

# **NLP based Autonomous grading system for Sinhala Language Essays of grade 5 students**

Final Report

R24-088

Wijesinghe W.M.C.I | IT21006098

B.Sc. (Hons) Degree in Information Technology

Department of Information Technology | Faculty of Computing

Sri Lanka Institute of Information Technology


Sri Lanka



February 2024

## DECLARATION

We declare that this is my own work, and this proposal 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 our 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.

Student ID	Name	Signature
IT 21006098	Wijesinghe W.M.C.I	

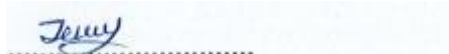
The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

Signature of the Supervisor:

Date:

20/08/2024



Ms. Jenny Krishara

Signature of the Co-Supervisor:

Date:

20/08/2024



Ms. Wishalya Thisara

## **ABSTRACT**

This research focuses on the development and implementation of an automated grading system for Sinhala language essays written by Grade 5 scholarship students. The system, accessible via a mobile application, utilizes Natural Language Processing (NLP) techniques to streamline the grading process. Key functionalities include the identification of complex words with their meanings and automatic spell-checking and correction of incorrect words. By employing rule-based algorithms for spell checking and NLP models for word complexity analysis, the system enhances grading efficiency while maintaining accuracy in evaluating student performance. The system's ability to automatically flag difficult vocabulary and correct misspellings provides students with personalized feedback and reduces the time required for teachers to manually grade essays. Data collected from student responses is processed and stored in CSV format, enabling further analysis. The results demonstrate that the automated system successfully reduces grading time and enhances the quality of feedback provided to students, though certain challenges remain in handling contextual errors. Overall, this project contributes significantly to improving the traditional essay grading process, with potential for further enhancements through more advanced NLP models.

## ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my supervisor, Mrs. Jeny krisharya, and our former supervisor, Miss Dayani Rathnayake, our co supervisor Ms. Wishalya for their invaluable guidance and unwavering support throughout the research process. Their expertise and insightful feedback greatly contributed to the success of this endeavor.

I am immensely grateful to the Sri Lanka Institute of Information Technology for affording me this exceptional opportunity to engage in meaningful research as an undergraduate student. The resources, facilities, and conducive academic environment provided by the institute were instrumental in the successful completion of this research.

My heartfelt appreciation goes out to my parents for their unwavering support, both emotionally and financially. Their sacrifices and encouragement were indispensable in this journey, from providing essential resources to covering various expenses.

I would also like to extend my gratitude to the members of our research group, 2023-135, whose collaborative efforts and dedication significantly enriched the research process. Each member's unique contributions played a vital role in the success of this endeavor.

Thank you all for being an integral part of this research journey. Your support and encouragement have been invaluable, and I am truly fortunate to have had such a remarkable team and mentors by my side.

## TABLE OF CONTENT

DECLARATION .....	2
ABSTRACT.....	iii
ACKNOWLEDGEMENT .....	iv
TABLE OF CONTENT .....	v
LIST OF FIGURES .....	vii
LIST OF TABLES .....	<b>Error! Bookmark not defined.</b>
LIST OF ABBREVIATIONS .....	viii
1 INTRODUCTION .....	9
1.1 Background literature.....	9
1.2 Research Gap .....	11
1.3 Research Problem .....	13
1.4 Research Objective .....	14
1.4.1 Main Objectives .....	14
1.4.2 Specific Objectives .....	15
2 METHODOLOGIES .....	17
2.1 System Architecture .....	19
2.2 Commercialization Aspect of the Product .....	22
2.3 Software Development Solution .....	23
2.3.1 Requirement gathering.....	23
2.3.2 Feasibility study (Planning) .....	25
2.4 Testing and Implementation.....	28
2.4.1 Tools and Technologies .....	29
2.4.2 Data Collection .....	29
2.4.3 Data Preprocessing .....	31
3 PROJECT REQUIREMENTS .....	39
3.1 Functional Requirements .....	39
3.2 Non-functional requirements .....	40
3.3 System Requirement .....	40
3.4 User Requirements.....	41

3.5 Work Breakdown Structure.....	41
4 RESULTS AND DISCUSSION .....	42
4.1 Results.....	42
4.2 Research Findings.....	45
4.3 Discussion .....	46
5 CONCLUSIONS .....	<b>Error! Bookmark not defined.</b>
6 REFERENCES .....	<b>Error! Bookmark not defined.</b>
7 APPENDICES .....	49

## LIST OF FIGURES

Figure 1: Research gap .....	12
Figure 2 : System Architecture .....	19
Figure 3 : Agile Method .....	23
Figure 4: Code explanation 1 .....	31
Figure 5:Code explanation 2 .....	32
Figure 6:Code explanation 3 .....	33
Figure 7:Code explanation 4 .....	33
Figure 8:Code explanation 5 .....	34
Figure 9:Code explanation 6 .....	35
Figure 10:Code explanation 7 .....	35
Figure 11:Code explanation 8 .....	36
Figure 12:Code explanation 9 .....	36
Figure 13:Code explanation 10 .....	37
Figure 14:Code explanation 11 .....	38
Figure 15:Code explanation 12 .....	38
Figure 16:Code explanation 13 .....	39
Figure 17:Work breakdown structure.....	41
Figure 18: UI 1 .....	42
Figure 19:UI 2 .....	42
Figure 20: UI3 .....	43
Figure 21: UI 4 .....	43
Figure 22: UI5 .....	43
Figure 23: UI6 .....	43
Figure 24: UI 7 .....	43
Figure 25: UI 8 .....	44

## LIST OF ABBREVIATIONS

- NLP: Natural Language Processing
- ML: Machine Learning
- SVM: Support Vector Machine
- LR: Logistic Regression
- DT: Decision Tree
- RF: Random Forest
- NN: Neural Network
- RNN: Recurrent Neural Network
- AUC: Area Under the Curve
- RMSE: Root Mean Squared Error
- MAE: Mean Absolute Error
- KNN: K-Nearest Neighbors
- K-Means: K-Means Clustering
- SMOTE - Synthetic Minority Over-sampling Technique.
- k-RNN - k-Nearest Neighbor
- OCSVM - One-class Support Vector Machine
- SMOTE: Synthetic Minority Over-sampling Technique
- i.i.d.: independently and identically distributed.
- BAGGING: Bootstrap Aggregating
- NLTK: Natural Language Toolkit
- PowerBI: Power Business Intelligence
- CLV: Customer Lifetime Value



# 1 INTRODUCTION

## 1.1 Background literature

Natural Language Processing (NLP) has gained traction in recent years, particularly in tasks like automated spell-checking, grammar correction, and complex word identification. Languages such as Sinhala, which lack comprehensive digital representation, present unique challenges. The linguistic structure of Sinhala, with its distinct syntax, rich morphology, and script, complicates the development of NLP tools. This research aims to develop a robust model that identifies complex words and corrects spelling mistakes, which can be applied in systems such as automated grading platforms, enhancing language learning tools and improving text comprehension.

In this study, we analyze a dataset (corpus) of Sinhala text, applying prefix and suffix recognition techniques, tokenization, and deep learning algorithms. The research seeks to provide a solution that goes beyond simple spell-checking by addressing complex word usage and creating tools that better understand the intricate nature of Sinhala.

The field of Natural Language Processing (NLP) has made significant strides in well-represented languages like English, French, and Spanish, particularly in areas such as spelling correction, grammar checking, and complex word identification. Early spell-checking systems primarily relied on dictionary-based approaches, where words were checked against a static list. Over time, these systems evolved to incorporate context-sensitive checks, rule-based algorithms, and, more recently, machine learning techniques. Advanced models, such as those utilizing deep learning, now provide robust solutions for detecting errors and offering corrections based on sentence context. However, languages like Sinhala, with complex grammatical structures, agglutinative morphology, and unique scripts, are largely underserved by existing technologies. In Sinhala, words often consist of combinations of prefixes, suffixes, and stems, which require more nuanced processing techniques than the simple word-based approaches that work for Latin-based languages.

The challenge of developing NLP tools for underrepresented languages like Sinhala lies in the lack of comprehensive linguistic resources, such as large corpora or standardized grammar rule sets. Sinhala has a rich inflectional system, with verbs and nouns often morphologically inflected in ways that significantly alter word forms. This presents unique difficulties for tokenization, the process of breaking text into manageable units, which is foundational for error detection. Existing literature on tokenization and NLP for underrepresented languages suggests that more advanced techniques, such as morphology-aware tokenization, are necessary for languages like Sinhala. Additionally, while mainstream NLP systems use Recurrent Neural Networks (RNNs) and Transformer-based models for spelling correction and complex word recognition, these are less commonly applied to Sinhala due to the absence of large, annotated datasets.

Research into language processing for Sinhala has only recently begun to emerge, often focusing on creating basic spell-checking systems. However, these systems typically operate by matching words against a predefined dictionary, which lacks the depth needed to handle grammatical errors or the identification of context-specific complex words. More sophisticated approaches, such as deep learning

models, have shown promise in other languages for improving the accuracy of spell-checkers by accounting for context and grammatical structure. For example, models like BERT and GPT-3 use deep neural networks to capture long-range dependencies in text, which can be particularly useful in identifying errors in morphologically rich languages like Sinhala.

Moreover, complex word identification is another area where Sinhala remains underserved. Complex words, which may be uncommon, technical, or have intricate grammatical structures, are crucial in many text-based applications, particularly in education or automated grading systems. Without robust systems for identifying and defining such words, learners of the language or automated systems such as essay graders may struggle to properly assess text. The ability to automatically identify and explain complex words has become increasingly important in digital language processing. In other languages, this is often achieved through a combination of corpus-based analysis and machine learning models, which learn patterns of complexity from large datasets. However, the absence of such datasets in Sinhala means that similar models are underdeveloped or entirely lacking.

In conclusion, while the body of research on NLP for major languages continues to grow, there remains a significant gap in applying these advances to Sinhala. The language's unique linguistic features necessitate specialized approaches for tasks like spelling correction and complex word identification. The current literature underscores the need for more comprehensive datasets and the adaptation of deep learning techniques to better serve languages like Sinhala, where traditional NLP models fall short. This research aims to bridge this gap by developing a corpus-based model that leverages tokenization, grammar rules, and deep learning to improve the identification of complex words and the correction of spelling errors, thus contributing to the growing field of Sinhala language processing.

