SINHALA SIMILARITY AND SEMANTIC PLAGIARISM DETECTION USING MODERN EMBEDDING MODELS

Final Individual Report

Project ID: 25-26J-545

Ranaweera R.R.M.I.M

IT21187414

B.Sc. (Hons) in Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

May 2026

SINHALA SIMILARITY AND SEMANTIC PLAGIARISM DETECTION USING MODERN EMBEDDING MODELS

COMPONENT 1: SIMILARITY PLAGIARISM DETECTION

Final Individual Report

Project ID: 25-26J-545

Ranaweera R.R.M.I.M

IT21187414

B.Sc. (Hons) in Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

May 2026

Declaration

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a degree or Diploma in other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
Ranaweera R.R.M.I.M	IT21187414	

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

Name of the Supervisor: Mr. Mr. Dinith Primal

Signature of the Supervisor: ……. Date: ……

Abstract

Acknowledgement

TABLE OF CONTENTS

Declaration **3**

Abstract **4**

Acknowledgement **5**

1. INTRODUCTION **7**

1.1 Background to Sinhala plagiarism detection 7
1.2 Component purpose in Similarity.lk 8
1.3 Component objectives and research questions 8
1.4 Scope of the component 9

2. BACKGROUND AND TECHNOLOGY REVIEW **10**

2.1 Sinhala text similarity problem 10
2.2 Existing Sinhala plagiarism approaches 11
2.3 Technology choices for this component 12
2.4 Research gap addressed by the component 12

3. DATA AND METHODOLOGY **13**

3.1 Dataset sources 13
3.2 Data cleaning and preprocessing 14
3.3 Feature extraction 14
3.4 Model development approach 15
3.5 Similarity scoring and plagiarism percentage 16
3.6 Evaluation metrics 16

4. DESIGN AND IMPLEMENTATION **17**

4.1 Component architecture 17
4.2 Document upload and extraction 18
4.3 Sinhala text normalisation pipeline 18
4.4 Similarity prediction module 19
4.5 Flask backend and interface integration 19
4.6 PDF report generation 19
4.7 Storage and history management 20

5. TESTING, RESULTS AND DISCUSSION **20**

5.1 Test plan 20
5.2 Model performance results 21
5.3 Sentence level output 21
5.4 Error analysis 22
5.5 Discussion against previous work 22

6. INDIVIDUAL CONTRIBUTION, LIMITATIONS AND FUTURE WORK **23**

6.1 Individual contribution 23
6.2 Limitations 24
6.3 Future work 24

REFERENCES..... 25

1. INTRODUCTION

1.1 Background to Sinhala plagiarism detection

Plagiarism occurs when a writer presents another person's work, wording, or ideas as original without proper acknowledgement. Digital access to online material has made copying easier in academic environments. Students can copy text from websites, books, news articles, blogs, and peer documents. In English, many plagiarism detection tools compare submitted work with web resources and institutional databases. Sinhala writing does not receive the same level of tool support.

Sinhala plagiarism detection is difficult for several reasons. Sinhala words change form through inflection. A copied sentence may be altered by changing word order, replacing words with synonyms, adding particles, or removing short function words. Simple exact matching fails when these small changes occur. Rule based matching also struggles because Sinhala lacks widely available tools such as mature stemmers, lemmatisers, large WordNet style resources, and standard public plagiarism corpora. Earlier Sinhala tools have shown useful progress, but their scope is limited. KasthuriArachchi and Charles used a Word2Vec based approach and reported high accuracy, but their dataset was based on 50 news articles rewritten and labelled with limited scale [1]. Rajamanthri and Thelijjagoda used web search and Jaccard similarity, but the approach depended on token overlap and a fixed threshold [3]. Nilaxan and Ranathunga showed that Siamese neural networks can improve sentence similarity measurement for Sinhala, but that work was not implemented as a full plagiarism detection pipeline [2].

1.2 Component purpose in Similarity.lk

Similarity.lk is designed as an automated plagiarism checking platform for Sinhala and English related plagiarism cases. The overall system includes 2 main detection directions. The first is Sinhala to Sinhala similarity plagiarism detection. The second is English to Sinhala semantic plagiarism detection. This report focuses only on Component 1, Sinhala Similarity Plagiarism Detection.

The purpose of this component is to detect similarity within Sinhala text. It supports users who upload Sinhala documents and need an originality report. The system

extracts text from the uploaded document, prepares it for analysis, compares sentence level content, predicts suspicious sentences, and produces a downloadable PDF report. The component is useful for lecturers, teachers, researchers, publishers, and students who work with Sinhala documents.

