

American University of Phnom Penh

School of Digital Technologies

Department of Information Technology

Sothyro Meas (UID:2021320)

**A Comparative Study of Plagiarism
Detection Methods on OCR-Extracted
Khmer Hard-Copy Texts with Elasticsearch
Implementation**

Final Year Project Report

Supervisor:

Tek Ming Ng, PhD

Co-supervisor:

Molika Meas, MSc

Phnom Penh, 25th April 2024

Author's declaration of originality

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used; these have been acknowledged.

Student Name: Sothyro MEAS (UID: 2021320)

Date of Submission: 25 April 2024

List of Abbreviations and Terms

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CLPD	Cross Language Plagiarism Detection
DB	Database
DOCX	Microsoft Word Open XML Document Format
FAISS	Facebook Artificial Intelligence Similarity Search
IDE	Integrated Development Environment
MoEYS	Ministry of Education, Youth and Sports of Cambodia
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
OCR	Optical Character Recognition
OS	Operating System
PDF	Portable Document Format
RESTful	Representational State Transfer (API)
SQL	Structured Query Language
TF-IDF	Term Frequency-Inverse Document Frequency
UI/UX	User Interface/User Experience
XML	Extensible Markup Language

Abstract

Plagiarism in the Khmer language remains a critical challenge in Cambodia, as the limited availability of digitized texts and continued reliance on hard-copy sources impede the development of effective digital detection tools. This gap has enabled widespread plagiarism in research papers, books, and educational documents published by students, researchers, and authors from various academic and research organizations, compromising academic integrity and highlighting the urgent need for a digital solution.

Although advanced plagiarism detection tools, such as Grammarly and Chegg, have significantly contributed to ensuring originality in many languages worldwide, they fail to detect plagiarism in under-resourced languages like Khmer.

Therefore, this project aims to develop a plagiarism detection tool specifically for the Khmer language and to identify the most efficient approach by comparing four different methods: Term Frequency–Inverse Document Frequency (TF-IDF) with cosine similarity; N-gram with Jaccard similarity stored in a PostgreSQL inverted index; Bidirectional Encoder Representations from Transformers (BERT); and Elasticsearch. Additionally, to analyze hard-copy source documents for plagiarism, they are first scanned and then processed using Optical Character Recognition (OCR) to extract the necessary text.

The system enables educational institutions, libraries, and publishers to upload large volumes of documents, books, and academic papers to detect plagiarized content, receive a plagiarism score, and identify matched sources within seconds. All uploaded documents are stored in a centralized storage system, allowing users to access their digital copies easily.

The tool is currently deployed on self-managed servers at the Cambodian Ministry of Education, the primary funder of this initiative. It is being used to assess official educational content and academic papers from 13 universities in Cambodia, helping evaluate the tool's effectiveness and identify areas for further improvement.

While the system effectively detects thousands of identical and similar plagiarized sentences, its accuracy is limited by the OCR text extraction. Common OCR inaccuracies can result in

distorted text, reducing the effectiveness of plagiarism detection. Future enhancements will focus on improving OCR performance and integrating internet-based plagiarism detection, ultimately expanding the system's capabilities and further strengthening research integrity in Cambodia.

Table of Contents

List of Abbreviations and Terms	3
Abstract	4
List of Figures	9
List of Tables	10
1. Introduction	11
1.1 Problem Statement	11
1.2 Motivation	12
1.3 Project Background	12
2. Literature Review	13
2.1 Plagiarism Detection for Under-Resourced Languages	13
2.2 Khmer Semantic Search Engine.....	13
3. Project Objective, Deliverable and Stakeholders	14
3.1 Main Objective and Key Deliverable.....	14
3.2 Stakeholder Involvement and Collaboration.....	15
4. Methodology and Implementation	16
4.1 Development Environment and Technology Stack.....	16
4.2 Data Preparation	17
4.2.1 Data Acquisition	18
4.2.2 Different Filetype Parsing	19
4.2.2.1 Portable Document File Parsing	19
4.2.2.2 Microsoft Word Document File Parsing.....	20
4.2.3 Data Preprocessing Pipeline	21

4.3 Plagiarism Detection Methods	24
4.3.1 TF-IDF and Cosine Similarity	25
4.3.1.1 Theoretical Overview.....	25
4.3.1.2 Process Flow with TF-IDF and Cosine Similarity.....	26
4.3.1.3 Technical Implementation	27
4.3.1.4 Results and Limitation	28
4.3.2 N-gram and Jaccard Similarity	29
4.3.2.1 N-gram Tokenization Technique	29
4.3.2.2 Measure Sentence Similarity with Jaccard	29
4.3.2.3 Process Flow with N-gram and Jaccard Similarity.....	30
4.3.2.4 Technical Implementation	31
4.3.2.5 Results and Limitation	33
4.3.3 Bidirectional Encoder Representations from Transformers (BERT)	33
4.3.3.1 Utilizing Multilingual BERT and FAISS	33
4.3.3.2 Process Flow with BERT and FAISS	34
4.3.3.3 Technical Implementation	34
4.3.3.4 Results and Limitation	35
4.3.4 Elasticsearch-Based Plagiarism Detection Method.....	36
4.3.4.1 Conceptual Overview.....	36
4.3.4.2 Process Flow with Elasticsearch.....	36
4.3.4.3 Technical Implementation	37
4.3.4.4 Results and Limitation	39
4.4 Methods Comparison and Discussion	39
4.4.1 Evaluation Methodology	39
4.4.2 Final Method Selection.....	40
 5. System Architecture and Workflow	 41
5.1 Overview of System Components.....	41
5.2 Document Indexing Workflow.....	43
5.3 Plagiarism Detection Workflow	44

6. Web Application Implementation	45
6.1 Plagiarism Detection Webpage	45
6.2 Indexing and Uploading Documents Webpage.....	49
6.3 File Storage System Webpage	50
7. Current Results and Limitations	52
8. User Feedback	52
9. Future Plan	52
10. Conclusion	53
Bibliography	54