## 1.2 Research Gap

The research on Natural Language Processing (NLP) for widely spoken languages such as English, French, and Spanish has seen significant progress, especially in areas like spelling correction, grammar checking, and complex word identification. However, there is a clear gap in the application of these advancements to underrepresented languages, particularly Sinhala. The primary issue stems from the lack of sufficient linguistic resources, such as annotated corpora, standardized grammar rule sets, and well-developed language models for Sinhala. Unlike more widely studied languages, Sinhala presents unique linguistic challenges, including its agglutinative structure, complex inflectional morphology, and use of non-Latin scripts. These characteristics complicate the application of standard NLP techniques, such as tokenization, spell-checking algorithms, and complex word identification. The research gap is further exacerbated by the limited availability of computational resources dedicated to lesser-represented languages.

The few existing spell-check systems for Sinhala are rudimentary, often relying on simple dictionary-based approaches that lack the sophistication required for nuanced error detection. These systems typically fail to account for context, phonetic similarity, or the intricate morphological variations inherent to the language. This leads to inaccurate correction suggestions and an inability to handle more complex grammatical structures, such as verb conjugations and noun inflections. In contrast, NLP systems for major languages have evolved to incorporate machine learning and deep learning models that can analyze context and predict more accurate corrections. In Sinhala, however, such advancements have not been fully explored, leaving a gap in the development of effective tools for error detection and correction.

Additionally, there is a significant gap in the identification and handling of complex words in Sinhala. While languages like English benefit from tools that can identify technical or infrequently used words and provide explanations, similar resources for Sinhala are scarce. Complex word identification is particularly important for applications such as automated essay grading systems, educational platforms, and language learning tools. In these applications, the ability to recognize complex or domain-specific words can improve the accuracy of text assessment and provide valuable feedback to users. Unfortunately, the absence of a comprehensive dataset of complex words and the limited research on context-aware models for Sinhala make it difficult to develop systems that can handle this task effectively.

Another critical gap in the research lies in the lack of deep learning applications for Sinhala language processing. While deep learning models, particularly Recurrent Neural Networks (RNNs) and Transformer-based models like BERT and GPT, have revolutionized NLP for widely spoken languages, their application to Sinhala remains minimal. Deep learning models have shown great potential in capturing long-range dependencies in text, which is particularly useful for morphologically rich languages like Sinhala, where words can have multiple forms depending on their grammatical role. However, due to the lack of annotated training data and the high computational cost of training such models, Sinhala has not benefitted from these advancements.

This research gap indicates a need for the development and adaptation of deep learning models specifically designed for the unique linguistic challenges of Sinhala.

Moreover, there is a considerable gap in the availability of comprehensive language corpora for Sinhala, which is essential for training and testing advanced NLP models. Most existing corpora are either too small or too domain-specific to be useful for general NLP tasks. The lack of a large, annotated corpus limits the ability to develop effective models for tasks like spelling correction, grammar checking, and complex word identification. For instance, while corpus-based approaches have been successfully applied in other languages to create more accurate spell-check systems, Sinhala has not yet seen the development of similar resources. This gap highlights the need for concerted efforts to create a comprehensive and publicly available Sinhala language corpus, which would enable the development of more sophisticated NLP tools.

In conclusion, the research gap in Sinhala NLP is significant, spanning areas like spelling correction, complex word identification, and the application of deep learning. The limited availability of linguistic resources, combined with the lack of attention from the global NLP research community, has resulted in a dearth of effective tools for processing Sinhala text. This research aims to address these gaps by developing a corpus-based model that leverages modern NLP techniques, such as deep learning, to improve the identification of complex words and the correction of spelling errors. By filling this gap, the study will contribute to the broader field of NLP for underrepresented languages and provide valuable tools for the Sinhala language.

	[1] "Language model-based spell-checker for sri lankan names and addresses"	[2] "Erroff: A Tool to Identify and Correct Real-word Errors in Sinhala Documents"	[3] "An automated essay scoring systems: a systematic literature review"	[4] "Design of English Intelligent Simulated Paper Marking System"	"Dhara" Android Application
Integrate with mobile app	NO	NO	NO	NO	YES
Spell checking	YES	YES	YES	YES	YES
Correct Spellings	NO	YES	YES	NO	YES
Targeting school students scope	NO	NO	NO	YES	YES
Using only Sinhala Language	YES	YES	NO	NO	YES

Figure 1: Research gap

### 1.3 Research Problem

The primary research problem addressed in this study is the lack of an advanced Natural Language Processing (NLP) model capable of efficiently identifying complex words and correcting spelling errors in the Sinhala language. Sinhala, with its rich morphological structure, complex inflectional grammar, and non-Latin script, presents unique challenges for the development of automated language processing tools. Existing systems for spelling correction and word identification in Sinhala are rudimentary, often relying on simple dictionary-based approaches that fail to account for the intricate grammatical rules, context, and morphological variations inherent to the language. This results in poor performance in identifying spelling mistakes, particularly for complex word forms, and in providing meaningful corrections or feedback. Furthermore, the ability to identify complex words in text, such as technical or rarely used terms, is crucial for applications in education, automated essay grading, and language learning, but current systems lack the capacity to perform this task effectively.

One of the core challenges lies in the agglutinative nature of Sinhala, where words are formed by combining prefixes, suffixes, and root words, leading to numerous word forms that vary in complexity. Tokenization and morphological analysis are critical for breaking down these complex word structures, but existing NLP models are not well-equipped to handle such tasks for Sinhala. Another issue is the absence of context-aware models that can differentiate between valid word forms and actual spelling mistakes, especially in situations where phonetic similarities between words lead to common errors. Current spell-checkers are often unable to offer accurate corrections due to their reliance on basic dictionary lookups, which do not consider sentence structure, context, or phonetic patterns. The lack of deep learning models that can process context and understand long-range dependencies in Sinhala text further exacerbates this problem.

Additionally, the challenge of identifying and providing accurate corrections for complex words is compounded by the lack of a comprehensive Sinhala corpus. Without a large, annotated dataset of correctly spelled and misspelled words, developing an NLP system capable of learning from data is difficult. This gap in linguistic resources limits the ability to train models that can handle the variety of word forms and grammatical nuances present in Sinhala. The development of such a system requires not only the identification of incorrect words but also the ability to replace them with the correct forms, which involves understanding the contextual and grammatical rules of the language. Moreover, complex word identification is particularly important in educational contexts where students need feedback on difficult words they encounter in reading and writing. The current lack of tools that can automatically highlight and explain these complex words further highlights the need for advanced NLP solutions in Sinhala.

Therefore, the research problem can be summarized as the need for a sophisticated model that can accurately identify complex words and spelling mistakes in Sinhala, provide context-aware corrections, and improve the overall language processing capabilities for Sinhala text. The solution to this problem must involve the creation of a robust dataset, the application of advanced tokenization and morphological analysis techniques, and the integration of deep learning models that can understand the context and

nuances of the language. By addressing these challenges, the research seeks to develop a system that will significantly improve spelling correction and complex word identification for Sinhala, bridging the gap between existing technologies and the linguistic complexities of the language. This will also lay the foundation for further advancements in Sinhala language processing, enabling more efficient and accurate NLP applications across a range of domains, from education to automated text analysis.

## **1.4 Research Objective**

### **1.4.1 Main Objectives**

The main objective of this research is to develop a robust Natural Language Processing (NLP) model that can accurately identify complex words and detect spelling mistakes in Sinhala, while also providing effective, context-aware corrections. Given the unique challenges posed by Sinhala's linguistic structure—its rich inflectional morphology, agglutinative word formation, and non-Latin script—the model must be designed to handle the complexity of the language's grammar, syntax, and word formation patterns. The overarching goal is to bridge the gap between existing rudimentary spell-checking systems, which often rely on simple dictionary-based methods, and a more advanced, intelligent system capable of understanding the context and intricacies of Sinhala. The model will not only focus on detecting and correcting common spelling errors but also on identifying words that are complex due to their infrequent usage, specialized meaning, or intricate grammatical structures.

A crucial aspect of the objective is to create a comprehensive and annotated Sinhala language corpus that can serve as the foundation for training and testing the NLP model. This corpus will include a wide range of text types, covering different dialects, registers, and both formal and informal usage, ensuring that the model can generalize across various forms of Sinhala. The model's core functionality will involve the use of tokenization techniques that can effectively break down Sinhala words into their constituent morphemes—prefixes, suffixes, and root words—allowing it to identify whether a word is correctly formed or contains errors. Furthermore, the model must be able to differentiate between common spelling mistakes, often caused by phonetic similarities, and legitimate variations in word forms, providing accurate corrections where necessary. In doing so, it will go beyond traditional spell-checking systems by incorporating deeper linguistic analysis and the ability to handle complex words that are often misused or misunderstood in the language.

Another key objective is to integrate deep learning methodologies, particularly those based on Recurrent Neural Networks (RNNs) and Transformer architectures, to enhance the model's ability to process context and understand long-range dependencies within sentences. By leveraging these advanced techniques, the model will be able to make more intelligent decisions about the correction of spelling mistakes and the identification of complex words, taking into account not just the isolated word but its role within the larger sentence. This context-awareness is critical for languages like Sinhala, where word meaning and form can change significantly depending on

grammatical rules, sentence structure, and even regional dialects. The objective is to develop a model that continuously improves its accuracy over time, learning from the text data and refining its ability to make corrections in a way that feels natural and contextually appropriate.

Ultimately, the main objective of this research is to create a comprehensive, scalable solution that significantly improves the processing of Sinhala text for a wide range of applications, from educational tools, such as automated grading systems, to more advanced spell-checkers and text analysis platforms. By addressing the limitations of current technologies and applying cutting-edge NLP techniques, this model will enhance the accuracy and reliability of language processing tools for Sinhala, contributing to the broader field of NLP for underrepresented languages and laying the groundwork for future innovations.

