්ක් එක කියන්නේ මොක...
● Sinhala with English letters

Figure 1.1.2. Summary of survey responses received for the question "What languages would you prefer to use the most for chatting if you had to interact with a chatbot?"


Would you train a model yourself or download a model that has been already trained and free to use?

110 responses

● I prefer downloading an existing model if it works for my usecase
● I prefer training a model from scratch because I have enough resources
● I prefer downloading an existing model since training a model can take a lot of time
● I prefer training a model since already trained model have not been trained o...
● I'm not sure what the question is about

Figure 1.1.3. Summary of survey responses received for the question "Would you train a model yourself?"

1.2 Research Gap

Although some research conducted around Sinhala has explored Sinhala-English code-switched text processing up to an extent, it is still considerably low. The survey mentioned in **Error! Reference source not found.** demonstrates much of the research done based on Sinhala and Sinhala-English code-switched text processing, and it is a valuable resource in identifying the current research gap. Few research papers have focused on building Sinhala chatbots **Error! Reference source not found.Error! Reference source not found.**. Especially the research paper mentioned in **Error! Reference source not found.** discusses how word embeddings models such as fastText models can be used to increase the accuracy and is closer to the objectives of this research. Another research **Error! Reference source not found.** has built a Sinhala chatbot claiming it is the first Sinhala chatbot, and it has incorporated NLP components such as morphological analyzers, Sinhala parsers, and knowledge bases. However, it lacks machine learning components such as word embeddings. The research in **Error! Reference source not found.** embodies training a word2vec model on the USCS Sinhala News dataset using the continuous bag of words (CBOW) method. However, the research paper has mentioned only elementary text-preprocessing techniques such as stop words removal and lemmatization. In addition to that, none of the research papers mentioned above highlight text-preprocessing methods for Sinhala-English code-switched textual data.

Few research papers **Error! Reference source not found.-Error! Reference source not found.** have considered Sinhala-English code-switched text data processing. Although they have discussed word-level language detection, code-switching point detection, and pretraining machine learning models, they do not significantly mention proper text-preprocessing techniques that can be used on code-switched textual data. The paper in **Error! Reference source not found.** has stated code-switching as a challenge in modern Sinhala text. The same research has introduced a technique called “dictionary mapping” to standardize the representation of Sinhala characters written in English characters. This technique is somewhat similar

Table 1.2. 1. Comparing and contrasting with previous related research work done

Research paper reference	**Languages used**	**Code-switched/ Code-mixed**	**Word Embeddings model type**	**Chatbot Framework**	**Special Data pre-processing techniques proposed**
[8] paper	Sinhala	-	fastText Embeddings	Rasa	Basic text preprocessing
[10] paper	Sinhala	-	Word2Vec	-	Basic text preprocessing
[11] paper	Sinhala (Main), English, Singlish	Code-mixed data	-	-	Dictionary Mapping (for characters)
This Research component	Sinhala (Main), English	Code-switched data	Word2Vec	Rasa	Equivalent Token Mapping and Character Mapping

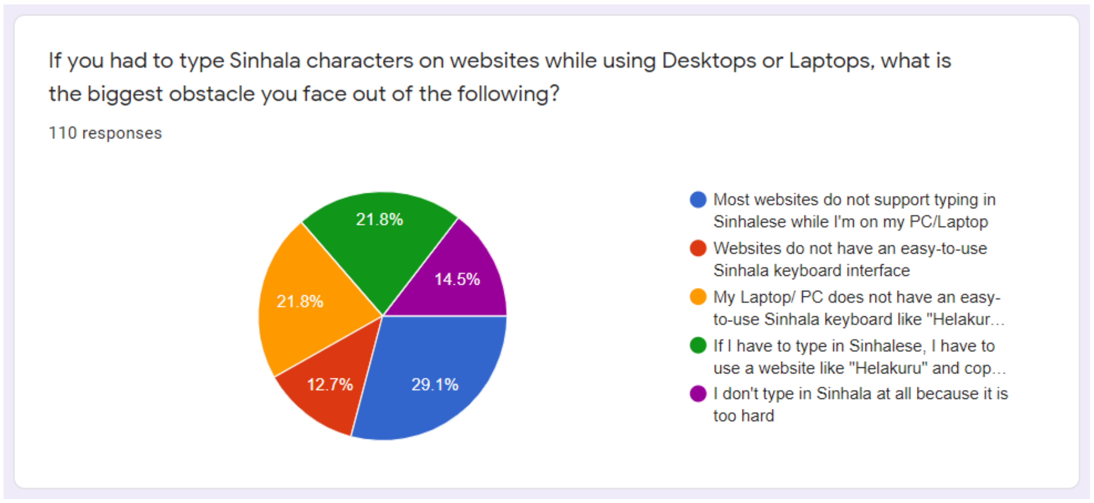
to the method proposed by this research to preprocess Sinhala-English code-switched text.

Although research papers **Error! Reference source not found.** and **Error! Reference source not found.** mention training word embeddings models for Sinhala, they have not considered Sinhala-English code-switched text processing. Although **Error! Reference source not found.** mentions training fastText models for Rasa chatbots, no alterations have been done to the pre-processing steps for the data or the fastText models. The focus of this research is developing "token mapping": a rule-based approach to identify and map tokens present in English in the text corpus to their corresponding representation in Sinhala characters to achieve lightweight word embedding models and handle out-of-vocabulary words efficiently. Thus, "token mapping" seems like a relatively novel approach considered by this research component, and there is a noticeable gap between the objectives of the related work done previously.

2. RESEARCH PROBLEM

NLP (Natural Language Processing) tools and models for processing Sinhala and Sinhala-English code-switched textual data and feature engineering are considerably low. Developing deep learning-based models for NLP such as word embeddings, language models, and named entity recognizers from scratch requires a large amount of language-specific training data which can be rare for low resource languages. Although widely used and high resource languages such as English have many NLP tools or are in the realm of the NLP research interests, languages such as Sinhala have a considerably low research interest. In **Error! Reference source not found.**, it is demonstrated that there is a lot of focus on Sinhala-based research and datasets, however, it also explains that keeping the research finding locked away from public access is one of the key reasons why Sinhala is still considered as a low resource language.

Since there are more bilingual and multilingual speakers today, many researchers focus on building bilingual and multilingual machine learning models that can handle more than a single language. Building multilingual models that can handle hundreds of languages to address modern issues such as code-mixing and code-switching may not be suitable for attaching to a conversational AI since those models can be heavy in size and require a lot of training data not in one language but many.


If you had to type Sinhala characters on websites while using Desktops or Laptops, what is the biggest obstacle you face out of the following?

110 responses

Figure 2.1. Summary of survey responses received for the question related to typing issues exist for Sinhala in various devices

Generally, users use a physical keyboard that has an English-specific layout when interacting with the computing devices. If they need to type in Sinhala or else code-switch between English and Sinhala, they will have to use a service like Helakuru to type in the Sinhala words they want and then copy and paste it into the conversational AI, which is a cumbersome task, and most users would not even bother to go through the trouble for it. Due to that reason, having a keyboard interface that can handle code-switching and code-mixing is a must. More precisely, the keyboard interface must be attachable and support most web applications.

3. RESEARCH OBJECTIVES

3.1 Main Objectives

The individual research component comprises two main objectives, and the first objective is to implement an approach to interact with chatbots effectively using Sinhala-English code-switching queries. The second objective is to efficiently and effectively handle out-of-vocabulary tokens present in Sinhala-English code-switching datasets using a technique introduced as Equivalent Token Mapping to improve the overall performance of Sinhala-English code-switching supported conversational AIs.

The specific objectives of this research component align with the overall research objective, which is developing NLP tools for pre-processing and handling Sinhala-English code-switched text data efficiently to improve the overall performance of conversational AIs.

3.2 Specific Objectives

The specific objectives of the first main objective are as follows.

1. Develop a character map or set of maps that map English characters into relevant Sinhala Unicode characters primarily based on phonetics (the sound of individual characters or combination of characters).

2. Develop an appropriate rule or ruleset to convert Sinhala words written using English characters into Sinhala Unicode characters based on the previously developed character mappings.

3. Order the ruleset logically to obtain a quality and accurate output.

4. Implement the ruleset as a modular website-attachable input component using the React frontend framework primarily focusing on re-usability.

5. Develop a web chat widget by integrating the developed keyboard interface to provide native support for typing code-switchable text queries in the chat widget efficiently.

6. Release the website-attachable input component with the native support for Sinhala-English code-switchable typing as a React component that any developer can freely download and install for NPM.

7. Integrate the webchat component with the integrated Sinhala-English code-switchable keyboard with *Kolloqe*, the chatbot development framework developed as the overall research solution.

The specific objectives of the second main objective are as follows.

1. Train a set of Rasa conversational AIs on code-switching training data extracted from the collected datasets and observe the chatbot responses for different equivalent words.

2. Discuss and evaluate possible solutions to handle out-of-vocabulary token issues in Sinhala-English code-switching datasets, including translation-based, phonetic-based, and rule-based solutions.

3. Select the best-suited token mapping approach and implement the SEETM algorithm backend logic using Python.

4. Develop a CLI and GUI interface for generating token maps.

5. Improve and re-introduce the Rasa WhitespaceTokenizer component as SEETMTokenizer that maps equivalent tokens before the tokenization step of the Rasa NLU pipeline.

6. Evaluate the impact of token maps and demonstrate how SEETM can handle OOV tokens effectively and increase ML model performance in Rasa chatbots.

7. Improve the backend implementation as a modular Python package that individuals can easily download and install on any machine that supports Python using Python Package Index (PyPI).

8. Integrate the developed SEETM Python package into *Kolloqe*, the chatbot development framework introduced by the overall research project.

Remaining sections of this dissertation elaborates on how the author managed to accomplish the main and specific objectives mentioned in this section and what are the overall research outcomes produced by the individual research component and how they have been integrated with the overall research solution.

4. METHODOLOGY

4.1 Data gathering and preparation

A total of two datasets were prepared for the overall research. Subsections 4.1.1 and 4.1.2 provide a detailed description of the two datasets, highlighting the significant differences. Both datasets used data augmentation techniques to overcome the limited resource nature of the Sinhala text data. Through augmentation, the authors captured as many code-switched phrases and distinctive writing styles as possible to eliminate biases. The quality of the data collected was ensured with the use of duplicate and irrelevant data removal.

4.1.1 General dataset for machine learning model training

The first dataset is a domain-specific dataset for ML model training and NLP text pre-processing tool building consisting of scraped Sinhala-English code-switched textual data from SLIIT websites and news articles. The dataset also contains data extracted from official and openly accessible Portable Document Format (PDF) and Microsoft Word Documents available on the same source sites[1]. The authors used data augmentation approaches to overcome the low-resource nature of the collected dataset. After data cleaning and augmentation, the whole dataset consists of 720 paragraphs. It

[1] SLIIT-related websites referred are https://support.sliit.lk/, https://sliitinternational.lk/, https://www.sliit.lk/, https://www.cahm.lk/, https://library.sliit.lk/, and http://sliit.lk/blog and the news articles were extracted from https://www.sundaytimes.lk/, http://www.dailynews.lk, and http://www.sundayobserver.lk/

is freely accessible as a public GitHub repository[2], where interested parties can clone and use it for future studies and other academic projects. The dataset described in this section assisted as a reference dataset to generate numerous Rasa conversational AI datasets required throughout the individual and overall research.

4.1.2 Conversational AI training dataset

The second dataset is a Rasa-specific dataset handcrafted by the authors entirely for training conversational AIs, and it contains knowledge and patterns extracted from the reference dataset mentioned in section 4.1.1. The second dataset is also a domain-specific dataset, created according to instructions provided in the Rasa documentation for intent classification, and also used data augmentation techniques to overcome the low resource issue and apprehend various sentence patterns. There were approximately 78 intents (classes) and 1700 data instances (examples), with a minimum of ten question examples initially present in each category as a standard. The authors improved the original chatbot training dataset and derived a new training dataset with 635 data instances and 71 intents. Notice that chatbot developers can subsequently change the number of question examples per intent to improve the overall performance of conversational AI at any moment. NLP researchers or other interested parties can clone and use the chatbot training dataset for future research, other relevant scholarly work, or domain-specific chatbot training from the same GitHub repository mentioned in section 4.1.1.

4.2 Functional and Non-functional Requirements

The requirement-gathering phase of this study primarily focused on identifying existing constraints in managing Sinhala-English code-switched textual data in modern applications and machine-learning models. Survey mentioned in section 1.2 aided in identifying many of the specific requirements of the research component. The two main findings that confirmed the research gap and assisted in defining specific requirements are (1) the necessity for a well-designed keyboard interface to handle

[2] The datasets mentioned in section 4.1.1 and 4.1.2 are publicly available under MIT license at https://github.com/kolloqe/datasets

Sinhala-English code-switching typing styles and (2) the demand for utilizing ML models in modern conversational AIs. Following are the functional and non-functional requirements derived by the author of this individual research component based on survey responses and analysis of current trends in Sinhala and Sinhala-English code-switching research.

4.2.1 Functional requirements

The following checklist summarizes the functional requirements of this individual research component.

1. The research component should introduce a Sinhala-English keyboard interface where users can type Sinhala and English mixed sentences.

2. The keyboard interface should provide an efficient technique to switch between the two languages.

3. The keyboard interface should not be an externally visible component. However, it should function as a native HTML input textbox that developers can integrate with any HTML page.

4. The research component also should introduce Sinhala-English Equivalent Token Mapping (SEETM): a novel Sinhala-English code-switching dataset pre-processing method that can effectively map multiple equivalent tokens to a single representation (base word).

5. SEETM should efficiently and effectively handle out-of-vocabulary (OOV) token issues in Sinhala-English code-switching datasets by generating a single vector representation for all equivalent words.

6. SEETM should be attachable to any Rasa conversational AI project through the NLU pipeline as a pre-processing component.

7. SEETM should allow users to create equivalent token maps easily before training conversational AIs on Sinhala-English code-switched datasets.

8. SEETM should provide the support for extending already define maps or add new token maps without having to re-train already-trained ML models.

4.2.2 Non-functional requirements

The non-functional requirements of the individual research component are as follows.

1. The Sinhala-English code-switching keyboard interface should be highly efficient since users directly interact with it.

2. The keyboard interface output should be reliable and consistent.

3. The character mappings utilized in the keyboard interface should be intuitive and easily perceivable.

4. The keyboard interface implementation should be modular and easily installable or attachable.

5. The SEETM approach should be efficient and intuitive.

6. Token mapping should also support CLI-based environments since it is common for chatbot developers to utilize cloud-based VMs to train chatbot models.

7. SEETM should offer a modular Python package that the developers can easily install, and each version of the SEETM package should ship with the necessary documentation.

The overall research outcome of this individual research component successfully addressed all the functional and non-functional requirements. The following sections of this dissertation provide an in-depth description of how the Sinhala-English code-switching keyboard interface and SEETM implementation satisfied these requirements and achieved the research component objectives.

4.3 Individual Component Architecture and Overview

The author of this individual research component has divided the research objectives into two main objectives as previously mentioned in section 3. The first objective addressed the issues encountered by the end users, conversational AI developers, and testers when interacting with conversational AIs and chatbots using Sinhala-English code-switching queries. The Sinhala-English code-switchable keyboard interface, which is the solution implemented by the author of this research component under the first research objective, comprises a set of character mappings that maps Sinhala Unicode characters to appropriate English characters or combination

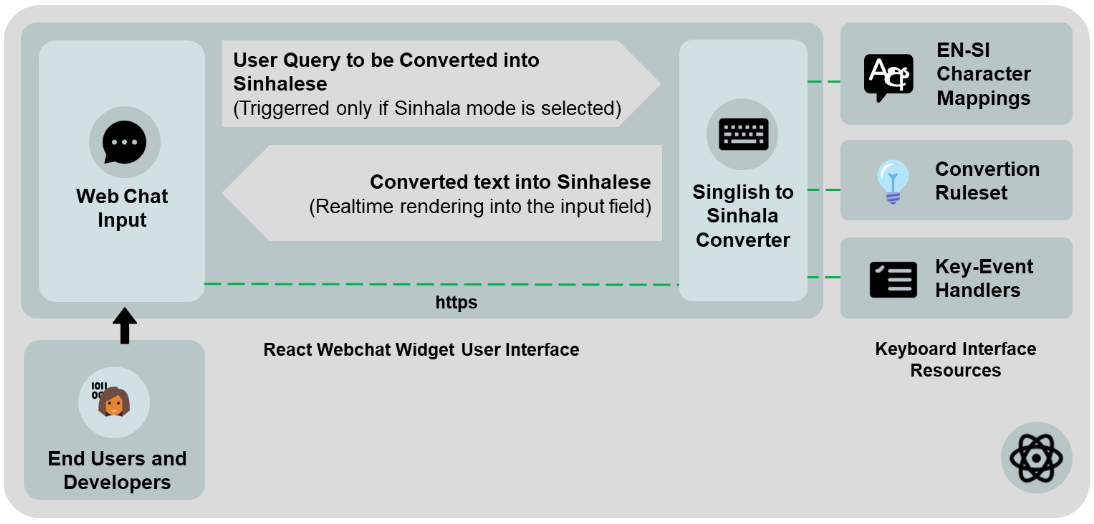

User Query to be Converted into Sinhalese
(Triggerred only if Sinhala mode is selected)
Web Chat Input
Converted text into Sinhalese
(Realtime rendering into the input field)
Singlish to Sinhala Converter
EN-SI Character Mappings
Convertion Ruleset
Key-Event Handlers
https
React Webchat Widget User Interface
Keyboard Interface Resources
End Users and Developers

Figure 4.3.1. High-level architectural diagram of the Sinhala-English keyboard interface

of characters primarily based on their specific sounds. This phonetic-inspired character mapping approach allows users to intuitively type matching English characters for selected Sinhala words (often referred to as Singlish), converts the text, and renders the output in real time. The implemented Sinhala-English code-switchable keyboard interface was released as an open-source NPM package and is actively maintained by the author of this individual research component along with comprehensive documentation. Section 4.4 elaborates on the approach utilized to implement the solution in detail. **Error! Reference source not found.** depicts the high-level architecture of the keyboard interface implementation.

According to the second objective, the author implemented SEETM, which stands for Sinhala-English Equivalent Token Mapping, to overcome out-of-vocabulary token issues encountered when training chatbots using Sinhala-English code-switching datasets. The SEETM approach allows developers to map equivalent words into a single base word by providing an easy-to-use GUI. Unlike the keyboard interface, SEETM was fully implemented using Python and chatbot developers can integrate SEETM with any Rasa 2.8.8 chatbot development project without any constraints. The structure of the SEETM Python package was logically divided into five sub-packages as *cli*, *core*, *server*, *shared*, and *utils*. The *cli* sub-package contains the SEETM CLI interface implementation which lets developers run the SEETM Developer Console and the *core* sub-package contains the core token mapping logic that is responsible for mapping equivalent tokens into their respective base words. The *server* sub-package contains the SEETM Developer Console frontend resources and the SEETM API implemented using Python Flask. The *shared* sub-package contains modules shared among all SEETM sub-packages such as constants and exceptions, whereas the *utils* sub-package consists of utility and I/O-related functions utilized for input-output operations such as reading and persisting token maps and initializing new SEETM projects. Section 4.5 elaborates on the approach utilized to implement the SEETM Python package in detail and section 5.1.2 describes how developers can use the SEETM Python package integrated with any Rasa chatbot development project. Figure 4.3.2 depicts the high-level architecture of the SEETM implementation.

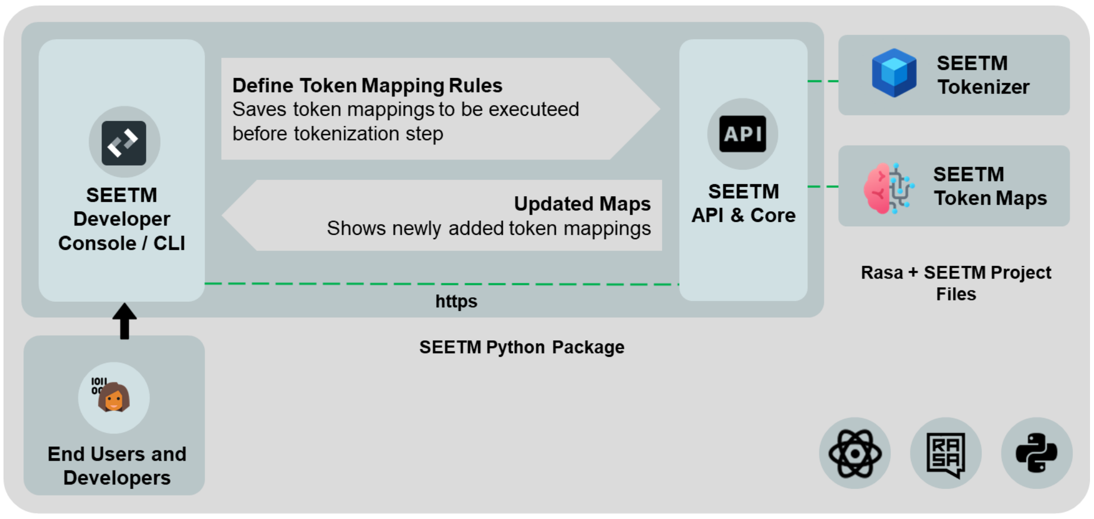

SEETM Developer Console / CLI
Define Token Mapping Rules
Saves token mappings to be executeed before tokenization step
Updated Maps
Shows newly added token mappings
SEETM API & Core
SEETM Tokenizer
SEETM Token Maps
Rasa + SEETM Project Files
https
SEETM Python Package
End Users and Developers

Figure 4.3.2. High-level architectural diagram of SEETM

Note that both implementations provide modular packages that were open-sourced and free to use. The keyboard interface React component is available for the developer to download at Node Package Manager (NPM) as previously mentioned and the SEETM Python package is available for the developers to download from the Python Package Index (PyPI). Since the released packages are open-source, the source code is properly maintained using dedicated public GitHub repositories[3].

Error! Reference source not found. depicts how the Sinhala-English code-switchable keyboard interface and the SEETM approach conform to the overall

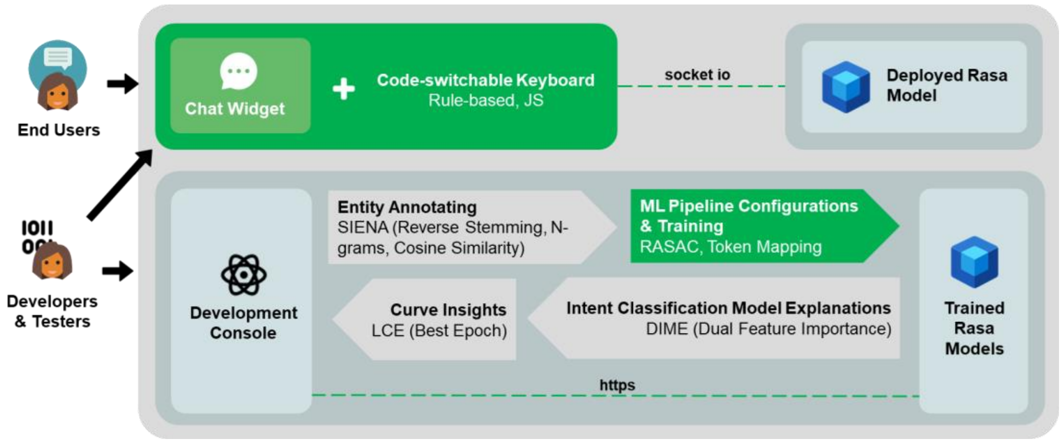


Figure 4.3.3. High-level architectural diagram of Kolloqe, the overall research solution, and indicated in green is where the individual research solutions conform

[3] The keyboard interface source code is available at https://github.com/kolloqe/react-kbi-si-en and the SEETM source code is publicly available at https://github.com/SEETM-NLP/seetm

research system architecture after integrating all research components. The keyboard interface integrated into Kolloqe allows developers to test the deployed version of the trained Sinhala-English code-switching supported chatbots directly via the chat widget in the dashboard page of the developer console and eliminates the need to utilize external or third-party Sinhala keyboard software when typing queries. The keyboard also supports efficient language switching between Sinhala and English. The SEETM approach integrated into Kolloqe allows the creation of token maps for handling out-of-vocabulary tokens through mapping equivalent words to a base word, increasing the overall performance of chatbot models trained using Kolloqe.

4.4 Implementation of Keyboard Interface

4.4.1 Development of character maps

The author of this individual research component has implemented the Sinhala-English code-switchable keyboard interface using a phonetics-based approach, since the main objective of the keyboard is to provide an interface to type Sinhala-English code-switching queries efficiently and intuitively in the chat widget. First the author mapped individual Sinhala Unicode characters into respective English characters or character sets based on their sound. Figure 4.4.1.1 clearly depicts the Sinhala-English character map constructed primarily based on phonetics and Singlish typing patterns.

The author gained insights into developing the character map by carefully studying the pronunciations of both Sinhala and English characters and thoroughly inspecting the English character usage when writing Sinhala words using English characters (code-mixing). To accomplish this, the author extracted Sinhalese words from the dataset collected initially and attempted to rewrite the word in English to resemble the same pronunciation as closely as possible. For example, the author wrote the Sinhala word *අම්මා* (mother) as *amma*, which closely preserves the initial Sinhala pronunciation. This phonetic-based approach derived the following insights based on the applied stress and articulation of the tested Sinhalese words. (The given mappings in the following list are in the format of *Source_Character-Mapped_Character*).

1. Most vowels without stress can be represented using a single English character.

අ-a, ඉ-i, එ-e, ඔ-o, උ-u

2. Some vowels without stress require a special English character mapping.
   ඇ-A, ඓ-I, ඖ-O

3. Most vowels with stress require same English characters typed twice.
   ආ-aa, ඊ-ii, ඒ-ee, ඕ-oo, ඌ-uu

4. Some vowels with stress require a special English character combination.
   ඈ-AA, ඒ-ea

5. Some vowels were given multiple representations considering the typing flexibility.
   ඈ - AA, Aa, ae
   ඊ - ii, ie
   ඒ - ea, ei
   ඓ - I, E
   ඕ - oo, oe
   ඖ - O, au

6. Most consonants without stress can be represented using a single English character.
   ක-k, ග-g, ජ-j, ට-t, ඩ-d, න-n, ප-p, බ-b, ම-m, ය-y, ර-r, ල-l, ව-w, ස-s, හ-h, ෆ-f

7. All prenasalized consonants require a special representation. Thus, *nn* and *mm* was used as a prefix to identify prenasalized consonants. *nn* was used for opened-lip prenasalized consonants and *mm* was used with closed-lip prenasalized consonants.
   ඟ-nng
   ඬ-nnd
   ඳ-nndh
   ඹ-mmb

8. All *Mahaprāna* characters were given *h* as a suffix to differentiate them from the non-*Mahaprāna* variant of them.
   ඛ-kh, ඝ-gh, ඡ-Ch, ඨ-Th, ඪ-Dh, ඵ-ph, භ-bh

9. Some *Mahaprāna* characters were given multiple representations for typing flexibility.
   ඛ-kh, K
   ඝ-gh, G
   ධ-dh, D
   ඡ-Ch, C
   ඵ-ph, P
   භ-bh, B

10. Some special consonants require special English character mappings that does not overlap with the other mappings.
    ං-x, ඃ-H, ඍ-R, ෲ-ruu, ෘ-ru, ඤ-KN, ඥ-GN, ඏ-Lu, ළ-L, ඞ-X, ණ-N, ද-dh, ච-ch, ත-th, ශ-sh, ෂ-Sh, ඣ-J, ට-T

11. Some consonants were given multiple representations considering the typing flexibility
    ය-y, Y
    ව-v, w
    ච-ch, c

12. In addition to the above mappings, there are special character set known as consonant modifiers (or vowel prefixes) that are used for modifying consonants and the mappings specified for vowels were used for the relevant vowel prefixes as well.
    (Highlighted in green in Figure 4.4.1.1 are the vowel prefixes that can modify a given consonant).

The character mappings depicted in Figure 4.4.1.1 are the finalized character mappings, according to relevant categories. The categories derived are as follows.

1. Pure vowels – vowels that are not utilized as suffixes

2. Vowel suffixes – also known as *consonant modifiers* and these are a derivative of pure vowels

3. Consonants – all consonants, including prenasalized consonants, *Mahaprāna* consonants, and special consonants, were considered as a single category.