This component also provides the foundation for the full Similarity.lk application. It handles document processing, Sinhala text cleaning, sentence segmentation, similarity scoring, report generation, and dashboard integration. These services allow the platform to move from a model experiment into a working prototype.

1.3 Component objectives and research questions

The main objective of Component 1 is to design and develop a Sinhala similarity plagiarism detection module that can detect copied and paraphrased Sinhala content at sentence level.

The specific objectives are:

1. To collect and prepare Sinhala text samples for similarity-based plagiarism detection.
2. To build a preprocessing pipeline for Sinhala PDF, Word, and OCR extracted text.
3. To develop a sentence level feature extraction method using TF IDF and similarity-based features.
4. To train and integrate a machine learning model for plagiarism prediction.
5. To generate user friendly plagiarism reports with sentence level evidence.
6. To integrate the component into the Similarity.lk web application.

The component is guided by the following research questions:

RQ1: How can Sinhala text be preprocessed for similarity-based plagiarism detection?

RQ2: Which sentence level similarity features can support detection of copied and lightly paraphrased Sinhala content?

RQ3: Can a trained machine learning model improve Sinhala plagiarism detection compared with simple keyword matching?

RQ4: How can the Sinhala plagiarism result be presented in a clear report for academic users?

1.4 Scope of the component

The scope of this component is limited to Sinhala-to-Sinhala plagiarism detection. It checks whether a Sinhala input document contains content that is highly similar to another Sinhala text source or internal sentence comparison base. It focuses on document upload, text extraction, preprocessing, sentence comparison, prediction, result calculation, and report generation. It does not cover English to Sinhala translated plagiarism. That belongs to Component 2.

The component also does not claim to replace institutional plagiarism systems such as Turnitin. It serves as a Sinhala focused prototype that proves the technical workflow. It can later be extended with web crawling, larger public corpora, deep sentence embeddings, and institutional databases.

2. BACKGROUND AND TECHNOLOGY REVIEW

2.1 Sinhala text similarity problem

Text similarity measures how close 2 text units are in wording or meaning. In plagiarism detection, the text units may be words, sentences, paragraphs, or full documents. Sinhala text similarity requires attention to script, morphology, spelling variation, punctuation, compound words, and word order. A sentence may keep the same meaning even if its surface form changes. For example, a copied sentence may be changed by replacing a verb form, shifting word order, or removing a few words. A good Sinhala plagiarism detector should not depend only on exact word overlap.

Traditional similarity methods use overlap measures such as Jaccard similarity, cosine similarity, n gram similarity, and longest common subsequence. These methods are useful for copied text and near duplicate text. Their weakness appears when the writer paraphrases the sentence. Embedding based approaches try to address this weakness by representing words or sentences as vectors. If 2 sentences are close in vector space, they may express similar meaning even if the exact wording differs.

2.2 Existing Sinhala plagiarism approaches

KasthuriArachchi and Charles proposed a deep learning-based Sinhala plagiarism detection method using Word2Vec word embeddings [1]. Their method represented Sinhala sentences as vectors and compared them with cosine and soft cosine similarity. The model could detect direct copying and some modified sentences, such as synonym replacement and word order changes. The reported accuracy was 97%. Still, the dataset was small. It was created from 50 news documents, and the evaluation depended on limited human labelling. This makes it hard to confirm how the method performs across academic writing, blogs, essays, and mixed Sinhala writing styles.