List of Figures

Figure 1: Dataset Provider	18
Figure 2: Text Extraction with Pdfplumber	19
Figure 3: Text Extraction with Pdminer	19
Figure 4: Text Extraction with PyMuPDF (Fitz)	20
Figure 5: Text Extraction with OCR (Tesseract)	20
Figure 6: Data Preprocessing Pipeline	22
Figure 7: OCR Spelling Errors Correction with Khmerlang API	24
Figure 8: TF-IDF Pipeline Implementation	27
Figure 9: Khmer Language N-Gram	29
Figure 10: Inverted Index for N-Gram	30
Figure 11: N-Gram and Jaccard Similarity Pipeline Implementation	31
Figure 12: Trigram Output from Sample Khmer Texts	31
Figure 13: Inverted Index Schema for N-Gram in PostgreSQL	32
Figure 14: BERT and FAISS Pipeline Implementation	34
Figure 15: Elasticsearch Pipeline Implementation	37
Figure 16: System Architecture	41
Figure 17: Documents Indexing Workflow	43
Figure 18: Plagiarism Detection Workflow	44
Figure 19: Plagiarism Detection Webpage	45
Figure 20: Redis Queue Dashboard Webpage	46
Figure 21: Results Report in Excel	47
Figure 22: Plagiarism Result Certificate	47
Figure 23: Plagiarism Detection Results Sent Via Email	48
Figure 24: Indexing and Uploading Documents Webpage	49
Figure 25: MinIO Webpage	50
Figure 26: MinIO Custom Storage Webpage	51

List of Tables

Table 1: Methods Comparison.....40

1. Introduction

Every educational institution in Cambodia has enforced strict regulations against plagiarism and demonstrated a strong commitment to upholding academic integrity. However, there is currently no dedicated tool for detecting plagiarism in the Khmer language, which creates significant constraints in validating the originality of authors' work. Since most Cambodian academic content remains in hard-copy format, this presents further obstacles for institutions attempting to analyze plagiarized texts effectively.

To address these challenges, the Ministry of Education, Youth, and Sports Secretariat proposed the development of plagiarism detection tools to deter unethical practices and encourage original research in Cambodia. This project aims to provide educational institutions, publishers, and researchers with a tool to verify the originality of Khmer texts efficiently.

This section outlines the problem statement, that is supported by relevant references, and presents the motivation and background of the project.

1.1 Problem Statement

According to insights shared during a personal interview with the Secretary of State and Minister at the Ministry of Education, Youth and Sport (MoEYS) in Cambodia, many authors, PhD researchers, and professors at more than thirteen universities in Cambodia, have reused redundant content to produce additional publications, despite being informed about the consequences of plagiarism. These practices have been identified in many education documents containing overlapping or recycled content within the Ministry of Education's annual books and research publications.

Furthermore, a recent official report from the Cambodian Education Forum (2023) found that plagiarism is common not only among students, but also among many academic staff [1]. The limited availability of digital Khmer texts worsens the problem, as most of the academic papers are not digitized. This makes it difficult for advanced internet-based plagiarism detection tools such as Grammarly or Chegg to work effectively with the Khmer language hard-copy texts.

1.2 Motivation

This plagiarism detection tool is designed to provide a solution for Cambodian university's academic departments, faculty members, document publishers, and educational institutions to verify original works in the Khmer language by comparing them with previously stored documents in the database. With this tool, institutions will save time and resources by analyzing large documents for plagiarism in seconds through a user-friendly web-based interface and receive plagiarism detection results via email. Additionally, the tool addresses challenges related to non-digitized hard-copy texts by utilizing Optical Character Recognition (OCR) to convert them into searchable digital text.

1.3 Project Background

Initiated and funded by the Cambodian Ministry of Education, Youth, and Sport (MoEYS), this project aims to detect plagiarism in academic publications across educational institutions. To enforce research integrity, the Secretary of State at MoEYS has also authorized thirteen public universities in Cambodia to test this platform for reviewing research papers, documents, and published books.

This initiative provides a centralized solution for detecting duplicated content, ensuring that Cambodia's academic works maintain high levels of originality and credibility.

Looking ahead to the future, this project will allow not just all institutions in Cambodia, but students nationwide to use the platform and check for plagiarism easily as digital document datasets will grow every year.

2. Literature Review

This section reviews existing research on Cross-Language Plagiarism Detection (CLPD) using multilingual Bidirectional Encoder Representations from Transformers (BERT) and Khmer semantic search engines. It discusses current limitations and demonstrates how addressing these issues is essential for building an effective plagiarism detection tool for the Khmer language.

2.1 Plagiarism Detection for Under-Resourced Languages

A recent study by Karen Avetisyan et al. (2023) explores advancements in Cross-Language Plagiarism Detection (CLPD) using a pre-trained multilingual BERT model, achieving high accuracy for under-resourced languages. The researchers used Armenian as a test language and demonstrated that BERT's contextual embeddings could effectively capture semantic similarities across languages [2]. However, the study has not yet been extended to Southeast Asian languages like Khmer, which presents distinct challenges such as a continuous script and the lack of standardized word segmentation tools. Inspired by this work, this project will also utilize this BERT-based technique specifically for the Khmer language, aiming to overcome its structural complexities and evaluate the model's effectiveness on Khmer academic content.

2.2 Khmer Semantic Search Engine

One study by Nimol Thuon et al. (2024) on a Khmer semantic search engine utilizing Term Frequency-Inverse Document Frequency (TF-IDF) focuses on enhancing document retrieval based on keyword searches in the Khmer language. The system also allows users to upload documents to the database and search for similar content within the uploaded files. While the system effectively retrieves relevant content, its functionality is limited to sentence- or keyword-based searching and does not support full-document plagiarism analysis of PDF or Word files. This limitation highlights the need for a more comprehensive approach to Khmer text analysis that includes OCR and plagiarism detection across various document formats [3].

3. Project Objective, Deliverable and Stakeholders

This section outlines the expected outcomes of the project and key components that need to be developed to deliver an efficient plagiarism detection tool for our stakeholders. The system was developed in alignment with the needs of the Ministry of Education, Youth and Sport (MoEYS), the primary stakeholder. Their involvement ensured that the final platform is both relevant and usable within the Ministry and across targeted universities in Cambodia.

3.1 Main Objective and Key Deliverable

The main objective of this project is to develop a Khmer-language plagiarism detection tool that allows educational institutions, publishers, and researchers in Cambodia to verify content originality. This tool addresses the limitations of existing plagiarism detection platforms such as Grammarly or Chegg, which lack support for the Khmer language, and a feature that allows users to upload documents into the system and compare new submissions against those previously uploaded. This system also utilizes OCR to extract Khmer text from PDF files. The specific objectives of this project include:

1. Data gathering, cleaning, and pipeline development
 - Compile hundreds of sample documents in Microsoft Word and PDF formats provided by the Cambodian Ministry of Education, Youth, and Sports (MoEYS).
 - Preprocess each document by text cleaning and tokenization.
 - Create metadata (e.g., document ID, title, author, university) for each document, and store it in both the database and the file storage system.
2. Integrating Optical Character Recognition (OCR) for hard-copy texts
 - Convert all PDF pages to images.
 - Extract text from each image using OCR to enable plagiarism detection.
3. Developing a scalable plagiarism detection functionality
 - Utilize a fast and scalable similarity search algorithm to detect plagiarized content within a large academic dataset.
 - Set a threshold to identify both exact and partial similarities.