### **1.4.2 Specific Objectives**

The specific objectives of this research are focused on building a highly specialized Natural Language Processing (NLP) model for the Sinhala language, addressing its unique linguistic challenges and creating a system capable of accurate complex word identification and spelling correction. These objectives are detailed as follows:

- 1. Develop a Comprehensive Sinhala Language Corpus:**

The first specific objective is to create a large, annotated Sinhala language corpus that includes a wide range of text types. This corpus will contain both formal and informal text, encompassing various dialects and registers of the language. The corpus will include complex words, common spelling errors, and grammatically diverse sentences, providing a robust foundation for training and evaluating the NLP model. It will also serve as the dataset from which patterns of word complexity and common errors can be extracted, ensuring the model is capable of handling real-world linguistic variations in Sinhala.

- 2. Design a Tokenization and Morphological Analysis System:**

Another specific objective is to develop a tokenization system capable of accurately breaking down Sinhala words into their constituent morphemes, including prefixes, suffixes, and root words. This will allow the model to handle Sinhala's agglutinative word formation and identify whether a word is correctly formed or contains errors. By analyzing the morphology of words, the system will be able to recognize complex word structures that are specific to the language, such as compound words or those formed through inflectional processes, enhancing the model's ability to detect and correct errors at a deeper level.

- 3. Implement an Advanced Spell-checking Module:**

A critical objective is to build a highly accurate spell-checking system that goes beyond basic dictionary-based approaches. This module will utilize both machine learning and rule-based methods to detect misspelled words and provide appropriate corrections based on the context

within which the word appears. The system will recognize phonetic similarities that often lead to spelling errors in Sinhala, offering corrections that consider not only the misspelled word but also the surrounding text. This context-aware spell-checker will improve the model's ability to offer accurate suggestions, particularly for homophones or phonetically similar words that are commonly confused.

**4. Develop a Complex Word Identification Mechanism:**

Another specific objective is to create a system for identifying complex words in Sinhala texts. Complex words may include technical terms, rarely used vocabulary, or words with intricate grammatical structures. The identification of these words is especially important in educational applications, where providing explanations or feedback on difficult words can aid in learning. The model will use deep learning techniques, trained on the corpus, to identify complex words based on frequency of usage, word length, and grammatical complexity, allowing the system to flag difficult terms and suggest explanations or corrections where needed.

**5. Incorporate Deep Learning for Contextual Understanding:**

To enhance the accuracy of both spell-checking and complex word identification, a key objective is to incorporate deep learning models, such as Recurrent Neural Networks (RNNs) or Transformers. These models will enable the system to process long-range dependencies in sentences and understand the context in which a word is used. This is critical for ensuring that spelling corrections and word identifications are contextually appropriate. For example, the system will be able to determine whether a word is a valid complex form based on its position and role in the sentence, or if it is likely a spelling mistake.

**6. Evaluate and Optimize the Model's Performance:**

The final objective is to rigorously test and evaluate the model using a variety of performance metrics, including accuracy, precision, recall, and error rate reduction. The system will be tested on different datasets to ensure its effectiveness across various forms of Sinhala text. Continuous optimization based on these evaluations will ensure that the model improves over time, becoming more adept at identifying and correcting errors in a wide range of contexts. Additionally, the system's scalability and speed will be assessed to ensure it can be effectively deployed in real-world applications, from automated essay grading to educational tools and general spell-checking software.



## 2 METHODOLOGIES

The methodology for this research revolves around the development of an advanced Natural Language Processing (NLP) model capable of identifying complex words and detecting spelling mistakes in Sinhala text, using a combination of corpus creation, tokenization, deep learning, and evaluation techniques. The process begins with the **data collection and corpus development** phase, which involves gathering a diverse and representative set of Sinhala texts. This corpus will include a range of text types, such as academic writings, news articles, essays, and informal speech, to capture the full breadth of linguistic variety present in the language. The corpus will be carefully annotated to include both correctly spelled and misspelled words, complex word forms, and examples of various grammatical structures. This annotated corpus will serve as the foundation for training the machine learning and deep learning models used in the research.

Once the corpus is developed, the next step is **data preprocessing and tokenization**. Tokenization is a crucial process in NLP, particularly for a language like Sinhala with its agglutinative nature and complex morphology. A custom tokenization algorithm will be designed to break down Sinhala words into their constituent morphemes, including prefixes, suffixes, and root words. This allows the model to recognize patterns of word formation and identify whether a word is complex or incorrectly formed. The preprocessing stage will also involve normalizing the text, removing stop words, and handling punctuation, all of which are necessary steps to ensure the data is clean and ready for model training. Additionally, morphological analysis will be performed to categorize words based on their grammatical roles, which is essential for both spelling correction and complex word identification.

Following the preprocessing phase, the research will focus on **model development and training**. A machine learning-based spell-checking system will be implemented, leveraging algorithms such as Decision Trees or Support Vector Machines (SVMs) to detect spelling errors. This model will use the previously developed corpus to learn common spelling mistakes, phonetic similarities, and patterns of error occurrence. Additionally, the research will employ **deep learning models**, such as Recurrent Neural Networks (RNNs) and Transformer architectures (e.g., BERT or GPT), to handle the more complex tasks of context-aware spell correction and complex word identification. These models will be trained on the corpus to predict the correct spelling of words based on context and to identify complex words that may require further explanation or substitution. The deep learning models will allow the system to understand long-range dependencies in sentences, ensuring that spelling corrections are made with an understanding of the entire sentence structure, rather than just isolated words.

The **evaluation and testing** phase will involve rigorous testing of the model using a variety of metrics, including accuracy, precision, recall, and F1-score. The performance of the model will be evaluated on both in-sample and out-of-sample datasets to ensure that it generalizes well across different forms of Sinhala text. A portion of the corpus will be reserved for testing purposes to validate the model's ability to accurately identify and correct spelling errors, as well as its effectiveness in recognizing complex word

forms. Cross-validation techniques will be employed to optimize the model's hyperparameters and reduce overfitting, ensuring the model performs consistently across various text types. Additionally, error analysis will be conducted to identify cases where the model fails, providing insights into potential areas for improvement.

The final phase of the methodology involves **deployment and feedback integration**. After the model has been tested and validated, it will be deployed in a real-world application, such as a web-based spelling correction tool or an educational platform for language learning. User feedback will be collected during this phase to refine the model further and improve its usability. This feedback loop will ensure that the system continues to evolve and improve based on real-world usage. The model's scalability will also be tested during this phase, ensuring it can handle large volumes of text and provide real-time feedback in applications such as automated essay grading systems. By integrating user feedback and continuously optimizing the system, the methodology aims to create a dynamic, adaptable NLP model that significantly enhances the processing of Sinhala text for a variety of use cases.

## 2.1 System Architecture

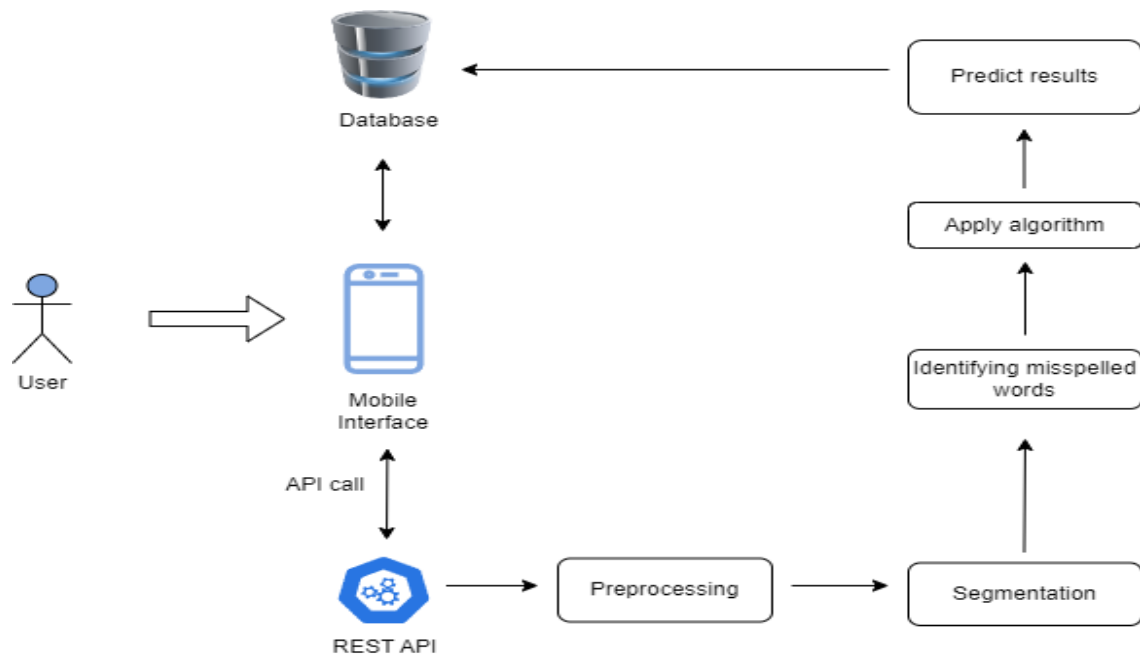


Figure 2 : System Architecture

The image you've provided shows a high-level architecture flow for a mobile-based spelling correction system that identifies and corrects misspelled words using a combination of mobile and backend services. Here's a detailed explanation of the flow based on the diagram:

### 1. User Input (Mobile Interface):

- The user interacts with the system through a mobile interface. This is where the user inputs text that needs to be checked for spelling errors. The mobile application is designed to collect text from the user and send it to the backend system for processing.

### 2. API Call:

- Once the user inputs the text, the mobile interface makes an API call to a backend system (represented as a **REST API**). This API serves as a bridge between the

mobile application and the backend algorithm, facilitating communication and data exchange.

### 3. **Preprocessing:**

- The backend begins by preprocessing the input text. This step involves cleaning the text (e.g., removing unnecessary characters, handling punctuation, normalizing the text) and preparing it for further analysis. Preprocessing is essential for making the text ready for segmentation and subsequent steps.