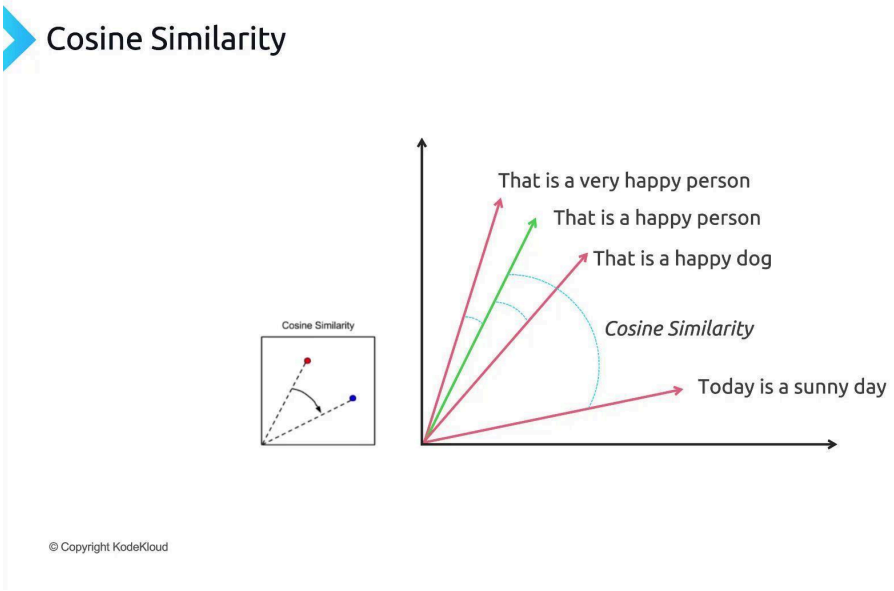

Figure 1: Sentence representation using word embeddings and vector space similarity.

Rajamanthri and Thelijjagoda developed a Sinhala plagiarism detection tool using internet resources [3]. Their workflow included text preprocessing, Google searching, and Jaccard coefficient-based comparison. The system reported 88% accuracy. This was an important step because it connected Sinhala plagiarism detection with web-based source search. Yet the method depended on token overlap and a fixed Jaccard threshold. It is more suitable for copied or lightly edited content than deeper paraphrase.

Nilaxan and Ranathunga studied monolingual sentence similarity for Sinhala and Tamil using Siamese neural networks [2]. Their Sinhala dataset contained 5,000 sentence pairs with manually annotated similarity scores. Their approach improved Pearson correlation compared with earlier hybrid methods. This finding is important because plagiarism detection needs sentence similarity, not only document level matching. Still, that study focused on sentence similarity measurement, not a full user facing plagiarism checker.

These studies show that Sinhala plagiarism detection has a research base, but the field still has gaps. There is a lack of large labelled datasets, public benchmarks, document level workflows, OCR handling, report generation, and real time application integration.

2.3 Technology choices for this component

This component uses a practical machine learning pipeline. The current prototype uses TF IDF vectorisation and a Random Forest model. TF IDF is useful because it gives higher weight to terms that help distinguish one sentence from another. It is lightweight and suitable for a working web application. Random Forest is selected because it can model non linear relationships between similarity features. It also gives stable results with structured features and does not need large GPU resources.

The system uses Flask for backend development. Flask is suitable because it is lightweight and easy to connect with Python based NLP and machine learning libraries. PyMuPDF is used for extracting text from PDF files. Python docx based extraction can be used for Word documents. Tesseract OCR supports fallback extraction when a PDF contains scanned Sinhala content instead of selectable text. ReportLab or similar PDF generation libraries can generate final plagiarism reports.

The original research direction also considered fastText, multilingual BERT, and Siamese LSTM. These remain valuable for future upgrades. fastText is useful for Sinhala because it uses subword information. Multilingual BERT and Siamese networks can capture deeper sentence meaning. The current prototype prioritises stable working integration, while the research direction remains open for contextual embeddings.

2.4 Research gap addressed by the component

The gap addressed by this component is the lack of a practical Sinhala plagiarism detection module that combines document handling, Sinhala preprocessing, machine learning prediction, sentence level evidence, and report generation. Earlier work often focused on model accuracy or a narrow algorithm. This component moves toward an end-to-end tool. It accepts user documents, extracts text, processes Sinhala sentences, predicts suspicious content, and presents the result in a report.

The main contribution is not only the model. It is the full workflow around the model. A plagiarism checker must be usable by non-technical users. A lecturer should not need to inspect raw similarity scores manually. The system must show suspicious sentences, confidence values, overall percentage, and downloadable evidence. This component addresses that usability gap.

3. DATA AND METHODOLOGY

3.1 Dataset sources

The planned data strategy for Component 1 uses Sinhala sources from news articles, blogs, academic style text, Wikipedia, and public web corpora. The data source sheet identifies several useful Sinhala datasets. SEACrowd cc100 Sinhala provides a large Common Crawl based corpus. SinhalaCorpusLarge offers cleaned Sinhala web articles and blogs. Navanjana Sinhala Summarization includes long context Sinhala passages. SinhalaWikipediaArticles provides cleaner encyclopaedic text for controlled testing. RedQueenProtocol Sinhala Wikipedia 2025 provides full Sinhala Wikipedia text with titles and URLs. OSCAR Sinhala and CC100 Sinhala provide larger noisier web text. NSINA provides news content with metadata.