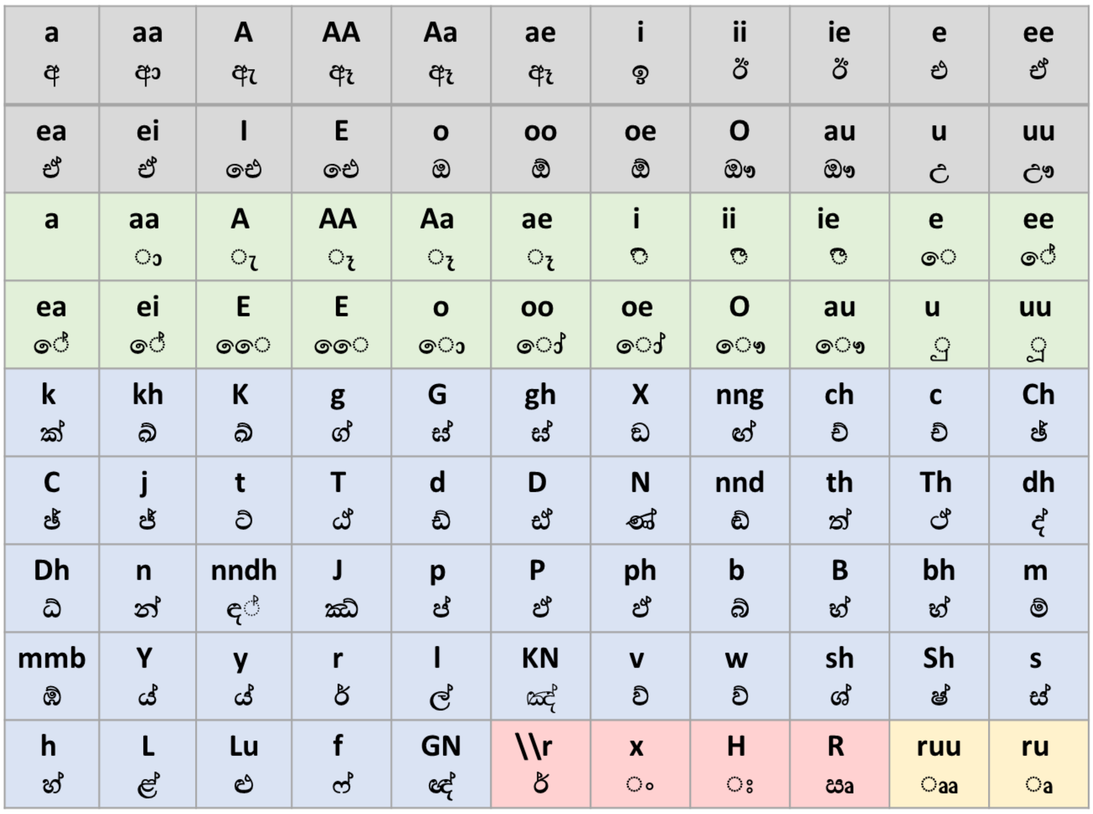
4. Non-joining characters – characters that are not joined with other characters.

5. Special characters – characters whose English mapping is overlapping with consonant mappings, thus needs high priority of execution.

4.4.2 Development of the ruleset

After the development of character mappings, the author studied extensively on how to utilize the character mappings to convert a given set of English mappings into Sinhalese. It was found by a trial-and-error approach that simply replacing arbitrary English characters by Sinhala Unicode characters does not generate the expected output. Thus, the author conducted further research on how to logically order the character mapping process, based on different categorizations of Sinhala characters. Initial assumption was that a ruleset can be derived based on different Sinhala character categorizations such as long vowels, short vowels, consonants, *Mahaprāna* consonants, prenasalized consonants, and special or rare characters.

The same distinct Sinhala word set extracted from the Sinhala-English code-switched dataset were utilized to test the logical order of converting words written in



a	aa	A	AA	Aa	ae	i	ii	ie	e	ee
අ	ආ	ඇ	ඈ	ඈ	ඈ	ඉ	ඊ	ඊ	එ	ඒ
ea	ei	I	E	o	oo	oe	O	au	u	uu
ඒ	ඒ	ඓ	ඓ	ඔ	ඕ	ඕ	ඖ	ඖ	උ	ඌ
a	aa	A	AA	Aa	ae	i	ii	ie	e	ee
	◌ා	◌ැ	◌ෑ	◌ෑ	◌ෑ	◌ි	◌ී	◌ී	◌ෙ	◌ේ
ea	ei	E	E	o	oo	oe	O	au	u	uu
◌ේ	◌ේ	◌ෛ	◌ෛ	◌ො	◌ෝ	◌ෝ	◌ෞ	◌ෞ	◌ු	◌ූ
k	kh	K	g	G	gh	X	nng	ch	c	Ch
ක්	ඛ්	ඛ්	ග්	ඝ්	ඝ්	ඞ	ඟ්	ච්	ච්	ඡ්
C	j	t	T	d	D	N	nnd	th	Th	dh
ඡ්	ජ්	ට	ඨ්	ඩ	ඪ්	ණ්	ඬ	ත්	ථ්	ද්
Dh	n	nndh	J	p	P	ph	b	B	bh	m
ධ	න්	ඳ◌්	ඣ	ප්	ඵ්	ඵ්	බ්	භ්	භ්	ම්
mmb	Y	y	r	l	KN	v	w	sh	Sh	s
ඹ	ය්	ය්	ර්	ල්	ඤ්	ව්	ව්	ශ්	ෂ්	ස්
h	L	Lu	f	GN	\\r	x	H	R	ruu	ru
හ්	ළ්	ළු	ෆ්	ඥ්	ර්	◌ං	◌ඃ	ඍa	◌aa	◌a

Figure 4.4.1.1. English to Sinhala character mappings. Color-coded categories: (1) gray – pure vowels, (2) green – consonant modifiers (vowel suffixes), (3) blue – consonants, (4) red – non-joining vowels and consonants, and (5) yellow – special suffixes

Table 4.4.2.1 An example of the trial-and-error approach incorporated to derive a logical order of character mapping execution

Attempt #	Initial Word	Character Category based on Priority	Output	Observation
1	ammaa	1. Short vowels	අmmඅඅ	Not successful. Order must be changed.
2	ammaa	1. Long vowels	ammආ	Correct
		2. Short vowels	අmmආ	Correct
		3. Normal consonants	අම්ම්ආ	Close but incorrect. Order must be changed.
3	ammaa	1. Normal consonants	aම්ම්aa	Incorrect. ම් and ආ are never merged.
4	ammaa	1. Normal consonants + any vowels	amමා	Correct
		2. Normal consonants alone	aම්මා	Correct
		3. Any vowel left alone	අම්මා	Correct. According to the test, the logical order up to now is, 1. Normal consonants + any vowel 2. Individual normal consonants 3. Individual vowels

English mappings into the respective Sinhalese representation. First, each word was written in using the corresponding English mappings. Then the author attempted to convert it back to the initial Sinhala form by applying different mappings in reverse according to different character categories. For example, if the initial word taken is *අම්මා*, it was converted to *ammaa* (අ-a, ම්-m, ම්-m, ආ-aa). Then it was tested in which order the mappings of the word *ammaa* need to be converted back to Sinhala. Table 4.4.2.1 depicts an example attempt on converting the word *ammaa* back to Sinhala by giving priority to different character categories. This was extensively studied to derive a ruleset that can convert most English mapped words of Sinhala words back to its original representation.

According to the observations of the previous study, the ruleset defined in Table 4.4.2.2 was derived. The same figure demonstrates how the English character mapped word *aakramaNaya* was converted to back to its original Sinhala representation, ආක්‍රමණය using the derived ruleset and the ruleset is ordered according to the descending order of execution priority.

The justification of the rule and the execution priority given are as follows.

1. Rule 1: Non-joining character mappings

   Non-joining characters are not coupled with any other Sinhala characters. Thus, they were given the highest priority since mapping them first does not affect the other mappings.

2. Rule 2: Consonant + Special character mappings

   Special character mappings contain conflicting English characters. For example, ◌ෘ is a special character with English mapping *ru* which shares a character with the English mapping of the consonant ර (English mapping of ර is *r*) and the vowel උ (English mapping of උ is *u*). This directly affects the execution order and if not mapped first, Rule 4 is executed and gives a faulty output. If rule 4 is executed prior to this rule, Sinhala words with special characters such as කෘමියා (*insect*) are only partially converted as ක්රුමියා.

3. Rule 3: *Rakarānshaya* mapping

   The *Rakarānshaya* mapping is a mapping that does not reflect in the previously presented character mappings presented in Figure 4.4.1.1, but it directly affects the execution order the same way Rule 2 did. The word ආක්‍රමණය is an example with the Sinhala character known as *Rakarānshaya*. If not mapped first, Rule 4 gives a faulty output by converting ආක්‍රමණය as ආක්රමණය, which is wrong. This rule essentially inspects consonants + r + vowel characters. Notice that the Rule 2 contains a vowel after *r*, which makes Rule 2 and Rule 3 (this rule) **not** interchangeable. If this rule was executed before Rule 2, the word කෘමියා is faultily converted as ක්‍රුමියා.

4. Rule 4: Consonant + Vowel mappings

This rule converts all the *consonants + vowel* mappings present in a specified word or phrase. For example, words such as මල (*mala*) utilizes only this rule since the English mapping of the word is a combination of vowels and consonants. In this case, *ma* gets converted into ම (m+a → ම්+අ) And la gets converted into ල (l+a → ල්+අ). If executed prior to Rule 5, a partially converted output is obtained as ම්අල්අ due to pure consonants and vowels getting converted first according to Rule 5 and Rule 6.

5. Rule 5: Pure consonant mappings

   Fifth rule converts all remaining pure consonants as previous rules take care of other different character mapping combinations with pure consonants. Note that Rule 5 (this rule) and Rule 6 are interchangeable since they do not affect each other.

6. Rule 6: Pure vowel mappings

   The final rule converts all remaining pure vowels as there are no other character mapping combinations left to inspect. As previously stated, Rule 5 and Rule 6 are interchangeable.

Algorithm 1 comprises the above ruleset and clearly states how the execution order of the ruleset was implemented. However, it comprises only the main logic of converting English mapping into the relevant Sinhala word and other parts of the actual implementation such as handling the shortcut keys and triggering the message sending are not included for the sake of simplicity.

Table 4.4.2.2. Ruleset derived and the order of rule execution when converting a word represented using English character mappings into its original Sinhala representation

Rule		**English Representation:** aakramaNaya	
		Character Mapping	**Output**
1.	Non-joining Character Mappings	skipped as there are no non-joining characters	aakramaNaya
2.	Consonant + Special Character Mappings	skipped as there are no special characters	aakramaNaya
3.	“Rakaranshaya” Mapping	kra → ක් + ර් + අ → ක්‍ර	aaක්‍රmaNaya
4.	Consonant + Vowel Mappings	ma → ම් + අ → ම Na → ණ් + අ → ණ ya → ය් + අ → ය	aaක්‍රමNaya aaක්‍රමණya aaක්‍රමණය
5.	Pure Consonant Mappings	skipped as there are no pure consonants left	aaක්‍රමණය
6.	Pure Vowel Mappings	aa → ආ	ආක්‍රමණය

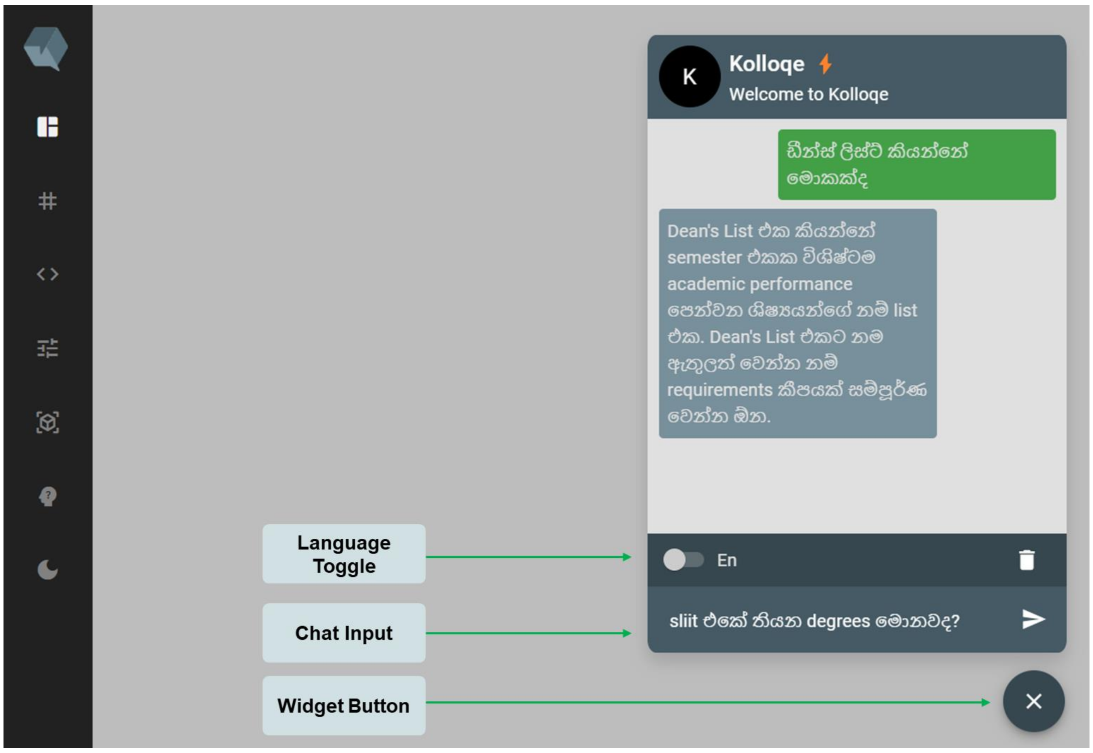
Algorithm 1 CHARACTERMAPPING ($instance$)

```
1   $output \leftarrow instance$        ▷ Initialize the output to the text. This is a string
2   $pureVowelMap \leftarrow$ INITIALIZE ($pureVowels$)          ▷ Maps are defined as either arrays or
3   $vowelSuffixMap \leftarrow$ INITIALIZE ($vowelSiffixes$)       objects. Since maps are large variables, it
4   $consonantMap \leftarrow$ INITIALIZE ($consonants$)            is indicated as INITIALIZE in the
5   $speacialCharMap \leftarrow$ INITIALIZE ($specialChars$)       algorithm
6   $nonJoiningCharMap \leftarrow$ INITIALIZE ($nonJoining$)
7   for each $char, mappedChar \in nonJoiningCharMap$ do        ▷ Rule #1
8       $output \leftarrow$ REPLACE ($mappedChar, char, output$)
9   for each $spchar, spMappedChar \in speacialCharMap$ do      ▷ Rule #2
10      for each $conChar, conMappedChar \in consonantMap$ do
11          $output \leftarrow$ REPLACE ($conMappedChar + spMappedCha, conChar + spChar, output$)
12  for each $vChar, vMappedChar \in vowelSuffixMap$ do         ▷ Rule #3
13      for each $conChar, conMappedChar \in consonantMap$ do
14          $output \leftarrow$ REPLACE ($conMappedChar + r + vMappedChar, conChar +$ "්ර" $+ vChar$ )
15  for each $vChar, vMappedChar \in vowelSuffixMap$ do         ▷ Rule #4
16      for each $conChar, conMappedChar \in consonantMap$ do
17          $output \leftarrow$ REPLACE ($conMappedChar + vMappedChar, conChar + vChar$, $output$)
10  for each $conChar, conMappedChar \in consonantMap$ do       ▷ Rule #5
11      $output \leftarrow$ REPLACE ($conMappedChar , conChar, output$)
12  for each $vChar, vMappedChar \in pureVowelMap$ do           ▷ Rule #6
14      $output \leftarrow$ REPLACE ($conMappedChar , conChar, output$)
18  return $output$
```

4.4.3 Implementation of the React component

The ruleset demonstrated in the previous section was implemented using JavaScript and attached to a chat widget designed utilizing React as the front-end framework. The primary reason to implement the keyboard interface and its ruleset using React was to increase the efficiency and reduce the latency when rendering a real-time output of the converted text into Sinhalese and to release it as a reusable React component.

Figure 4.4.3.1 depicts the React-based chat widget component implemented. Note that the chat widget provides an input field to type user queries and a language toggle switch to switch between Sinhalese and English even within sentences. This enables typing both Sinhala and English words within the same query. To provide the users with efficient support for switching in between languages, an easy-to-use shortcut was implemented as *Control Key + Q*, which the users can utilize without having to use the mouse to change the language. The JavaScript implementation of the Algorithm 1 was set to execute whenever the chat widget users trigger *Keystroke* events and all three keystroke events, *key down*, *key press*, and *key up*, were utilized properly to handle the conversion of English mappings to Sinhalese. Table 4.4.3.1 comprises how the keystroke events were handled and the corresponding behaviour of the keyboard interface. It is noteworthy that the conversion only takes place when the language is selected as Sinhalese. If the language selected is English, the English to Sinhalese conversion algorithm is not triggered at all, allowing the users to type in English in the input field.



ඩීන්ස් ලිස්ට් කියන්නේ මොකක්ද

Dean's List එක කියන්නේ semester එකක විශිෂ්ටම academic performance පෙන්වන ශිෂ්‍යයන්ගේ නම් list එක. Dean's List එකට නම ඇතුලත් වෙන්න නම් requirements කීපයක් සම්පූර්ණ වෙන්න ඕන.

sliit එකේ තියන degrees මොනවද?

Figure 4.4.3.1. React chat widget component with code-switchable keyboard interface attached to the HTML input element.

Table 4.4.3.1. JavaScript Keystroke Events Handling when converting Character Mappings into the relevant Sinhalese representation

Keystroke Event	**Sinhala-English code-switchable keyboard interface behaviour language selection is set to Sinhalese**
Key Press	
If keycode between 20 and 126 including 20, 126 (Including a-z, A-Z, 0-9, and symbols)	Accumulates the key value each keystroke.
Key down	
If keycode is 8 (Backspace)	Inserts a backspace by copying the current content in the input field and eliminating the last character. If a selection is backspaced, all selection content is removed.
If keycode is 13 (Enter)	Triggers the message sending button in the widget.
Key up	
For any keycode	Triggers the English mappings to Sinhalese conversion and convers the accumulated input text saved in React state by *Key Press* event and renders the output to the input field of the widget.

4.4.4 Publishing the React component

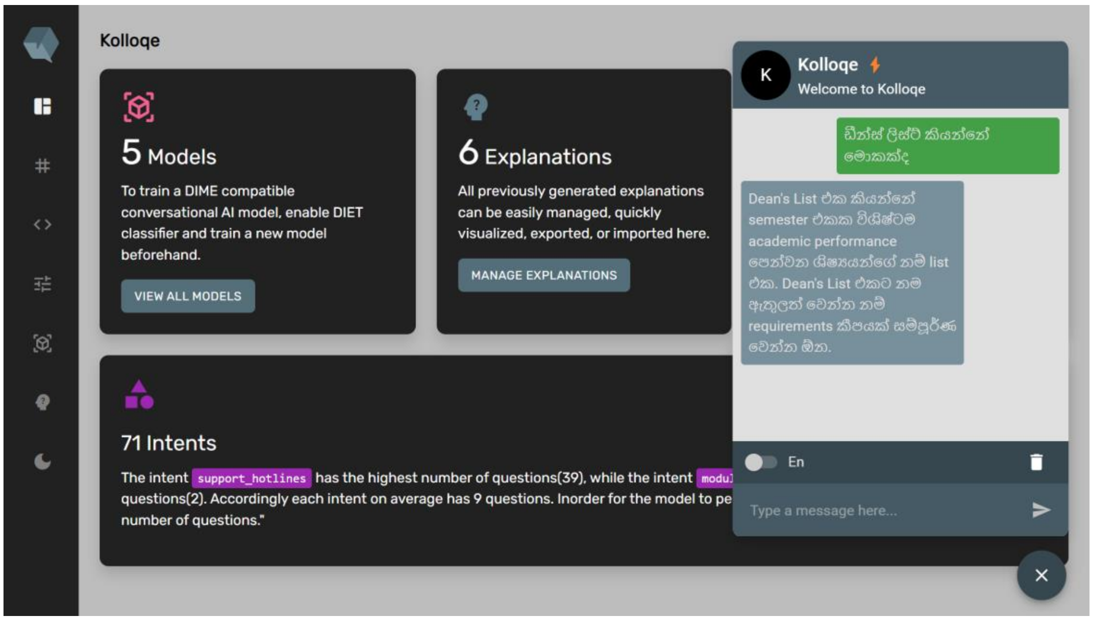
The implementation of the React chat widget with integrated Sinhala-English code-switchable keyboard interface was packaged as an individual React component that can be attached to any React based project. Also, the same component was transpiled into a single JS file that can be attached with any HTML-based websites coded without any specific front-end development frameworks such as React, Vue.JS, and Angular JS. The Webchat widget component was not released to the public as an open-source component since it is utilized in *Kolloqe*, the proprietary chatbot development platform built as the overall research outcome when all four research components integrated and releasing the widget as a free package would degrade the competitive advantage of *Kolloqe*.

However, a separate React component named *@Kolloqe/input*[4] was built and released as an open-source package which any developer can utilize to integrate built-in support for Sinhala-English code-switchable typing support in React based websites. The *@Kolloqe/input* React component appears to be a normal Input field, but it is seamlessly integrated with the Sinhala-English code-switching algorithm demonstrated in section 4.4.2. Any front-end or full-stack developer who wants to enable native typing support for Sinhala-English code-switching queries can freely obtain the React input component from NPM and consume it in their projects under the Apache 2.0 license[5].

4.4.5 Integrating with the overall research

The chat widget built with React was later integrated into the *Kolloqe* chatbot development console to allow chatbot developers to test deployed versions of already trained Sinhala-English code-switching supported chatbot models directly using the chat widget integrated into the UI. In addition to the chat widget, the Sinhala-English code-switchable keyboard interface was also integrated to all the input fields in the *Kolloqe* developer console to provide the support for natively type both Sinhala and English anywhere in the UI, which makes *Kolloqe* the first web application that natively supports typing in both Sinhala and English within any input field in the UIs of the application without the assistance of third-party keyboard utilities or external software. Figure 4.4.5.1 through Figure 4.4.5.3 depicts where the keyboard interface designed in this individual component was utilized in the overall product.

[4] Kolloqe input React package can be downloaded from https://www.npmjs.com/package/@kolloqe/input
[5] https://github.com/kolloqe/react-kbi-si-en/blob/main/LICENSE allows interested individuals to make changes to the source code with clearly highlighting the changes made.



To train a DIME compatible conversational AI model, enable DIET classifier and train a new model beforehand.

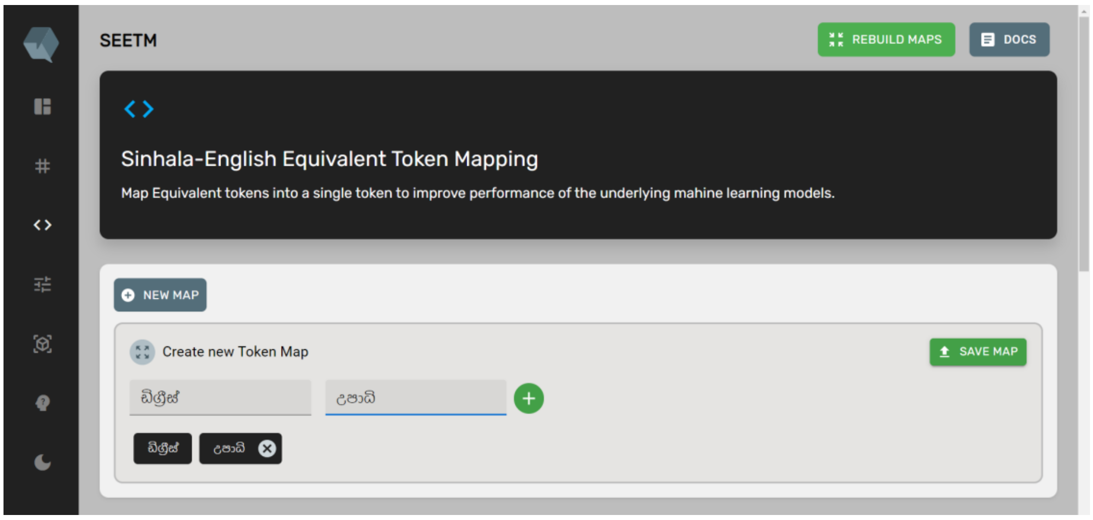
All previously generated explanations can be easily managed, quickly visualized, exported, or imported here.

The intent `support_hotlines` has the highest number of questions(39), while the intent `modul` questions(2). Accordingly each intent on average has 9 questions. Inorder for the model to pe number of questions."

ඩීන්ස් ලිස්ට් කියන්නේ මොකක්ද

Dean's List එක කියන්නේ semester එකක විශිෂ්ටම academic performance පෙන්වන ශිෂ්‍යයන්ගේ නම් list එක. Dean's List එකට නම ඇතුලත් වෙන්න නම් requirements කීපයක් සම්පූර්ණ වෙන්න ඕන.

Figure 4.4.5.1. Sinhala-English code-switchable Keyboard interface attached chat widget integrated with the overall research outcome, *Kolloqe* developer console.



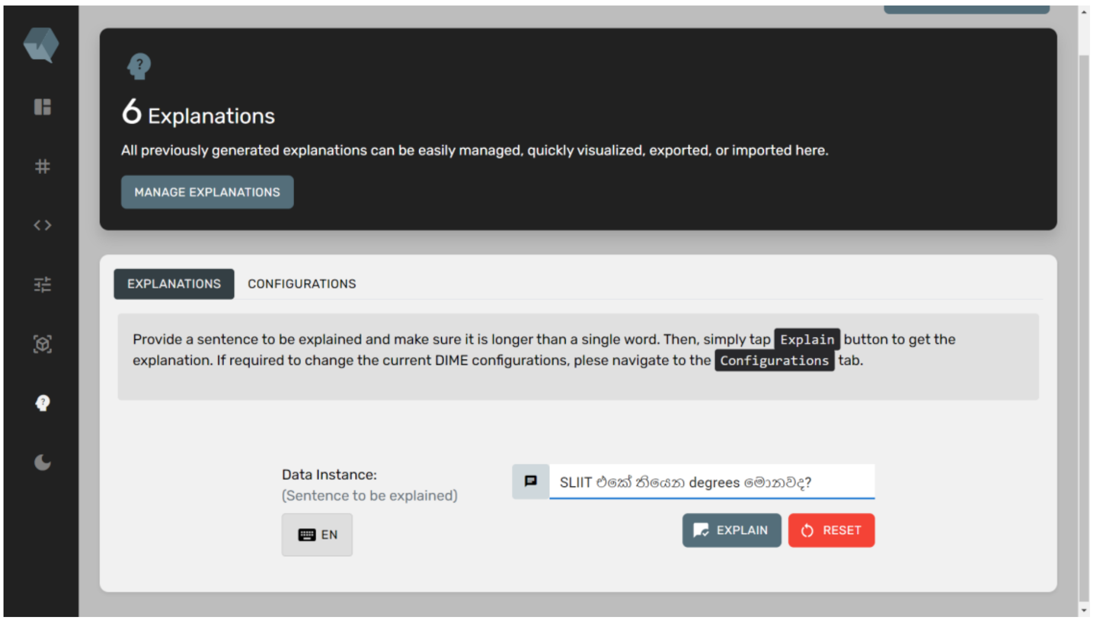
Sinhala-English Equivalent Token Mapping

Map Equivalent tokens into a single token to improve performance of the underlying mahine learning models.

ඩිග්‍රීස් | උපාධි

ඩිග්‍රීස් උපාධි

Figure 4.4.5.2. Sinhala-English code-switchable Keyboard interface attached to the token mapping UI in the *Kolloqe* developer console.


6 Explanations

All previously generated explanations can be easily managed, quickly visualized, exported, or imported here.

Provide a sentence to be explained and make sure it is longer than a single word. Then, simply tap `Explain` button to get the explanation. If required to change the current DIME configurations, plese navigate to the `Configurations` tab.

Data Instance:
(Sentence to be explained)

SLIIT එකේ තියෙන degrees මොනවද?

Figure 4.4.5.3. Native support for Sinhala-English code-switching queries provided in the Explainable AI UI in the *Kolloqe* developer console.

4.5 Implementation of SEETM

The next part of the implementation of the individual research component is related to the second main objective mentioned in section 3.1 and is independent of the Sinhala-English code-switchable keyboard interface implementation described in detail in section 4.4 The SEETM component primarily attempted to solve the OOV tokens issue when training chatbot models and its concept is relevant for any NLP-based machine learning or deep learning model with a code-switching training dataset. The author of this individual research component managed to demonstrate how the implementation of SEETM can effectively handle OOV tokens and the following sub-sections elaborates on the attempts and techniques utilized to develop the SEETM approach.

4.5.1 The theory behind equivalent token mapping

The task of handling equivalent tokens is often given a low priority since modern deep learning-based complex ML models can effectively handle OOV tokens by producing relatively close word embeddings for equivalent tokens, synonyms, and other closely related words. In linguistics, the term *equivalent words* or *equivalent tokens* represents two or more words or phrases that can be interchanged within the same context. However, as clearly described in section 2, code-switching and code-mixing datasets that consists of data instances belong to two or more distinct languages can contain numerous equivalent tokens outspread across the dataset, and if the code-switching datasets utilized to train machine learning models are not highly representative from the perspective of equivalent words or phrases, the models often fail to detect equivalent tokens or handle equivalent words given as an input that they have not seen during the training. The author suggested that mapping all possible equivalent words or phrases that can occur within the domain of a chatbot to a single representation during model training can vastly improve the overall model performance at the model inference time. This idea was implemented and introduced as Equivalent Token Mapping (ETM) in this research component and SEETM is the specific implementation of ETM specifically designed to address the equivalent OOV token issue encountered in Sinhala-English code-switching datasets.