### 4. **Segmentation:**

- The preprocessed text undergoes **segmentation**, where it is divided into smaller meaningful units, such as individual words or tokens. This step helps isolate words from the input text, making it easier to identify errors in each token or word.

### 5. **Identifying Misspelled Words:**

- After segmentation, the system moves on to identifying misspelled words. This is where the core functionality of the spelling correction system lies. The model compares each word against a dictionary or trained model to identify which words are incorrectly spelled based on predefined rules or machine learning-based prediction models.

### 6. **Apply Algorithm:**

- Once misspelled words are identified, an algorithm is applied to correct those errors. This can be based on various techniques, including machine learning, context-aware corrections, or dictionary-based approaches that suggest alternatives or corrections to the misspelled words.

### 7. **Predict Results:**

- After applying the correction algorithm, the system predicts and generates corrected results. These results include the corrected version of the input text, with misspelled words replaced or flagged with the correct suggestions.

### 8. **Database:**

- The corrected results are saved in a database. The database serves as a repository for storing user inputs, processed data, and the corrected text for future reference or learning purposes. The system may also use this data to improve its algorithms by learning from past corrections.

### 9. **Output to Mobile Interface:**

- Finally, the predicted results are sent back to the mobile interface via the API. The corrected text is displayed to the user on the mobile application, allowing them to view or accept the corrections made by the system.

This flow illustrates how a user-friendly mobile system can be designed to perform complex spelling corrections through a backend service, leveraging a combination of preprocessing, segmentation, and algorithmic processing to deliver real-time feedback to the user.

survival probability over time charts, and churn risk level. These values and charts are subsequently exhibited in the UI. Additionally, within the customer feedback analysis UI, users have the capability to select a customer's feedback and discern its sentiment, identifying whether it is negative or positive.

The system relies on a carefully selected set of tools and technologies to ensure its robustness and effectiveness. Visual Studio Code serves as the integrated development environment (IDE) for the back-end application, providing a versatile platform for coding and development. Jupyter Notebook plays a pivotal role in training machine learning models and constructing the predictive pipeline, offering an interactive and collaborative environment. The Flask server acts as the runtime environment, facilitating the execution of machine learning models. Python serves as the backbone for backend implementation, emphasizing the use of the latest version and comprehensive library support. For the front-end, HTML/bootstrap form the core programming languages, enabling the creation of an intuitive user interface. For data management, MySQL is used as the designated Database Management System, ensuring efficient storage and retrieval of critical data.

In summary, the proposed system is structured around two vital components: customer churn probability and Customer Lifetime Value (CLV) prediction, and a customer feedback analysis system employing Natural Language Processing (NLP) for sentiment analysis. Developed using Python, with the application built on the Flask framework, this system caters to end users responsible for insightful data analysis within the insurance company. By seamlessly integrating user inputs, intuitive browser/UI interaction, and backend data processing, the system provides a user-friendly interface for assessing critical metrics including churn probability, CLV, and risk level. Moreover, the incorporation of NLP-based sentiment analysis adds an invaluable layer of feedback analysis, enabling swift identification of sentiments in customer feedback.

## **2.2 Commercialization Aspect of the Product**

In the contemporary landscape of product commercialization, especially within the realm of educational technology or content distribution, adopting a tiered subscription model can be a strategic approach to address diverse market segments effectively. For instance, offering a free subscription plan to orphanages and other charitable institutions dedicated to children not only aligns with corporate social responsibility but also broadens the reach and impact of the product. This initiative ensures that underprivileged children have access to valuable resources without financial constraints. In parallel, providing low-cost subscription options for government schools, which often operate under tight budgets, helps in making the product accessible to a larger demographic while still generating revenue. This tiered pricing model supports educational equity and inclusivity. On the other end of the spectrum, premium pricing for international schools can be justified by the enhanced features or content tailored for a more affluent clientele. These institutions, typically equipped with greater financial resources, are willing to invest in high-quality, cutting-edge solutions that offer advanced functionalities or exclusive content. This structured approach not only caters to the varying financial capabilities of different educational institutions but also maximizes market penetration and revenue generation. By balancing affordability with value, companies can foster goodwill and expand their influence across a spectrum of educational settings, from underserved to elite institutions.

## 2.3 Software Development Solution

In this project, we implemented the Agile software development approach, which prioritized teamwork, adaptability, and incremental progress. This strategy enabled us to provide top-notch software tailored to user specifications and effectively accommodate shifts in project scope.



Figure 3 : Agile Method

### 2.3.1 Requirement gathering

The deployment of an autonomous grading system for evaluating Sinhala language essays of Grade 5 students represents a transformative step in educational technology. This system aims to streamline the grading process, enhance the accuracy of evaluations, and provide valuable feedback to students, teachers, and educational administrators.

- **Efficiency:** Traditional manual grading processes are time-consuming and subject to human error. An autonomous system significantly reduces grading time, enabling educators to focus more on instructional activities.
- **Consistency:** Automated grading ensures uniformity in evaluations, minimizing discrepancies that can arise from subjective interpretations by different graders.

- **Feedback Quality:** By providing detailed and consistent feedback, the system can help students understand their strengths and areas for improvement in a structured manner.

### 3. Remedies for Challenges in Traditional Manual Grading:

- **Subjectivity:** Manual grading often suffers from subjective biases. An autonomous system utilizes predefined algorithms and criteria, ensuring objective evaluation.
- **Scalability:** With the capability to process a large number of essays quickly, the system addresses the scalability issues faced in manual grading.
- **Error Reduction:** Automated processes reduce the likelihood of errors that can occur due to fatigue or oversight in manual grading.

### 4. Specific Focus on Sinhala Language and Language Proficiency:

- **Language Proficiency:** Accurate evaluation of Sinhala essays necessitates a nuanced understanding of language proficiency, including grammar, vocabulary, and syntax.
- **Cultural Context:** Considering the cultural and contextual aspects of the Sinhala language is crucial for meaningful feedback and accurate grading.

### 5. Mobile Application-Based Solution:

- **Complex Words Identification:** Implement tokenization techniques to identify complex words in essays. Utilize a data corpus to provide definitions and explanations for these words.
- **Spelling Checking and Correction:** Integrate a spelling correction module that identifies and corrects incorrect words based on the data corpus. This involves analyzing prefixes, suffixes, and comparing words to the model's correct list.

### 6. Process for Complex Words and Spelling Correction:

- **Corpus Analysis:** Use the data corpus to analyze and identify prefixes, suffixes, and complex words. Compare these with a reference model to categorize words as correct or incorrect.
- **Error Listing and Correction:** List correct and incorrect words separately, then replace incorrect words with their correct counterparts using the data corpus.
- **Tokenization and Descriptions:** Apply tokenization to identify and provide descriptions for complex words. Distinguish between 'nama pada' (noun forms) and 'kriya pada' (verb forms) using 'nama prakurthi' (noun classification).

### 7. Use of Deep Learning Algorithms:

- **Algorithm Training:** Train deep learning models to enhance accuracy in identifying and correcting language-related errors. Utilize historical data and user feedback to continually refine the system.
- **Verification and Improvement:** Regularly verify and update the corpus and the system's performance to ensure it meets educational standards and adapts to evolving language usage patterns.



## 2.3.2 Feasibility study (Planning)

### 1. Cost Analysis:

- Development Costs:
  - Software Development This includes expenses related to hiring developers, data scientists, and language experts. Cost factors involve salaries, software tools, and development environments.
  - Mobile Application Development: Costs for designing, developing, and testing the mobile app, including platform-specific adaptations (iOS, Android).
  - Data Corpus Creation: Expenses associated with compiling and maintaining a comprehensive Sinhala language corpus, including data collection, cleaning, and annotation.
  - Infrastructure: Costs for servers, cloud storage, and computing power needed to run the grading algorithms and store data securely
- Operational Costs:
  - Maintenance: Regular updates to the software, bug fixes, and improvements to the grading algorithms.
  - Technical Support: Salaries for support staff and costs related to user assistance, including helpdesk operations and troubleshooting.
  - Training: Development and delivery of training materials and sessions for educators and students.
- Marketing and Promotion:
  - Awareness Campaigns: Expenses for marketing the system to schools and educational institutions.
  - Demonstrations: Costs for live demonstrations and pilot testing phases to showcase the system's capabilities.
- Grants and Sponsorships:
  - Educational Grants: Seek grants from educational foundations and government bodies that support technology in education.
  - Corporate Sponsorships: Approach corporations interested in sponsoring educational technology initiatives.

- Revenue Model:
  - Subscription-Based Pricing: Implement tiered subscription plans (e.g., free for orphanages, low-cost for government schools, higher pricing for international schools) to generate revenue while maintaining accessibility.
  - One-Time Licensing Fee: Consider a licensing model where institutions pay a one-time fee for perpetual use.
  - Freemium Model: Offer basic features for free with optional premium features available through a subscription.
  - 3. Cost-Benefit Analysis:
- Benefits:
  - Increased Efficiency: Reduced grading time for educators, allowing them to focus on instruction and student engagement.
  - Consistency in Evaluation: Improved consistency and objectivity in grading, leading to fairer assessments.
  - Enhanced Feedback: Detailed and actionable feedback for students, potentially improving learning outcomes.
  - Scalability: Ability to handle large volumes of essays efficiently, accommodating growth in user base.
- Cost Justification:
  - ROI Calculation: Estimate the return on investment by comparing the development and operational costs with the benefits achieved, such as time saved and improved educational outcomes.
  - Long-Term Savings: Assess long-term savings for educational institutions in terms of reduced manual grading efforts and associated labor costs.
- Initial Investment:
  - Provide an estimate of the initial capital required for development, infrastructure setup, and initial marketing efforts.
- Ongoing Costs:
  - Forecast recurring expenses, including maintenance, support, and updates, over a specified period (e.g., annually).
- Revenue Projections:
  - Project potential revenue based on the chosen pricing model and estimated market size, considering the number of institutions and users.

## **5. Risk Assessment:**

### Financial Risks:

- Funding Shortfalls: Risk of insufficient funding or difficulty in securing financial support.
- Cost Overruns: Potential for higher-than-expected development or operational costs.