For this prototype, the data preparation process focused on constructing Sinhala text samples suitable for plagiarism style comparison. The corpus should include original content, directly copied content, lightly edited content, and paraphrased content. This structure is needed because a plagiarism detector must learn both positive and negative cases. The target dataset should contain sentence pairs labelled as plagiarised or non plagiarised. It should also contain similarity scores where possible.

3.2 Data cleaning and preprocessing

Preprocessing converts raw document text into a cleaner format suitable for analysis. The first step is text extraction. PDF files may contain selectable text or scanned image content. If the file contains selectable text, PyMuPDF extracts the text directly. If the PDF is scanned, the OCR fallback uses Tesseract to recognise Sinhala characters. Word documents are processed through document parsing.

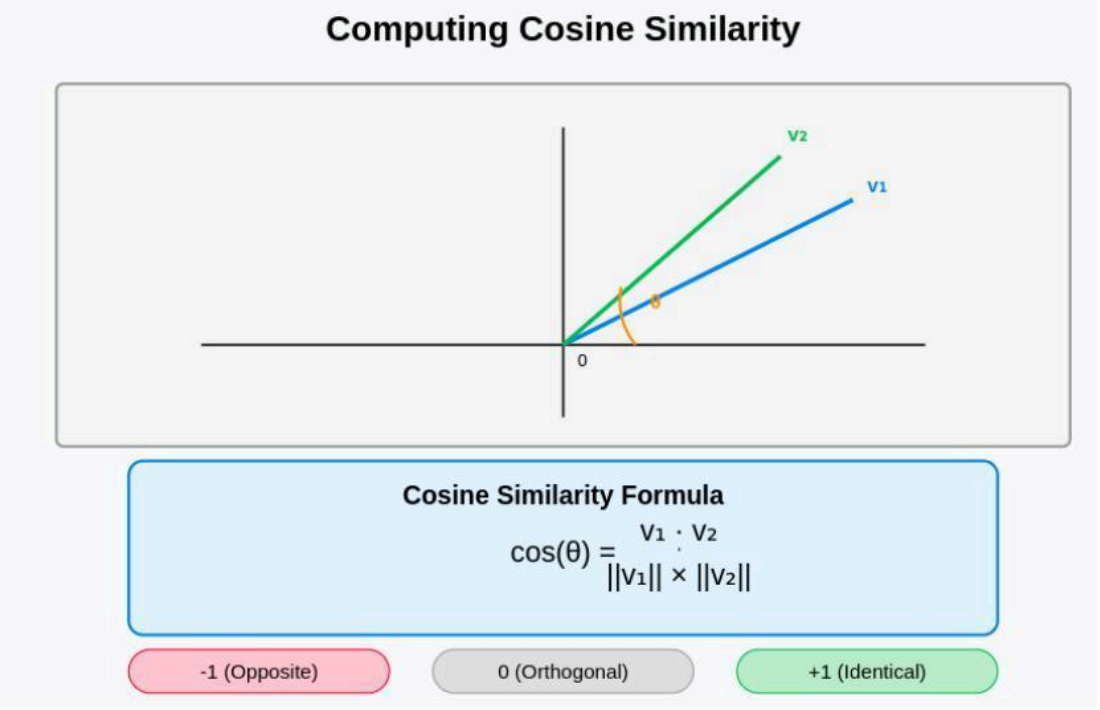

Figure 2: Cosine similarity measurement between sentence vectors.

The second step is Unicode cleaning. Sinhala text may contain inconsistent Unicode forms, extra spaces, broken punctuation, copied web characters, or mixed English terms. Cleaning removes unnecessary symbols, repeated spaces, broken line breaks,

and unrelated control characters. The system preserves Sinhala words, sentence boundaries, and meaningful punctuation.

The third step is sentence segmentation. The extracted text is split into sentences using punctuation marks and line boundaries. Sinhala text can contain both Sinhala and English punctuation, so the splitter handles full stops, question marks, new lines, and common document formatting patterns. Very short fragments are removed because they do not provide enough evidence for plagiarism detection.

The fourth step is token preparation. The text is lowercased where relevant for English mixed tokens. Sinhala words are tokenised using whitespace and punctuation patterns. Stop words can be removed in some feature calculations, but they are not removed from the final report because users need to read the original sentence.