4. Providing a user-friendly web-based platform
 - Develop a web interface that allows users to upload documents for plagiarism detection.
 - Deliver plagiarism detection results (e.g., similarity score, matching reference sentences from the database, and associated university names) in downloadable Excel files via email.
 - Allow users to view, delete, or download all uploaded files in the database as needed.
 - Display real-time storage usage and alert users when storage is full.
 - Provide login access for admin and standard users.
5. Deployment on a self-managed server at the Cambodian Ministry of Education, Youth, and Sports
 - Host the file storage system and web application on a self-managed server at MoEYS.
 - Allow all authorized users to access the web application via the domain name provided by MoEYS (domain name: plagiarism-checker.duraseksa.gov.kh)

3.2 Stakeholder Involvement and Collaboration

The Ministry of Education, Youth and Sport (MoEYS) served as the primary stakeholder in this project, as the MoEYS team will be responsible for providing the document dataset, defining the system requirements to meet the expected outcome and the practical needs of Cambodian education institutions and support final system deployment in the self-managed MoEYS server.

4. Methodology and Implementation

This section outlines the methodology and technical implementation of the Khmer-language plagiarism detection system. The methodology consists of three main stages: preparing the input data, applying different plagiarism detection methods, and evaluating their effectiveness.

The section begins with an overview of the development environment and tools used. It then explains the data preparation process to extract clean and structured text, which is then used as input for four distinct detection methods: TF-IDF with cosine similarity, N-gram with Jaccard similarity stored in a PostgreSQL inverted index, BERT embeddings with FAISS, and full-text search using Elasticsearch.

Each method is examined in terms of its theoretical foundation, technical implementation, and performance limitations. Finally, the results are compared to assess which techniques are most effective for plagiarism detection in the context of the Khmer language.

4.1 Development Environment and Technology Stack

The development environment includes the software frameworks, tools, and services used to build, train, test, and deploy the Khmer-language plagiarism detection system. It spans several components, covering backend and frontend development, OCR processing, machine learning tools, storage system, and deployment. The technologies are grouped as follows:

- Machine learning Tools
 - Scikit-learn – Used to implement TF-IDF vectorization and cosine similarity
 - Transformers – Used to experiment with the multilingual BERT model for plagiarism detection
- Text extraction and search Tools
 - Tesseract OCR – Used to extract text from images
 - Elasticsearch – Enables efficient similarity search and large-scale text retrieval.
 - FAISS – Used for integrating with BERT to support fast vector-based document retrieval

- Storage and background task management
 - MinIO – Used to store uploaded documents, supporting secure backup and file access.
 - Redis – Used for managing background tasks and job queues, enabling asynchronous processing of plagiarism detection tasks.
- Programming languages and frameworks
 - Python / Flask – Used to build a backend development framework responsible for API endpoints, processing logic, and integration with search components
 - React.js – Used for developing a responsive and interactive web-based frontend
- Deployment
 - Ministry of Education, Youth and Sport (MoEYS) Server – Used for Production hosting environment
 - Docker – Used to containerize and manage all services, ensuring consistent deployment.
- Development tools
 - Visual Studio – Used for source code development
 - Postman – Used for RESTful API testing and validation during backend development
 - Jupyter Notebook – Enabled model testing without separate file export
 - Google Colab – Used to test BERT-based methods on GPU-enabled environments for improved performance during experimentation

4.2 Data Preparation

This section outlines the steps taken to prepare data for plagiarism detection, including data acquisition, file type parsing, and preprocessing. In the data acquisition phase, documents are provided by official sources from the Ministry of Education, Youth and Sport (MoEYS). Since the system accepts two file formats, each file is processed according to its type. For instance, scanned PDF files are processed using Optical Character Recognition (OCR) with Tesseract, while Microsoft Word documents are parsed directly to extract text using a Python library. Once

the raw text is extracted, it is cleaned and tokenized using a custom preprocessing pipeline. This same pipeline is applied to both user-uploaded documents stored in the database and those being checked for plagiarism, ensuring consistent formatting and reliable comparison.

4.2.1 Data Acquisition

The dataset used in this project was provided via Google Drive by the Ministry of Education, Youth, and Sport and includes a total of 551 published books collected from thirteen universities across Cambodia. These documents cover a wide range of topics, including agriculture, technology, literature, and more. Among these, 541 files are in PDF format while the remaining 10 are in Microsoft Word format. Since most of the PDF files are scanned, Optical Character Recognition (OCR) is required to extract readable text before all plagiarism detection methods can be performed.

Shared with me > 42-កម្រងសៀវភៅឧត្តមសិក្សា ▾

Type ▾ People ▾ Modified ▾ Source ▾












































Name	Owner	Last modified ▾ ↓	File size	
 តារាងកង្វះតម្រូវទំនិញផ្ទៃដី	 snhuem	Mar 29, 2024 snhuem	—	    
 សាកលវិទ្យាល័យភ្នំពេញអន្តរជាតិ	 snhuem	Apr 3, 2022 snhuem	—	
 សាកលវិទ្យាល័យស្វាយរៀង	 snhuem	Apr 3, 2022 snhuem	—	
 វិទ្យាស្ថានបច្ចេកវិទ្យាព័ត៌មាន	 snhuem	Apr 3, 2022 snhuem	—	
 វិទ្យាស្ថានបច្ចេកវិទ្យាព័ត៌មានឈើទាល	 snhuem	Apr 3, 2022 snhuem	—	
 សាកលវិទ្យាល័យជាតិបុរាណវិទ្យា	 snhuem	Apr 3, 2022 snhuem	—	
 សាកលវិទ្យាល័យភូមិន្ទវិទ្យាសាស្ត្រសង្គម	 snhuem	Apr 3, 2022 snhuem	—	
 សាកលវិទ្យាល័យភូមិន្ទភ្នំពេញ	 snhuem	Apr 3, 2022 snhuem	—	
 វិទ្យាស្ថានជាតិកសិកម្ម ប្រៃសណីយ៍	 snhuem	Apr 3, 2022 snhuem	—	
 សាកលវិទ្យាល័យភូមិន្ទកសិកម្ម	 snhuem	Apr 3, 2022 snhuem	—	
 សាកលវិទ្យាល័យជាតិប្រៃសណីយ៍	 snhuem	Apr 3, 2022 snhuem	—	
 សាកលវិទ្យាល័យ ហេង សំរិទ្ធ វប្បធម៌	 snhuem	Apr 3, 2022 snhuem	—	
 សាកលវិទ្យាល័យភូមិន្ទវិទ្យាសាស្ត្រ	 snhuem	Apr 3, 2022 snhuem	—	

Figure 1: Dataset Provider

4.2.2 Different Filetype Parsing

This section outlines how different file types are handled during the parsing phase to ensure consistent text extraction. Each format requires a specific processing method, for instance, PDF files, are processed using Optical Character Recognition (OCR) to extract text, while Microsoft Word documents are parsed directly using Python library.