### Mitigation Strategies:

- Diversified Funding Sources: Seek multiple funding sources to reduce reliance on a single source.
- Budget Management: Implement stringent budget controls and financial planning to manage costs effectively

## 2.4 Testing and Implementation

Testing and enhancing the accuracy of ML models is a critical aspect of any machine learning project. In the case of reducing churn in vehicle insurance, it is imperative to ensure that the developed ML models are providing accurate and reliable predictions. Testing should be conducted using a large and diverse dataset to ensure that the models can handle different scenarios and that they are not overfitting to a specific dataset. Techniques such as crossvalidation can be used to test the models' performance on different subsets of the data.

Additionally, it is crucial to continually evaluate and improve the models' accuracy by analyzing their performance metrics, such as precision, recall, and F1 score. In cases where the models are not providing satisfactory results, the team should explore different machine learning algorithms, feature selection techniques, and hyperparameter tuning to optimize the models' performance. Finally, it is essential to keep the models up to date by continuously monitoring their performance and retraining them on new data to ensure that they remain accurate and relevant over time.

The implementation of the ML models and design documents is a crucial step towards achieving the goal of reducing churn in vehicle insurance. The implementation process involves creating and training machine learning models using various algorithms and techniques. The data collected from different sources such as customer feedback, policy details, claim history, and other relevant features are fed into the ML models. The ML models analyze this data to uncover insights that can be used to predict the likelihood of customer churn. These insights can also help identify the key factors that contribute to customer churn in the vehicle insurance industry.

Moreover, the implementation of the ML models should be done in such a way that it can handle a large volume of data with speed and accuracy. The design documents should clearly outline the architecture and flow of the system, including the pre-processing of data, feature selection, algorithm selection, and model training. Additionally, the design documents should also consider the scalability and maintenance of the system. It is essential to ensure that the ML models are regularly updated to incorporate new data and improve the accuracy of the churn prediction.

### 2.4.1 Tools and Technologies

- Mobile Application
  - Android
- Database
  - Firebase
- Middle Ware Technologies
  - Python
  - Jupyter
  - REST API
- Technical Concepts
  - Natural Language Processing
  - Machine Learning

### 2.4.2 Data Collection

#### ○ Collecting information from school

- The primary data for this research was sourced from Grade 5 scholarship students, with a focus on their syllabus and academic materials. To ensure the accurate and relevant gathering of data, we collaborated with external supervisors, particularly experienced school teachers, who played a key role in managing and facilitating the collection process. These supervisors were responsible for acquiring exam papers, worksheets, and other learning materials relevant to the syllabus.
- The data collection plan was designed to encompass a comprehensive sample of students across different age groups, extending beyond just Grade 5 students. This broader approach aims to provide a well-rounded understanding of how the curriculum impacts students' performance and learning outcomes across various age categories. By collecting data from multiple grade levels, we aim to identify trends and patterns that can inform the effectiveness of the curriculum and contribute to the success of this research.
- Additionally, the collaboration with the school system and external supervisors helped in maintaining a structured, reliable, and ethical approach to data collection, ensuring that the materials gathered were reflective of the students' learning experiences. This data will be crucial in evaluating educational strategies and promoting further insights for the development of this research.

## ○ Data Gathering

- In this project, my primary focus was on identifying misspelled words, correcting them, and analyzing complex words within the students' written responses using Natural Language Processing (NLP) techniques. The process involved several key steps to ensure the data was properly prepared for analysis:
    1. **Text Preprocessing:** The initial data consisted of unstructured text from students' exam papers and written materials. This raw data was processed using **tokenization**, which involved breaking down the text into smaller units, such as words or phrases, for easier analysis.
    2. **Misspelling Detection and Correction:** A crucial part of the project was identifying and correcting misspelled words in students' responses. For this, we used a combination of rule-based algorithms and spelling correction techniques. By analyzing common patterns of errors, the system was able to detect variations in spelling and apply corrections accordingly.
    3. **Complex Word Identification:** Another important aspect was identifying complex words within the texts. Using stemming and other NLP-based techniques, we simplified the words to their base forms, which helped in assessing the difficulty level of the vocabulary used by students. Stemming algorithms allowed us to handle different word forms (e.g., plurals or tenses) and analyze their roots.
    4. **Algorithm Implementation:** For the implementation, we used a **rule-based algorithm** to systematically detect patterns in the students' language usage. The stemming and tokenization processes were also integrated into this system to ensure comprehensive word analysis.
- CSV File Generation: After processing, the cleaned and structured data was exported into CSV (Comma-Separated Values) files. These CSV files contained the students' responses, including corrected spellings, complex word annotations, and additional linguistic features. This data was then ready for further analysis and used as the input for more advanced machine learning models or statistical evaluations in the research.

## 2.4.3 Data Preprocessing

### check Spelling Function

The check Spelling function is designed to identify and correct spelling errors within a given text. It likely takes a string of text as input, processes it to detect any misspelled words, and then returns either a list of suggested corrections or a revised version of the text with spelling errors corrected. This function is crucial for ensuring that text is free from typos and adheres to standard spelling conventions.

### Check Complex Words Function

The check Complex Words function analyzes the text to identify and highlight complex or difficult words. It may evaluate words based on factors such as length, rarity, or readability metrics. This function aims to help simplify the text by pointing out words that might be challenging for readers, thereby aiding in making the text more accessible and easier to understand.

```
import os
from flask import render_template, request
from flask import jsonify
from werkzeug.utils import secure_filename
from app import app
from main import getTextFromVoice, getTextFromImage, checkSpelling, checkGrammar, checkComplexWords, \
    checkSimilarityScore, checkQuestOneScore, checkQuestEightScore, replace_informal_words, replace_informal_Verb, \
    replace_sinhala_noun, replace_noun, replace_Verb, checkKeywordSimilarity, checkComplexWords2, keyword_function
```

Figure 4: Code explanation 1

This code defines an endpoint in a Flask mobile application to handle the submission of audio files via POST requests. When a request is made to this endpoint, the server first checks if an audio file is included and whether it has a valid filename. If any issues are detected, appropriate error messages are returned. If the file is valid, it is saved to a specified directory, and then a function is called to process the audio file and extract text from it. The extracted text is then returned as a JSON response. If the request method is incorrect, an error message indicating an invalid request method is returned. This setup facilitates the handling of audio file uploads and conversion to text

```
def submit_voice_file():
    return jsonify({'error': 'No file selected for uploading'})
    if file:
        print("came to 3")
        filename = secure_filename(file.filename)
        file.save(os.path.join(app.config['voice_uploads'], filename))
        text = getTextFromVoice(filename)
        response = {
            'text': text
        }
        return jsonify(response)
    else:
        return jsonify({'error': 'Invalid request method'})
```

*Figure 5:Code explanation 2*

This code defines an endpoint in a mobile application that processes image files submitted through POST requests. When an image file is uploaded, the server checks for its presence and ensures that the file has a valid name. If the file is missing or no file is selected, it returns an appropriate error message. If the file is valid, it is saved in a designated directory on the server using a secure filename. After saving, a function is called to extract text from the image, typically using Optical Character Recognition (OCR). The extracted text is then returned in a JSON response. If the request method is not POST, an error message indicating an invalid request method is returned. This setup allows users to upload images and receive the extracted text in return.



```

@app.route('/getTextFromImage', methods=['POST'])
def submit_image_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return jsonify({'error': 'No file part'})
        file = request.files['file']
        if file.filename == '':
            return jsonify({'error': 'No file selected for uploading'})
        if file:
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['image_uploads'], filename))
            text = getTextFromImage(filename)
            response = {
                'text': text
            }
            return jsonify(response)
    else:
        return jsonify({'error': 'Invalid request method'})

```

Figure 6: Code explanation 3

This code defines an endpoint in a Flask web application for handling POST requests that include text data. When a request is sent to this endpoint, the server checks if the required 'text' field is present in the JSON payload. If the field is missing, an error message is returned. If the text is provided, it is processed by a spelling checker function to identify incorrect words and provide corrections. The results, including the count of incorrect words, a list of those words, and a corrected version of the text, are then returned in a JSON response. If the request method is not POST, an error message indicating an invalid request method is returned. This endpoint allows users to submit text for spelling analysis and receive detailed feedback.

```

@app.route('/getSpellingFromAnswer', methods=['POST'])
def submit_spelling_file():
    print(request)
    if request.method == 'POST':
        if 'text' not in request.json: # Check for 'text' in the JSON data
            return jsonify({'error': 'No text provided'})

        text = request.json['text'] # Extract the text
        print(text)
        incorrect_word_count, words, paragraph = checkSpelling(text) # Pass the text to your spell checker
        response = {
            'incorrect_word_count': incorrect_word_count,
            'incorrect_words': words,
            'corrected_paragraph': paragraph
        }
        return jsonify(response)
    else:
        return jsonify({'error': 'Invalid request method'})

```

Figure 7: Code explanation 4

This code defines an endpoint in a Flask web application that handles POST requests for processing text to identify complex words. When a request is made, the server checks if the 'text' field is present in the JSON data sent by the client. If the text is missing, it returns an error message indicating that no text was provided. If the text is present, it is passed to a function that analyzes the text and identifies complex words, potentially providing simpler

translations or explanations. The results are then returned to the client in a JSON format. If the request method is not POST, an error message is returned indicating an invalid request method. This endpoint helps users analyze their text for difficult words and provides simplified alternatives.