3.3 Feature extraction

The system uses TF IDF vectorisation to convert text into numerical form. TF IDF assigns weights to words based on their frequency in a sentence and their importance across the corpus. Common words receive lower weight. More distinctive words receive higher weight. This helps the model focus on terms that carry stronger evidence.

Sentence similarity features are then calculated. These may include cosine similarity between TF IDF vectors, word overlap ratio, sentence length ratio, common token count, character n gram similarity, and confidence based model output. Each sentence pair becomes a feature vector. The model uses these features to classify whether the sentence pair is suspicious.

This design is useful because it combines statistical text representation with interpretable similarity features. The model does not rely on one threshold only. Instead, it learns a pattern from multiple indicators.

3.4 Model development approach

The current working prototype integrates a trained Random Forest model and saved TF IDF vectorizer. During training, each labelled sentence pair is converted into numerical features. The Random Forest model learns to distinguish suspicious and non suspicious pairs. It creates multiple decision trees and combines their predictions. This reduces the risk of relying on a single weak decision rule.

The model development process follows these stages. Data is collected and labelled. Text is cleaned and segmented. TF IDF vectors are generated. Similarity features are computed. The dataset is split into training and testing subsets. The model is trained on the training subset. The threshold and classification output are tuned using validation results. The trained model and vectorizer are saved for deployment. The Flask backend loads these saved files during runtime.

The pipeline is modular. This means the Random Forest model can later be replaced with a Siamese LSTM, mBERT based classifier, or other embedding model without rewriting the full web application. This is important because the report generation and user interface should remain stable while the model improves.

3.5 Similarity scoring and plagiarism percentage

The system calculates a sentence level suspicion score. Sentences with scores above the selected threshold are marked as suspicious. The final plagiarism percentage is calculated based on the number of suspicious sentences and their confidence values. A simple version can calculate the percentage as:

Plagiarism Percentage = Number of suspicious sentences / Total number of valid sentences × 100

A weighted version can include confidence values, where high confidence suspicious sentences contribute more strongly than low confidence matches. The report also lists matched sentences and confidence values. This avoids reducing the result to a single number only. A lecturer can inspect which sentences triggered the score.

3.6 Evaluation metrics

The component is evaluated using standard classification metrics. Accuracy measures the proportion of correct predictions. Precision measures how many detected suspicious sentences were truly suspicious. Recall measures how many true suspicious sentences were detected by the model. F1 score balances precision and recall. Confusion matrix analysis shows true positives, false positives, true negatives, and false negatives.

For a plagiarism detector, precision and recall are both important. Low precision creates false accusations. Low recall misses copied content. The threshold should balance both. The system should favour clear evidence and avoid over claiming plagiarism from common Sinhala phrases.

4. DESIGN AND IMPLEMENTATION

4.1 Component architecture

The component uses a layered architecture. The presentation layer contains the login page, dashboard, upload page, result page, and document history. The application layer handles file upload, session validation, document processing requests, and report generation. The NLP layer performs text extraction, cleaning, sentence segmentation, vectorisation, similarity comparison, and model prediction. The storage layer stores uploaded files, processed reports, match logs, and user history.