The concept of ETM expands beyond the context of chatbots and can be employed by any NLP model that suffers from the equivalent OOV tokens issue primarily caused by having a code-mixing or code-switching training dataset. ETM essentially maps various words or phrases with the same meaning to a single base word, given that they can occur within the same context. This prevents the NLP-based ML models from encountering multiple different representations of equivalent words while training and the model only train on the specified base tokens, reducing the model vocabulary. For instance, a word embedding model like Word2Vec gives distinct embeddings (vector representations) for the words *Degree*, ඩිග්‍රී, උපාධි, and *Upādi* because, even though the specified words are equivalent in the code-switching context, their characters are different and originating from different lexicons. Although modern ML models can figure out that they have a similar meaning, the embeddings given by the models are never going to be the same as their character representations are distinct. Instead of being exactly the same, the embeddings of equivalent words receive almost similar embeddings, indicating that they can be utilized interchangeably within the same context. Instead of getting almost similar embeddings, EMT is focused on getting the same embedding or feature set at the feature generation step for all equivalent tokens and phrases. There are two advantages of utilizing EMT.

1. With SEETM, the models can figure out that University එකේ තියෙන ඩිග්‍රීස් මොනවද? (*What are the degrees available at the University?*), University එකේ තියෙන degrees මොනවද? and University එකේ තියෙන උපාධි මොනවද? are the same. Without token mapping, the ML models are unaware that they are the **same** due to varying embeddings given to ඩිග්‍රීස්, degrees, and උපාධි.

2. In a scenario where an NLP model has seen only the words ඩිග්‍රීස් and degrees during training, the model will often detect the word උපාධි as a OOV token and that directly degrades the overall model performance.

Figure 4.5.1.1 and Figure 4.5.1.2 depict the theoretical aspect of SEEMT. The author has attempted various approaches to perform SEETM and sections 4.5.2 - 4.5.4 of the dissertation explain these approaches in detail.

4.5.2 A translation-based approach

To practically implement SEETM, the author first attempted a translation-based approach. It should be noted that this approach was later discarded due to several limitations explained in the latter part of this section in detail. Since the main goal of

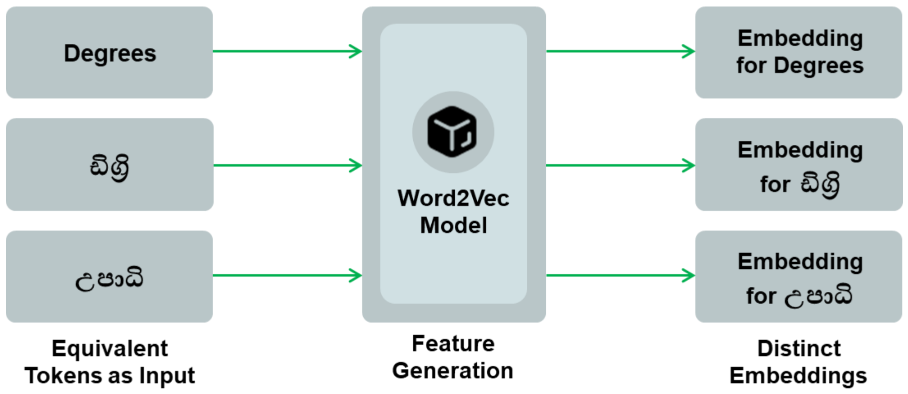


Figure 4.5.1.1. Generating Embedding without SEETM

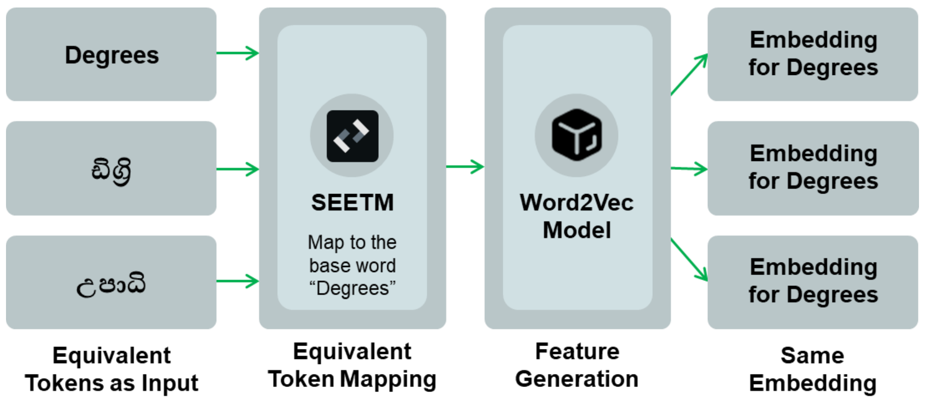


Figure 4.5.1.2. Generating Embedding with SEETM

SEEMT approach is to map equivalent tokens into a single representation, translation was considered as a possible technique that can essentially map English words to their corresponding Sinhala equivalent by translating the English version of the token into Sinhala. For an example, consider the equivalent text instances, *University එකේ තියෙන උපාධි මොනවද?* and *University එකේ තියෙන Degrees මොනවද?* (What are the degrees available at the University). In the given text instances, the words *උපාධි* and *Degrees* are equivalent since they are interchangeable within the same context. The translation approach attempted to automatically detect English tokens present in a specified text instance and translate them to Sinhalese. Figure 4.5.2.1 and Figure 4.5.2.2 depict the expected behavior of the translation-based approach.

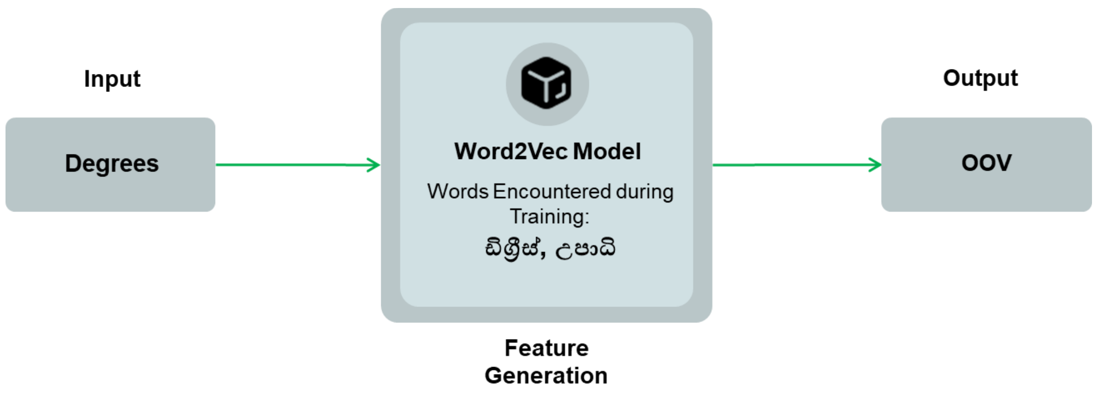


Figure 4.5.2.1. The model does not output a valid embedding since it has not encountered the word *Degree* during its training, and considers the word as an OOV token

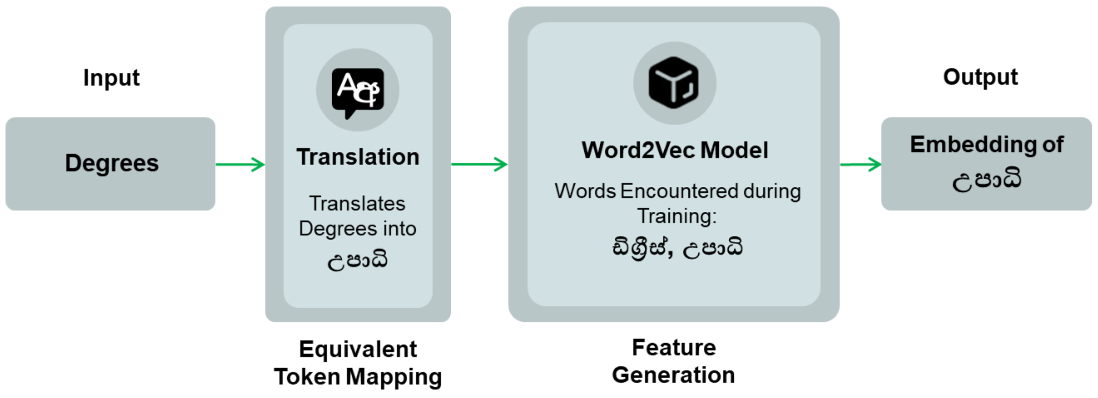


Figure 4.5.2.2. Handling OOV tokens through translating English words into Sinhalese

However, the translation-based approach was not utilized for the implementation of SEETM due to the following limitations.

1. Translation requires either an already existing pre-trained translator or training a translation model from scratch. Training a translation model from scratch was not considered as it did not align with the overall research timeline.

2. Existing English-to-Sinhala translators such as Google translator provided as a cloud service was not preferred due to often encountering obsolete words that are not utilized in day-to-day conversions in Sinhala, especially in a chatbot. (See Figure 4.5.2.3).

3. Translation-based SEETM is limited for a single token pair. For example, the translation based SEETM can only map *Degree* into *උපාධිය*, however, there can be additional equivalent tokens such as *ඩිග්‍රිය*, which are not covered by this technique.

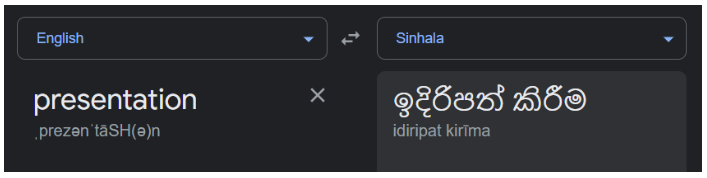


Figure 4.5.2.3. The translation of Presentation from English to Sinhala, generated the Google translator, which is not widely used when conversing in Sinhalese.

4.5.3 A pronunciation-based approach

After discarding the translation-based approach, the author attempted generating the Sinhala representation of English words, which the translation approach failed to accomplish, by employing a phonetics-based approach. The idea was to convert a given English token to an intermediate representation based on its pronunciation and then convert the intermediate state to the relevant Sinhala representation of the original word. The author accomplished this by utilizing the International Phonetics Alphabet (IPA) to convert a specific English word into the intermediate representation using the IPA character set. Since converting English words to IPA is a manual task done by professionals, the author utilized an existing Python package[6] that queries the Carnegie Mellon University (CMU) pronouncing dictionary and provides the IPA representation for valid English word inputs. However, the most challenging task was to convert the IPA representation into the respective Sinhala representation and the author had to develop IPA to Sinhala character mappings to accomplish this task. The generation of IPA to Sinhala character maps was done following the same strategy utilized while generating Sinhala to English character mappings described extensively in section 4.4.1. A set of IPA-based websites[7] that offered clear pronunciations of the IPA characters was also referred in the character mapping development phase. Figure 4.5.3.1 clearly depicts IPA to Sinhala character mappings developed by the author. It is noteworthy that some Sinhala pronunciations were mapped into more than one IPA characters primarily because the CMU pronouncing dictionary contained a few obsolete IPA characters. Table 4.5.3.1 depicts how a given English word is converted to the intermediate representation using English to IPA mappings and then the intermediate representation is converted to the Sinhala representation using the introduced IPA to Sinhala phonetic mappings. The introduced IPA to Sinhala character mappings were categorized based on the IPA character categorization as follows.

[6] The Python package utilized by the author to convert English words into the relevant IPA representation is available at https://pypi.org/project/eng-to-ipa/
[7] The websites related to IPA and phonetics referred are https://americanipachart.com/, http://www.speech.cs.cmu.edu/cgi-bin/cmudict, and https://en.wikipedia.org/wiki/International_Phonetic_Alphabet

1. Diphthongs – Pronunciation formed by two vowels utilized in the same syllable, indicated in blue in Figure 4.5.3.1.

2. Monophthongs – Pure vowels that are articulated the same manner, indicated in green in Figure 4.5.3.1.

3. Monophthongs + stress – Pure vowels present with stress characters employed by IPA alphabet to represent clear articulations, indicated in yellow in Figure 4.5.3.1.

4. Consonants – IPA characters that are pronounced with complete or partial vocal tract closure, indicated in gray in Figure 4.5.3.1.

Although the IPA to Sinhala character mappings were able to convert the phonetic representation of an English word to the Sinhala representation, it is only a partial conversion. For example, English to IPA and IPA to Sinhala mapping process only converts the word *Degree* to the partial Sinhala representation, ඩඉග්ර්ඊ. To properly convert the intermediate IPA mappings into the complete Sinhala representation, the author had to develop a comprehensive ruleset from scratch apply IPA to Sinhala character mappings and convert the partial Sinhala words obtained into complete Sinhala representation. The process followed a similar approach introduced and discussed in detail in section 4.4.2. The finalized IPA to Sinhala word conversion

Table 4.5.3.1. Conversion of English words Degree and Fee into the intermediate IPA mappings and then into the corresponding Sinhala representation

English Word		**English to IPA Mapping**	**IPA to Sinhala Mapping**	**Sinhala Representation**	
				Character Mapping	**Output**
1.	Degree	dɪˈgri	d → ඩ ɪ → ඉ g → ග් r → ර් i → ඊ	ඩ + ඉ → ඩි ග් + ර් + ඊ → ග්‍රී	ඩිග්‍රී
2.	Fee	fi	f → ෆ් i → ඊ	ෆ් + ඊ → ෆී	ෆී

eɪ ඒ	aɪ අයි	aʊ අව්	ɔ අව්	ɔɪ ඕ	oʊ ඕ	ˈæ ඇ	ˌæ ඇ	
I ඊ	u ඌ	ɪ ඉ	ʊ උ	ə අ	ɛ එ	ɝ අර්	æ ඈ	ɑ ආ
tʃ ච්	dʒ ජ්	p ප්	b බ්	t ට්	d ඩ්	k ක්	g ග්	θ ත්
ð ද්	f ෆ්	v ව්	s ස්	z ස්	ʃ ශ්	ʒ ෂ්	w ව්	m ම්
n න්	ŋ න්ග්	ɹ ර්	r ර්	j ය්	h හ්	ɫ ල්	l ල්	

Figure 4.5.3.1. IPA to Sinhala character mappings introduced based on pronunciation categorized and indicated as (1) blue – diphthongs, (2) green – monophthongs, (3) yellow – monophthongs + stress, and (4) gray – consonants

ruleset is depicted in Figure 4.5.3.2, and the justifications for the priority of execution given to each rule are as follows.

1. Rule 1: Diphthongs mapping

   Diphthongs were mapped first since Diphthongs and Monophthongs have overlapping IPA characters. If IPA Monophthongs were mapped first, Diphthongs eɪ (ඒ), aɪ (අයි), and ɔɪ (ඕ) characters are getting altered by the Monophthong ɪ (ඉ), which produces a faulty output.

2. Rule 2: Monophthongs + Stress mapping

   IPA characters can often contain stress marks, which represent necessary articulations. The author noticed that the original IPA Monophthong character æ (ඈ) presented with a preceding stress symbol becomes the Sinhala sound ඇ, which is a short vowel and not stressed when pronounced. If Monophthongs were first replaced (Rule 3) without considering the stress (this rule), A faulty output is generated that replaces ඇ with ඈ.

3. Rule 3: Monophthongs mapping

   After the Monophthongs with stress were mapped, the rest of the Monophthongs were mapped using this rule. However, there is no particular order for the Rule 3 and Rule 4 since IPA Monophthongs and Consonants do not contain any overlapping characters.

4. Rule 4: Consonants Mapping

   Consonants mapping rule maps the remaining IPA characters which are consonant sounds. When mapping consonants, the consonants tʃ (ච), and dʒ (ජ) were mapped first to avoid other consonant characters such as ʃ (ශ්), ʒ (ෂ්), and d (ඩ) from altering the individual characters, which produces faulty outputs.

5. Auxiliary Rule: Stress removal

   After the character mapping is done, the author observed that there are remaining stress characters utilized by CMU pronouncing dictionary and implemented the stress removal rules. This rule is responsible for removing the stress Symbols, including ˈ and ˌ to output a clean Sinhala character set.

6. Auxiliary Rule: Combination of Sinhala consonants and vowels

   Although rules until Rule 5 can produce a Sinhala word, as previously mentioned, the output is a partial output (*Degree* converted into ඩඉග්ර්ී). The role of the sixth rule is to combine the individual Sinhala sound representations and construct a valid Sinhala word. Since all characters present at this stage are Sinhalese, the author had to only implement two-character combination rules identical to the third and the fifth rule of English to Sinhala mapping rules introduced in section 4.4.2. The first combination rule combines *consonant* + *Rakarānshaya* + *vowel* combinations (maps ග් + ර් + ී into ග්‍රී). The second combination rule is responsible for combining *consonant* + *vowel* combinations (maps ඩ + ඉ into ඩි).

Table 4.5.3.2. Ruleset derived and the order of rule execution when converting an English word represented using IPA character mappings into its original Sinhala representation

Rule		**IPA to Sinhala Conversion:** dɪˈgri (IPA representation of Degree)	
		Character Mapping	**Output**
1.	Diphthongs Mapping	skipped as there are no Diphthongs	dɪˈgri
2.	Monophthongs + Stress Mapping	skipped as there are no Monophthongs with stress	dɪˈgri
3.	Monophthongs Mapping	ɪ → ඉ i → ඊ	dඉˈgrඊ
4.	Consonants Mapping	d → ඩ g → ග් r → ර්	ඩඉˈග්ර්ඊ
5.	Stress Removal	ˈ → empty string	ඩඉග්ර්ඊ
6.	Sinhala consonant + vowel combining	ග් + ර් + ඊ → ග්‍රී	ඩඉග්‍රී
		ඩ + ඉ → ඩි	ඩිග්‍රී

Although the IPA-based approach works fine as depicted in Table 4.5.3.1, the technique discussed in this section also comprises a set of limitations and the author had to find an alternative solution that addresses the limitations observed in the translation-based and pronunciation-based (IPA) techniques. The limitations encountered in the pronunciation-based approach are as follows.

1. Pronunciation of certain words is distinct in different countries or regions and the existing phonetic mappings extracted from CMU pronouncing dictionary are not universally acceptable. For example, the word *Banana* is pronounced as *bəˈnaːnə* in British IPA and as *bəˈnænə* in American IPA. The CMU comprises only North American pronunciations. Specially in Sinhala-English code-switching datasets, the author observed a few unrealistic mappings caused primarily due to varying accents and dialects. Table 4.5.3.2 contains a few unrealistic mappings the author observed during the implementation of the pronunciation-based approach.

2. The pronunciation based SEETM is limited to a single token pair. For example, it can only map Degree into ඩිග්‍රී, however, there can be additional equivalent tokens such as උපාධිය, which are not covered by this technique.

3. The proposed pronunciation based SEETM is a one-way conversion. Notice that the translation-based approach suggested can either translate a Sinhala word into English or vice-versa. However, in the pronunciation-based approach, the conversion happens only from English to Sinhala.

Despite the minor limitations, the author implemented the pronunciation approach as stated in Algorithm since it can be further developed and utilized as an English to Sinhala representation suggesting algorithm in Sinhala-centric applications and future Sinhala NLP research. Algorithm 2 clearly states how the author implemented English to IPA representation conversion in the pronunciation based SEETM approach. First, a specific text data instance is passed as an input to the algorithm and the algorithm tokenizes the instance using a *Whitespace Tokenizer*. Then the tokens are inspected one at a time and checked whether it exists in the CMU pronouncing dictionary to verify that the token is a valid English word that exists in the CMU pronouncing dictionary. If the token is a valid English token with an existing IPA mapping, it is then converted into the IPA representation using the previously mentioned *eng-to-ipa* Python package. All the IPA-mapped instances are then sent to the IPA to Sinhala mapping algorithm clearly stated in the Algorithm 3. The Algorithm 3 is responsible

```
Algorithm 2 PronunciationTokenMapping ($instance$)
1   $tokens$ ← Tokenize ($instance$)          ▷ Tokenizing the input text instance
2   $mappedInstance$ ← ""                       ▷ Initialize mapped instance to an empty
                                                  string
3   for each $token \in tokens$ do
4       if IsInCMU ($token$) do
5           $ipaRepresentation$ ← EngToIPAConvert ($token$)
6           $sinhalaWord$ ← IPAToSinhalaMapping ($ipaRepresentation$)
7           $mappedInstance$ ← Add ($sinhalaWord$)
8       else
9           $mappedInstance$ ← Add ($token$)
10  return $mappedInstance$
```

```
Algorithm 3 IPATOSINHALAMAPPING ($token$)
1   $output \leftarrow token$      ▷ Initialize the output to the text. This is a word
2   $diphMap \leftarrow$ INITIALIZE ($diphthongs$)      ▷ Maps are defined as either arrays or objects. Since maps are large variables, it is indicated as INITIALIZE in the algorithm
3   $monoStressMap \leftarrow$ INITIALIZE ($monophthongs\&Stress$)
4   $monoMap \leftarrow$ INITIALIZE ($monophthongs$)
5   $constMap \leftarrow$ INITIALIZE ($consonants$)
6   $stressMap \leftarrow$ INITIALIZE ($stress$)
7   $sinVowelMap \leftarrow$ INITIALIZE ($sinhalaVowels$)
8   $sinConstMap \leftarrow$ INITIALIZE ($sinhalaConsonants$)
9   $sinVowelSuffixMap \leftarrow$ INITIALIZE ($sinhalaVowelSuffixes$)

10  for each $diphChar, mappedDiphChar \in diphMap$ do      ▷ Rule #1
11      $output \leftarrow$ REPLACE ($mappedDiphChar , diphChar, output$)
12  for each $mStChar, mStMappedChar \in$      ▷ Rule #2
    $monoStressMap$ do
13      $output \leftarrow$ REPLACE ($mStMappedChar , mStChar, output$)
14  for each $monoChar, monoMappedChar \in monoMap$ do      ▷ Rule #3
15      $output \leftarrow$ REPLACE ($monoMappedChar , monoChar, output$)
16  for each $constChar, constMappedChar \in constMap$ do      ▷ Rule #4
17      $output \leftarrow$ REPLACE ($constMappedChar , constChar, output$)
18  for each $stChar, stMappedChar \in stressMap$ do      ▷ Rule #5
19      $output \leftarrow$ REPLACE ($stMappedChar , stChar, output$)
20  for each $constChar, constMappedChar \in sinConstMap$ do      ▷ Rule #6
21      for each $vChar, vMappedChar \in sinVowelMap$ do
22          $output \leftarrow$ REPLACE (C$onstMappedChar +$ ර් $+ vMappedChar,$
            $constChar +$ ◌්ර $+ vChar , output$)
23          $output \leftarrow$ REPLACE (C$onstMappedChar + vMappedChar,$
            $constChar + vChar , output$)
24  return $output$
```

for converting an IPA mapped instance to its Sinhala representation based on the IPA-to-Sinhala mappings depicted earlier in Figure 4.5.3.1 and the IPA to Sinhala conversion ruleset defined in Figure 4.5.3.2. The result of Algorithm 3 is returned back to the Algorithm 2 and the final output is constructed by combining individually mapped words. Figure 4.5.3.2 demonstrate the high-level concept of pronunciation based SEETM and how it addresses the OOV issue illustrated previously in Figure 4.5.2.1.

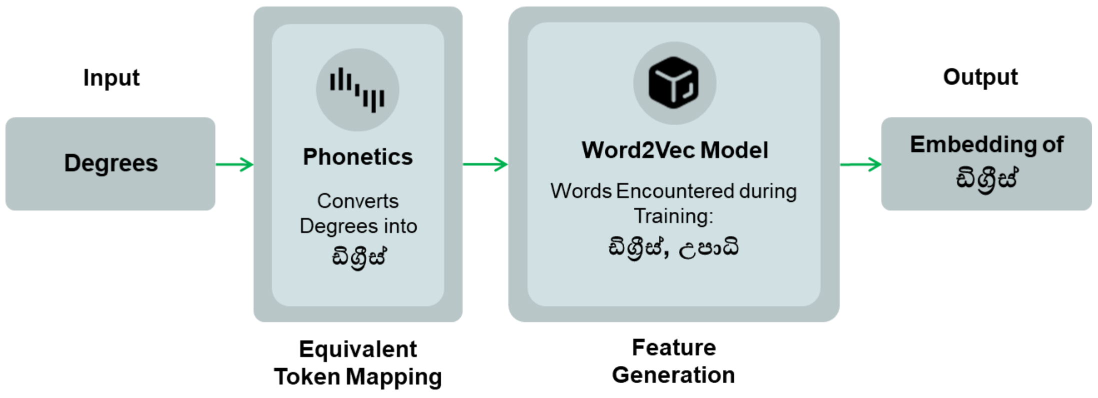


Figure 4.5.3.2. Handling OOV tokens through converting English words into Sinhala representation using pronunciation-based approach

To overcome the limitations of the pronunciation based SEETM, the author introduced rule based SEETM which is described in detail in the following section. Section 5.2 demonstrates the evaluation results obtained for both pronunciation-based and rule-based algorithms and clearly states why the rule-based approach is preferred over the pronunciation-based approach for performing SEETM.

4.5.4 The rule-based approach

The author introduced the rule based SEETM approach to address the limitations encountered in both the translation and pronunciation-based approaches previously introduced. The suggested approach is not that complex when compared to the previous approaches and relies on user defined equivalent token maps to map all equivalent words and phrases to a singe base word. Table 4.5.4.1 clearly demonstrates the idea of creating equivalent token maps. The first example given in Table 4.5.4.1 is a word directly taken from the English lexicon and the second example represents a neologism. Both examples demonstrate that rule-based maps can be utilized to map any type of equivalent words to a base word. The user-defined or simply rule-based SEETM addresses the following drawbacks of the previous techniques.

1. Rule-based mapping allows to map more than one token pair. Both translation and pronunciation-based approaches were limited to only one mappable token pair. The translation-based approach can only detect that *degrees* and උපාධි are

equivalent while the pronunciation-based approach is only able to detect that *degrees* and ඩිග්‍රීස් are equivalent. However, the rule-based approach can be configured to detect *degrees,* ඩිග්‍රීස්, and උපාධි are all equivalent words.

2. Rule-based mapping does not employ automatic translation or representation conversion from a specified language to another. As explained in sections 4.5.2 and 4.5.3, translation and pronunciation-based approaches primarily relied on the idea of automatically converting word representations based on translation or convert the representation of a word of one language to another based on character mappings. In rule-based approach, equivalent token maps are constructed primarily depending on the domain of the ML model training datasets and the developers of the ML models can decide which equivalent words they require to map to a base word and construct a set of custom mapping rules.

As explained in section 4.5.1, with SEETM is enabled, the ML models will be trained on only the base word. The model is entirely unaware of the other equivalent words mapped to the base, and the correct identification of equivalent words that can be interchanged within the same context is a crucial task in constructing rule-based equivalent token maps. Developers of the ML models with code-switching datasets should study the dataset thoroughly before constructing rule-based token maps and correctly and clearly identify what are the possible equivalent tokens and which base word is the most suitable. It is always wise to select a widely utilized base word since it can occur frequently in the training data instances and after the model is deployed in a production setting. The Algorithm 4 elaborates on how rule based SEETM can be implemented.

Table 4.5.4.1. The concept of user-defined equivalent token maps

Base Token		**Equivalent Tokens**
1.	උපාධි	උපාධි, degrees, ඩිග්‍රි, ඩිග්‍රිස්, ඩිග්‍රී, ඩිග්‍රීස්
2.	sliit	sliit, SLIIT, ශ්‍රී ලංකා තොරතුරු තාක්ෂණ ආයතනය, ස්ලිට්, ස්ලිට් එක

Algorithm 4 RULEBASEDTOKENMAPPING ($instance$)

```
1  $output \leftarrow instance$        ▷ Duplicating input instance for string replacement
2  $ruleBasedMaps \leftarrow$ INITIALIZE ($ruleBasedMaps$)        ▷ Initialize the user-defined token mappings
3  $ruleBasedMaps \leftarrow$ SORT ($ruleBasedMaps$ , **reverse**)        ▷ Sort the token mapping list in the descending order
4  **for each** $baseToken, eqToken \in ruleBasedMaps$ **do**
5      $output \leftarrow$ REPLACE ($eqToken, baseToken, output$)        ▷ Replacement should be done only at word level. A word should not be partially replaced
6  **return** $output$
```

The *RuleBasedTokenMapping* algorithm first retrieves all user-defined mappings and sorts all equivalent tokens regardless of the base word. The sorting is done in the descending order to avoid replacing individual words in phrases before replacing phrases. For example, if SEETM comprise two token maps as (1) ඩිග්‍රී එක, උපාධිය mapped to the base word *degree* and (2) ඩිග්‍රී, උපාධි mapped into the base word *degrees*, and when processing the instance, මට IT ඩිග්‍රී එක කරන්න පුලුවන්ද, SEETM might replace only the ඩිග්‍රී word in the phrase ඩිග්‍රී එක with *degrees* according to the second token map specified and the instance then becomes මට IT degrees එක කරන්න පුලුවන්ද, which is wrong. Looking at the example, it is evident that longer phrases must be replaced first to avoid partial text replacements and sorting in descending order was incorporated to accomplish that. After the sorting, the algorithm iterates through all the maps and replaces equivalent words with its relevant base word and finally, the mapped instance is returned.

Section 4.5.5 offers a clear and concise walkthrough on the implementation process the author has undergone during the development of SEETM done using Python language.