4.2.2.1 Portable Document File Parsing

Several Python libraries were explored for extracting text directly from PDF files, including PyMuPDF, pdfminer, and pdfplumber. Although these tools are widely used for handling digital PDFs, they do not produce accurate results when applied to Khmer text. As a result, Tesseract OCR was selected as the primary method for processing scanned PDFs, as it provided more reliable text recognition for the Khmer language.

ផ្នែកទី១៖ មូលហេតុនៃការកំណត់ព្រំដែនសមុទ្រ
១-សញ្ញា ណាទូហៅ
កាលណាគេនិយាយអំពីការណ៍ កំពុងនៃសមុទ្រ គេគង់នឹងកត់ចំណាំគោលនិក្ខេបមានសារសំខាន់
មួយ គឺ “ផែនដីរាបបំផុតនៃសមុទ្រ(The land dominates the sea)”។ មានន័យសំគាល់ថា បែបអាចឆ្លង
បានសិទ្ធិរាបបំផុតនៃសមុទ្រដែលគេជាប់នៃផែនដីគោករបស់ខ្លួន ដោយអនុគោលការណ៍ ឆ្លងបំបែក
ជាតិពាក់ព័ន្ធ។ គោកមាត់ លំដាប់ (Malcolm N. Shaw) តាមរយៈគម្រោងនីតិវិធីអន្តរជាតិរបស់ខ្លួនបាន

Figure 2: Text Extraction with Pdfplumber

ផ្នែកទី១៖ មូលហេតុនៃការកំណត់ព្រំដែនសមុទ្រ
១-សញ្ញា ណាទូហៅ
កាលណាគេនិយាយអំពីការ ំណត់ព្រំដែនសមុទ្រ
គេគង់នឹងកត់ចំណាំគោលនិក្ខេបមានសារសំខាន់
មួយ គឺ “ផែនដីរាបបំផុតនៃសមុទ្រ(The land dominates the sea)”។ មានន័យសំគាល់ថា បែបអាចឆ្លង
បានសិទ្ធិរាបបំផុតនៃសមុទ្រដែលគេជាប់នៃផែនដីគោករបស់ខ្លួន

Figure 3: Text Extraction with Pdminer

ផ្នែកទី១៖ មូលហេតុនៃការកំណត់ព្រំដែនសមុទ្រ
 ១-សញ្ញាណទូហៅ

កាលណាគេនិយាយអំពីការកំណត់ព្រំដែនសមុទ្រ គេនឹងនឹកគួរគេលំអចិត្តនៃតំបន់សមុទ្រមួយ ដែលនឹងប្រាប់ប្រងនៃសមុទ្រ(The land dominates the sea)។ មានន័យសំគាល់ថា បែបអាចទទួលបាននិទ្ទិព្រំដែនតំបន់សមុទ្រដែលនៅជាប់នឹងតំបន់សមុទ្រនេះ ដោយអនុលោមតាមវិធានចាប់អនុវត្តជាតិពាក់ព័ន្ធនឹងតំបន់សមុទ្រ (Malcolm N. Shaw) តាមរយៈការស្រាវជ្រាវនិងការសិក្សាជាប្រពៃណី។

Figure 4: Text Extraction with PyMuPDF (Fitz)

To install Tesseract for the Khmer language on our Windows environment, we downloaded the trained data file from the Tesseract OCR GitHub repository [4].

Once installed, Tesseract was used to perform OCR on scanned PDF documents. To extract text using this OCR engine, each page of the PDF was first converted into an image using the *pdf2image* library [5]. After retrieving all pages, the *pytesseract.image_to_string()* function was then used to extract text from each image. However, processing one page at a time can slow down the text extraction. Therefore, we utilized Python’s *ProcessPoolExecutor* to extract text from multiple pages concurrently. As shown in the following figure, Tesseract OCR produced the most accurate text extraction compared to the other libraries.

ផ្នែកទី១៖ មូលហេតុនៃការកំណត់ព្រំដែនសមុទ្រ
 ១-សញ្ញាណទូហៅ

កាលណាគេនិយាយអំពីការកំណត់ព្រំដែនសមុទ្រ គេនឹងនឹកគួរគេលំអចិត្តនៃតំបន់សមុទ្រមួយ ដែលនឹងប្រាប់ប្រងនៃសមុទ្រ(២០៨ ៨០៣៨៥ ហា ៥៨១)។ មានន័យសំគាល់ថា បែបអាចទទួលបាននិទ្ទិព្រំដែនតំបន់សមុទ្រដែលនៅជាប់នឹងតំបន់សមុទ្រនេះ ដោយអនុលោមតាមវិធានចាប់អនុវត្តជាតិពាក់ព័ន្ធនឹងតំបន់សមុទ្រ (Malcolm N. Shaw) តាមរយៈការស្រាវជ្រាវនិងការសិក្សាជាប្រពៃណី។

Figure 5: Text Extraction with OCR (Tesseract)

4.2.2.2 Microsoft Word Document File Parsing

To extract text from Microsoft Word documents in .docx format, we used the Python library *python-docx*, which provides access to the document's structure, including paragraphs and tables. Using this library, the system iterates through all elements in the document body and retrieves the available text content for further processing. The following pseudocode illustrates the process of extracting text elements from a .docx file using *python-docx*:

Algorithm: Pseudocode for Extracting Text from a Word Document

```
Input: file_path – path to the .docx file
Output: text_content – extracted document text
1 Open the Word document from the given file path;
2 Initialize an empty list text_content;
3 for each block in the document body do
4     if block is a paragraph then
5         | Append the paragraph text to text_content;
6     end
7     else if block is a table then
8         | for each row in the table do
9             | Initialize an empty list row_data;
10            | for each cell in the row do
11                | Extract the text and append it to row_data;
12            end
13            | Join row_data with a separator and append to
              | text_content;
14        end
15    end
16 end
17 Return text_content;
```

Initially, an empty list called *document_content* is created to store the extracted text. The system then iterates through each element in the document body using the *iterchildren* function from the *python-docx* library. If the element is a paragraph (i.e., its XML tag ends with 'p'), the text is extracted and added directly to the *document_content* list. If the element is a table (tag ends with 'tbl'), the system iterates through each row and collects text from all cells. The cell contents are then joined using the pipe symbol (|) as a separator to form a single formatted row, which is appended to the *document_content* list. This process continues until there is no further content in the document body.