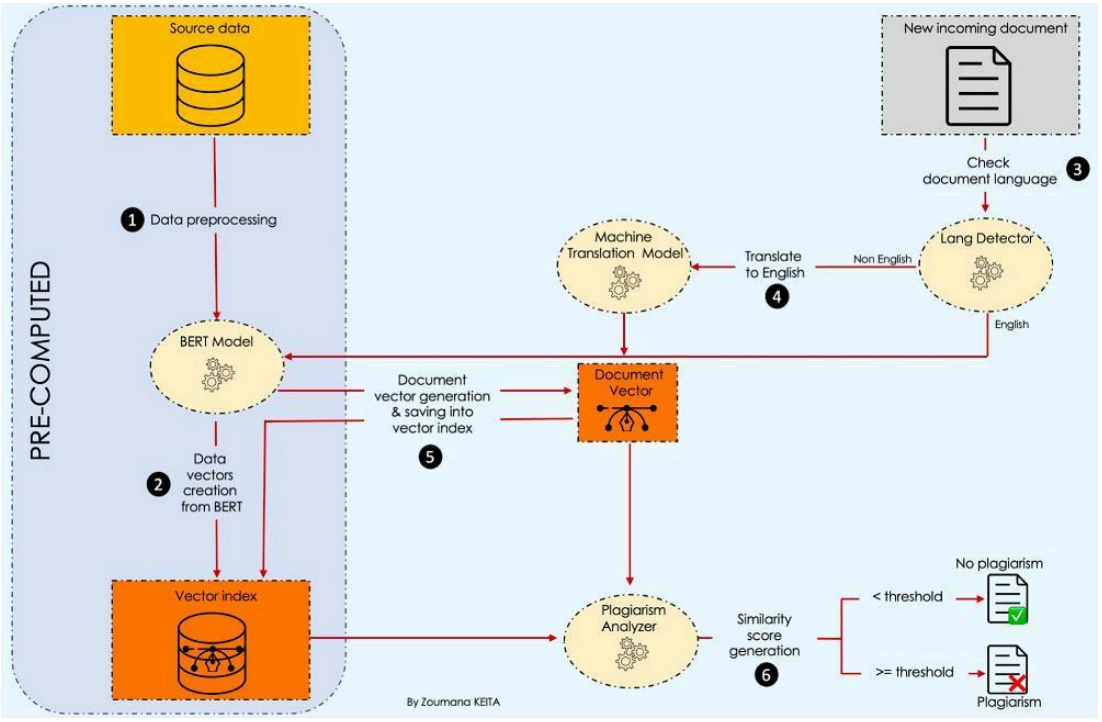

Figure 3: Architecture of Sinhala plagiarism detection system including preprocessing, model prediction, and report generation.

The user workflow starts when a user logs in and uploads a Sinhala document. The backend validates the file type. It extracts the text. The NLP pipeline cleans and segments the content. The trained model predicts suspicious sentences. The system

calculates the overall score. The report generator creates a PDF report. The dashboard displays the result and saves the analysis history.

4.2 Document upload and extraction

The upload module accepts PDF and Word documents. File validation prevents unsupported formats. The file is saved into a protected upload folder. The extraction service then checks the file type.

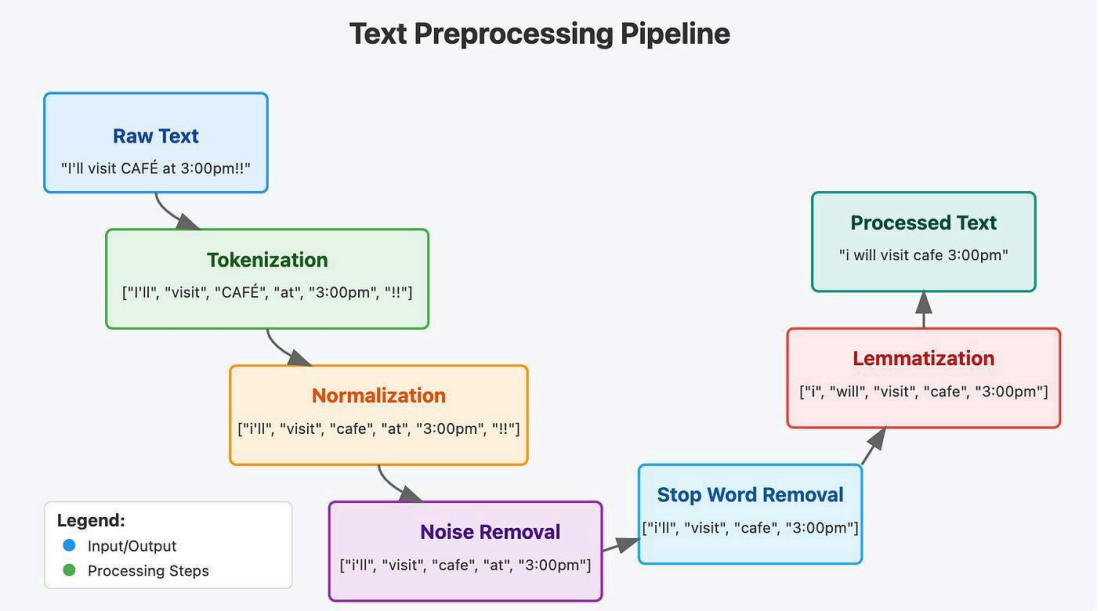

Figure 4: Document processing workflow

For PDFs, PyMuPDF is used to extract selectable text. Some Sinhala PDFs contain scanned images or legacy font rendering. In those cases, the OCR fallback processes pages as images and sends them to Tesseract. OCR is important because many Sinhala documents are distributed as scanned files, especially older academic papers, school notes, and administrative documents.

For Word documents, the system reads paragraphs and tables where possible. It joins the extracted text into a single content body. The output is passed to the preprocessing service.