4.5.5 SEETM Tokenizer

Since the author introduced SEETM as a data pre-processing technique, SEETM tokenizer was developed to attach token mapping support to the NLU pipeline of Rasa chatbot projects. SEETM tokenizer is an extension of the existing Whitespace Tokenizer component available in Rasa chatbot development framework, and the functionality was extended by implementing the SEETM just before the tokenization step. The reason behind performing SEETM before the tokenization step is that otherwise it is impossible to replace phrases with the base word if the tokenizer separates them by the whitespace. Figure 4.5.5.1 illustrates the importance of implementing SEETM before the tokenization step. The extended tokenizer was attached to the Rasa NLU pipeline as depicted in Figure 4.5.5.2.

Rasa loads the training data and passes them to the SEETM tokenizer as depicted in Figure 4.5.5.2 and the SEETM tokenizer performs ETM on the data instances prior to the tokenization step and maps all equivalent words defined by the developer to their respective base word. The NLU pipeline then does not encounter any of the equivalent words replaced, which ensures the advantages stated in section 4.5.1 are met.

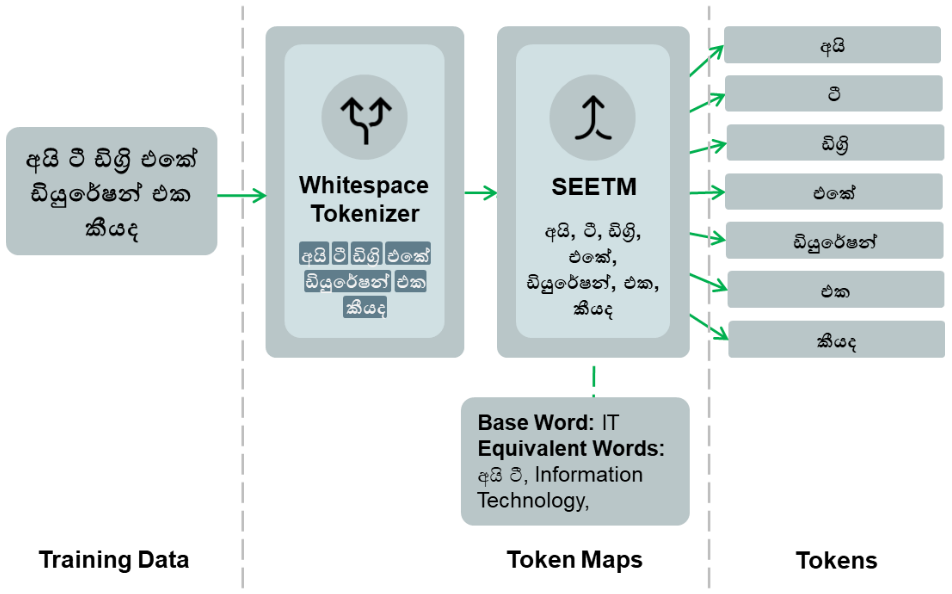
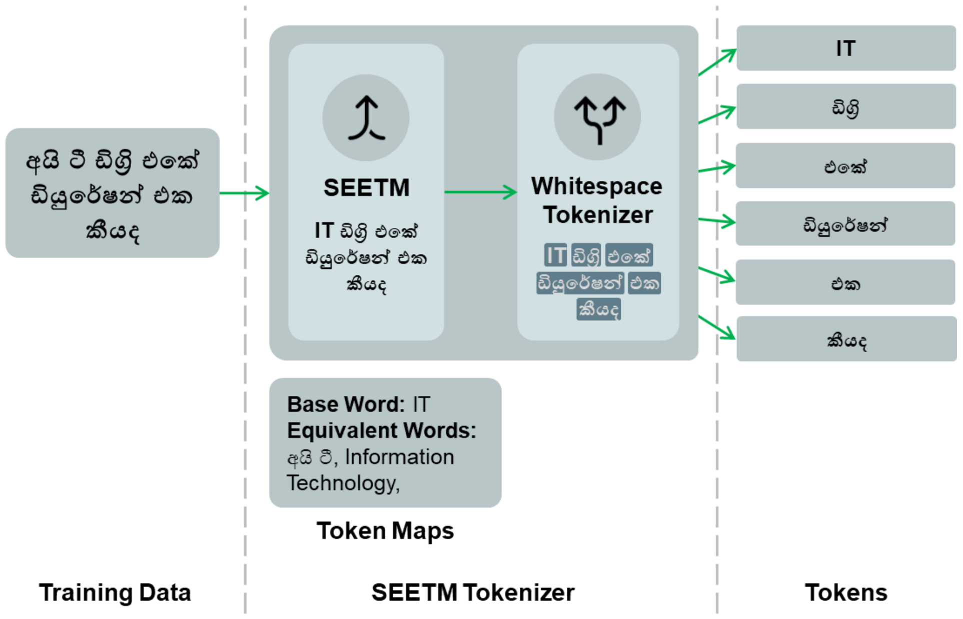

අයි ටී ඩිග්‍රි එකේ ඩියුරේෂන් එක කීයද
Whitespace Tokenizer
අයි ටී ඩිග්‍රි එකේ ඩියුරේෂන් එක කීයද
SEETM
අයි, ටී, ඩිග්‍රි, එකේ, ඩියුරේෂන්, එක, කීයද
Base Word: IT
Equivalent Words: අයි ටී, Information Technology,
අයි
ටී
ඩිග්‍රි
එකේ
ඩියුරේෂන්
එක
කීයද
Training Data
Token Maps
Tokens

Figure 4.5.5.1. SEETM attached after tokenization does not maps equivalent tokens as expected



අයි ටී ඩිග්‍රි එකේ ඩියුරේෂන් එක කීයද
SEETM
IT ඩිග්‍රි එකේ ඩියුරේෂන් එක කීයද
Whitespace Tokenizer
IT ඩිග්‍රි එකේ ඩියුරේෂන් එක කීයද
IT
ඩිග්‍රි
එකේ
ඩියුරේෂන්
එක
කීයද
Base Word: IT
Equivalent Words: අයි ටී, Information Technology,
Token Maps
Training Data
SEETM Tokenizer
Tokens

Figure 4.5.5.2. SEETM Tokenizer that combines SEETM and the whitespace tokenizer, preserving the order of the components.

4.6 SEETM Python package

The pronunciation and rule based SEETM techniques discussed in section 4.5.3 and 4.5.4 and the SEETM tokenizer introduced in section 4.5.5 were implemented as a modular Python package using Python as the primary language. The Python package was named SEETM[8] and published to the Python Package Index (PyPI) as an open-source package under the Apache 2.0 license. SEETM fully supports Rasa 2.2.8 through 2.8.8 and Rasa-based chatbot developers can get SEETM installed on their machines simply by running *pip install seetm*. The SEETM package is continuously released by the author with proper versioning done according to the semantic versioning[9] and the source code of SEETM[10] is publicly available on a GitHub repository along with all the PyPI package releases. Interested individuals are given the opportunity to contribute to the project via submitting issues, pull requests (PRs), and the source code is also available under the Apache 2.0 license which requires

[8] The SEETM Python package is available at https://pypi.org/project/seetm/
[9] More information on semantic versioning can be found at https://semver.org/
[10] The source code of SEETM is publicly available at https://github.com/SEETM-NLP/seetm

individuals who improve or modify the code base to clearly state the changes made, making the source code further reliable.

Since developers need to integrate SEETM tokenizer to their Rasa NLU pipeline to enable token mapping, the compatibility between the SEETM and Rasa Python packages matters when integrating the two packages. Table 4.6.1 comprises the compatibility matrix between SEETM and Rasa Python package releases.

4.6.1 SEETM CLI Interface

The SEETM Python package is fully compatible with the CLI and the primary CLI interface offers exactly five sub-interfaces, namely, SEETM Init, SEETM Map, SEETM Server, SEETM Extract and SEETM Evaluate. This section of the dissertation clarifies the intent of each interface in detail, whereas Figure 4.6.1.1 depicts the interfaces offered by SEETM and their purpose briefly.

The main SEETM CLI interface is responsible for parsing arguments passed via *seetm* CLI command after downloading and installing the SEETM Python package. The main CLI interface is responsible for handling ETM requests via invoking the respective sub-interfaces. Table 4.6.1.1 contains different arguments and optional arguments supported by the main SEETM CLI interface.

Table 4.6.1. SEETM and Rasa compatibility matrix

SEETM Version		**Rasa Open-Source Version**
MAJOR.MINOR	**PATCH (Released Bug fixes)**	
1.1.x	1.1.0, 1.1.1	2.2.8 - 2.8.8
1.0.x	1.0.2, 1.0.1, 1.0.0	
0.0.x	0.0.1a2, 0.0.1a1	

Table 4.6.1.1. Main SEETM CLI Interface arguments

CLI Command	**Purpose**
Main Command	
seetm	Entry point of the primary SEETM CLI Interface.
Optional Arguments	
-v or --version	Outputs SEETM package version details to CLI.
-h or --help	Provides instructions on how to use SEETM CLI.

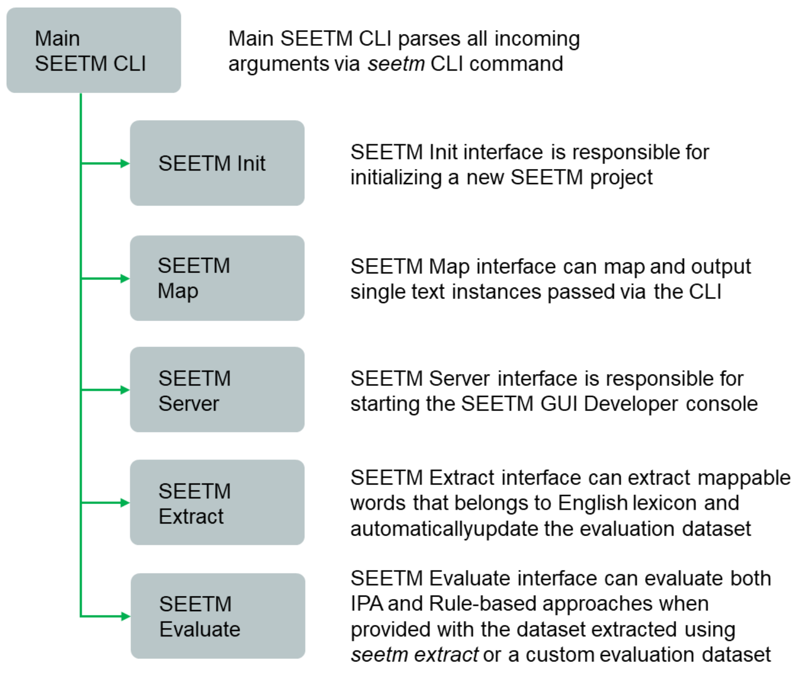

Main SEETM CLI
Main SEETM CLI parses all incoming arguments via *seetm* CLI command
SEETM Init
SEETM Init interface is responsible for initializing a new SEETM project
SEETM Map
SEETM Map interface can map and output single text instances passed via the CLI
SEETM Server
SEETM Server interface is responsible for starting the SEETM GUI Developer console
SEETM Extract
SEETM Extract interface can extract mappable words that belongs to English lexicon and automaticallyupdate the evaluation dataset
SEETM Evaluate
SEETM Evaluate interface can evaluate both IPA and Rule-based approaches when provided with the dataset extracted using *seetm extract* or a custom evaluation dataset

Figure 4.6.1.1. Interfaces and sub-interfaces offered by SEETM Python package

Table 4.6.1.2 through Table 4.6.1.6 contains the positional and optional arguments that can be passed to each SEETM sub-interface, and the intent of the arguments passed.

Table 4.6.1.2. SEETM Init sub-interface arguments

CLI Command	Purpose
Sub-interface Command	
seetm init	Initializes a new SEETM project in a specified directory.
Optional Arguments	
--debug	Sets the logging level to debug mode from info mode.
--quiet	Initializes a new project without prompting the user for a project location. The new project is initialized as the same directory where the CLI command originated from.

Table 4.6.1.3. SEETM map sub-interface arguments

CLI Command	Purpose
Sub-interface Command	
seetm map	Runs SEETM CLI mapper, a terminal based SEETM tool.
Optional Arguments	
-i or --instance	Specify data instance to be mapped.
-m or --method	Specify the method to use for SEETM (*ipa* or *rule-based*).
-c or --config	Specify a custom SEETM configuration file.
--no-case	Preserve or ignore the case sensitivity of the data instance provided. If no case is enabled, data instance is converted to lower-case (not recommended as it reduces input features).
--no-persist	If specified, SEETM CLI mapper skips persists mapping results as JSON objects in the *seetm exports* directory.
--debug	Sets the logging level to debug mode from info mode.
--quiet	Runs SEETM mapper without printing any logging statements

Table 4.6.1.4. SEETM server sub-interface arguments

CLI Command	Purpose
Sub-interface Command	
seetm server	Run SEETM server, a web-based GUI map constructing tool.
Optional Arguments	
-p or --port	Specify the DIME server port. The default port is 6066
-c or --config	Specify a custom SEETM configuration file
-m or --method	Specify the method to use for SEETM (*ipa* or *rule-based*).
--debug	Sets the logging level to debug mode from info mode.

Table 4.6.1.5. SEETM extract sub-interface arguments

CLI Command	Purpose
Sub-interface Command	
seetm extract	Run SEETM CLI extractor that allows extracting valid English words present in a Sinhala-English code-switching dataset
Optional Arguments	
-d or --data-path	Specify the dataset path. Default is *data* directory in the project root
-c or --config	Specify a custom SEETM configuration file for retrieving the dataset path
--clean	If specified, the existing evaluation dataset is deleted. Otherwise, obsolete data points are removed, and new data points are added while preserving valid data points in the existing evaluation dataset which is the preferred option.
--debug	Sets the logging level to debug mode from info mode.
--no-persist	If specified, SEETM extractor generates the evaluation dataset but does not persist it
--quiet	Runs SEETM extractor without printing any logging statements

Table 4.6.1.6. SEETM evaluate sub-interface arguments

CLI Command	Purpose
Sub-interface Command	
seetm evaluate	Run SEETM CLI evaluator that allows to evaluate the performance of IPA (pronunciation based) or Rule-based approaches against the evaluation dataset generated by the extractor or a custom evaluation dataset specified.
Optional Arguments	
-m or --method	Specify the method to use for SEETM (*ipa* or *rule-based*).
-c or --config	Specify a custom SEETM configuration file for retrieving the dataset path
-m or --metric	Allows evaluating using either *edit-distance* or *sequence matching*. This is only valid when --level is set to *token*.
-l or --level	Specify the evaluation level. If set to *token* level, it evaluates only IPA mappings. If set to *vector* level, the SEETM extractor trains two *Word2Vec* models and evaluates the performance based on the vector similarity against the evaluation dataset. (When evaluating the Rule-based approach, only the data available in token maps constructed are tested)
-e or --epochs	Specify the number of epochs to train the *Word2Vec* models when --level is set to *vector*.
--debug	Sets the logging level to debug mode from info mode.
--no-persist	If specified, SEETM extractor generates the evaluation dataset but does not persist it
--quiet	Runs SEETM extractor without printing any logging statements

4.6.2 SEETM project structure

The SEETM Init sub-interface lets developers initialize a SEETM project from scratch and integrate it with an existing or entirely new Rasa project. The *seetm init* command generates an initial project structure as depicted in Figure 4.6.2.1 which generates *seetm_components*, *seetm_eval*, *seetm_exports*, and *seetm_maps* directories and the *seetm_config.yml* configuration file. The *seetm_components* directory contains the SEETM tokenizer that is required to be added to the target Rasa projects NLU pipeline by the developer. Without doing so, the token maps constructed have no effect on chatbot model training or inference. Figure 4.6.2.2 and Figure 4.6.2.3 depict how the SEETM tokenizer can be added to the Rasa NLU pipeline configuration file. The *seetm_exports* directory contains all CLI generated mappings and mappings generated while training a new Rasa chatbot model. The *seetm_eval* directory comprises the evaluation dataset that the developers can automatically generate via *seetm extract* or a custom evaluation dataset prepared for testing the performance of the IPA (pronunciation-based) or Rule-based SEETM techniques, and the list of previously generated evaluation reports. Finally, the *seetm_maps* directory consists of SEETM mapping files where rule-based mappings are persisted.

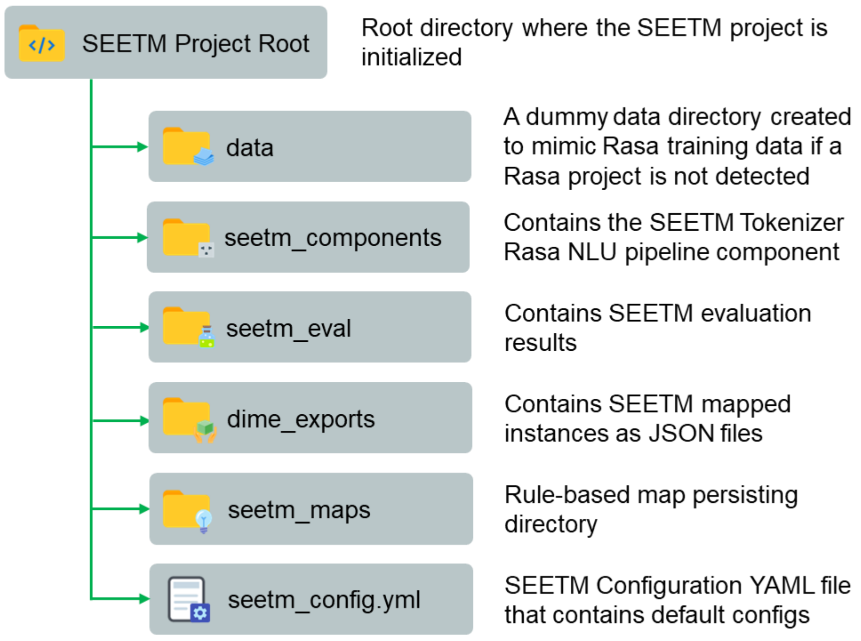
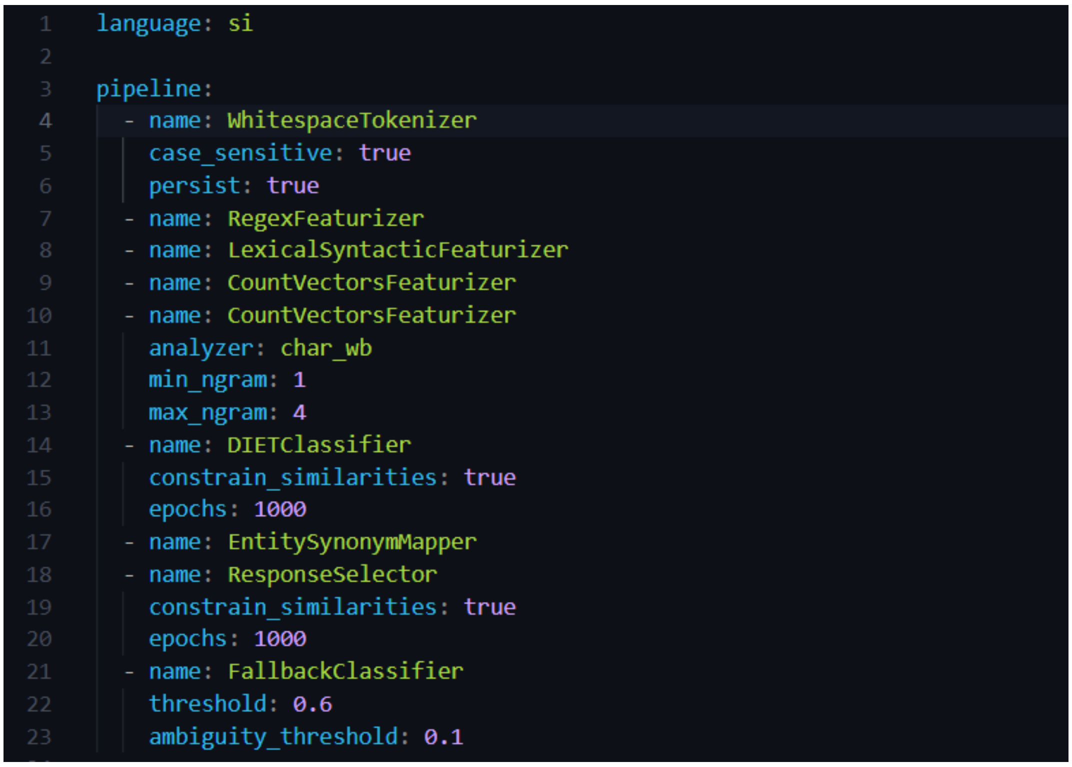
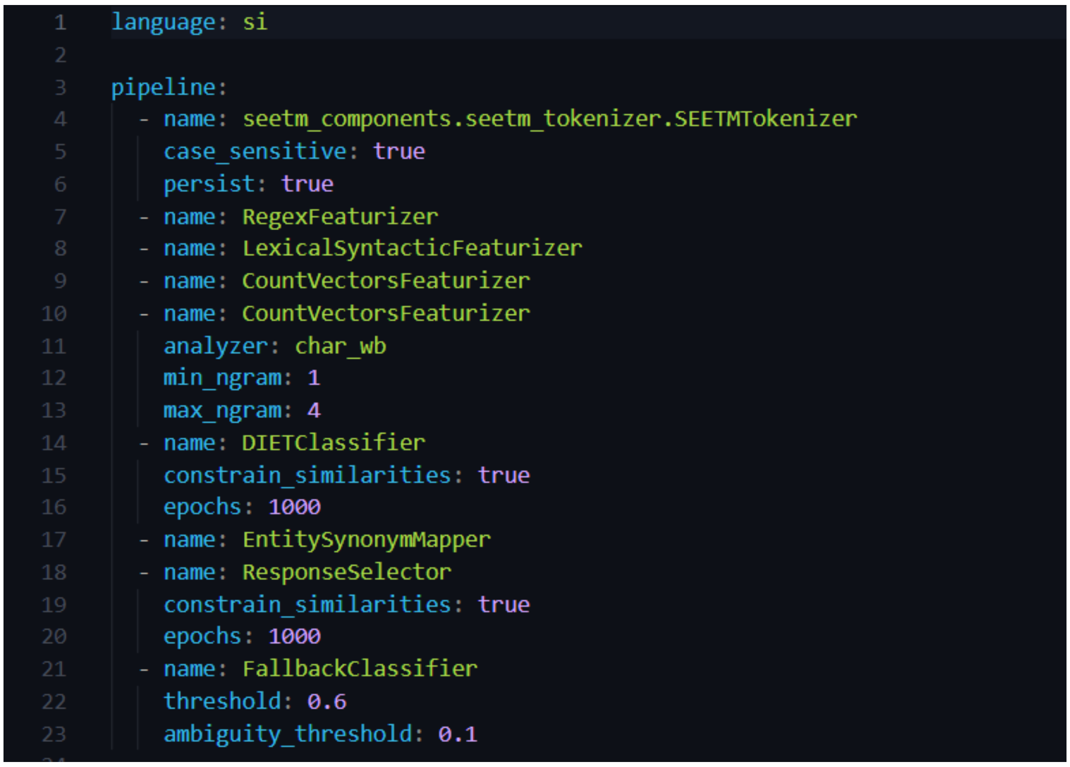

SEETM Project Root
Root directory where the SEETM project is initialized
data
A dummy data directory created to mimic Rasa training data if a Rasa project is not detected
seetm_components
Contains the SEETM Tokenizer Rasa NLU pipeline component
seetm_eval
Contains SEETM evaluation results
dime_exports
Contains SEETM mapped instances as JSON files
seetm_maps
Rule-based map persisting directory
seetm_config.yml
SEETM Configuration YAML file that contains default configs

Figure 4.6.2.1. SEETM project directory structure



```yaml
language: si

pipeline:
  - name: WhitespaceTokenizer
    case_sensitive: true
    persist: true
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: char_wb
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    constrain_similarities: true
    epochs: 1000
  - name: EntitySynonymMapper
  - name: ResponseSelector
    constrain_similarities: true
    epochs: 1000
  - name: FallbackClassifier
    threshold: 0.6
    ambiguity_threshold: 0.1
```

Figure 4.6.2.3. Rasa NLU pipeline before integrating SEETM tokenizer



```yaml
language: si

pipeline:
  - name: seetm_components.seetm_tokenizer.SEETMTokenizer
    case_sensitive: true
    persist: true
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: char_wb
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    constrain_similarities: true
    epochs: 1000
  - name: EntitySynonymMapper
  - name: ResponseSelector
    constrain_similarities: true
    epochs: 1000
  - name: FallbackClassifier
    threshold: 0.6
    ambiguity_threshold: 0.1
```

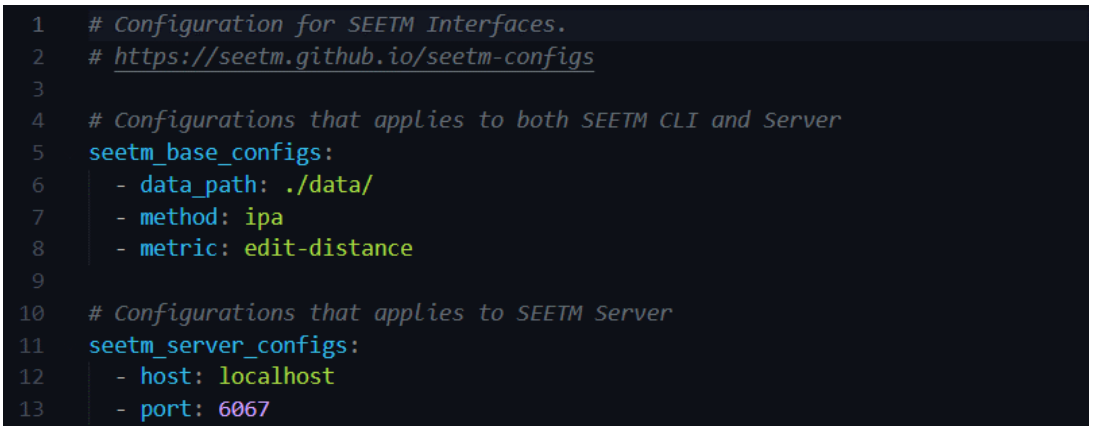
Figure 4.6.2.2. Rasa NLU pipeline after integrating SEETM tokenizer

4.6.3 SEETM default configurations

The default SEETM configuration file mentioned in section 4.6.2 comprises initial settings required to run SEETM Server, SEETM Map, SEETM Extract, and SEETM Evaluate interfaces. Figure 4.6.3.1 depicts the content of *seetm_config.yml* where settings are defined under two main configuration categories, *seetm_base_configs* and *seetm_server_configs*. Table 4.6.3.1 contains all customizable SEETM configuration settings, default values, and allowed options.

Table 4.6.3.1. SEETM configuration settings, default values, and allowed options

Configuration	Value		Description
Base configurations			
data_path	*Default value*	./data/	Specifies the Rasa data directory. If a separate dataset is used for mapping, the directory path should be specified explicitly
	Valid values	a valid directory path where Rasa data files reside	
method	*Default value*	rule-based	SEETM method for the SEETM CLI mapper
	Valid values	Ipa, rule-based	
metric	*Default value*	edit-distance	Evaluation metric to compare the quality of IPA mapped instances compared to the expected output
	Valid values	edit-distance, sequence-matching	
Server configurations			
host	*Default value*	localhost	Host name to host the SEETM server
	Valid values	localhost, 0.0.0.0, 127.0.0.1	
port	*Default value*	6067	Port to start the SEETM server
	Valid values	a valid port number	



```yaml
# Configuration for SEETM Interfaces.
# https://seetm.github.io/seetm-configs

# Configurations that applies to both SEETM CLI and Server
seetm_base_configs:
  - data_path: ./data/
  - method: ipa
  - metric: edit-distance

# Configurations that applies to SEETM Server
seetm_server_configs:
  - host: localhost
  - port: 6067
```

Figure 4.6.3.1. The *seetm_config.yml* file

4.6.4 SEETM Server

The SEETM Server interface is noteworthy as it offers the developers with an easy-to-use GUI for constructing equivalent token maps. The author implemented the GUI using React as the primary front end development framework. The developed front end was built and served using the flask server built into the SEETM Python package and can be deployed both on debug and production level modes. The UI facilitates adding new token maps, deleting existing maps, or editing the content of already constructed equivalent token maps. When adding a new token map, developers are given the option to define a base word and a set of equivalent words and persist it to the *seetm_maps* directory effortlessly. There are validations in place to make sure that the same mappings are not added more than once and two ore more maps cannot contain the same equivalent words or base words. Another important fact is that SEETM has the built-in support for typing support for both Sinhala and English within the UI as clearly explained in section 4.4.5. See Appendix C for the complete list of SEETM server user interfaces.

4.6.5 Integrating with the overall research

The overall SEETM Python package was then integrated with *Kolloqe*, the overall research output which is essentially a chatbot development platform built by integrating all four individual research components. The SEETM UI integrated with Kolloqe allows developer to define equivalent token maps and enable the SEETM tokenizer through its pipeline configuration user interface easily without having to manually configure the backend configuration files. In the pipeline configuration UI of Kolloqe, developers can turn SEETM on or off by switching between SEETM tokenizer and Whitespace tokenizer as required. Figure 4.4.5.2 previously attached in section 4.4.5 depicts the SEETM token mapping UI that was integrated with Kolloqe. Section 5.1 highlights and elaborates more on the final outcome of the individual research, whereas sections 5.2.1 and 5.2.4 clearly demonstrates how different approaches of SEETM was evaluated by the author. Section 5.2.6 further demonstrates the impact of rule-based SEETM techniques on Rasa conversational AIs by a human-grounded experiment conducted.