4.2.3 Data Preprocessing Pipeline

After the text is extracted, a series of preprocessing steps are applied to clean and prepare it for plagiarism detection. The raw text often contains inconsistent line breaks, extra spaces, unnecessary characters, and irregular newlines. Therefore, the following steps are applied to ensure text consistency.

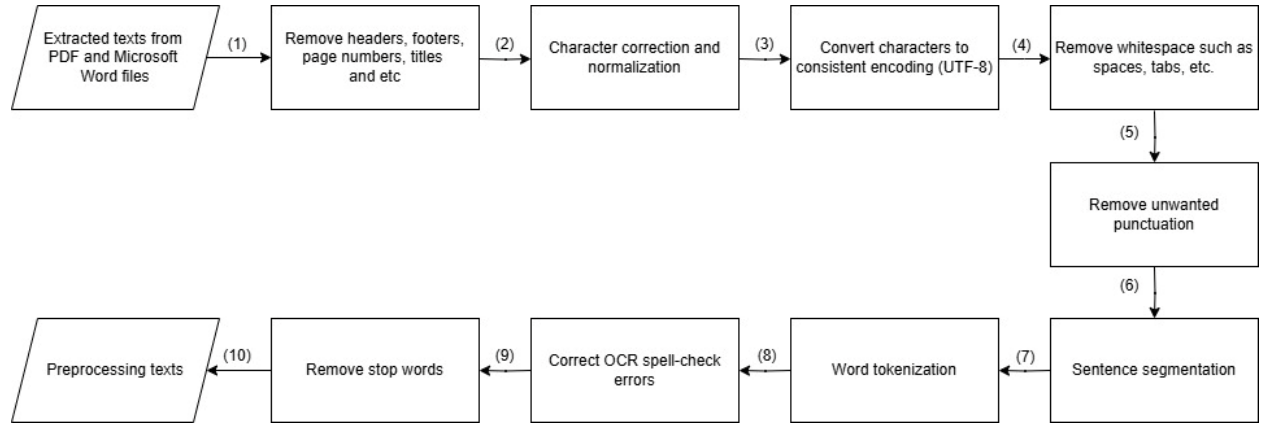


Figure 6: Data Preprocessing Pipeline

1. Remove headers, footers, page numbers, and titles: The raw extracted text often contains non-content elements such as page numbers, document headers and footers, and chapter titles. These are considered textual noise and must be removed to improve the accuracy of plagiarism detection. To filter them out, we observed the following patterns:

- Lines consisting only of two to three-digit numbers with symbols are page numbers.
- Repeated lines across multiple pages are headers or footers.
- Lines that follow a numbering format (e.g., "1.", "1.1") and have fewer than 20 characters are removed using regular expressions (regex), as they are titles.

2. Character correction and normalization:

- Reordered characters for consistency: In Khmer, certain characters can appear visually correct but are actually in different Unicode sequences due to incorrect ordering. For example, the word "ស្រី" (female) can be encoded as either ស + ្រ + ី or ស + ្រ + ី + ្រ + ី or ស + ្រ + ី + ្រ + ី + ្រ. Although these variants look similar on screen, they produce different underlying character sequences, which can negatively affect text comparison [6]. To address this issue, the system ensures that the character រ is always placed between the diacritic marks (្រ) to maintain consistent and correctly ordered Unicode representations [7].

- Incorrect character replacement: Some characters may be incorrectly used or misrecognized by OCR. For example, "ស៊ី" (scold) should be represented as ស + ៊ី + ័, but an OCR error most certainly produces ស + ៊ី + ័ + ័, substituting the correct consonant (ស) with a similar-looking but incorrect one (័). To solve this issue, we can use the replace function in Python to always convert words that consist of ៊ី + ័ to return ៊ី + ស. [6]
 - Number normalization: To maintain consistency across the dataset, all Khmer numerals (e.g., ១២៣) are converted into Arabic numerals (e.g., 123). [6]
3. Convert characters to consistent encoding (UTF-8): All content is encoded in UTF-8 to ensure consistent character representation.
 4. Remove spaces: Extra spaces, tabs, and line breaks are stripped to ensure clean texts.
 5. Remove unwanted punctuation: Punctuation that does not contribute to meaning such as ៀ, ៊, “, «, ^”, (, /), etc. are removed.
 6. Sentence segmentation: The cleaned text is then divided into sentence-level units using Khmer ending punctuation and symbols such as: ្រ, ័, ! and ?.
 7. Word tokenization: For Khmer, a language that does not use spaces between words, word tokenization is a critical preprocessing step. Methods like TF-IDF, N-gram, and Elasticsearch rely on clearly defined word boundaries to compute meaningful terms. To achieve this, we use the *khmercut* library to segment sentences into smaller word units for indexing and similarity detection [8].
 8. Correct OCR spell-check errors: OCR can introduce spelling errors, especially with the Khmer script. This step identifies and corrects OCR spelling mistakes to improve

accuracy in plagiarism detection. To implement this, we use a tool provided by Khmerlang spell check correction API to process the tokenized words [9].



Figure 7: OCR Spelling Errors Correction with Khmerlang API

As shown in the above figure, the API returns results that allow us to identify words with OCR errors. We then use the suggestions from the API response to replace those words and reduce errors caused by OCR.

9. Remove stop words: Common but semantically weak words such as “គាត់”, “ការ” or “នេះ” are removed to improve search relevance. To implement this, we have used the Khmer stop word from the Khmer Natural Language Processing community to filter all of the unnecessary words [10].

After preprocessing, the cleaned and standardized text is ready for analysis. This processed data is then used as input across the following plagiarism detection methods. Each method is described in the following sections along with its implementation, and evaluation results.

4.3 Plagiarism Detection Methods

This section outlines different methods that can be used to detect plagiarized content in the preprocessed data. After exploring each method, the following section will evaluate which one offers the best results, and that method will be integrated into our web application system. Each method uses a distinct strategy to measure text similarity, including vector-based approaches like TF-IDF with cosine similarity; set-based techniques like n-gram and Jaccard similarity (stored in

an inverted index in PostgreSQL); advanced retrieval engines like Elasticsearch; and deep learning models such as BERT combined with FAISS for similarity matching.

The following subsections explain each method in detail, including its theoretical overview, technical implementation aligned with system requirements, results, and limitations.

4.3.1 TF-IDF and Cosine Similarity

Term Frequency-Inverse Document Frequency (TF-IDF) and cosine similarity are widely used methods for measuring textual similarity. They convert each sentence into a vector based on word importance and calculate similarity using the cosine of the angle between the two vectors. This section explains how TF-IDF works in theory, its technical implementation, performance optimization, results and limitation.