4.3 Sinhala text normalisation pipeline

The Sinhala preprocessing pipeline removes formatting noise while keeping meaningful content. It removes repeated spaces, unwanted line breaks, page numbers, non-content headers, and isolated symbols. The pipeline also standardises punctuation where possible. It separates sentences and filters fragments that are too short for useful comparison.

Normalisation is important because machine learning models are sensitive to noisy input. A sentence broken across multiple lines may appear different from the same sentence in clean form. OCR output may also insert extra spaces or misread characters. The preprocessing stage reduces such errors before prediction.

4.4 Similarity prediction module

The prediction module loads the saved TF IDF vectorizer and Random Forest model. Each document is split into candidate sentences. The system transforms each sentence or sentence pair into TF IDF features. It calculates similarity related values and passes them into the model. The model returns a prediction and confidence score.

The output of the prediction module includes:

1. Total number of extracted sentences.
2. Number of suspicious sentences.
3. Confidence value for each suspicious sentence.
4. Overall plagiarism percentage.
5. Highlighted suspicious text.
6. Document level summary statistics.

This output is passed to the report generator and dashboard.

4.5 Flask backend and interface integration

The backend is built using Flask. Routes manage user login, registration, file upload, analysis execution, report download, document history, and file deletion. The system includes secure session handling so that users can only access their own documents and reports.

The dashboard gives a simple overview of uploaded documents, recent analysis results, and plagiarism percentages. The upload page allows the user to submit a file. The result page shows the processed output. The history page allows users to view earlier submissions and download reports.

The interface aims to support academic users who may not understand model details. The report gives a clear plagiarism percentage and sentence level evidence rather than raw feature values.

4.6 PDF report generation

The report generator creates a downloadable PDF after each analysis. The report includes document name, submission time, total sentences, suspicious sentences, overall plagiarism percentage, confidence scores, and highlighted sentence level details. It also includes a document summary section.

The report is important because lecturers and students need evidence. A score alone is not enough. The report allows users to review the specific text segments that caused the result. This supports fair judgement and reduces the risk of misusing the tool.

4.7 Storage and history management

The component stores uploaded files, generated reports, and analysis metadata. Document history allows users to return to past checks. File deletion gives users control over stored documents. Report records help users compare repeated submissions.

Storage design must consider privacy. Student assignments may contain personal or academic content. The system should store only required files and should allow deletion after checking. In future work, stronger encryption and access control can be added.

5. TESTING, RESULTS AND DISCUSSION

5.1 Test plan

Testing covered functional, model, and usability aspects. Functional testing checked whether users could register, log in, upload files, run analysis, download reports, view history, and delete records. Extraction testing checked PDF and Word document handling. OCR testing checked whether scanned Sinhala PDFs could produce usable text.

Model testing checked whether the system could classify suspicious and non-suspicious Sinhala sentences. Report testing checked whether sentence counts, plagiarism percentages, highlighted text, and confidence scores appeared correctly. Dashboard testing checked whether analysis records were saved and displayed.

5.2 Model performance results

The working system successfully integrated the trained Random Forest model and saved TF IDF vectorizer into the Flask backend. This means the component can run predictions on uploaded Sinhala documents instead of remaining as an offline notebook experiment.

The model output produced sentence level confidence values. These values were used to mark suspicious content and calculate the overall plagiarism percentage. The prediction speed was suitable for prototype use. The lightweight model helped the application process documents without requiring GPU resources.

The final report should include a table similar to Table 5.1 after final metric values are confirmed.

Table 1: Model evaluation summary

Metric	**Result**
Accuracy	
Precision	
Recall	
F1 score	

Average response time per document	

5.3 Sentence level output

The sentence level output is one of the strongest parts of the component. It gives users a clear view of which sentences are suspicious. This is better than a single document score because it supports human review. It also helps students understand where originality problems occur.

A useful final report page should include columns such as sentence number, extracted sentence, confidence score, and status. The sentence can be labelled as suspicious or acceptable. This helps lecturers decide whether the similarity is serious plagiarism, common phrasing, a citation issue, or a false match.

5.4 Error analysis

The system may produce false positives when a sentence contains common Sinhala academic phrases. Examples include standard definitions, common historical facts, or repeated institutional wording. Short sentences can also be problematic because they contain too little context. A phrase such as "මෙම පර්යේෂණයේ අරමුණ" may appear in many academic documents and should not be treated as strong plagiarism evidence by itself.