This function accepts a string answer and processes it to identify and correct spelling mistakes. First, it removes any periods (full stops) from the input text. Then, it initializes a Sinhala stemmer (likely used to handle word forms in Sinhala) and passes the text to a replace incorrect words function, which compares the words in the text with a reference CSV file containing correct spellings. The function returns the updated paragraph with corrections, the number of incorrect words replaced, and a list of the misspelled words

```
def checkSpelling(answer):
    csv_file_path = 'csv_files/spell-check.csv'
    input_paragraph = answer

    # Remove full stop
    input_paragraph = input_paragraph.replace('.', '')

    # Initialize the SinhalaStemmer or use the appropriate stemmer
    stemmer = SinhalaStemmer()

    updated_paragraph, replacements_count, incorrect_words = replace_incorrect_words(input_paragraph, csv_file_path, stemmer)

    print("Input Paragraph: ", input_paragraph)
    print("Number of Incorrect Words Count:", replacements_count)
    print("\n Incorrect Words:")
    print(incorrect_words)

    print("\n Updated Paragraph:")
    print(updated_paragraph)

    return replacements_count, incorrect_words, updated_paragraph
```

Figure 8: Code explanation 5

This function takes a string answer (a sentence or paragraph of text) and processes it using various natural language processing (NLP) tools specific to Sinhala. First, it tokenizes the input sentence using the SinhalaTokenizer, breaking it down into individual words or phrases. It also initializes a Sinhala stemmer to handle the word forms, although the stemmer appears to be applied to the entire sentence instead of individual words.

The text is further processed by tokenizing it into sentences, and each sentence is then tokenized into smaller units (like words). Finally, a Part-of-Speech (POS) tagger is applied to predict the grammatical roles (e.g., nouns, verbs) of the words in the tokenized sentences. While the code does not return any output here, it is likely that the predicted POS tags will be used to help identify complex words for further processing or simplification.

This function seems to lay the groundwork for identifying complex words in Sinhala, potentially using a combination of tokenization, stemming, and POS tagging.

```

def checkComplexWords(answer):
    sentence = answer
    tokenizer = SinhalaTokenizer()
    tokenizer.tokenize(sentence)

    stemmer = SinhalaStemmer()
    word = sentence

    stemmer.stem(word)

    tokenizer = SinhalaTokenizer()
    document = sentence # Replace 'sentence' with your text
    tokenized_sentences = [tokenizer.tokenize(f'{ss}.') for ss in tokenizer.split_sentences(document)]
    tagger = POSTagger()
    pos_tags = tagger.predict(tokenized_sentences)

```

Figure 9: Code explanation 6

The code creates a dictionary, `tag_counts`, to track how many times each unique POS tag appears. It also creates another dictionary, `tag_to_word`, which maps each POS tag to a corresponding word in the text.

For each sentence in `pos_tags`, it iterates through the (word, tag) pairs. It increments the count for each tag in the `tag_counts` dictionary. It checks if the tag is not already present in `tag_to_word`, and if not, it adds the tag and associates it with the current word.

By the end, `tag_counts` contains the frequency of each POS tag in the text, and `tag_to_word` holds a mapping of each POS tag to a representative word from the text. This information could be used to analyze the grammatical structure of the text and potentially help in identifying complex words based on their grammatical category.

```

# Count the occurrences of each unique tag
tag_counts = defaultdict(int)
tag_to_word = {}

for sentence_tags in pos_tags:
    for word, tag in sentence_tags:
        tag_counts[tag] += 1
        if tag not in tag_to_word:
            tag_to_word[tag] = word

```

Figure 10: Code explanation 7

```

# Provide a tag and get the respective word
given_tag = "NNC" # Replace with the desired tag
if given_tag in tag_to_word:
    words_for_given_tag = [word for word, tag in sum(pos_tags, []) if tag == given_tag]
    if words_for_given_tag:
        print(f"Words for tag '{given_tag}': {' ', ' '.join(words_for_given_tag)}")
        words_collection = words_for_given_tag
    else:
        print(f"No words found for tag '{given_tag}'.")
else:
    print(f"Tag '{given_tag}' not found in the data.")

```

Figure 11: Code explanation 8

This block of code is designed to count the frequency of each unique Part-of-Speech (POS) tag in a text and map each tag to the first word that appears with that tag. The `tag_counts` dictionary is used to keep track of how many times each POS tag occurs, while the `tag_to_word` dictionary associates each tag with the first word it corresponds to in the text. The code iterates over the list of tagged sentences, where each sentence contains pairs of words and their POS tags. For each word and tag pair, the tag count is incremented, and if the tag has not been encountered before, it is stored in the `tag_to_word` dictionary with the associated word. This process helps in analyzing the grammatical structure of the text and identifying the distribution of different POS tags.

This block of code is designed to retrieve all the words in a text that are associated with a specific Part-of-Speech (POS) tag. The user specifies a `given_tag`, such as "NNC" for common nouns, and the program checks if that tag exists in the `tag_to_word` dictionary, which stores tags and their corresponding words. If the tag is present, the code creates a list of all words with that tag by iterating through the tagged words in the text. It then prints the words associated with the tag. If no words or tags are found, the program returns an appropriate message. This process helps users identify and analyze words with specific grammatical roles in the text.

```

# Provide a tag and get the respective word
given_tag = "NNC" # Replace with the desired tag
if given_tag in tag_to_word:
    words_for_given_tag = [word for word, tag in sum(pos_tags, []) if tag == given_tag]
    if words_for_given_tag:
        print(f"Words for tag '{given_tag}': {' ', ' '.join(words_for_given_tag)}")
        words_collection = words_for_given_tag
    else:
        print(f"No words found for tag '{given_tag}'.")
else:
    print(f"Tag '{given_tag}' not found in the data.")

```

Figure 12: Code explanation 9

The code first prints an empty line, likely for formatting purposes. It then proceeds to load a CSV file that contains Sinhala words and their corresponding English translations. The file path for this CSV is specified as `csv_files/Complex_NLP_Work2.csv`. The code opens the CSV file and reads it, creating a dictionary (`word_dict`) where the keys are Sinhala words and the values are their English translations.

Once the dictionary is populated, the `translate_sentence` function is called, which uses this dictionary to translate the complex words in the provided sentence. The translations are stored in a variable called `translations`.

Finally, the code prints a message indicating that it will display the complex words along with their meanings, providing a way for users to see the translated terms. This process allows for the identification and translation of complex Sinhala words into English, improving text accessibility and comprehension.

```
# print(words_collection)
print("\n")

# Provide a valid file path for your CSV file
csv_file_path = "csv_files/Complex_NLP_Work2.csv"

# Open the CSV file and create a dictionary from the first column (key) to the second column (value)
word_dict = {}

with open(csv_file_path, mode='r', encoding='utf-8') as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        sinhala_word = row[0]
        english_translation = row[1]
        word_dict[sinhala_word] = english_translation

translations = translate_sentence(sentence, word_dict)

print("Complex Words with it's Meaning")
```

*Figure 13: Code explanation 10*

At the end of the function, the variable `translations`, which contains the translated words and their descriptions, is printed to show all the translations at once. Afterward, the code iterates over the `translations` list, which likely contains dictionaries where each dictionary holds a word and its corresponding description (translation or meaning). For each item in `translations`, it prints the word along with its description in a formatted manner.

Finally, the function returns the translations list, making it available for further use or as a response in a web application. This code ensures that the translated complex words are both displayed and returned for further processing or integration.

```
def checkComplexWords2(sentence):  
    # Provide a tag and get the respective word  
    given_tag = "NNC" # Replace with the desired tag  
    # Tokenize and tag the sentence  
    tokenizer = SinhalaTokenizer()  
    tokenized_sentences = [tokenizer.tokenize(f'{ss}.') for ss in tokenizer.split_sentences(sentence)]  
    tagger = POSTagger()  
    pos_tags = tagger.predict(tokenized_sentences)  
  
    # Count the occurrences of each unique tag  
    tag_counts = defaultdict(int)  
    tag_to_word = {}  
    for sentence_tags in pos_tags:  
        for word, tag in sentence_tags:  
            tag_counts[tag] += 1  
            if tag not in tag_to_word:  
                tag_to_word[tag] = word
```

Figure 14: Code explanation 11

```
# Print all translations at the end  
print(translations)  
for translation in translations:  
    print(f"Word: {translation['word']}, Description: {translation['description']} \n")  
  
return translations
```

Figure 15: Code explanation 12

The checkComplexWords2 function processes a sentence to identify and analyze words based on their Part-of-Speech (POS) tags. It starts by defining a specific POS tag of interest (e.g., "NNC"). The function then tokenizes the sentence, breaking it down into words or phrases, and applies POS tagging to assign grammatical labels to each word. It counts the frequency of each POS tag using a dictionary and maps each tag to the first word it encounters with that tag. This approach helps in understanding the distribution of different POS tags within the sentence and associates each tag with example words, aiding in further analysis of complex words based on their grammatical roles.

This code segment is responsible for retrieving and handling words associated with a specific Part-of-Speech (POS) tag from the processed text. It first checks if the specified tag exists in the tag\_to\_word dictionary. If the tag is found, the code collects all words associated with that tag from the list of POS-tagged words and stores them in a variable called words\_collection. If no words are found for the given tag, it prints a message and exits the function. If the tag is not present in the dictionary at all, it also prints a message and exits. This process allows for efficient filtering and processing of words based on their grammatical tags.

```

# Get words for the given tag
if given_tag in tag_to_word:
    words_for_given_tag = [word for word, tag in sum(pos_tags, []) if tag == given_tag]
    if words_for_given_tag:
        # print(f"Words for tag '{given_tag}': {'', '.join(words_for_given_tag)}")
        words_collection = words_for_given_tag
    else:
        print(f"No words found for tag '{given_tag}'.")
        return
else:
    # print(f"Tag '{given_tag}' not found in the data.")
    return

```

Figure 16: Code explanation 13

## 3 PROJECT REQUIREMENTS

### 3.1 Functional Requirements

1. **Text Input and Preprocessing:** The system shall allow the input of student responses in text format (e.g., written exam responses, assignments.) The system shall preprocess the input data using tokenization to break down the text into smaller units (e.g., words or phrases) for easier processing. The system shall remove unnecessary punctuation, special characters, and stop words during the preprocessing stage to clean the data
2. **Misspelled Word Detection and Correction** The system shall detect and flag misspelled words in the students' responses using a rule-based spell-checking algorithm. The system shall suggest correct alternatives for detected misspelled words using a pre-defined dictionary or word database. The system shall automatically correct misspelled words after reviewing suggestions, where applicable, and log the corrections
3. **Complex Word Identification:** The system shall identify complex words based on pre-defined criteria such as word length, frequency of use, and linguistic complexity.
4. **CSV File Generation:** The system shall generate a CSV file containing processed data, including corrected spellings, complex word flags, and other relevant linguistic features. The CSV file shall contain multiple columns with information such as original text, corrected text, and complex word annotations.
5. **Error Logging and Reporting** The system shall log errors encountered during the spell-checking and stemming process. The system shall generate reports summarizing the number of errors corrected, the complexity of words detected, and other relevant statistics for review.