4.3.1.1 Theoretical Overview

Term frequency (TF) measures how frequently a term appears in a document, normalized by the total number of terms in that document.

$$TF(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

However, Term Frequency alone is not sufficient to determine whether each document is similar because there could be common terms that may appear frequently in many documents but carry no meaning which lead to false positive. Therefore, we combine Inverse Document Frequency to also measure how rare a term is across all documents.

$$IDF(t) = \log \frac{\text{Total number of document}}{\text{Number of documents containing term } t}$$

If a term appears in many documents, the denominator becomes large, and the overall fraction becomes small. By taking the logarithm of a small number, the IDF score becomes low, making the frequent words appear to be less considered and prioritized words with rarity across the entire document. By multiplying TF and IDF, we ensure a term receives a high TF-IDF score only when it is both frequent and also rare in the overall dataset. [11]

$$TF - IDF (t, d) = TF(t, d) \times IDF (t)$$

To apply this method in our plagiarism detection tool, we first compute the TF-IDF scores for all terms (words) across both the original documents in the database and the newly uploaded document that the user wants to check for plagiarism. These scores are combined into vectors, with each sentence represented as a vector.

After converting each sentence into a TF-IDF vector, we compare the sentence vectors from the newly uploaded document with those from the existing documents in the database. To measure how similar two sentences are, we apply cosine similarity, which calculates the angle between two vectors in a high-dimensional space.

The smaller the angle, the more similar the sentences are. A cosine similarity scores close to 1 indicates high similarity, while a score near 0 suggests little to no similarity. The formula for cosine similarity between two vectors, A and B, is shown below

$$\text{cosine similarity score } (A, B) = \frac{A \cdot B}{\|A\| \|B\|}$$

4.3.1.2 Process Flow with TF-IDF and Cosine Similarity

TF-IDF vectors are computed based on term frequencies and document frequencies across the entire dataset, including the original documents in the database and the newly uploaded documents that the user wants to check for plagiarism. This process is computationally expensive and inefficient, especially when dealing with many documents.

To address this limitation, we compute the TF-IDF vectors once for each sentence in the original documents and utilize the *joblib* library to save the results. *Joblib* is a Python library that efficiently saves and loads large objects like TF-IDF models and vectors to avoid recomputing them repeatedly [12]. Later, when a new document is uploaded, we simply load the saved results from *joblib* and transform only the new sentences into TF-IDF vectors. These vectors are then compared to each vector in the precomputed matrix using cosine similarity to produce similarity scores. This approach avoids redundant preprocessing by saving the results once and loading them when needed. The diagram below illustrates the implementation of this method:

Data Preprocessing for TF-IDF

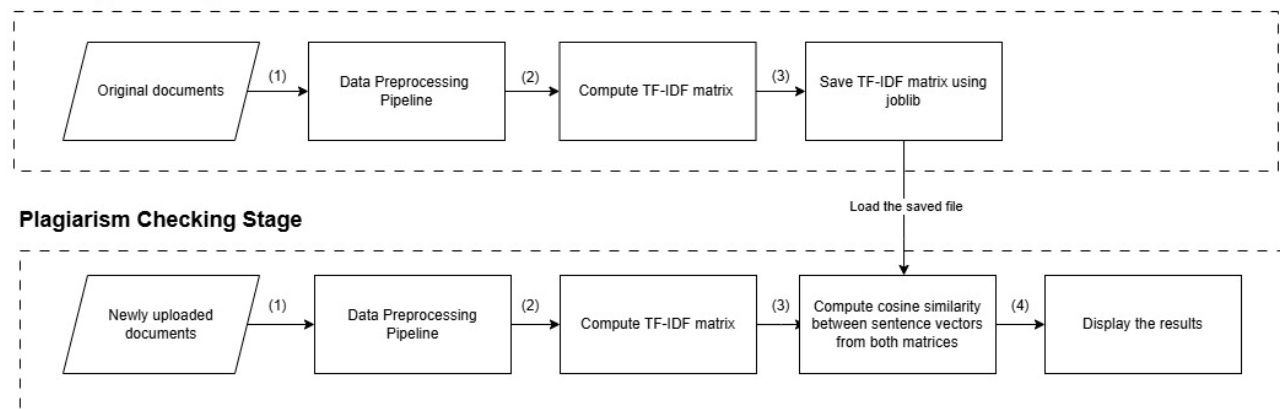


Figure 8: TF-IDF Pipeline Implementation

4.3.1.3 Technical Implementation

For this implementation, we utilize `TfidfVectorizer` function from Scikit-learn feature extraction library to calculate TF-IDF score and convert each sentence into vector. We divide the implementation into four main stages:

1. **Preprocessing and sentence extraction:** We load our entire dataset, which contains metadata such as document ID, university, title, author, original filename, content type and a list of sentences. Each sentence is converted to TF-IDF vector and mapped to its corresponding metadata, allowing us to trace any detected similarity back to the original source.

2. TF-IDF vectorization and joblib caching: After computing TF-IDF matrix with Scikit-learn, we use the joblib from the Python library to save the vectorizer training model, TF-IDF matrix of the original documents and its metadata. This allows us to avoid computing TF-IDF for the entire original documents in the database again every time a user upload a document to check for plagiarism.

```
from sklearn.feature_extraction.text import TfidfVectorizer
import joblib

vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(all_sentences)

joblib.dump(vectorizer, "vectorizer.pkl")
joblib.dump(tfidf_matrix, "tfidf_matrix.pkl")
joblib.dump(index_to_metadata, "metadata.pkl")
```

3. Transforming uploaded documents: When a user uploads new documents for plagiarism checking, the documents are passed through the same data preprocessing pipeline and transform function from Scikit-learn using the saved TfidfVectorizer from joblib.
4. Measure textual similarity: Cosine similarity is used to compare each uploaded sentence vector with all original sentences vector in order to measure the similarity.

```
def cosine_similarity(vec1, vec2):
    dot_product = sum(a * b for a, b in zip(vec1, vec2))
    magnitude1 = math.sqrt(sum(a ** 2 for a in vec1))
    magnitude2 = math.sqrt(sum(b ** 2 for b in vec2))
    if magnitude1 == 0 or magnitude2 == 0:
        return 0.0
    return dot_product / (magnitude1 * magnitude2)
```

4.3.1.4 Results and Limitation

The processing time of this approach increases with the number of sentences being compared, as the system performs sentence-by-sentence similarity calculations. A key limitation of this method lies in its scalability: frequently adding new documents to the original dataset would require retraining the TF-IDF vectorizer on the entire corpus, which becomes increasingly time-consuming as the dataset grows.

4.3.2 N-gram and Jaccard Similarity

N-gram and Jaccard similarity are another technique for comparing sets of text based on shared sequences. N-gram splits text into sequences of n words, while Jaccard similarity measures the overlap between two texts. This section outlines how n-gram and Jaccard similarity works in theory, along with their technical implementation to meet the requirements of our plagiarism detection system, their output results, and their limitations [13].