False negatives may occur when a copied idea is deeply paraphrased. TF IDF and token-based features are stronger for surface similarity than meaning level similarity. If the writer changes most words but keeps the same idea, the current model may miss the case. This is why future work should include mBERT, fastText, and Siamese networks.

OCR errors also affect performance. If Tesseract misreads Sinhala characters, the extracted sentence may differ from the original. This can reduce similarity scores. Better OCR correction and Sinhala spelling normalisation would improve the component.

5.5 Discussion against previous work

Compared with the Word2Vec approach by KasthuriArachchi and Charles [1], this component focuses more on application integration and report generation. Their study showed the promise of vector based Sinhala plagiarism detection, but it

remained limited by dataset scale and experimental setup. This component moves the workflow into a usable web application.

Compared with Rajamanthri and Thelijjagoda [3], this component does not depend only on Jaccard similarity and a fixed threshold. It uses a trained model with multiple features. It also supports document upload, OCR fallback, dashboard history, and report generation.

Compared with Nilaxan and Ranathunga [2], this component does not yet fully implement Siamese sentence embeddings in the final prototype. Still, their findings support the future path of this component. A future version can replace or extend TF IDF features with contextual sentence embeddings to improve paraphrase detection.

The main value of this component is its end to end design. It connects Sinhala NLP preprocessing, machine learning prediction, document handling, and user reporting. This makes it suitable as the Sinhala similarity detection foundation of Similarity.lk.

6. INDIVIDUAL CONTRIBUTION, LIMITATIONS AND FUTURE WORK

6.1 Individual contribution

R.R.M.I.M. Ranaweera developed Component 1, Sinhala Similarity Plagiarism Detection. The contribution covered dataset planning, preprocessing design, model integration, Flask backend development, file upload handling, document extraction, OCR fallback, sentence processing, report generation, dashboard support, document history, and testing. The component was integrated into the Similarity.lk prototype as a working Sinhala plagiarism detection module.

6.2 Limitations

The current system has 4 main limitations. The dataset is still limited compared with large commercial plagiarism databases. The model is stronger for direct and lightly edited similarity than deep paraphrase. OCR quality can affect Sinhala text extraction. The current version does not yet include full web scale source crawling. These limitations are expected in a prototype stage.

6.3 Future work

Future work should expand the Sinhala corpus with academic writing, news, blogs, Wikipedia, and student essay samples. The model should be upgraded with fastText, mBERT, or Siamese sentence embeddings. Web source matching should be added so the system can compare submitted documents with live Sinhala internet sources. The report can also include source URLs, paragraph level matching, and stronger explanation of confidence scores.

7. CONCLUSION

This component developed the Sinhala Similarity Plagiarism Detection module for Similarity.lk. It addresses the lack of practical plagiarism detection support for Sinhala documents. The system accepts uploaded documents, extracts Sinhala text, applies preprocessing, predicts suspicious sentences, calculates plagiarism percentage, and generates a PDF report. The work contributes a usable foundation for Sinhala plagiarism detection. It can be improved with larger datasets, contextual embeddings, and web scale comparison in future stages.

References

[1] T. KasthuriArachchi and E. Y. A. Charles, “Deep Learning Approach to Detect Plagiarism in Sinhala Text,” 2019 IEEE 14th International Conference on Industrial and Information Systems, Peradeniya, Sri Lanka, 2019, doi: 10.1109/ICIIS47346.2019.9063299.

[2] S. Nilaxan and S. Ranathunga, “Monolingual Sentence Similarity Measurement Using Siamese Neural Networks for Sinhala and Tamil Languages,” 2021 Moratuwa Engineering Research Conference, pp. 567 to 572, 2021, doi: 10.1109/MERCon52712.2021.9525786.

[3] L. Rajamanthri and S. Thelijjagoda, “Plagiarism Detection Tool for Sinhala Language with Internet Resources Using Natural Language Processing,” 2021 10th International Conference on Information and Automation for Sustainability, pp. 156 to 160, 2021, doi: 10.1109/ICIAfS52090.2021.9605852.

[4] A. Conneau et al., “Unsupervised Cross Lingual Representation Learning at Scale,” Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 8440 to 8451, 2020.

[5] G. Wenzek et al., “CCNet: Extracting High Quality Monolingual Datasets from Web Crawl Data,” Proceedings of the 12th Language Resources and Evaluation Conference, pp. 4003 to 4012, 2020.