4.7 Tools and Technologies

4.7.1 Sinhala-English code-switchable keyboard development

As explained in sections 4.4.3 and 4.4.4, the author utilized React front-end development framework running on Node 16.15.0 to develop the Sinhala-English code-switchable keyboard interface and integrate it seamlessly with the chat widget developed as a modular react component. Although the chat widget was not published to NPM, a separate react input component was developed with built-in support for Sinhala-English code-switchable typing support and published to NPM under *@Kolloqe/input*. To manage the versioning of the keyboard input component, semantic versioning was utilized, and the source code of the component is publicly available in a public GitHub repository under the Apache 2.0 license. To develop the chat widget UI, Material UI (MUI) react component library was incorporated.

4.7.2 SEETM package development

The author selected Python 3.8 to implement the SEETM Python package since Rasa is a primary backend dependency of SEETM and it requires either Python 3.7 or 3.8[11]. Rasa open source was utilized to extend its Whitespace Tokenizer as SEETM tokenizer and to test SEETM implementation. The *eng -to-ipa* Python package was utilized to implement the pronunciation-based ETM technique. The *gensim* and *rich* Python packages were used to train Word2Vec models for evaluating IPA and rule based SEETM techniques and display the task progress in CLI respectively. The *regex* Python package was utilized to perform string replacement conveniently for Sinhala Unicode characters.

4.7.3 SEETM server development

The backend of the SEETM server integrated into the SEETM Python package is a Python Flask based server and *waitress*[12] was utilized as a Web Server Gateway

[11] The installation requirements of Rasa are clearly stated at https://rasa.com/docs/rasa/2.x/installation
[12] *waitress* is a pure Python WSGI server that supports production-grade deployments, available at https://pypi.org/project/waitress/

Interface (WSGI) server to enable support for production level deployments. SEETM API is a fully flask-based API that was reconfigured to serves the React front-end build files.

4.7.4 SEETM front-end development

SEETM server front-end was developed using React installed on Node version 16.15.0. The *MUI* react component library was utilized to implement the required user interfaces and *axios* and *uuid* were incorporated to handle asynchronous API requests and generate unique identifications required for backend API requests.

4.7.5 Source code, versioning, and release management

PyCharm 2022.1.3 was utilized as the primary Integrated Development Environment (IDE) for Python development and Visual Studio Code (VS Code) was used for React based front-end development. Anaconda Python distribution used to effectively manage virtual environments for Python dependency management as there were numerous packages involved with the backend development tasks. As previously mentioned, semantic versioning was used to create the releases for both Sinhala-English keyboard interface and the SEETM Python package while individual public GitHub repositories were used to maintain the open-source packages with required documentation and version releases. Each stable version of SEETM was built, released, and pushed to PyPI to make the package accessible for the majority of Rasa developers.

The complete list of tools and technologies used in this research component are concluded in Table 4.7.5.1.

Table 4.7.5.1. Tools and Technologies used throughout the individual research

Task	**Tools and technologies utilized**
Sinhala-English code-switching Keyboard Interface	Node 16.15.0, MUI 5.8.4
React chat widget	Node 16.15.0, MUI 5.8.4, socket.io, axios 0.27.2, framer-motion 6.3.16, react-indiana-drag-scroll 2.2.0, uuid 8.3.2, VS code
SEETM Python package	Python 3.8, Python Dotenv, Ruamel YAML, rich, Rasa 2.8.8, Flask, Flask-cors, Waitress, Gensim, Regex, PyCharm, Anaconda Distribution
SEETM server frontend	Node 16.15.0, MUI 5.8.4, axios 0.27.2, framer-motion 6.3.16, Bootstrap 5, React JS, prop-types 15.8.1, react-router 6.3.0
Chatbot development	Rasa 2.8.8, Rasa-SDK 2.8.0
Source code and version management	GitLab, GitHub, semantic-versioning, NPM, PyPI
Testing	pytest, CoLab

4.8 Commercialization Aspect of the Research Outcomes

The commercialization value that the Sinhala-English code-switchable keyboard interface holds is immeasurable as it can be employed by a wide variety of tools to extend typing support in websites beyond English. Despite that, the author decided to release the solution as an open-source tool to make the research accessible to a wider audience. However, the Sinhala-English code-switchable typing supported webchat widget was not released to the public as doing so would decrease the competitive advantage of *Kolloqe*, the overall research solution.

The SEETM Python package has a higher commercial value as well. SEETM was introduced as a novel pre-processing technique for Sinhala-English code-switching datasets and there are no other production-grade tools that can compete with SEETM. However, the author has released both Sinhala-English code-switchable keyboard interface and the SEETM Python package as open-source tools because of the following reasons.

1. There is a lack of freely accessible Sinhala-English NLP research, the keyboard interface and SEETM package allows the ML and NLP communities to gain

open access to the free and open-source tools and their source code as an step taken to make the research accessible to a wider audience.

2. The open-sourced tools have a higher potential to gain more attention of the community and it allows the author of this individual research to improve the tools by inspecting and fixing the issues and bugs reported, addressing the limitations and drawbacks of the current implementation, and constantly improve the tools with invaluable community contributions.

3. The open-sourced tools have the primary objectives of increasing the interest in conducting novel Sinhala and Sinhala-English related research and promoting open-access to invaluable research.

4. The Sinhala-English code-switchable keyboard input component was released as an open-source tool mainly to demonstrate how the native typing support in websites can be extended beyond English for Sinhala and other languages since the author has not found any other tool or website that is capable of doing so0.

However, both Sinhala-English code-switchable keyboard interface and SEETM implementation provides *Kolloqe*, the overall research solution which is essentially a cloud-based chatbot development platform, the competitive advantage over widely utilized chatbot building frameworks, including Google DialogFlow and Rasa, by enabling native support for Sinhala-English code-switching typing in the chat widget and within the user interface and allowing the platform to handle OOV tokens in code-switching datasets accordingly. The following points elaborate on how the commercial value of Kolloqe was increased by the individual research component outcomes.

1. SEETM provides Kolloqe the ability to effectively handle OOV token issues caused by Sinhala-English code-switched datasets.

2. SEETM allows Kolloqe to increase the overall model performance by effectively handling OOV tokens even when a chatbot model is deployed for production.

3. SEETM increases the overall model performance by allowing the developers train chatbot models on a single base word by replacing multiple equivalent words, which allows the models to comprehend that equivalent words must be given the same exact features.

4. The Sinhala-English code-switchable keyboard interface allows Kolloqe to provide native support for typing in Sinhala, English or both Sinhala-English within the same text instance, throughout the Kolloqe developer console. It is noteworthy that the author did not encounter any similar tools or websites that has extended the native typing support in websites beyond English.

5. The Sinhala-English code-switchable keyboard interface utilized in Kolloqe chat widget allows both developers, testers, and end-users of the chatbot to frequently test the chat widget by typing code-switching queries natively, and it also provides Kolloqe the ability to develop chatbots that is accessible to a wider population, increasing its market value over similar products.

5. TESTING & IMPLEMENTATION RESULTS & DISCUSSION

5.1 Result

The author of this individual research component managed to deliver two primary research outcomes, (1) The Sinhala-English code-switchable keyboard interface and (2) The Rasa 2.8.8 supported Sinhala-English Equivalent Token Mapping Python implementation, that achieved the objectives and the requirements stated in sections 3 and 4.2 respectively.

The Keyboard solution enables native support for typing Sinhala, English, and Sinhala-English code-switching queries and was integrated into a chat widget developed using React as a modular react component. The keyboard interface also integrated into all the user interfaces of the SEETM and Kolloqe developer consoles, enabling developers to type Sinhala in the UI instantly while training and testing chatbots. The implementation of the keyboard was done in the front-end to increase

Table 5.1.1. Summarized SEETM experiment setup details

Bot #	**Dataset Size**	**Difference of Bots**	**Epochs**	**Validation Accuracy**
Bot-1	Total: 50 Train: 35 Test: 15 Total intent: 4	Bot with mappings	25	96.67%
Bot-2	Total: 50 Train: 35 Test: 15 Total intent: 4	Bot without mappings	25	90%

the efficiency when providing a real time output to its end-users. An open-source version of the keyboard interface has been made available via NPM and the source code is publicly available as previously mentioned.

The SEETM Python implementation was done for both pronunciation and rule-based techniques and provides support for both CLI and web-based GUI. Developers can construct equivalent token maps using the UI or directly editing the mapping files in the relevant project directories and effectively handle the OOV token issues caused by Sinhala-English code-switching datasets when training Rasa chatbots. Any Rasa developer is able to download and install SEETM using PyPI which offers full support for Rasa 2.8.8, one of the actively maintained recent version of Rasa chatbot development framework.

Both research outcomes allow processing Sinhala-English code-switching data effectively and the importance of the open-source releases is clearly mentioned in section 4.8. Most importantly, the outcomes of the individual research add a tremendous amount of commercial value to the overall research outcome, which is clearly stated in the same section of this dissertation. Section 5.2 demonstrates overall research findings and the human-grounded experiments done to properly test both solutions provided by the individual research.

5.2 Research Findings

Sections 5.2.1 to 5.2.4 of this dissertation clearly demonstrate the experiment setups, alpha testing carried out by the developers, integration testing done in each integrated research outcome, and beta testing done based on human-grounded experiments for a proper evaluation of the research outcomes.

5.2.1 Intuitiveness of the Keyboard interface

The intuitiveness of the Sinhala-English code-switchable keyboard interface was tested using a human-grounded experiment by allowing ten (10) testers to test the chat widget where the keyboard interface was integrated and the Helakuru keyboard, a widely utilized Sinhala keyboard, by instructing them to type a series of eleven (11) words taken from the Sinhala and English lexicons, without revealing the English to Sinhala character mappings. To preserve the integrity of the experiment, the testers were divided into two groups where the first group of testers used the Sinhala-English code-switching keyboard interface, and the other group used the Helakuru keyboard. Each tester was given one minute to type each word and for all failed attempts, the testers were given a second attempt to see if they can succeed. The results presented in Table 5.2.1.1 indicates the observations of the experiment.

The observations and insights obtained are as follows.

Table 5.2.1.1. Observations of Keyboard intuitiveness testing experiment

Tester #	**Average Time– Sinhala-English Keyboard (this tool)**			**Average Time– Helakuru (හෙළකුරු) Keyboard**		
	1st Attempt Duration (Minutes)	Failed Words in 1st Attempt	Failed Words in 2nd Attempt	1st Attempt Duration (Minutes)	Failed Words in 1st Attempt	Failed Words in 2nd Attempt
Tester 1	4.95	1	0	3.03	1	1
Tester 2	5.63	3	1	6.67	4	1
Tester 3	3.27	1	0	2.45	0	0
Tester 4	0.65	0	0	3.48	1	1
Tester 5	2.02	0	0	6.20	3	0
Total	16.52	5	1	22.23	9	3

1. Out of five testers who tested the Sinhala-English keyboard interface, only three testers have failed to type all words correctly within the first attempt, while four out of five testers who tested the Helakuru keyboard have failed within the first attempt. Interestingly, only one tester out of the three testers failed during the first attempt has failed to recover during the second attempt when testing the Sinhala English keyboard, whereas three out of four failed testers of the Helakuru keyboard have failed the second attempt, which indicates a higher failure rate.

2. Testers who tested the Sinhala-English code-switchable keyboard interface have completed the first attempt within 16.52 minutes, whereas the Helakuru keyboard testers have spent 22.23 minutes to complete their first attempt. This indicates that the overall typing time a user must spend is lower in the Sinhala-English code-switching keyboard interface.

3. Based on the failed words in the first attempt, the success rate of the Sinhala-English code-switchable keyboard interface is 90.91% which is higher than the success rate of Helakuru which is 83.64%. (The success rate is the total number of correctly typed words out of all words within the first attempt).

4. Based on the failed words in the first and second attempts, the testers of the Sinhala-English keyboard interface have achieved a recovery rate of 80%, whereas the testers of the Helakuru keyboard have only achieved a recovery rate of 66.67%. (The recovery rate is the number of correctly typed words out of the ones that failed).

The exact experiment setup and the observations of collected are available as a public GitHub repository[13] where interested individuals can easily setup the experiment environment and redo the experiment. Appendix D: Keyboard Interface Testing Setup depicts the experiment setup utilized for the experiment.

[13] https://github.com/dinushiTJ/Keyboard-experiment-setup

5.2.2 Keyboard interface and chat widget integration testing

Test cases 001 to 004 reveal the integration test cases ran against the keyboard interface and the React-based chat widget. The given test cases were utilized to test the query typing functionality of the Sinhala-English code-switching keyboard interface, test language switching functionality of the React-based chat widget with the integrated keyboard interface, and the overall chat widget functionality when carrying out a conversation with a chatbot connected with the web chat widget.

Table 5.2.2.1. Integration test case for Sinhala-only query typing

Project ID: 2022-056-IT19075754					
Project Name: Sinhala-English code-switching Keyboard Interface					
Project Function: Sinhala-English code-switching query typing in the chat widget					
Test case ID: 001		**Test case designed by: ID No:** IT19075754 **Name:** Jayasinghe D.T.			
Test Priority (High/Medium/Low): High					
Test Description: Type Sinhala-only queries in the integrated chat widget					
Prerequisite: A HTML page opened in the browser with chat widget attached.					
Test Steps: Step 1: Click on the widget icon on the bottom-left corner. Step 2: Use the input field of the chat widget to type a Sinhala-only query. Step 3: Observe if the output is rendered to the input box for Sinhala words.					
Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
001	Any Sinhala query or series of queries.	Output should be rendered to the input box with Sinhala Unicode words.	Sinhala words were converted correctly, and the output was rendered in real time.	pass	Tested for various queries with varying patterns.

Table 5.2.2.2. Integration test case for Sinhala-English query typing

Project ID: 2022-056-IT19075754					
Project Name: Sinhala-English code-switching Keyboard Interface					
Project Function: Sinhala-English code-switching query typing in the chat widget					
Test case ID: 002			**Test case designed by:** **ID No:** IT19075754 **Name:** Jayasinghe D.T.		
Test Priority (High/Medium/Low): High					
Test Description: Type code-switching queries in the integrated chat widget					
Prerequisite: A HTML page opened in the browser with chat widget attached.					
Test Steps: Step 1: Click on the widget icon on the bottom-left corner. Step 2: Use the input field of the chat widget to type a code-switching query. Step 3: Observe if the real time output is rendered to the input box for both Sinhala and English words.					
Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
002	Any Sinhala-English code-switching query or series of queries	Real time output should be rendered to the input box and the Sinhala words must be converted; English words must be intact	Sinhala words were converted correctly, and the output was rendered in real time.	pass	Tested for various queries with varying patterns

Table 5.2.2.3. Integration test case for Sinhala-English language switching

Project ID: 2022-056-IT19075754					
Project Name: Sinhala-English code-switching Keyboard Interface					
Project Function: Sinhala-English language switching in the chat widget					
Test case ID: 003			**Test case designed by:** **ID No:** IT19075754 **Name:** Jayasinghe D.T.		
Test Priority (High/Medium/Low): High					
Test Description: Test the language switching functionality of the chat widget					
Prerequisite: A HTML page opened in the browser with chat widget attached.					
Test Steps: Step 1: Click on the widget icon on the bottom-left corner. Step 2: Use the language toggle button right above the message input and observe if the language was correctly switched by typing a query in the input box. Step 3: Use CTRL+Q, the shortcut for language switching, and observe if the language switching is functional by typing a query and observing the output					
Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
003	Any Sinhala-English code-switching query Language switching shortcut keystrokes (CTRL+Q)	The language should be changed, and the input query should support typing queries in the selected language.	Both the language switch toggle button and the CTRL+Q shortcut worked as expected.	pass	Tested for various queries.

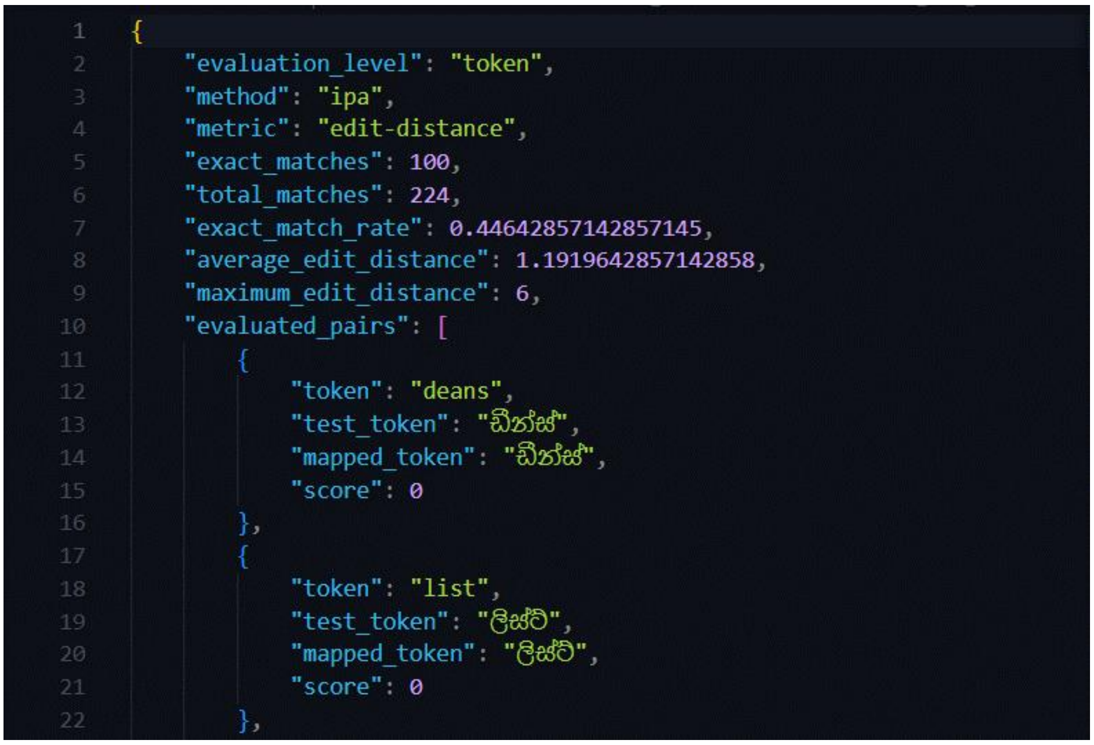
Table 5.2.2.4. Integration test case for the chat widget

Project ID: 2022-056-IT19075754					
Project Name: Sinhala-English code-switching Keyboard Interface					
Project Function: Chat widget functionality					
Test case ID: 004			**Test case designed by:** **ID No:** IT19075754 **Name:** Jayasinghe D.T.		
Test Priority (High/Medium/Low): Low					
Test Description: Test the message sending and receiving functionality of the chat widget.					
Prerequisite: A HTML page opened in the browser with chat widget attached and a running Rasa chatbot on the port pointed by the chat widget.					
Test Steps: Step 1: Click on the widget icon on the bottom-left corner. Step 2: Type a message and send it using send button or pressing the enter button. Step 3: Observe if the sent message bubble is right aligned. Step 4: Observe if the received bot response is displayed and the bubble is left-aligned. Step 5: Observe if the chat bubbles are distinguishable and properly color coded and aligned.					
Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
004	Any query	Left aligned user queries and right aligned bot responses. The enter button should handle message sending when the input box is on focus	Chat bubbles were properly displayed with expected colors; Enter button handled the message sending task.	pass	Tested for various queries.

5.2.3 Evaluation of IPA mappings

The SEETM CLI tool offers built-in support for evaluations through the SEEM Evaluate interface. The author conducted testing for IPA mappings (pronunciation-based approach) by incorporating a testing dataset that consisted of two hundred and twenty-four (224) words taken from the English lexicon and their expected Sinhala representation. Figure 5.2.3.1 depicts a part of the evaluation report persisted by the SEETM evaluator.

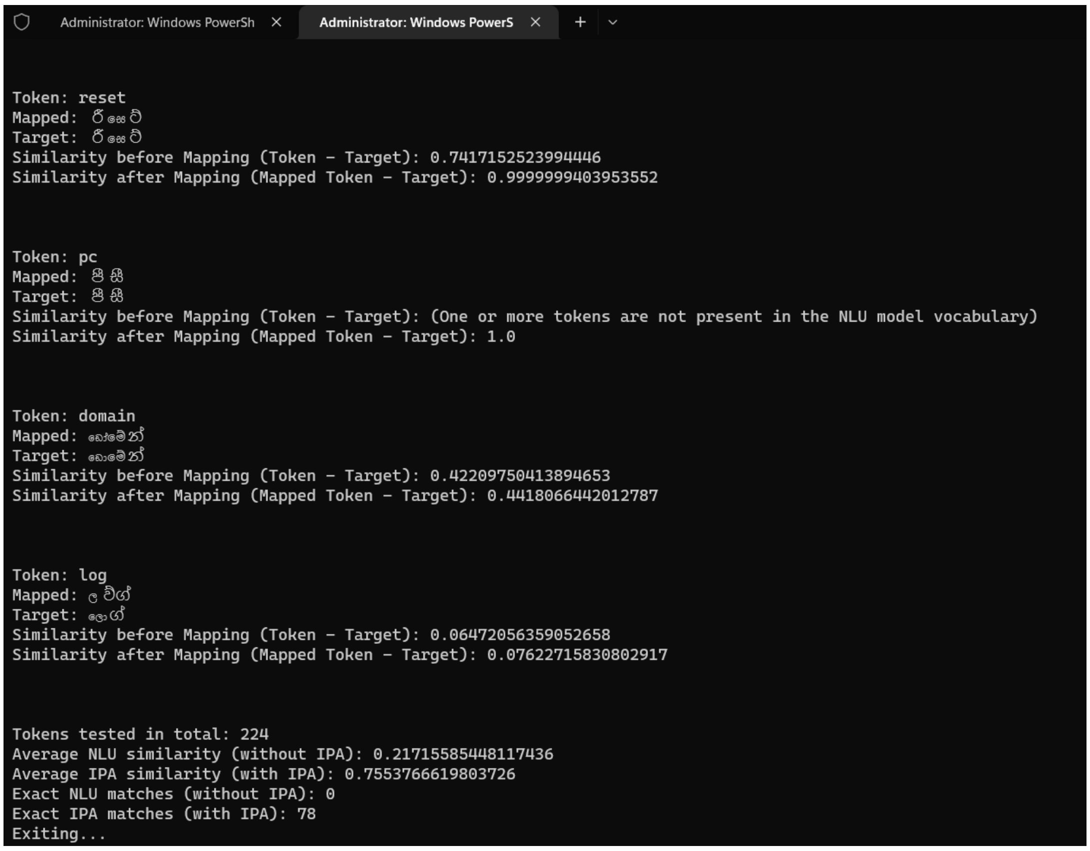
According to the test conducted by the author, the IPA mappings were able to convert 100 out of 224 English words that were presented in the evaluation dataset to their exact Sinhala representation. Thus, there were 44.7% exact matches. Based on *edit distance* utilized as the metric that finds exact matches, which states the minimum operations should be carried out to turn one sequence of characters to the expected sequence, the overall evaluation dataset had only 1.19 edit distance on average. The average edit distance clearly demonstrates the there are only one or two mistakes present on an IPA-based token mapping on average, which is surprisingly low.



```json
{
    "evaluation_level": "token",
    "method": "ipa",
    "metric": "edit-distance",
    "exact_matches": 100,
    "total_matches": 224,
    "exact_match_rate": 0.44642857142857145,
    "average_edit_distance": 1.1919642857142858,
    "maximum_edit_distance": 6,
    "evaluated_pairs": [
        {
            "token": "deans",
            "test_token": "ඩීන්ස්",
            "mapped_token": "ඩීන්ස්",
            "score": 0
        },
        {
            "token": "list",
            "test_token": "ලිස්ට්",
            "mapped_token": "ලිස්ට්",
            "score": 0
        },
```

Figure 5.2.3.1. A part of the evaluation report for the pronunciation-based approach

The SEETM evaluator also allows evaluation of the pronunciation-based method at vector level, by utilizing vector similarities given by two Word2Vec models trained on unmapped and IPA-mapped versions of the Rasa chatbot dataset. This evaluation algorithm loads the Rasa chatbot training dataset previously mentioned in section 4.1.2 and creates another version of the training dataset by mapping all English tokens with the pronunciation-based approach. The two datasets are then utilized to train two Word2Vec models using the *gensim* Python library. Then the evaluation dataset mentioned earlier that consists of 224 tokens is utilized to evaluate the similarities between the original and expected words using two Word2Vec models and the results are compared. Figure 5.2.3.2 depicts a part of evaluation results observed by the author. The evaluation results clearly state that the similarity of a target and expected token pair is increased by 53.8% when pronunciation-based approach is utilized.

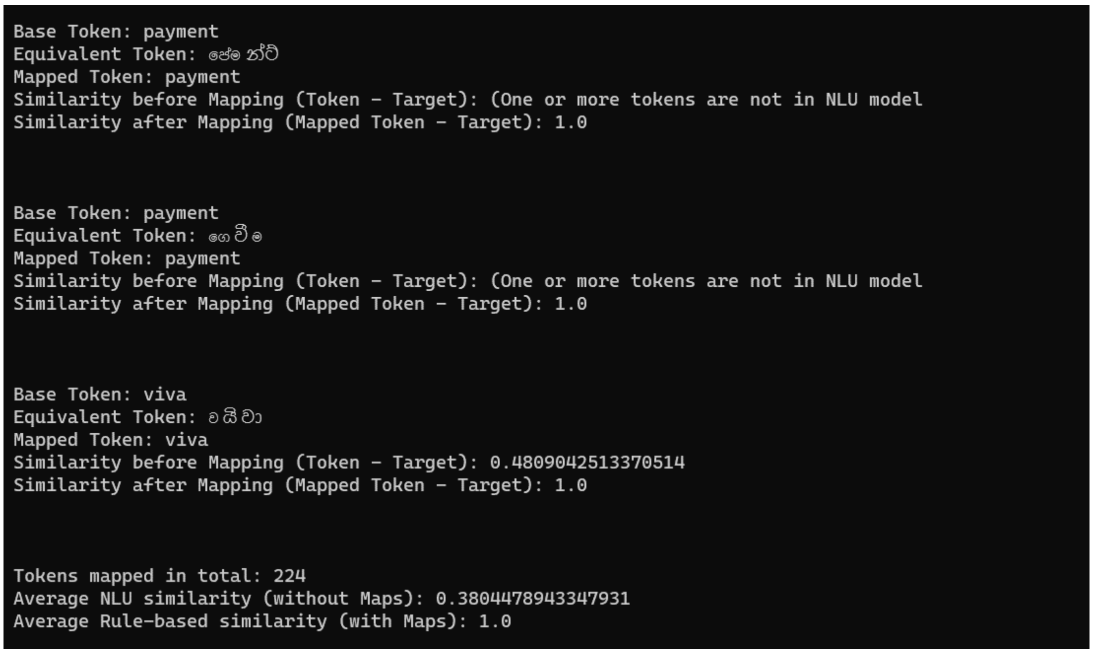


```
Token: reset
Mapped: රීසෙට්
Target: රීසෙට්
Similarity before Mapping (Token - Target): 0.7417152523994446
Similarity after Mapping (Mapped Token - Target): 0.9999999403953552


Token: pc
Mapped: පී සී
Target: පී සී
Similarity before Mapping (Token - Target): (One or more tokens are not present in the NLU model vocabulary)
Similarity after Mapping (Mapped Token - Target): 1.0


Token: domain
Mapped: ඩෝමේන්
Target: ඩොමේන්
Similarity before Mapping (Token - Target): 0.42209750413894653
Similarity after Mapping (Mapped Token - Target): 0.4418066442012787


Token: log
Mapped: ලඕග්
Target: ලොග්
Similarity before Mapping (Token - Target): 0.06472056359052658
Similarity after Mapping (Mapped Token - Target): 0.07622715830802917


Tokens tested in total: 224
Average NLU similarity (without IPA): 0.21715585448117436
Average IPA similarity (with IPA): 0.7553766619803726
Exact NLU matches (without IPA): 0
Exact IPA matches (with IPA): 78
Exiting...
```

Figure 5.2.3.2. Observations of the Word2Vec based evaluations on the pronunciation based SEETM technique

5.2.4 Evaluation of rule-based mappings

The SEETM evaluator also allows conducting rule based SEETM evaluations based on vector level similarities obtained using two *gensim* Word2Vec word embedding models. The word embedding models are dynamically trained each time the evaluator is invoked, and the SEETM evaluator creates two versions of the Rasa training dataset mentioned in section 4.1.2 by creating an additional version of the database using rule-based maps specified. Then the evaluator trains two word embedding models on the unmodified dataset and the mapped dataset using the rule-based technique. When evaluating, all of the rule-based maps are loaded and the similarity between each equivalent and base token pair from both word embedding models is inspected and compared. The author comprehensively evaluated nine (9) rule-based word pairs and observed that the average similarity between the base word and the equivalent word pairs was increased by 61.95%. Its noteworthy that the rule-based approach 1.0 cosine similarity for all nine (9) token maps that were tested.



```
Base Token: payment
Equivalent Token: පේම න්ට්
Mapped Token: payment
Similarity before Mapping (Token - Target): (One or more tokens are not in NLU model
Similarity after Mapping (Mapped Token - Target): 1.0



Base Token: payment
Equivalent Token: ගෙවීම
Mapped Token: payment
Similarity before Mapping (Token - Target): (One or more tokens are not in NLU model
Similarity after Mapping (Mapped Token - Target): 1.0



Base Token: viva
Equivalent Token: වයිවා
Mapped Token: viva
Similarity before Mapping (Token - Target): 0.4809042513370514
Similarity after Mapping (Mapped Token - Target): 1.0



Tokens mapped in total: 224
Average NLU similarity (without Maps): 0.3804478943347931
Average Rule-based similarity (with Maps): 1.0
```

Figure 5.2.4.1. A part of the evaluation report of the rule based SEETM approach

5.2.5 SEETM integration testing

The Python implementation of SEETM was tested after integration to test the overall functionality of the individual component. Test cases 005 to 008 contains black box integration test cases carried out on SEETM.