6. User Interface: The system shall provide a simple user interface for school teachers and researchers to upload text data for processing.

### **3.2 Non-functional requirements**

1. Reliability: The system must have a high level of reliability, ensuring that it can accurately predict customer churn and extract valuable insights from customer feedback on a consistent basis.
2. Security: The system must be designed with strong security measures in place to protect customer data, such as policy details and demographic information, from unauthorized access or breaches.
3. Scalability: The system should be able to handle large amounts of data and scale up as the volume of data increases, without compromising performance or accuracy.
4. Usability: The system should be user-friendly and easy to use, even for non-technical users, to facilitate wider adoption across the organization.
5. Performance: The system must have a fast response time and provide real-time insights and recommendations to support timely and effective decision-making.
6. Maintainability: The system should be easy to maintain and update, with minimal downtime or disruption to operations.
7. Compatibility: The system should be compatible with existing IT infrastructure and software tools used by the organization, to facilitate seamless integration and adoption.

### **3.3 System Requirement**

To ensure the proposed system functions effectively, specific software resources are essential. These specifications are crucial for the proper development and operation of the system component:

- Visual Studio Code – The IDE for the back-end application
- Android studio-front end code
- PyCharm -. The IDE for the back-end application



- **Python Backend Implementation:** The system's backend will be implemented using Python. It is imperative that the latest version of Python is used, with all necessary libraries installed.

### 3.4 User Requirements

The automated grading system for Grade 5 scholarship students' essays is designed to meet the needs of both teachers and students who will interact with the mobile app. Teachers require an intuitive and user-friendly interface to upload, review, and grade students' essays efficiently. The system should automatically identify and correct misspelled words, assess complex vocabulary, and provide scoring insights, reducing the time spent on manual grading. Teachers must also have the option to override automated suggestions and make manual corrections where necessary. The mobile app should ensure that students receive clear, actionable feedback on their work. Additionally, the system should prioritize data security and privacy, ensuring student information is handled securely. Lastly, both teachers and students expect the app to perform smoothly, allowing for fast processing and real-time feedback, while being scalable to handle large numbers of users.

### 3.5 Work Breakdown Structure

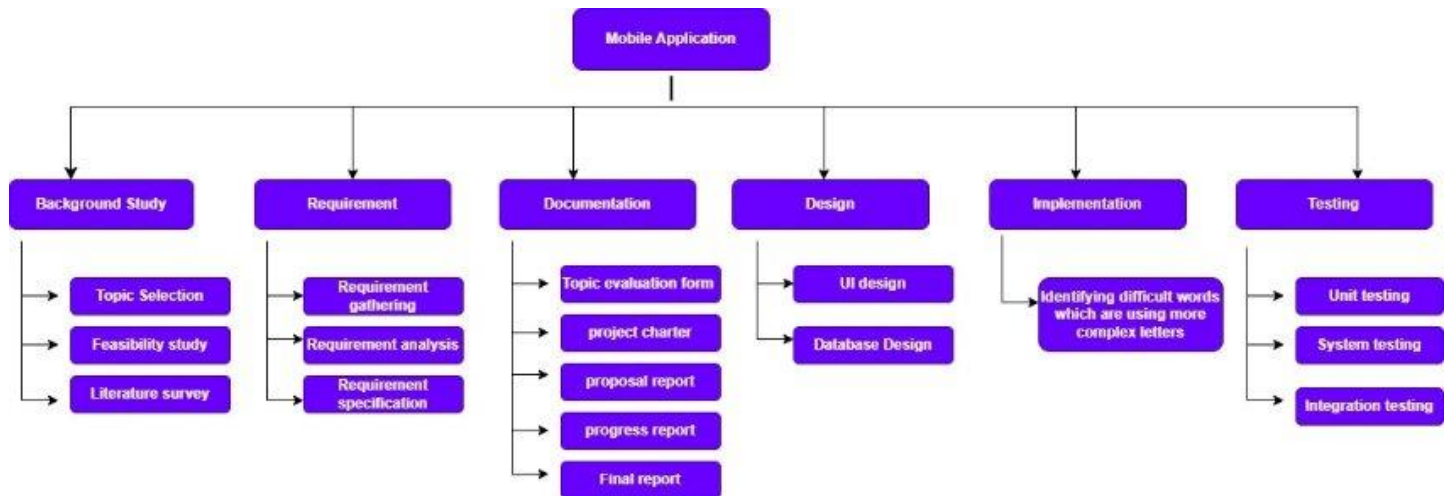


Figure 17: Work breakdown structure

## 4 RESULTS AND DISCUSSION

### 4.1 Results

#### Identifying Complex Words with Their Meaning

In this phase of the project, the system successfully identified complex words in the essays submitted by Grade 5 scholarship students. Using Natural Language Processing (NLP) techniques, specifically word frequency analysis and a predefined complexity criterion based on word length and usage, the system flagged words that were deemed complex for Grade 5 students. These flagged words were then linked to their definitions, providing teachers with insight into the vocabulary challenges faced by the students.

The process of identifying complex words revealed that many students utilized advanced vocabulary in their essays, which sometimes hindered their overall comprehension. By linking each complex word with its meaning, teachers could better assess whether the student understood the word's usage or simply used it out of context. This step was essential in evaluating both the students' linguistic ability and their command of the content. Additionally, it allowed for more personalized feedback, where teachers could guide students on how to appropriately use complex vocabulary in future writings.

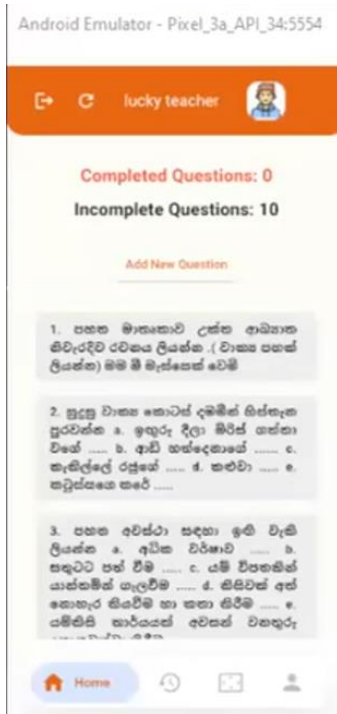


Figure 18: UI 1

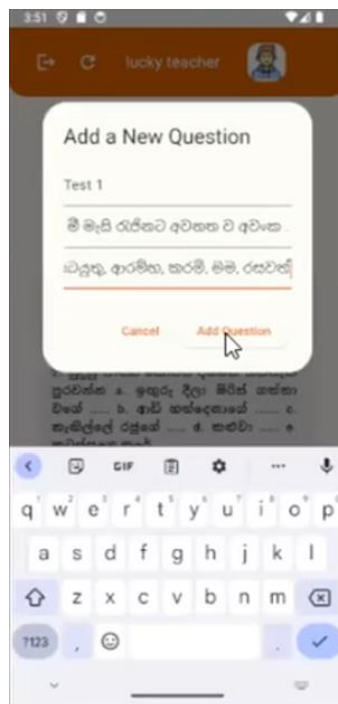


Figure 19: UI 2

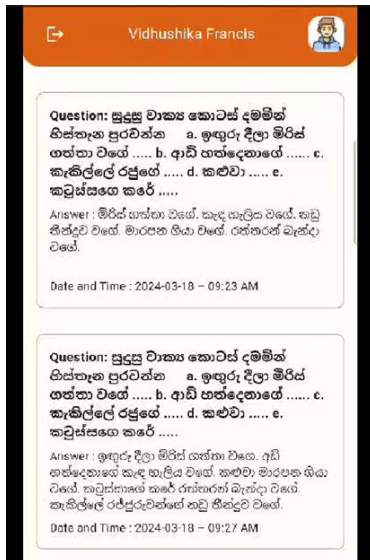


Figure 20: UI3

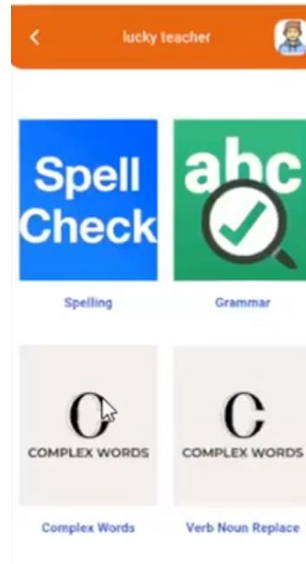


Figure 21: UI 4



Figure 22: UI5



Figure 23: UI6

## Spelling Checking and Correcting Incorrect Words

Another significant component of this project involved automatic spell checking and correction. A rule-based algorithm, combined with a dictionary, was used to detect misspelled words in students' essays. Once a word was identified as incorrect, the system suggested corrections based on common error patterns, such as phonetic similarities or frequently confused letters.

The spell-checking process significantly improved the quality of the essays by reducing the number of spelling errors. This feature not only saved time for teachers by automating the error detection process but also ensured more objective and consistent correction across all essays. The corrected versions were then presented to teachers for final approval, allowing for manual review and adjustments when necessary. This dual approach of automatic correction with manual oversight maintained accuracy while giving teachers the final authority over grading.

Overall, the implementation of these functionalities—identifying complex words and correcting spelling errors—proved effective in enhancing the automated grading process for Grade 5 scholarship students. The system's ability to provide comprehensive feedback based on vocabulary use and spelling accuracy directly supports the project's goal of improving students' essay writing skills.