4.3.2.1 N-gram Tokenization Technique

N-gram is a sequence of terms extracted from a given text. In our plagiarism detection tool, we use n-gram to break sentence into smaller segments and identify overlapping segments between sentences.

N-gram on Khmer Language

N = 1	N = 2	N = 3
គាត់	គាត់ទៅ	គាត់ទៅសាលា
ទៅ	ទៅសាលា	
សាលា		

Figure 9: Khmer Language N-Gram

4.3.2.2 Measure Sentence Similarity with Jaccard

Jaccard similarity is a method for comparing the similarity between two sets. In this context, let set A represent the n-grams of the original documents, and set B represent the n-gram of the uploaded document being checked for plagiarism. The Jaccard similarity between those two sets is defined as:

$$Jaccard\ Similarity(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

The Jaccard similarity ranges from 0 to 1, where a higher similarity score indicates more overlapping n-grams between the two sets, while a lower score indicates less overlap.

4.3.2.3 Process Flow with N-gram and Jaccard Similarity

Calculating Jaccard similarity based on shared n-grams between all sentence pairs can be memory and computationally expensive, especially when we have a large number of documents. To address this, we use an inverted index method. Each n-gram from every sentence in the original documents is stored in the database, and mapped to the sentence IDs where it appears.

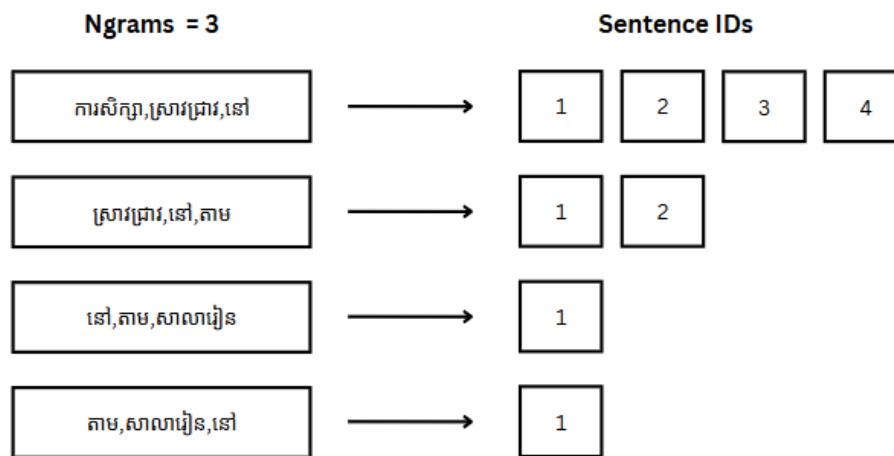


Figure 10: Inverted Index for N-Gram

During the plagiarism detection process, the newly uploaded document is split into sentences, and each sentence is broken into n-grams. These n-grams are then used to search for matches in the n-gram table in the database, allowing the system to quickly retrieve the sentence ID and its full sentence from the original document if they share at least one n-gram. Jaccard similarity is computed only between these candidate sentences, rather than performing a full sentence-by-sentence comparison. The diagram below illustrates the implementation of this method:

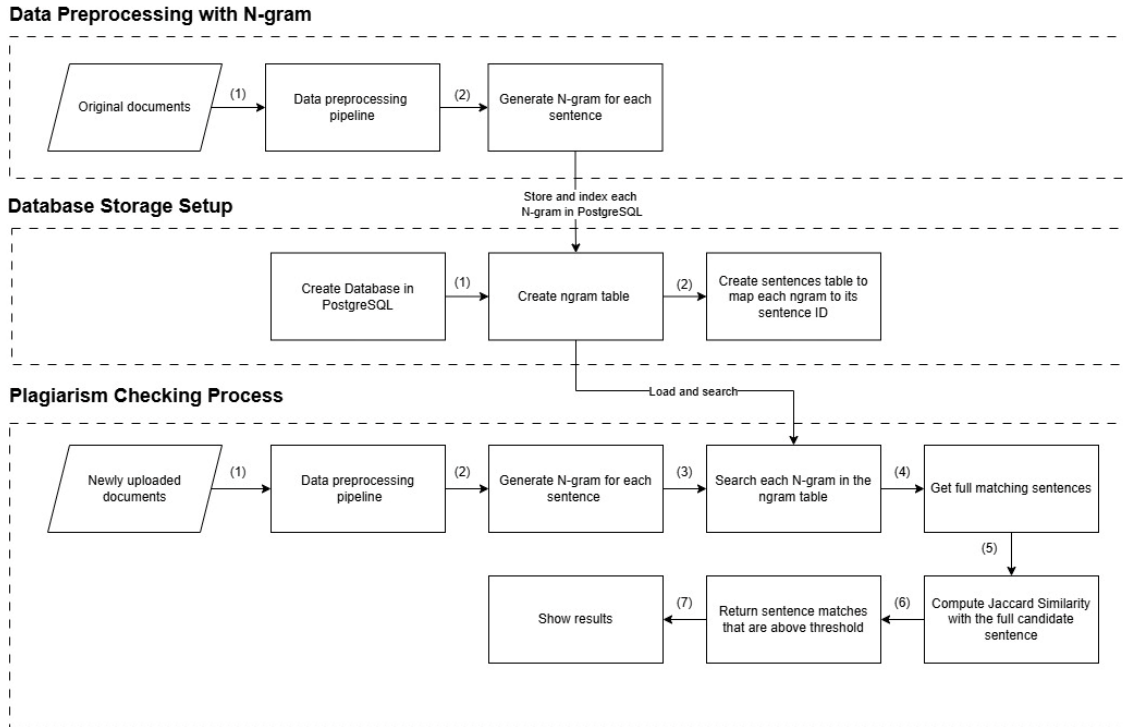


Figure 11: N-Gram and Jaccard Similarity Pipeline Implementation

4.3.2.4 Technical Implementation

First, we iterate through each sentence and its word tokens to create n-gram sequences. These unique n-grams are then indexed in a PostgreSQL database to support fast lookups. When a user uploads a document for plagiarism checking, we extract its n-grams and compare them against the indexed n-grams to retrieve candidate matches and then apply Jaccard to compute the similarity score. The overall implementation is divided into three main stages:

1. Each sentence is processed by splitting it into a list of words, then iterating through the list to group every n consecutive word into n-grams. In the following example, we use $n = 3$.