Figure 5.2.5.1. Integration test case for SEETM project initialization

Project ID: 2022-056-IT19075754					
Project Name: SEETM					
Project Function: SEETM project initialization					
Test case ID: 005			**Test case designed by:** **ID No:** IT19075754 **Name:** Jayasinghe D.T.		
Test Priority (High/Medium/Low): High					
Test Description: Create a new SEETM project using CLI and verify if required directories and files are created					
Prerequisite: The integrated SEETM package must be installed in the virtual environment					
Test Steps: Step 1: Open a terminal and activate the virtual environment. Step 2: Go to a new directory for creating a new SEETM project. Step 3: Run *seetm init* command. Step 4: Press enter to select the current directory Step 5: Wait until the project is created and observe the created directory structure.					
Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
005	*seetm init* in CLI	SEETM directories and files created in the new folder.	Observed the initial dirs. And files were created as expected as follows. - *seetm_config.yml* - *seetm_maps* - *seetm_exports* - *seetm_eval* - *seetm_components* - *data*	pass	The *seetm init* command is functional.

Figure 5.2.5.2. Integration test case for SEETM CLI mapper

Project ID: 2022-056-IT19075754	
Project Name: SEETM	
Project Function: SEETM CLI mapping	
Test case ID: 006	**Test case designed by:** **ID No:** IT19075754 **Name:** Jayasinghe D.T.
Test Priority (High/Medium/Low): High	
Test Description: Run the *seetm map* command and inspect if the specified data instance is correctly mapped, using both *ipa* and *rule-based* methods.	
Prerequisite: Installed integrated SEETM Python package in the virtual env.	
Test Steps: Step 1: Open a terminal and activate the virtual environment. Step 2: Run *seetm map* with a data instance under --*instance* flag and --*method* as *ipa* Step 3: Observe the CLI output and verify that it is correctly mapped. Step 4: Run *seetm map* with a data instance under --*instance* flag and --*method* as *rule-based* Step 5: Observe the CLI output and verify that it is correctly mapped.	

Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
006	*seetm map --instance* "any text instance" --*method* ipa and *seetm map --instance* "any text instance" –*method* rule-based	The English words in data instance must be mapped when method is IPA. The words in rule-based maps must be mapped in Rule-based method.	Observed that both IPA mapping and Rule-based mappings are functional for various data instances.	pass	The functionality was tested for varying data instances and patterns.

Figure 5.2.5.3. Integration test case for SEETM Server

Project ID: 2022-056-IT19075754					
Project Name: SEETM					
Project Function: SEETM project server deployment					
Test case ID: 007			**Test case designed by:** **ID No:** IT19075754 **Name:** Jayasinghe D.T.		
Test Priority (High/Medium/Low): High					
Test Description: Test if the SEETM server is functional in both debug and production modes.					
Prerequisite: Installed integrated SEETM Python package in the virtual env.					
Test Steps: Step 1: Open a terminal and activate the virtual environment. Step 2: Go to an existing SEETM project directory in the CLI. Step 3: Run *seetm server --debug* command. Step 4: Observe if the SEETM server is starting in debugging mode at port 6067 Step 5: Visit http://localhost:6067 and confirm that the SEETM developer console is functional.					
Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
007	*seetm server --debug* command and *seetm server* command	SEETM Server must start in debug mode and production modes; the developer console must be accessible at port 6067.	Observed that the server deployment is functional, and the developer console UI is operational.	pass	Both debugging and production level deployments worked as expected.

Figure 5.2.5.4. Integration test case for developer console – constructing token maps

Project ID: 2022-056-IT19075754					
Project Name: SEETM					
Project Function: Creating token maps using the SEETM developer console.					
Test case ID: 008			**Test case designed by:** **ID No:** IT19075754 **Name:** Jayasinghe D.T.		
Test Priority (High/Medium/Low): High					
Test Description: Test if the SEETM developer console works as expected and the developers can construct, edit, and delete token maps using the UI.					
Prerequisite: Installed integrated SEETM Python package in the virtual env.					
Test Steps: Step 1: Open a terminal and activate the virtual environment. Step 2: Go to an existing SEETM project directory in the CLI. Step 3: Run *seetm server --debug* command. Step 4: Visit http://localhost:6067 and confirm that the SEETM developer console is functional. Step 5: In the dashboard, try creating a new token map and saving it. Step 6: Try editing the token map and see if its functional by saving the changes. Step 7: Try deleting the map and see if its functional.					
Test ID	Test Inputs	Expected Outputs	Actual Output	Result (Pass/Fail)	Comments
008	*seetm server --debug* command and equivalent words required to construct mappings	Create, edit, and delete maps should be functional.	Observed that the maps were created, created maps were editable, and as well as removable.	pass	The generated maps were extensively tested by training new Rasa bots on changed mappings.

5.2.6 Effect of rule-based token maps

The author conducted an additional human-grounded evaluation experiment of the SEETM approach to quantify the effect of the rule-based token mappings. The experiment setup comprised two Rasa chatbots, (1) a chatbot trained using SEETM and (2) a chatbot trained without SEETM. The ultimate objective of the experiment was to find the effect of rule-based token mapping and if it is able to handle inputs with varying equivalent words the models may have not encountered during training.

The training process of the two chatbots followed an identical approach. Both chatbots were trained on the same training dataset that comprised one-hundred and seventy-three (173) training data instances and validated using a validation dataset of thirty (30) data instances. It is noteworthy that the overall dataset consisted of nineteen (19) intent classes, however, the author carefully crafted four (4) out of the nineteen intent classes to contain only Sinhala and Sinhala representation of English words to highlight the OOV token issue. The author then integrated SEETM with the first chatbot and created relevant token mappings to map the possible equivalent words that the users might input under the four crafted intent classes. The second chatbot was not integrated with SEETM. Then the two chatbots were exposed to the testers and they were instructed to ask the same query from both bots and observe the chatbot response. Table 5.2.6.1 demonstrates the summary of the observations collected by the testers.

Based on the observations it is evident that 64% of the responses obtained from the first chatbot were accurate, whereas the second chatbot has given only 24% accurate responses. The observations clearly highlight the impact of rule-based token mapping.

Table 5.1.2. Observations of SEETM experiment

Tester #	**Bot 1 Responses (with SEETM)**		**Bot 2 Responses (without SEETM)**	
	correct	wrong	correct	wrong
Tester 1	4	1	1	4
Tester 2	4	1	1	4
Tester 3	3	2	0	5
Tester 4	3	2	1	4
Tester 5	2	3	3	2
Total	16	9	6	19

The experiment setup is available as a public GitHub repository[14] with relevant instructions, where interested individuals can easily setup the experiment environment and redo the experiment if required. The repository also contains the SEETM token maps utilized within the first bot[15]. See Appendix E: Token Mapping Testing Setup and Appendix F: Token Mapping Testing Observations for more details on the experiment conducted.

5.3 Discussion

The Sinhala-English code-switchable keyboard interface is the first individual research outcome produced by this individual research component and as mentioned in section 4.4.1, the English to Sinhala character mappings were carefully crafted by the author, preserving the intuitiveness by incorporating a phonetic-based approach. For example, the character mappings were constructed in a way that a user can intuitively identify letter a is for අ, letter k is for ක්, and to construct the letter ක, they must type ka. The intuitiveness of the Sinhala-English code-switchable keyboard was demonstrated by the observed results in section 5.2.1. When compared with similar solutions that allow typing Sinhala Unicode characters primarily based on phonetics, the Sinhala-English code-switchable keyboard revealed that it is more efficient, exemplifying that the users can easily guess the character mappings to type Sinhala swiftly. The author has also addressed issues in currently available Sinhala-only keyboard solutions where existing mappings by (1) replacing ක්-K with ඛ-K, (2) replacing ඊ-I with ඓ-I, (3) replacing ඳ-q with ඳ-dh, and (4) replacing ඪ-qh with ඪ-Dh to preserve the intuitiveness. During testing the keyboard interface, the author also noticed that some character mappings like ඇ-A, ඈ-AA, ◌ං-x, ර්‍-z, ඪ-Dh, එ-e, ඒ-ee are hard to guess in keyboard interface and as well as existing solutions due to less predictable character mappings and users also seem to struggle when typing එ-e and ඒ-ee as they entangle the mapping with ඉ-i, ඊ-ii which since e is utilized in English to represent the sound ɪ (ඉ) or I (ඊ) in IPA. As mentioned in section 4.4.5, the Sinhala-

[14] https://github.com/dinushiTJ/SEETM-experiment-setup
[15] The entire list of token maps utilized to test the first bot are available at https://github.com/dinushiTJ/SEETM-experiment-setup/blob/main/bot_with_maps/seetm_maps/token_to_token_map.json

English code-switchable keyboard interface introduced during this research is the first code-switchable keyboard interface that can be integrated into any React-based website that extends native typing support in websites beyond English. However, currently, the users must manually switch between Sinhala and English when typing code-switching text instances using the language toggle switch or the shortcut key provided. Thus, enabling automatic language detection and switching between languages without requiring explicit triggers is a significant improvement that can be done over the current solution.

Since the prevailing character mapping issues are not addressed solely based on additional character mappings and other rule-based solutions, a light-weight machine learning or deep learning-based solution seems like a more appropriate solution that can correctly identify different typing patterns of users such as *degree*, *digrii*, *digree*, *digri*, and *degree* all maps to the word ඩිග්‍රී and it remains as a possible future improvement that was unexplored within the scope of this research. Another possible improvement that can be explored is enabling word suggestions within the keyboard interface. Although there are third party tools that can suggest Sinhala words, there is no tool found during this research that can enable word suggestions natively for any input field on a given website or a web application.

As mentioned in section 4.4.5, the Sinhala-English code-switchable keyboard interface introduced during this research is the first code-switchable keyboard interface that can be integrated into any React-based website that extends native typing support in websites beyond English. However, currently, the users must manually switch between Sinhala and English when typing code-switching text instances using the language toggle switch or the shortcut key provided. Thus, enabling automatic language detection and switching between languages without requiring explicit triggers is a significant improvement that can be done over the current solution.

As explained in section 4.5, the author identified three techniques to solve the OOV token issue previously mentioned, namely, (1) the translation-based approach, (2) the pronunciation-based approach, and (3) the rule-based approach. Out of the suggested solutions, the author excluded the translation-based approach due to its limitations and practical usage concerns, as numerous translated phrases from English to Sinhala resulted in obsolete formal Sinhala words usually not present in conversations. Although the author invested time in fully implementing the pronunciation-based approach to reveal the importance of the matter, section 4.5.3 clearly states the limitations of the pronunciation-based approach considering the scope of the individual research. Sections 5.2.3 and 5.2.4 highlight how the pronunciation-based and rule-based solutions improved the similarity between the equivalent words and phrases after the ETM process using word embeddings generated by Word2Vec models. The human-grounded experiment previously mentioned in section 5.2.6 accentuates that the Rasa chatbot with SEETM integrated performed better when compared with the other chatbot, which the author trained on the same training dataset and the NLU pipeline configurations without SEETM.

Although the author utilized only the rule-based technique in the SEETM Python package specifically for training Rasa conversational AI models, it would be significant to see a pronunciation-based and rule-based hybrid approach incorporated in the equivalent token map building. Since the current implementation of the rule-based SEETM technique relies on user-defined maps, the impact of the maps directly depends on the developers constructing them. Interested researchers can focus on enhancing the current approach of token mapping construction by partially or fully automating the process beyond manually crafted token maps.

5.4 Individual Contribution

Sections 4.4.5, 4.6.5, and 4.8**Error! Reference source not found.** clearly state the individual research contribution made by the two primary research outcomes of this research component, towards the overall research. To conclude the sections, the individual research outcomes assisted the overall research solution, *Kolloqe*, to expand native typing support in the user interfaces beyond English and support Sinhala-English code-switching typing support, which allows chatbots trained using Kolloqe to be highly accessible to a vast audience. The SEETM approach introduced by the author allowed the overall research outcome to be immune to OOV issues caused by equivalent words or phrases present in code-switching datasets, allowing Kolloqe to train high performance chatbot models.

6. CONCLUSION

ML and NLP are relatively novel domains to the Sinhala Language that belong to the category of low-resource language due to its undersized digital presence. Although numerous research is available on ML and NLP topics for Sinhalese, research on Sinhala-English code-switching and code-mixing data pre-processing for NLP are infrequent and openly accessible research is rare. There is also a deficiency in production-grade implementations of Sinhala and Sinhala-English code-switching-related research outcomes. More specifically, many chatbot development frameworks with advanced ML and NLP support do not offer relevant tools or typing support required to interact with the bots in Sinhalese, and Sinhala-supported chatbot development platforms do not comprise a training process that can effectively handle equivalent word-related OOV issues. Addressing these issues is essential to deliver chatbots and other Sinhala or Sinhala-English code-switching-related production-grade NLP tools that are reliable and widely accessible to a large audience. The value of providing open access to Sinhala-related research should also be a prominent concern since both Sinhala and other related low-resource languages can employ similar approaches to overcome language-related challenges in the digital world. Not having access to previous research and proper implementations is one of the primary reasons why Sinhala is still considered a low-resource language.

The author of this research component produced two invaluable research solutions that address the previously mentioned drawbacks and limitations in chatbot development frameworks, focusing on Sinhala-English code-switching datasets utilized to train chatbot models. The Sinhala-English code-switchable keyboard allows developers to extend the native typing support in websites beyond English and spotlighted a potential typing standard for multilingual web users. The Sinhala-English Equivalent Token Mapping technique has introduced a novel pre-processing strategy for Sinhala-English code-switched datasets, but the idea is not limited to Sinhala and English. The author has provided open access for both research solutions with proper implementations where the source codes are publicly available for any interested individual to continue from where the author concluded the research and utilize the knowledge gained from the study as an inspiration to conduct new research that is beneficial for Sinhala and other low-resource languages.

REFERENCES

[1]. [1] K. Mote, "Natural Language Processing-A Survey," *arXiv preprint arXiv:1209.6238*, 2012.

[2]. [2] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pp. 604-624, 2020.

[3]. [3] I. Smith and U. Thayasivam, "Language Detection in Sinhala-English Code-mixed Data," in *2019 International Conference on Asian Language Processing (IALP)*, 2019: IEEE, pp. 228-233.

[4]. [4] T. Fonseka, R. Naranpanawa, R. Perera, and U. Thayasivam, "English to sinhala neural machine translation," in *2020 International Conference on Asian Language Processing (IALP)*, 2020: IEEE, pp. 305-309.

[5]. [5] G. Caldarini, S. Jaf, and K. McGarry, "A literature survey of recent advances in chatbots," *Information*, vol. 13, no. 1, p. 41, 2022.

GLOSSARY

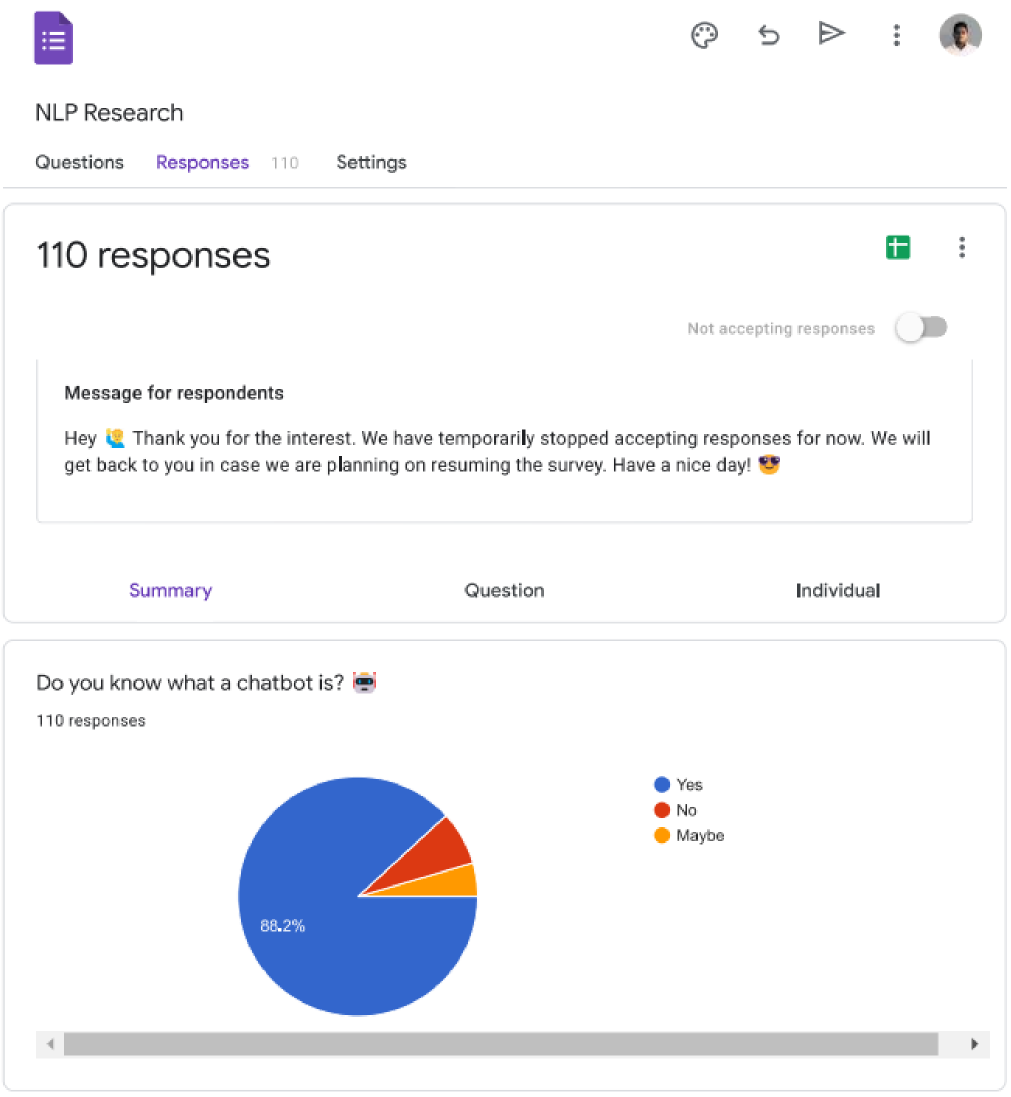
Term	Definition
Artificial Intelligence (AI)	A field of study that frequently refers to machine intelligence as opposed to natural intelligence.
Machine Learning (ML)	A subfield of AI that concerns the development of methods that leverage data to improve performance in executing a task or set of tasks.
Natural Language Processing (NLP)	A subfield of linguistics, computer science, and artificial intelligence that concerns how to enable machines to process enormous quantities of natural language data.
Rasa Open-Source	An open-source conversational AI development platform developed and maintained by Rasa HQ.
Training Dataset	A large collection of data utilized to train machine learning and deep learning models.
Word Embeddings	Vector representations of words for text analysis in NLP, constructed in a way such that similar words comprise similar vectors.
Equivalent Tokens	A set of words in the same language or different languages that has the same underlying meaning within the same context.

Code-switching	The practice of alternating between two or more languages while conversing, within the same conversation.
Code-mixing	The practice of mixing two or more languages while conversing, within the same conversation.
Equivalent Token Mapping	A technique introduced in this research component that effectively maps multiple equivalent tokens into a single base token is utilized to handle out-of-vocabulary token issues in code-switching datasets.
Word2Vec	A technique surfaced in 2013 for NLP, that learns word associations from a large corpus of text using a deep neural network.
Lexicon	The vocabulary of a specific language.

APPENDICES

Appendix A: Survey Form

Figure A.1: Complete survey form questions and responses – part 1



NLP Research

Questions Responses 110 Settings

110 responses

Not accepting responses

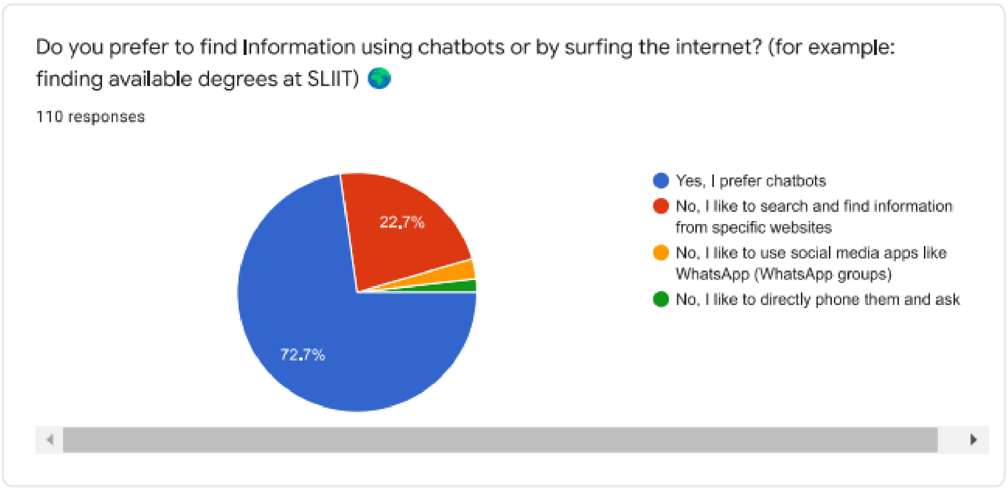
Message for respondents

Hey 🙋 Thank you for the interest. We have temporarily stopped accepting responses for now. We will get back to you in case we are planning on resuming the survey. Have a nice day! 😎

Summary Question Individual

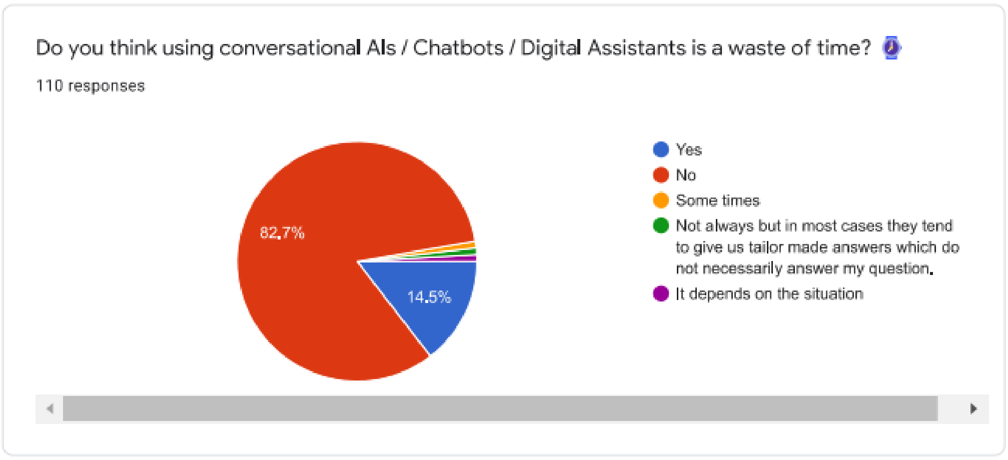
Do you know what a chatbot is? 🤖

110 responses



Do you prefer to find Information using chatbots or by surfing the internet? (for example: finding available degrees at SLIIT) 🌍

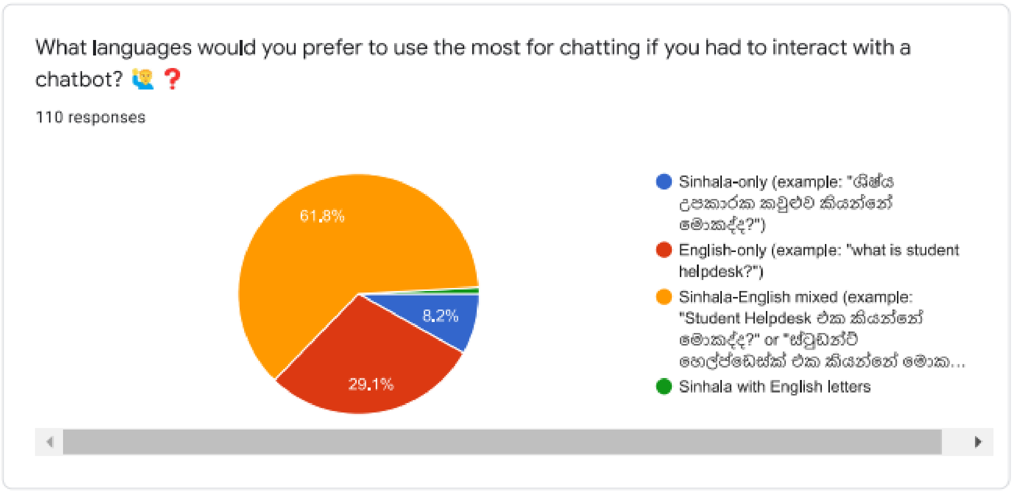
110 responses



Do you think using conversational AIs / Chatbots / Digital Assistants is a waste of time?

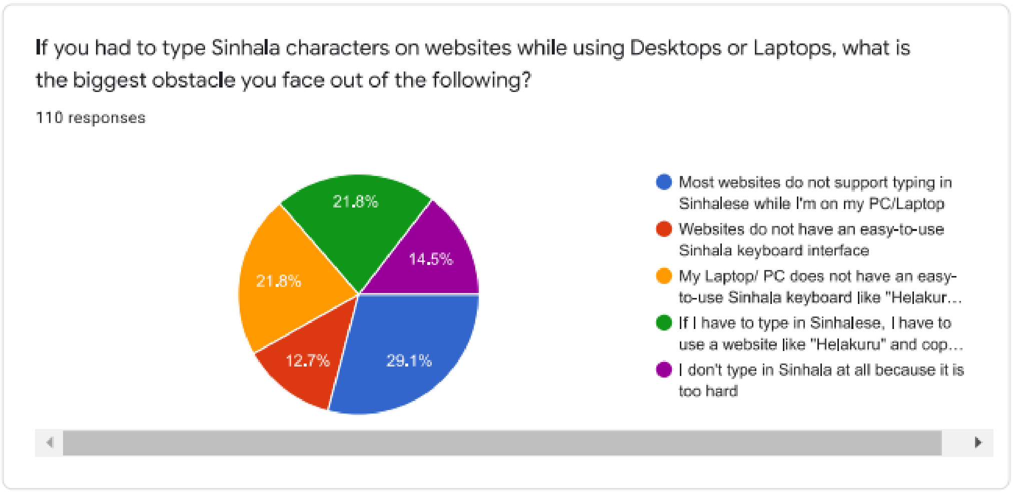
110 responses

Figure A.2: Complete survey form questions and responses – part 2


What languages would you prefer to use the most for chatting if you had to interact with a chatbot? 🙋 ❓

110 responses

- ● Sinhala-only (example: "ශිෂ්‍ය උපකාරක කවුළුව කියන්නේ මොකද්ද?")
- ● English-only (example: "what is student helpdesk?")
- ● Sinhala-English mixed (example: "Student Helpdesk එක කියන්නේ මොකද්ද?" or "ස්ටුඩන්ට් හෙල්ප්ඩෙස්ක් එක කියන්නේ මොක…
- ● Sinhala with English letters


If you had to type Sinhala characters on websites while using Desktops or Laptops, what is the biggest obstacle you face out of the following?

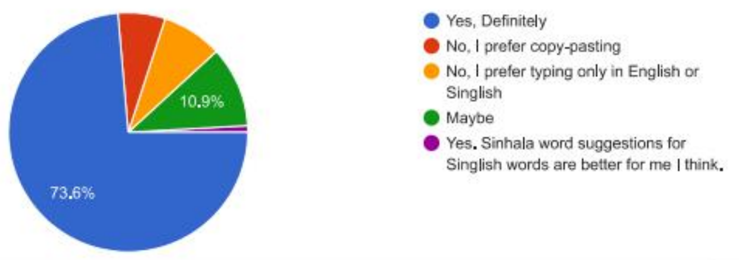
110 responses

- ● Most websites do not support typing in Sinhalese while I'm on my PC/Laptop
- ● Websites do not have an easy-to-use Sinhala keyboard interface
- ● My Laptop/ PC does not have an easy-to-use Sinhala keyboard like "Helakur…
- ● If I have to type in Sinhalese, I have to use a website like "Helakuru" and cop…
- ● I don't type in Sinhala at all because it is too hard

Figure A.3: Complete survey form questions and responses – part 3

Do you prefer if websites offered Sinhala and English mixed Typing facilities out of the box without having to install additional software? 💻

110 responses



If you were given the following options to ask any quick question related to SLIIT you have, what option would you choose? (Please note that the question can only be a simple, general and a frequently asked question such as "ස්ලිට් VPN එක කියන්නේ මොකක්ද?" but not as complicated as "SE මිඩ් එක්සෑම් paper එකේ structure එක මොකක්ද?")