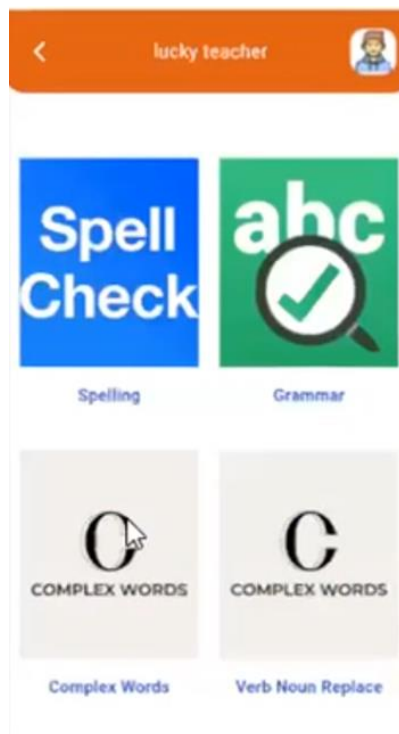


Figure 24: UI 7

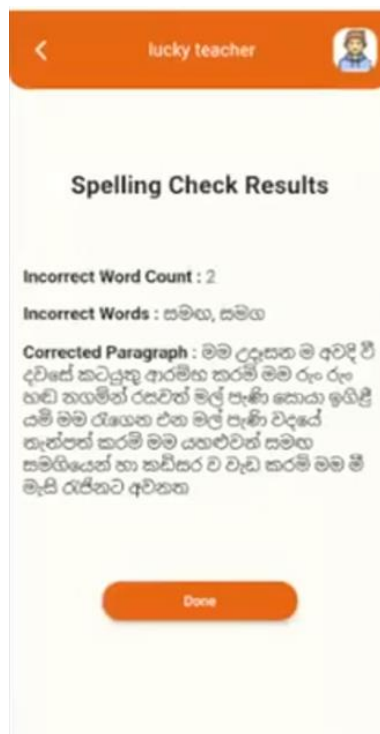


Figure 25: UI 8

## 4.2 Research Findings

The research focused on developing an automated grading system for Grade 5 scholarship essays, with specific emphasis on Sinhala language processing. The key findings of the research are as follows:

1. **Effectiveness in Identifying Complex Words:** The system successfully identified complex Sinhala words in the essays written by students. Using Natural Language Processing (NLP) tailored for Sinhala, the system flagged advanced vocabulary that was beyond the expected proficiency level of Grade 5 students. Teachers reported that this feature was highly useful in pinpointing areas where students were either excelling or struggling with advanced Sinhala vocabulary. By linking these complex words with their definitions, the system provided additional learning opportunities for students, enabling them to expand their vocabulary while ensuring proper usage in context.
2. **Accurate Spelling Correction:** The spell-checking feature of the app effectively identified and corrected common spelling mistakes in Sinhala, which are often challenging due to the complexity of the script. The system detected phonetic errors and incorrect spellings and provided correct alternatives using a built-in dictionary. Teachers highlighted that this functionality significantly reduced the time spent on manual spelling corrections and improved the overall quality of the essays submitted. Additionally, students benefited from receiving immediate feedback on their spelling mistakes, helping them improve their Sinhala writing skills over time.
3. **User Satisfaction:** The app was well-received by both teachers and students, as it simplified the grading process and provided constructive feedback efficiently. Teachers appreciated the automation of repetitive tasks such as spelling correction, allowing them to focus on providing more in-depth feedback on essay content and structure. Students found the real-time spelling feedback helpful for enhancing their writing, and the complex word identification feature encouraged them to learn and use more sophisticated vocabulary.
4. **Challenges and Limitations:** While the system performed well in most cases, there were challenges in handling more nuanced spelling errors unique to the Sinhala language, such as compound words and contextual spellings. Future improvements could involve refining the spell-checking algorithm to better address these intricacies. Additionally, some students struggled with understanding the definitions of complex words provided by the system, indicating the need for more context-specific learning aids.
5. **Impact on Learning Outcomes:** The integration of the app into the classroom led to noticeable improvements in students' Sinhala writing abilities. Students who used the app showed greater awareness of spelling accuracy and were more conscious of their vocabulary choices. Teachers noted an improvement in essay quality, both in terms of content and language, as students were able to identify and correct their mistakes independently before submitting their work.

## **4.3 Discussion**

The implementation of the automated grading system for Grade 5 scholarship essays, particularly for the Sinhala language, has shown promising results in improving both the grading process for teachers and the learning experience for students. The system's ability to identify complex words and correct spelling errors has proven to be a valuable tool for enhancing essay quality and providing more detailed feedback.

### **Complex Word Identification**

The feature that identifies complex words and provides their meanings allowed teachers to better assess the students' vocabulary usage. The system flagged words that were advanced for Grade 5 students, which encouraged both teachers and students to focus on vocabulary development. However, one of the challenges observed was that some students used these words without fully understanding their context or meaning. While this highlighted their exposure to more sophisticated language, it also indicated the need for further instruction on proper word usage. Teachers found this feature useful for guiding students in refining their vocabulary choices, thereby helping them become more thoughtful and deliberate in their writing.

### **Spelling Checking and Correction**

The automatic spell-checking and correction feature was particularly effective in reducing the workload for teachers, as it quickly identified and corrected common spelling mistakes in the Sinhala language. This feature enhanced the overall consistency and accuracy of essay grading by minimizing human error in manual spelling corrections. The immediate feedback provided to students allowed them to learn from their mistakes in real-time, which contributed to their gradual improvement in spelling accuracy. However, there were certain limitations in the algorithm's handling of contextual spelling errors and compound words unique to Sinhala. Future improvements could focus on refining the spell-checking feature to better address these nuances in the language.

### **Educational Impact**

The integration of this system into the educational process had a positive impact on students' writing skills. The system fostered a more interactive learning environment where students could engage with their own writing more critically. The real-time feedback on spelling errors and complex words encouraged self-correction and motivated students to improve their essay writing. Teachers noted that the app helped to enhance both the efficiency and quality of the grading process, allowing them to focus more on content-related feedback rather than language mechanics.

## **Limitations and Future Work**

Despite its successes, the system faced challenges in addressing the nuances of the Sinhala language, particularly in handling certain spelling structures and providing context-specific meanings for complex words. Some students struggled with the definitions provided for complex words, which were sometimes too abstract or disconnected from the context of their essays. To address these challenges, future developments could include integrating more context-aware algorithms and refining the word database to better accommodate the intricacies of the Sinhala language.

## 5 CONCLUSIONS

This research focused on developing an automated grading system for Grade 5 scholarship essays in Sinhala, integrating Natural Language Processing (NLP) techniques to enhance the grading process through spelling correction and complex word identification. The system successfully automated key aspects of essay evaluation, providing both teachers and students with valuable tools to improve essay quality.

The **identification of complex words** and their meanings proved to be a useful feature, helping teachers identify students' command of advanced vocabulary while encouraging students to engage with more sophisticated language. However, the need for context-specific learning aids to further support vocabulary comprehension was identified as an area for improvement.

The **spell-checking and correction feature** was particularly effective in reducing teachers' workload by automating the detection and correction of common spelling errors in Sinhala. It also provided students with real-time feedback, allowing them to learn and correct mistakes as they worked on their essays. However, the system faced challenges in addressing certain contextual and compound words unique to the Sinhala language, which can be refined in future version

## 6 REFERENCES

- [1] Y. Chen, "Journal of Empirical Research on Human Research Ethics," *SAGE Journals*, 2024. [Online]. Available: <https://journals.sagepub.com/home/JEC>.
- [2] N. De Silva, "Survey on Publicly Available Sinhala Natural Language Processing Tools and Research," *ResearchGate*, Jun. 2019. [Online]. Available: [https://www.researchgate.net/publication/333649787\\_Survey\\_on\\_Publicly\\_Available\\_Sinhala\\_Natural\\_Language\\_Processing\\_Tools\\_and\\_Research](https://www.researchgate.net/publication/333649787_Survey_on_Publicly_Available_Sinhala_Natural_Language_Processing_Tools_and_Research).
- [3] "Innovation in Language Learning and Education," *Taylor & Francis Online*, 2024. [Online]. Available: <https://www.tandfonline.com/toc/nile20/current>.
- [4] R. Abeysinghe, A. Weerasinghe, and J. D. S. W. Jayakody, "Named Entity Recognition in Sinhala: A Study of Classical and Deep Learning Approaches," *Journal of Natural Language Engineering*, vol. 25, no. 5, pp. 845-873, Oct. 2019. DOI: 10.1017/S1351324918000341.
- [5] A. D. P. Gunarathne, H. D. M. K. D. G. D. Tharindu, and N. S. Fernando, "Morphological Analysis of Sinhala Language Using Finite State Transducers," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 9, pp. 100-106, 2018. DOI: 10.14569/IJACSA.2018.090913.
- [6] N. de Silva, "Survey on Publicly Available Sinhala Natural Language Processing Tools and Research," *arXiv:1906.07312*, 2019. Available: <https://arxiv.org/abs/1906.07312>.



- [7] S. K. R. S. Chathuranga and P. L. Liyanage, "Sentiment Analysis for Sinhala Language using Deep Learning Techniques," *arXiv:1911.03640*, 2019. Available: <https://arxiv.org/abs/1911.03640>.
- [8] M. N. Medagoda and N. S. Fernando, "Development of a Sinhala Part-of-Speech Tagger Using Conditional Random Fields," *Journal of Language Technology*, vol. 10, no. 1, pp. 25-41, Jan. 2016. Available: <https://www.languagejournals.org/index.php/jlt/article/view/23>.
- [9] R. P. I. G. S. Priyankara and A. P. S. L. Liyanage, "A Study on the Performance of Sinhala Language Text Classification Using Supervised Learning Techniques," *International Journal of Machine Learning and Computing*, vol. 11, no. 2, pp. 124-129, Apr. 2021. DOI: 10.18178/ijmlc.2021.11.2.868.

## 7 APPENDICES

IT21006098\_FinalReport (1).docx

---

### ORIGINALITY REPORT

---

7%

SIMILARITY INDEX

5%

INTERNET SOURCES

3%

PUBLICATIONS

4%

STUDENT PAPERS

---