110 responses

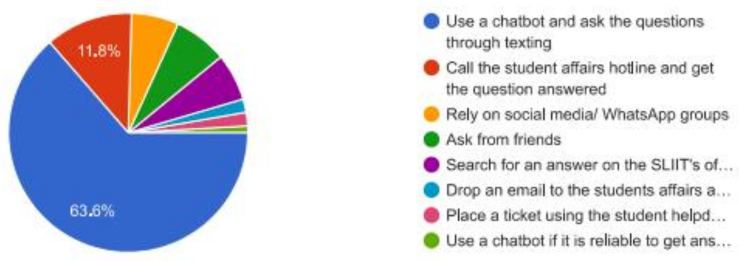
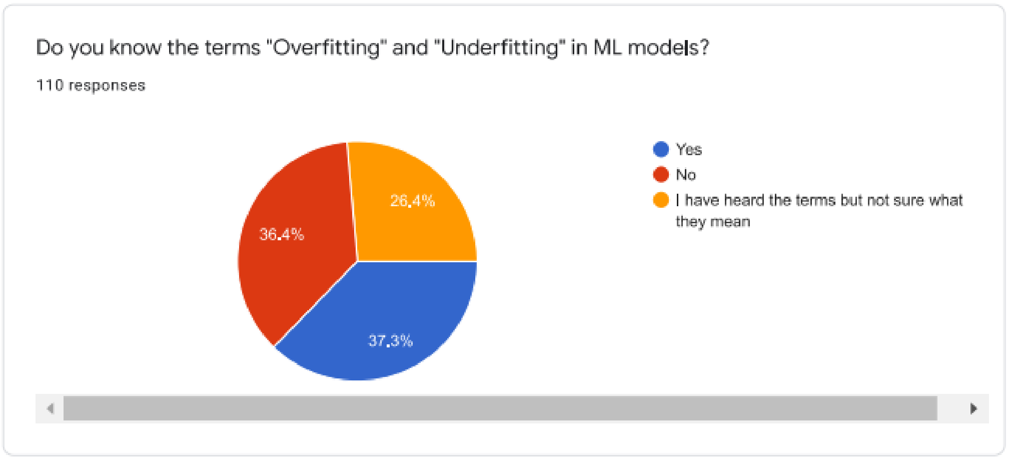
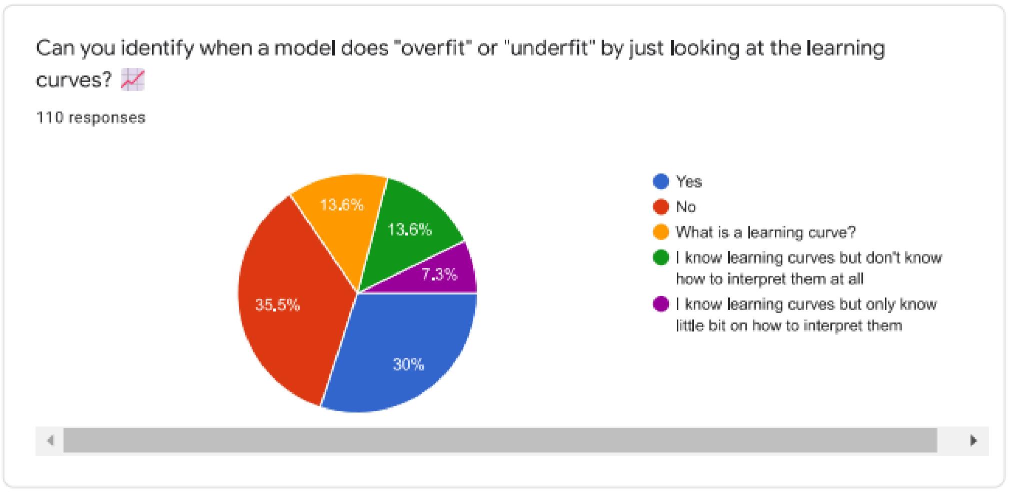


Figure A.4: Complete survey form questions and responses – part 4

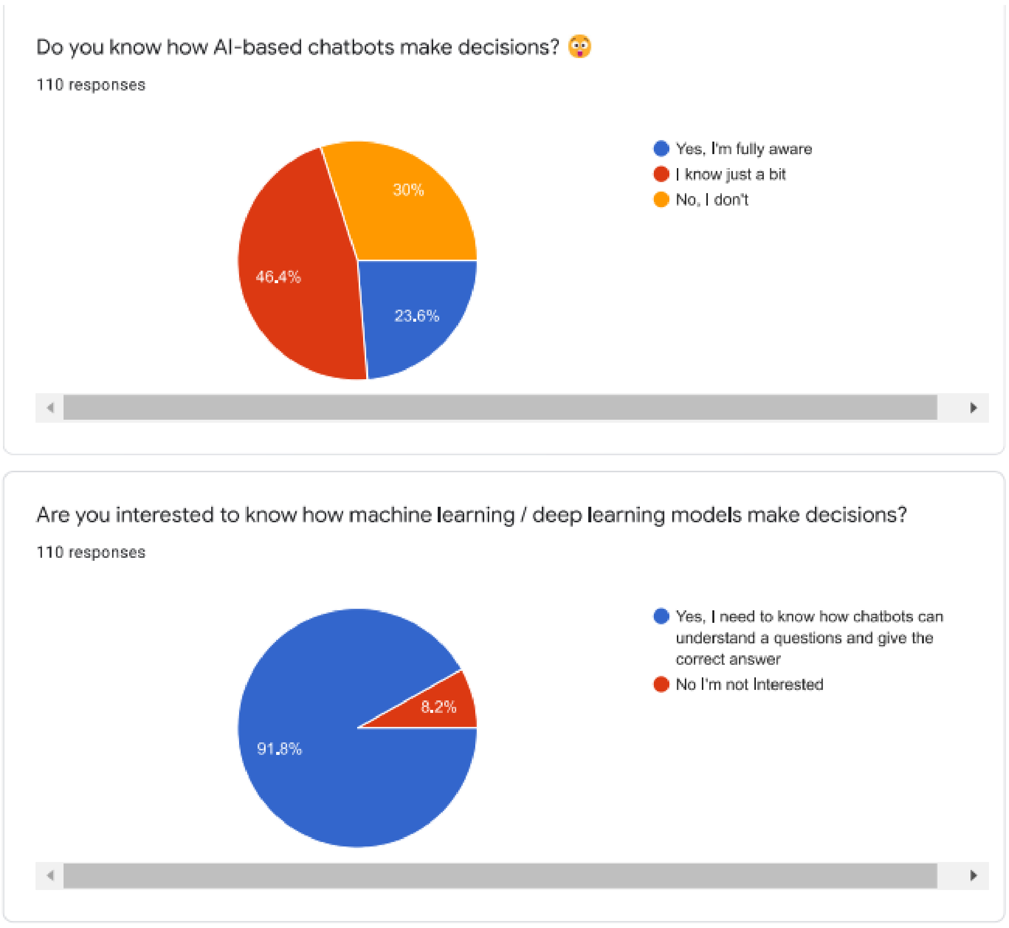

Do you know the terms "Overfitting" and "Underfitting" in ML models?

110 responses


Can you identify when a model does "overfit" or "underfit" by just looking at the learning curves? 📈

110 responses

Figure A. 5: Complete survey form questions and responses – part 5

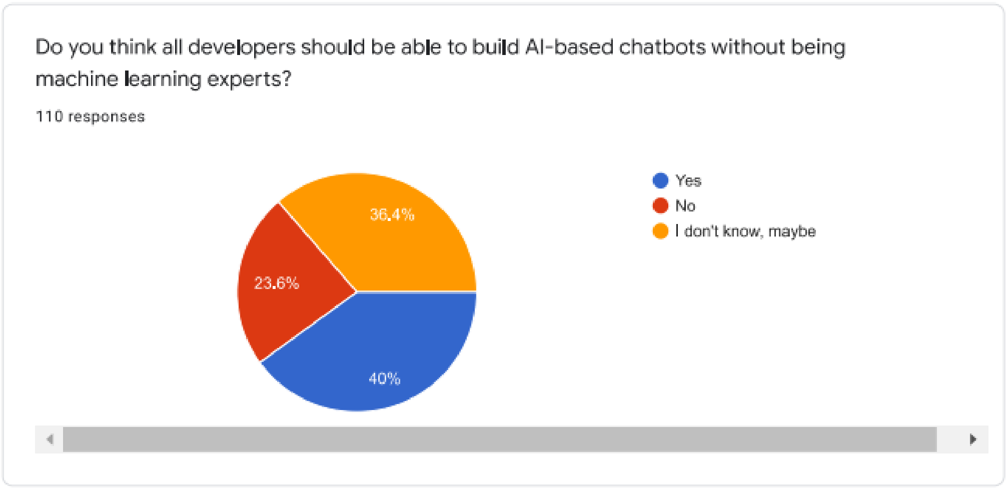

Do you know how AI-based chatbots make decisions? 😳

110 responses

Are you interested to know how machine learning / deep learning models make decisions?

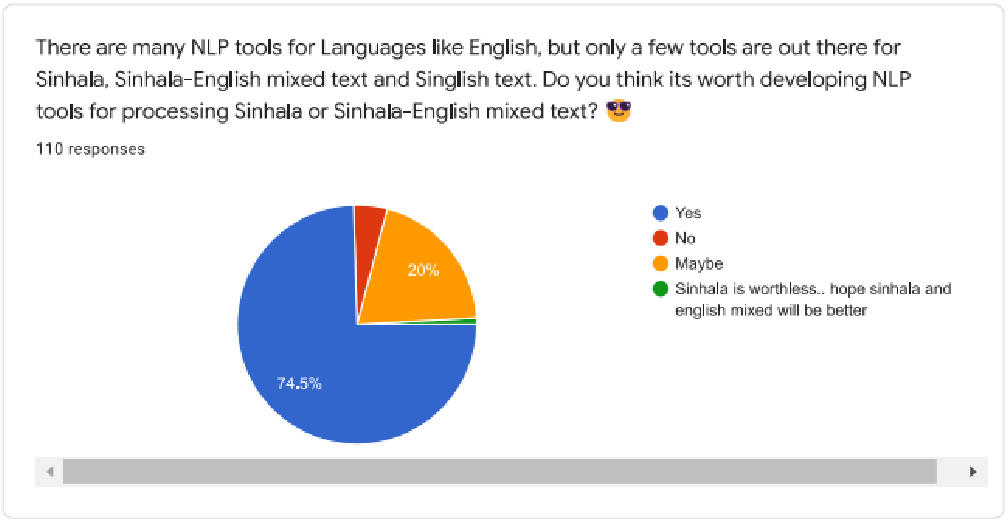
110 responses

Figure A.6: Complete survey form questions and responses – part 6



Do you think all developers should be able to build AI-based chatbots without being machine learning experts?

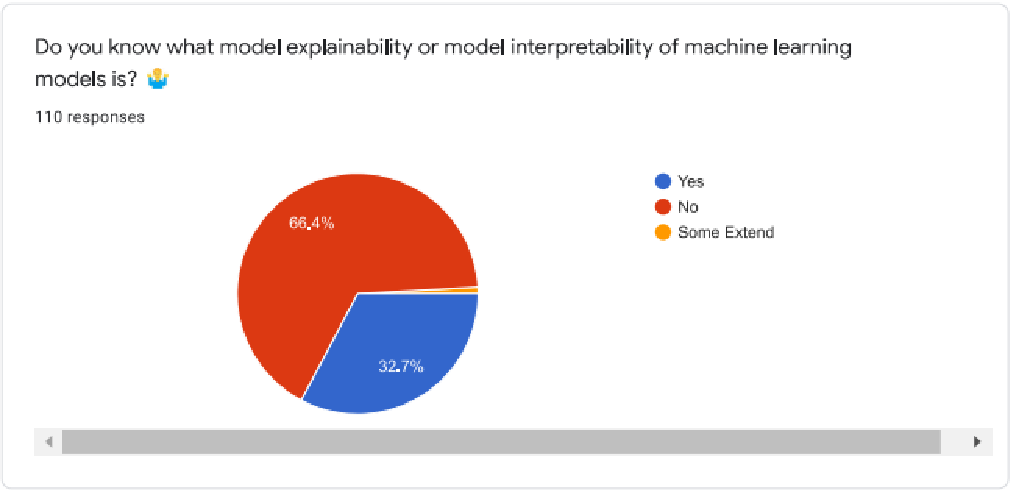
110 responses



There are many NLP tools for Languages like English, but only a few tools are out there for Sinhala, Sinhala-English mixed text and Singlish text. Do you think its worth developing NLP tools for processing Sinhala or Sinhala-English mixed text? 😎

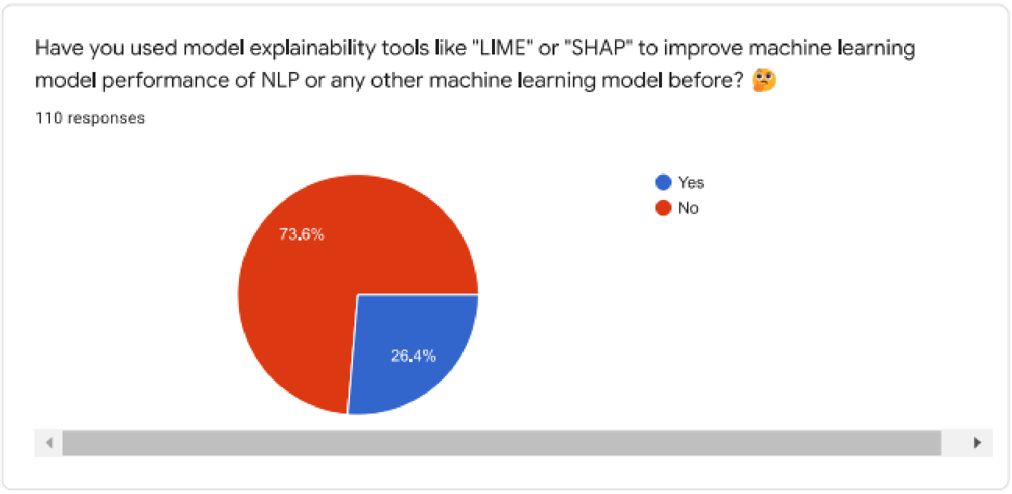
110 responses

Figure A.7: Complete survey form questions and responses – part 7



Do you know what model explainability or model interpretability of machine learning models is?

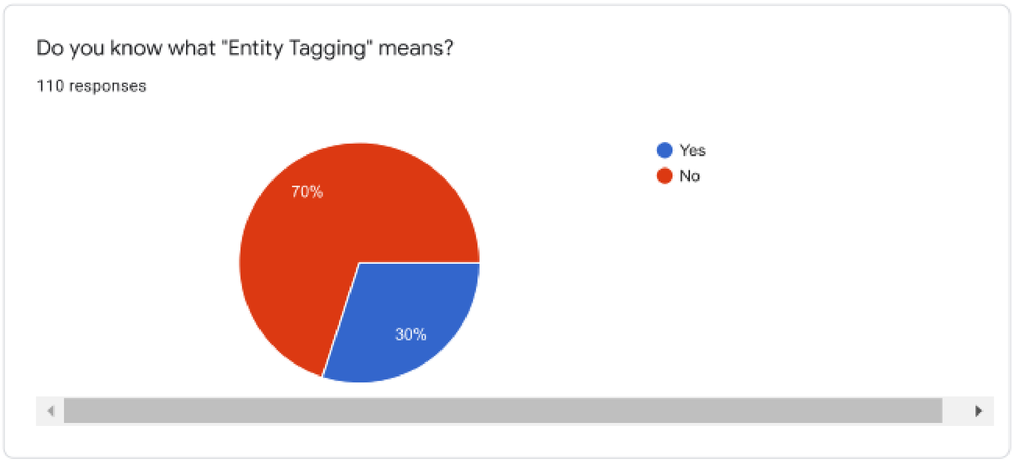
110 responses



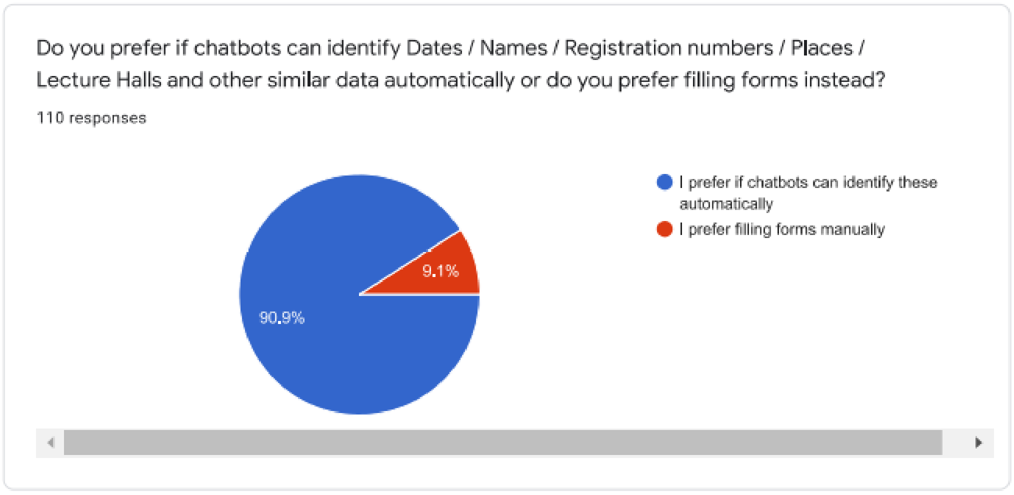
Have you used model explainability tools like "LIME" or "SHAP" to improve machine learning model performance of NLP or any other machine learning model before?

110 responses

Figure A.8: Complete survey form questions and responses – part 8


Do you know what "Entity Tagging" means?

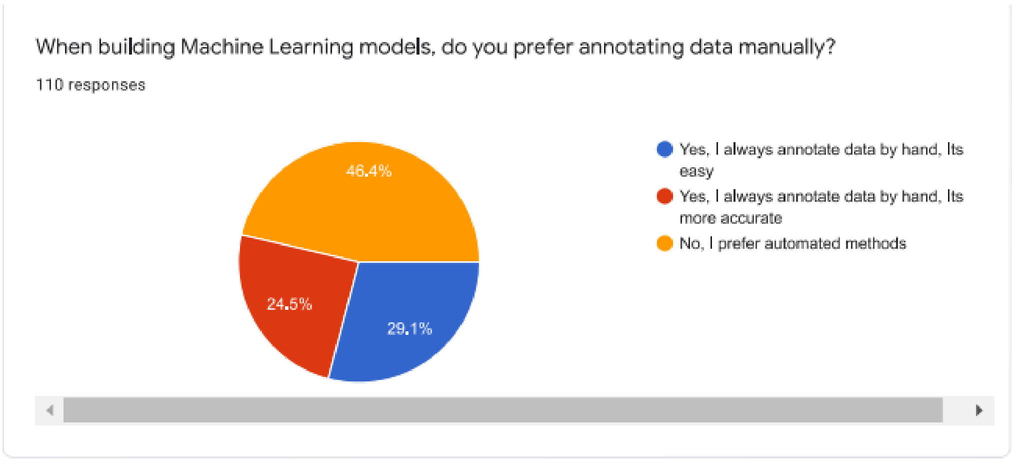
110 responses


Do you prefer if chatbots can identify Dates / Names / Registration numbers / Places / Lecture Halls and other similar data automatically or do you prefer filling forms instead?

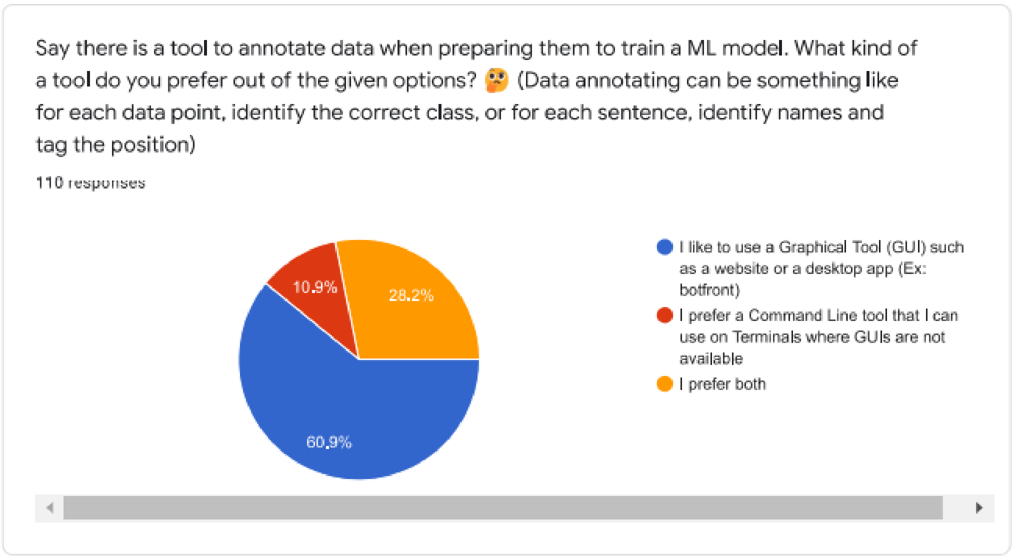
110 responses

9.1%
90.9%
● I prefer if chatbots can identify these automatically
● I prefer filling forms manually

Figure A.9: Complete survey form questions and responses – part 9

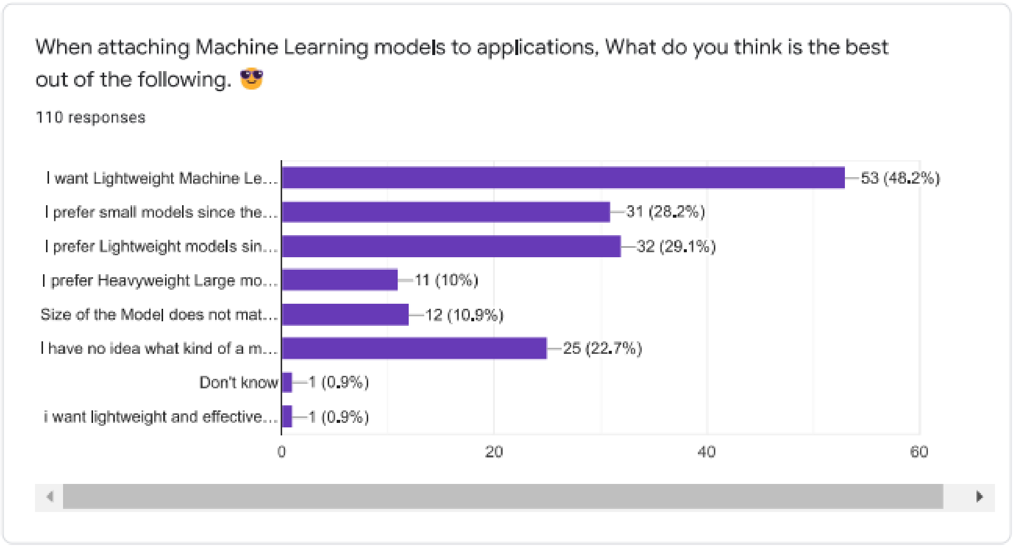

When building Machine Learning models, do you prefer annotating data manually?

110 responses

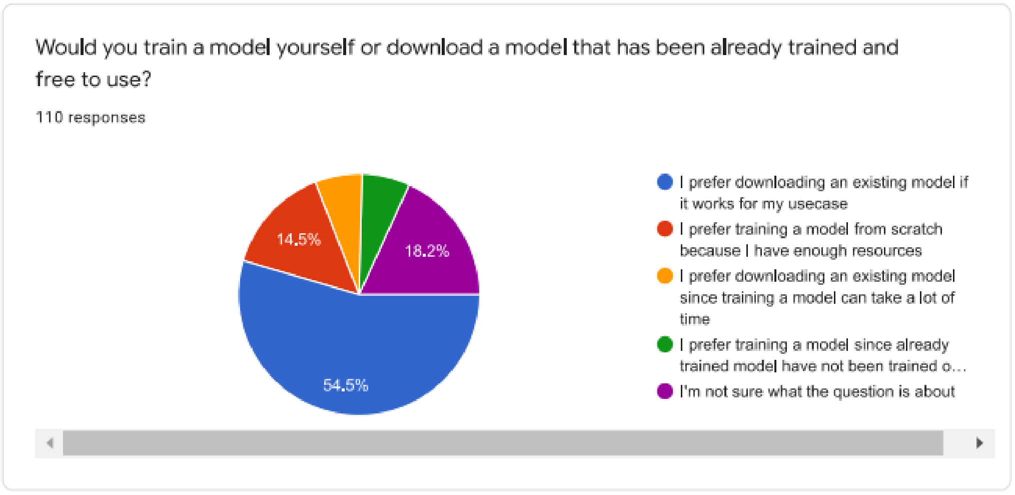

Say there is a tool to annotate data when preparing them to train a ML model. What kind of a tool do you prefer out of the given options? 🤔 (Data annotating can be something like for each data point, identify the correct class, or for each sentence, identify names and tag the position)

110 responses

Figure A.10: Complete survey form questions and responses – part 10


When attaching Machine Learning models to applications, What do you think is the best out of the following. 😎

110 responses

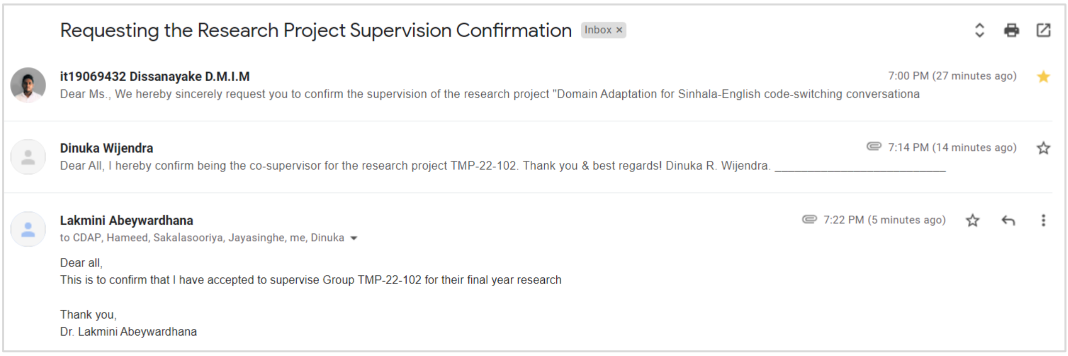

Would you train a model yourself or download a model that has been already trained and free to use?

110 responses

Thank you!

Figure A.11: Complete survey form questions and responses – part 11

Appendix B: Supervision Confirmation Emails


Requesting the Research Project Supervision Confirmation Inbox ×

it19069432 Dissanayake D.M.I.M 7:00 PM (27 minutes ago)
Dear Ms., We hereby sincerely request you to confirm the supervision of the research project "Domain Adaptation for Sinhala-English code-switching conversationa

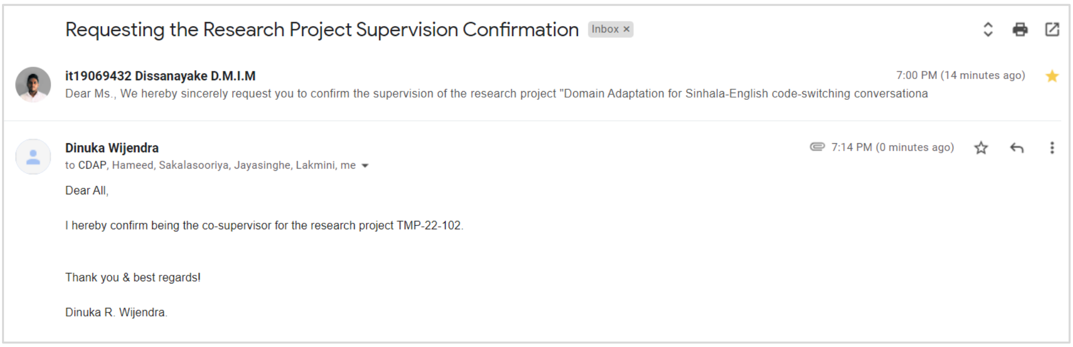
Dinuka Wijendra 7:14 PM (14 minutes ago)
Dear All, I hereby confirm being the co-supervisor for the research project TMP-22-102. Thank you & best regards! Dinuka R. Wijendra. ________________________

Lakmini Abeywardhana 7:22 PM (5 minutes ago)
to CDAP, Hameed, Sakalasooriya, Jayasinghe, me, Dinuka

Dear all,
This is to confirm that I have accepted to supervise Group TMP-22-102 for their final year research

Thank you,
Dr. Lakmini Abeywardhana

Figure B.1. Research project supervision confirmation email


Requesting the Research Project Supervision Confirmation Inbox ×

it19069432 Dissanayake D.M.I.M 7:00 PM (14 minutes ago)
Dear Ms., We hereby sincerely request you to confirm the supervision of the research project "Domain Adaptation for Sinhala-English code-switching conversationa

Dinuka Wijendra 7:14 PM (0 minutes ago)
to CDAP, Hameed, Sakalasooriya, Jayasinghe, Lakmini, me

Dear All,

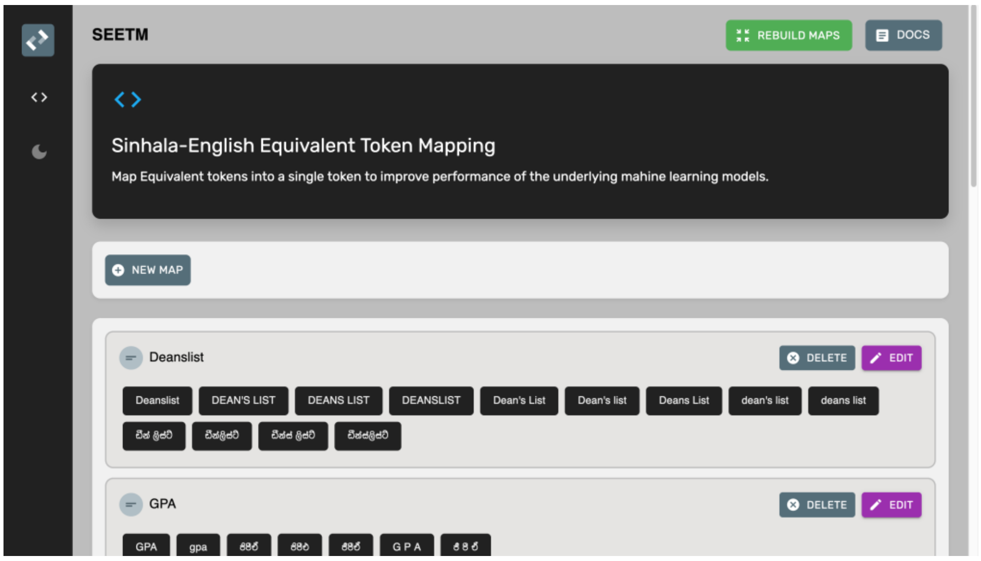
I hereby confirm being the co-supervisor for the research project TMP-22-102.

Thank you & best regards!

Dinuka R. Wijendra.

Figure B.2. Research project co-supervision confirmation email

Appendix C: SEETM Developer Console User Interfaces



Sinhala-English Equivalent Token Mapping

Map Equivalent tokens into a single token to improve performance of the underlying mahine learning models.

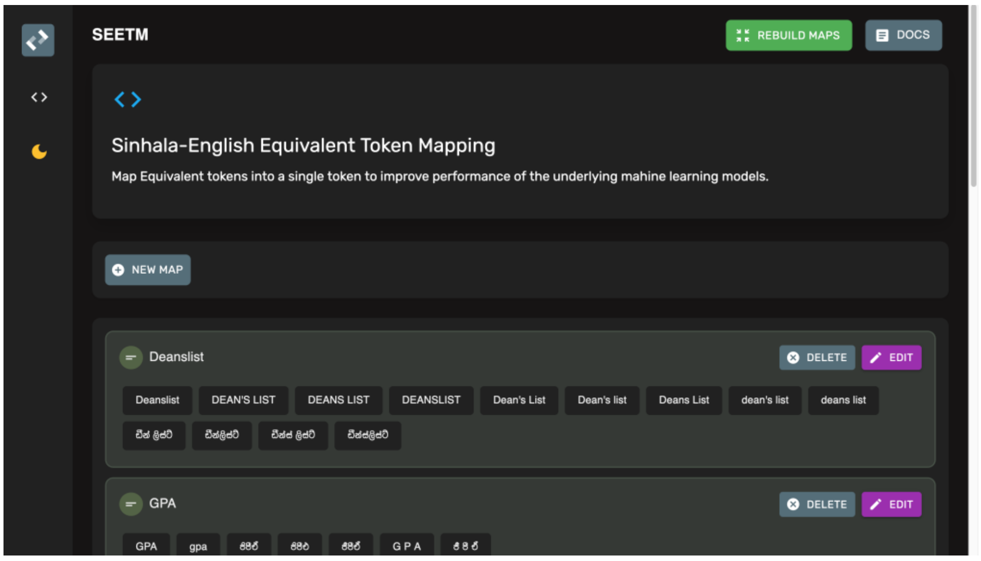
Deanslist

Deanslist | DEAN'S LIST | DEANS LIST | DEANSLIST | Dean's List | Dean's list | Deans List | dean's list | deans list | ඩීන් ලිස්ට් | ඩීන්ලිස්ට් | ඩීන්ස් ලිස්ට් | ඩීන්ස්ලිස්ට්

GPA

GPA | gpa | ජීපීඒ | ජීපීඑ | ජීපීඒ | G P A | ජී පී ඒ

Figure C.1. SEETM dashboard – light mode



Sinhala-English Equivalent Token Mapping

Map Equivalent tokens into a single token to improve performance of the underlying mahine learning models.

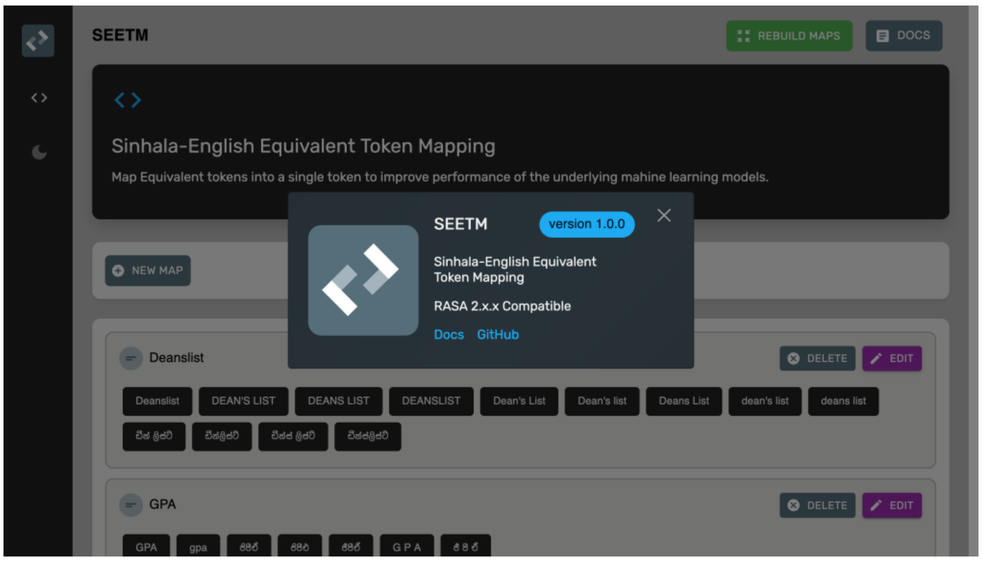
Deanslist

Deanslist | DEAN'S LIST | DEANS LIST | DEANSLIST | Dean's List | Dean's list | Deans List | dean's list | deans list | ඩීන් ලිස්ට් | ඩීන්ලිස්ට් | ඩීන්ස් ලිස්ට් | ඩීන්ස්ලිස්ට්

GPA

GPA | gpa | ජීපීඒ | ජීපීඑ | ජීපීඒ | G P A | ජී පී ඒ

Figure C.2. SEETM dashboard – dark mode


SEETM

REBUILD MAPS | DOCS

Sinhala-English Equivalent Token Mapping

Map Equivalent tokens into a single token to improve performance of the underlying mahine learning models.

SEETM version 1.0.0

Sinhala-English Equivalent Token Mapping

RASA 2.x.x Compatible

Docs GitHub

NEW MAP

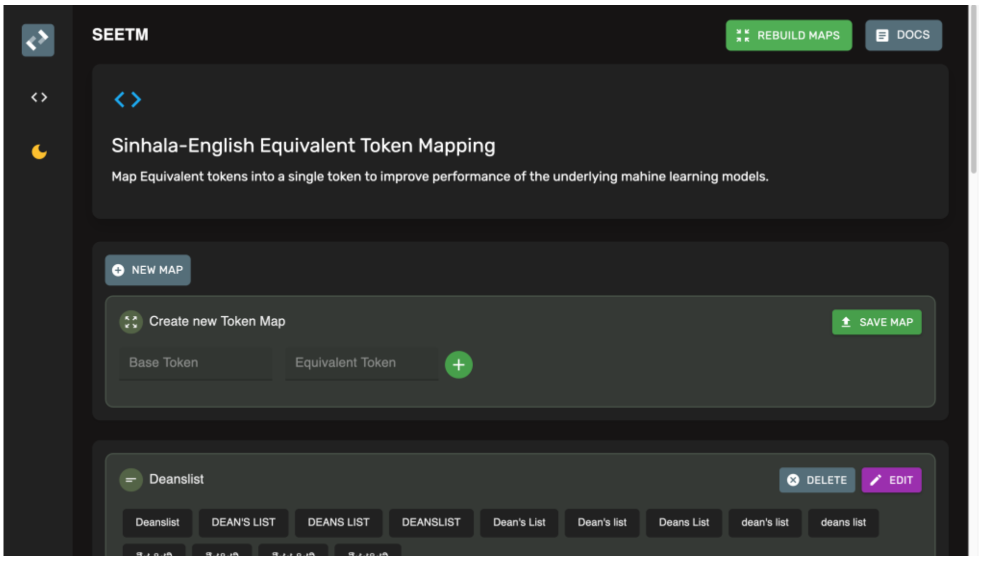
Deanslist — DELETE | EDIT

Deanslist | DEAN'S LIST | DEANS LIST | DEANSLIST | Dean's List | Dean's list | Deans List | dean's list | deans list | ඩීන් ලිස්ට් | ඩීන්ලිස්ට් | ඩීන්ස් ලිස්ට් | ඩීන්ස්ලිස්ට්

GPA — DELETE | EDIT

GPA | gpa | ජීපීඒ | ජීපීඑ | ජීපීඒ | G P A | ජී පී ඒ

Figure C.3. SEETM versioning information


SEETM

REBUILD MAPS | DOCS

Sinhala-English Equivalent Token Mapping

Map Equivalent tokens into a single token to improve performance of the underlying mahine learning models.

NEW MAP

Create new Token Map — SAVE MAP

Base Token | Equivalent Token

Deanslist — DELETE | EDIT

Deanslist | DEAN'S LIST | DEANS LIST | DEANSLIST | Dean's List | Dean's list | Deans List | dean's list | deans list

Figure C.4. New token maps creation UI component

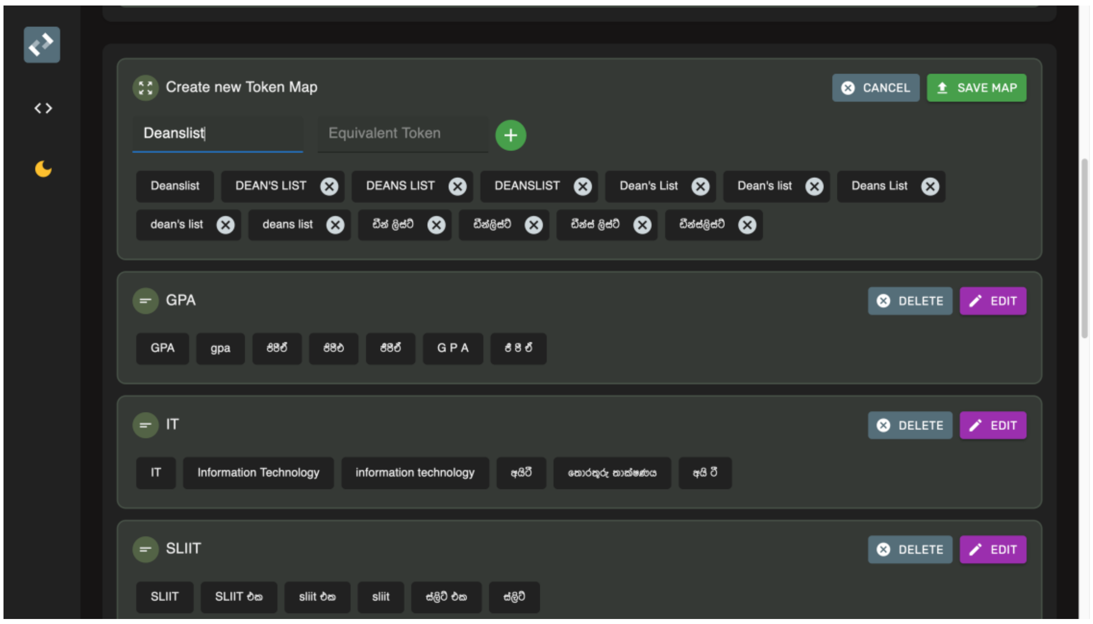
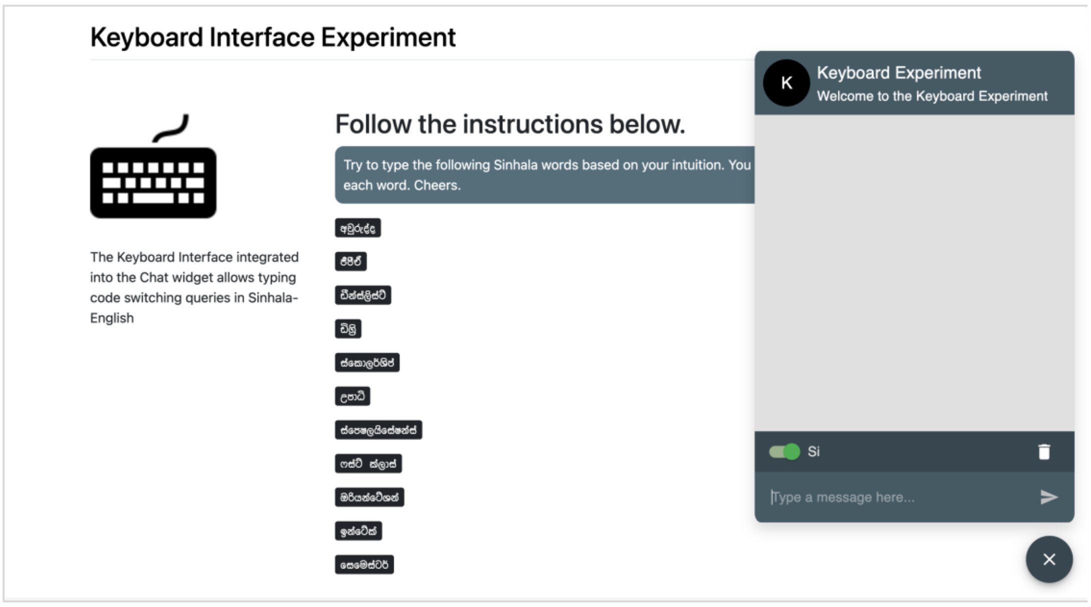


Figure C.5. Edit or remove already created token maps Feature

Appendix D: Keyboard Interface Testing Setup


Keyboard Interface Experiment

The Keyboard Interface integrated into the Chat widget allows typing code switching queries in Sinhala-English

Follow the instructions below.

Try to type the following Sinhala words based on your intuition. You
each word. Cheers.

අවුරුද්ද

ජීපීඒ

ඩීන්ස්ලිස්ට්

ඩිග්‍රි

ස්කොලර්ෂිප්

උපාධි

ස්පෙෂලයිසේෂන්ස්

ෆස්ට් ක්ලාස්

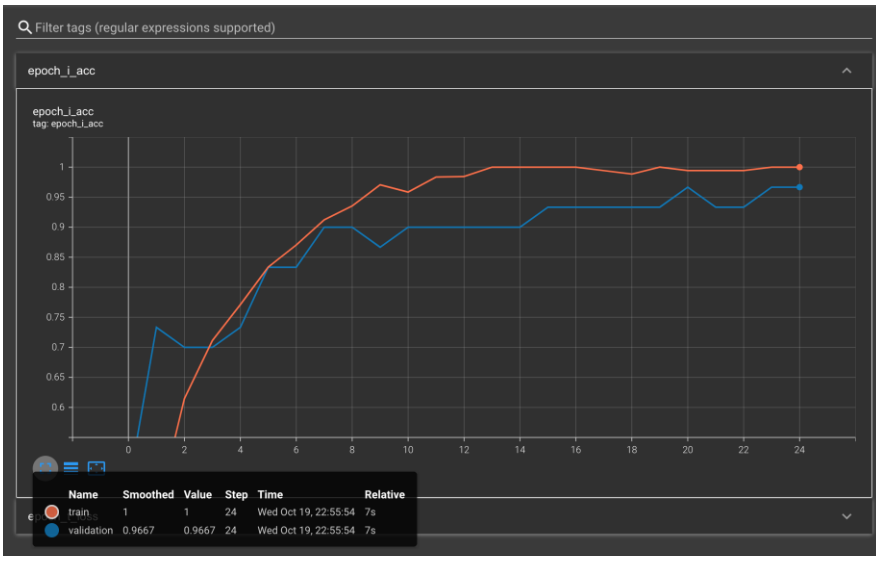
ඔරියන්ටේෂන්

ඉන්ටේක්

සෙමෙස්ටර්

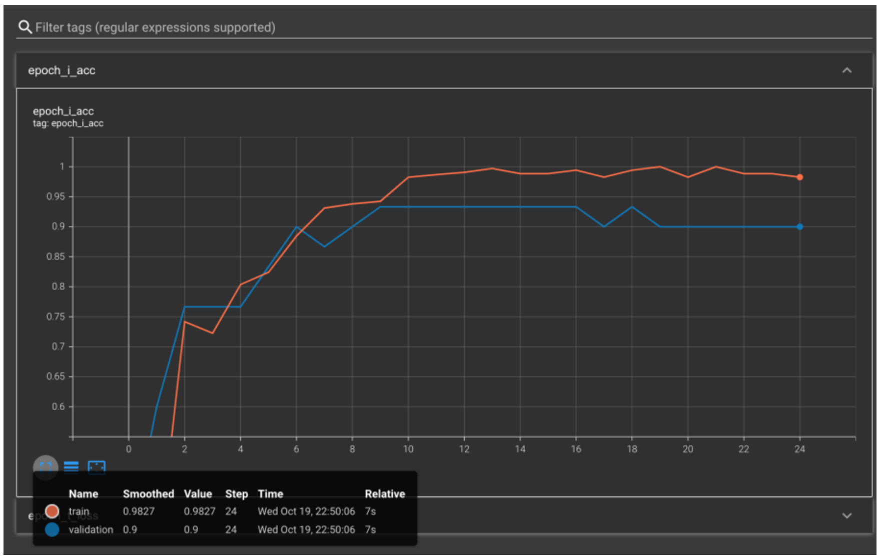
Figure D.1. UI of the bot trained without SEETM, utilized in the SEETM testing setup

Appendix E: Token Mapping Testing Setup



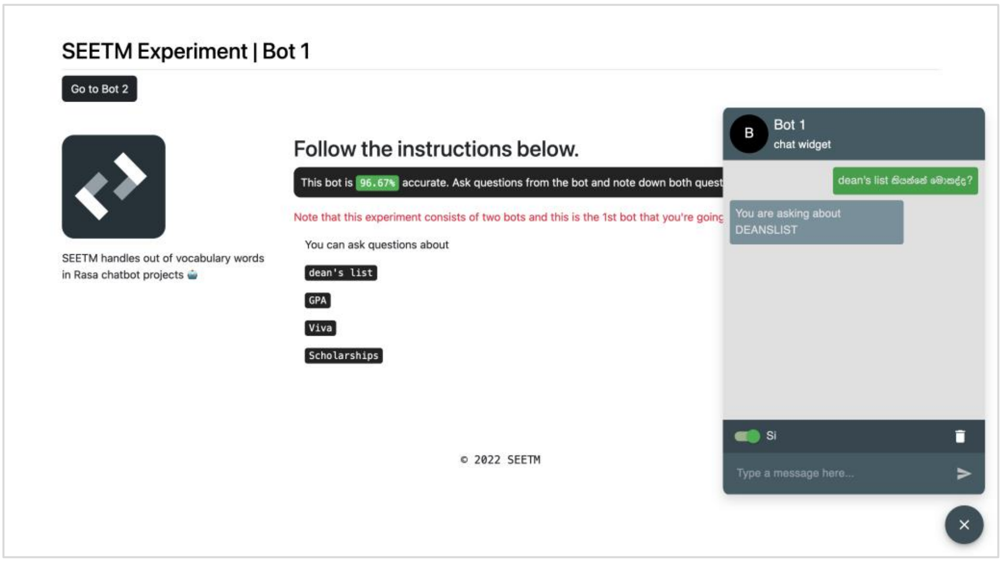
Name	Smoothed	Value	Step	Time	Relative
● train	1	1	24	Wed Oct 19, 22:55:54	7s
● validation	0.9667	0.9667	24	Wed Oct 19, 22:55:54	7s

Figure E.1. Accuracy score of the bot trained with SEETM, utilized in the SEETM testing setup



Name	Smoothed	Value	Step	Time	Relative
● train	0.9827	0.9827	24	Wed Oct 19, 22:50:06	7s
● validation	0.9	0.9	24	Wed Oct 19, 22:50:06	7s

Figure E.2. Accuracy score of the bot trained without SEETM, utilized in the SEETM testing setup


SEETM Experiment | Bot 1

Go to Bot 2

SEETM handles out of vocabulary words in Rasa chatbot projects

Follow the instructions below.

This bot is 96.67% accurate. Ask questions from the bot and note down both quest

Note that this experiment consists of two bots and this is the 1st bot that you're going

You can ask questions about

`dean's list`

`GPA`

`Viva`

`Scholarships`

© 2022 SEETM

Bot 1
chat widget

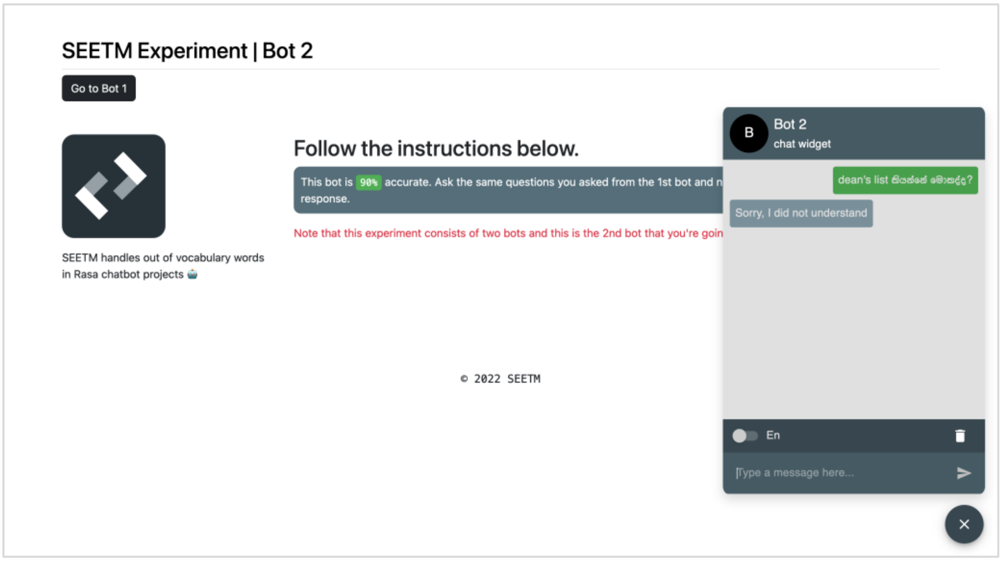
dean's list කියන්නේ මොකද්ද?

You are asking about DEANSLIST

Si

Type a message here...

Figure E.3. UI of the bot trained with SEETM, utilized in the SEETM testing setup


SEETM Experiment | Bot 2

Go to Bot 1

SEETM handles out of vocabulary words in Rasa chatbot projects

Follow the instructions below.

This bot is 90% accurate. Ask the same questions you asked from the 1st bot and n response.

Note that this experiment consists of two bots and this is the 2nd bot that you're goin

© 2022 SEETM

Bot 2
chat widget

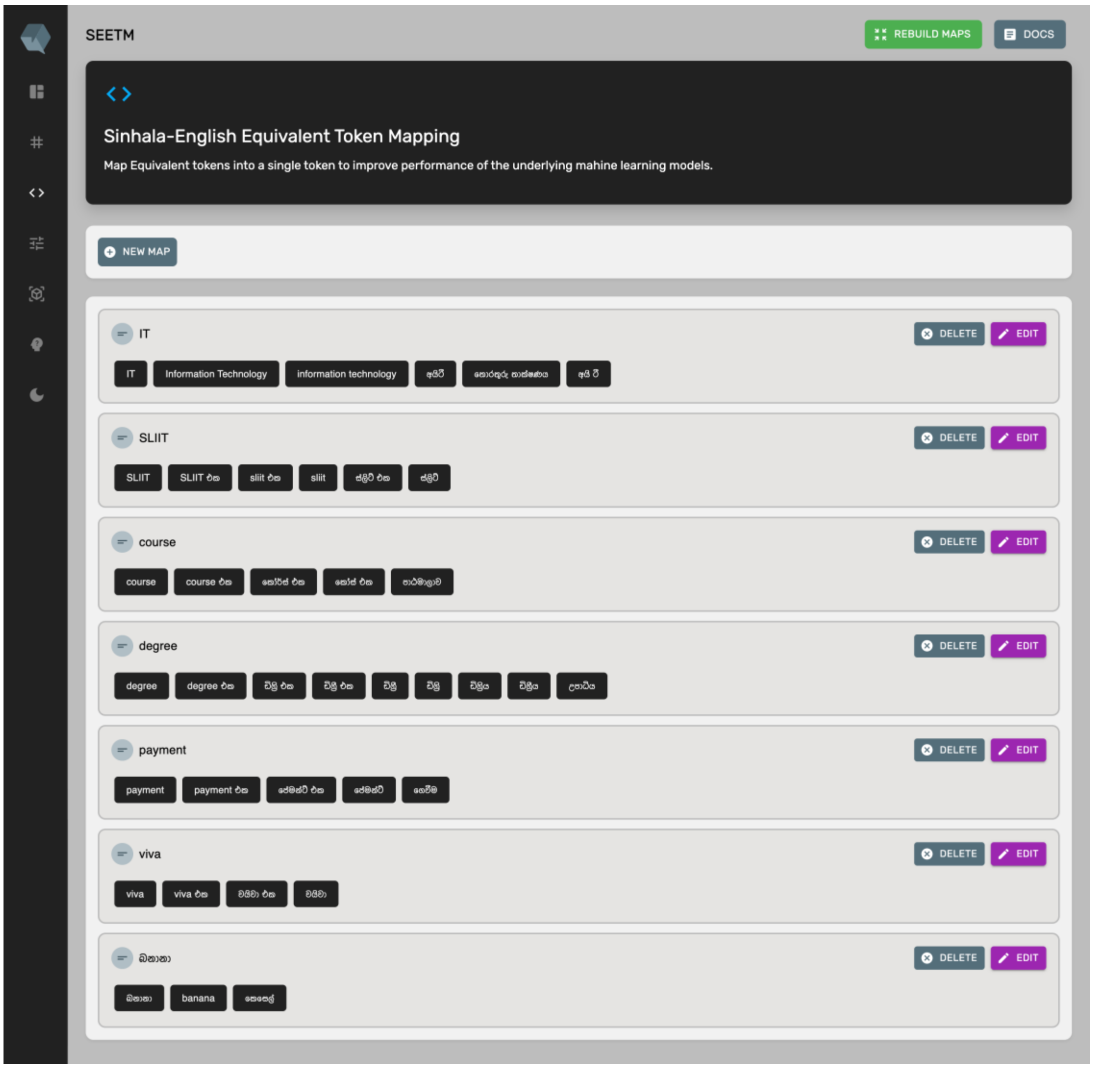
dean's list කියන්නේ මොකද්ද?

Sorry, I did not understand

En

Type a message here...

Figure E.4. UI of the bot trained without SEETM, utilized in the SEETM testing setup



Sinhala-English Equivalent Token Mapping

Map Equivalent tokens into a single token to improve performance of the underlying mahine learning models.

IT
IT | Information Technology | information technology | අයිටී | තොරතුරු තාක්ෂණය | අයි ටී

SLIIT
SLIIT | SLIIT එක | sliit එක | sliit | ස්ලිට් එක | ස්ලිට්

course
course | course එක | කෝර්ස් එක | කෝස් එක | පාඨමාලාව

degree
degree | degree එක | ඩිග්‍රී එක | ඩිග්‍රී එක | ඩිග්‍රී | ඩිග්‍රී | ඩිග්‍රිය | ඩිග්‍රීය | උපාධිය

payment
payment | payment එක | පේමන්ට් එක | පේමන්ට් | ගෙවීම

viva
viva | viva එක | වයිවා එක | වයිවා

බනානා
බනානා | banana | කෙසෙල්

Figure E.5. Token maps added to the bot trained using SEETM tokenizer (bot with maps)

Appendix F: Token Mapping Testing Observations

SEETM experiment observations are available at
https://jp7me8926m3.typeform.com/report/TAIYndZV/prP4Rbn1myC0qH3S