[('លោក', 'ប្រធាន', 'អនុសាសន៍'), ('ប្រធាន', 'អនុសាសន៍', 'បាន'), ('អនុសាសន៍', 'បាន', 'បន្ត'), ('បាន', 'បន្ត', 'ទៀត'), ('បន្ត', 'ទៀត', 'ថា'), ('ទៀត', 'ថា', 'ជាតិប្បកាល'), ('ថា', 'ជាតិប្បកាល', 'សម្តេច'), ('ជាតិប្បកាល', 'សម្តេច', 'តែងតែ'), ('សម្តេច', 'តែងតែ', 'យកចិត្តទុកដាក់'), ('តែងតែ', 'យកចិត្តទុកដាក់', 'ខ្ពស់'), ('យកចិត្តទុកដាក់', 'ខ្ពស់', 'ចំពោះ'), ('ខ្ពស់', 'ចំពោះ', 'សុខុមាល័យ'), ('ចំពោះ', 'សុខុមាល័យ', 'របស់'), ('សុខុមាល័យ', 'របស់', 'បង្កើន'), ('របស់', 'បង្កើន', 'ប្រជាពលរដ្ឋ'), ('បង្កើន', 'ប្រជាពលរដ្ឋ', 'រងគ្រោះ'), ('ប្រជាពលរដ្ឋ', 'រងគ្រោះ', 'និង'), ('រងគ្រោះ', 'និង', 'ងាយ'), ('និង', 'ងាយ', 'រងគ្រោះ'), ('ងាយ', 'រងគ្រោះ', 'គ្រប់'), ('រងគ្រោះ', 'គ្រប់', 'រូប'), ('គ្រប់', 'រូប', 'ងាយ'), ('រូប', 'ងាយ', 'មិន'), ('ងាយ', 'មិន', 'ប្រកាន់'), ('មិន', 'ប្រកាន់', 'វណ្ណៈ'), ('ប្រកាន់', 'វណ្ណៈ', 'ពណ៌'), ('វណ្ណៈ', 'ពណ៌', 'សម្បុរ'), ('ពណ៌', 'សម្បុរ', 'ងឿ'), ('សម្បុរ', 'ងឿ', 'សាសនា'), ('ងឿ', 'សាសនា', 'រដ្ឋ'), ('សាសនា', 'រដ្ឋ', 'និទ្ទាការ'), ('រដ្ឋ', 'និទ្ទាការ', 'នយោបាយ'), ('និទ្ទាការ', 'នយោបាយ', 'ណាមួយ'), ('នយោបាយ', 'ណាមួយ', 'ឡើយ')]

Figure 12: Trigram Output from Sample Khmer Texts

2. We created three tables such as sentences, inverted index and ngrams in PostgreSQL. The sentences table stores each sentence along with its associated metadata, including the document ID, sentence ID (as the primary key), full sentence, author, title, university, and year of publication. The ngrams table stores each unique n-gram from all sentences, with ngram ID as the primary key. The inverted index table maps each n-gram to the sentence it appears in by storing both sentence ID and ngram ID as foreign keys. This structure enables better retrieval of candidate sentences for Jaccard similarity comparison.

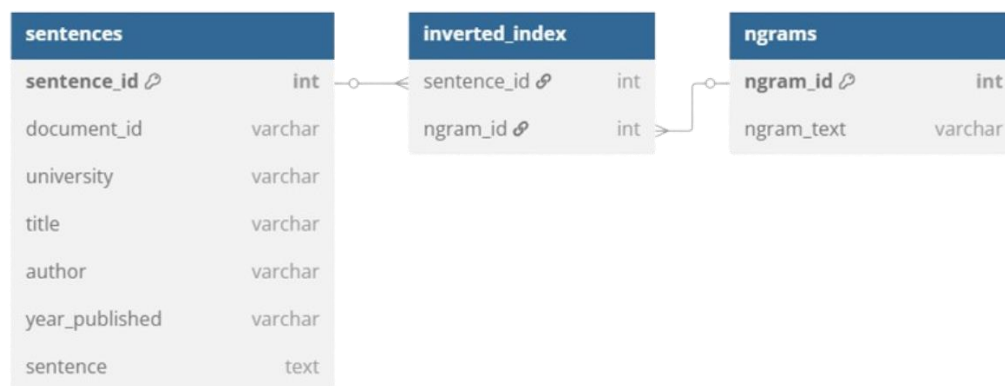


Figure 13: Inverted Index Schema for N-Gram in PostgreSQL

3. Once all candidate sentence IDs are obtained, we retrieve their corresponding full n-grams and compute for Jaccard similarity. The similarity score is calculated as the ratio of the intersection size to the union size of the two n-gram sets. This comparison is implemented using set operations in Python.

```
def jaccard_similarity(set1, set2):
    intersection = set1 & set2
    union = set1 | set2
    if not union:
        return 0.0
    return len(intersection) / len(union)
```


4.3.2.5 Results and Limitation

This approach consumes a large amount of memory, especially with large datasets consisting of hundreds of thousands of sentences. Since every sentence is broken down into multiple n-grams and stored in the ngrams table, data can grow rapidly, consuming significant storage and memory resources. This leads to slower response times and reduced scalability for real-time applications.

4.3.4 Bidirectional Encoder Representations from Transformers (BERT)

Bidirectional Encoder Representations from Transformers (BERT) is an open-source language model developed by Google that can capture the semantic meaning of sentences. This allow us to apply it in our tool to detect similar content. This section outlines the use of this method and the implementation of BERT with FAISS (Facebook AI Similarity Search) to perform plagiarism detection.

4.3.4.1 Utilizing Multilingual BERT and FAISS

For under-resourced languages like Khmer, we utilize a multilingual BERT model—paraphrase-multilingual-MiniLM-L12-v2—which could capture the semantic meaning of Khmer text and supports cross-lingual similarity [2]. To enable efficient similarity search, we employ FAISS (Facebook AI Similarity Search), which allows for fast approximate nearest neighbor retrieval in large vector spaces, significantly reducing computation time compared to brute-force pairwise comparisons. In our implementation, we use the IndexFlatL2 index, which performs search using the L2 (Euclidean) distance [14]. The L2 distance between two vectors x and y is defined as:

$$L2(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

4.3.4.2 Process Flow with BERT and FAISS

To implement our plagiarism detection tool with BERT and FAISS, we divide the process into two main stages: the original document preprocessing stage and the plagiarism checking stage.

1. In the preprocessing stage, after each sentence from the original documents is cleaned and tokenized using our data preprocessing pipeline, it is encoded using a pre-trained BERT model and then indexed using FAISS for efficient similarity search. This index is saved into a model file to be reused during the plagiarism detection process.
2. In the plagiarism checking stage, when a user uploads a new document, the same data preprocessing, and BERT encoding process is applied to the new document. The saved FAISS model is then loaded, and the newly encoded vectors are searched against the index to retrieve similar sentences.

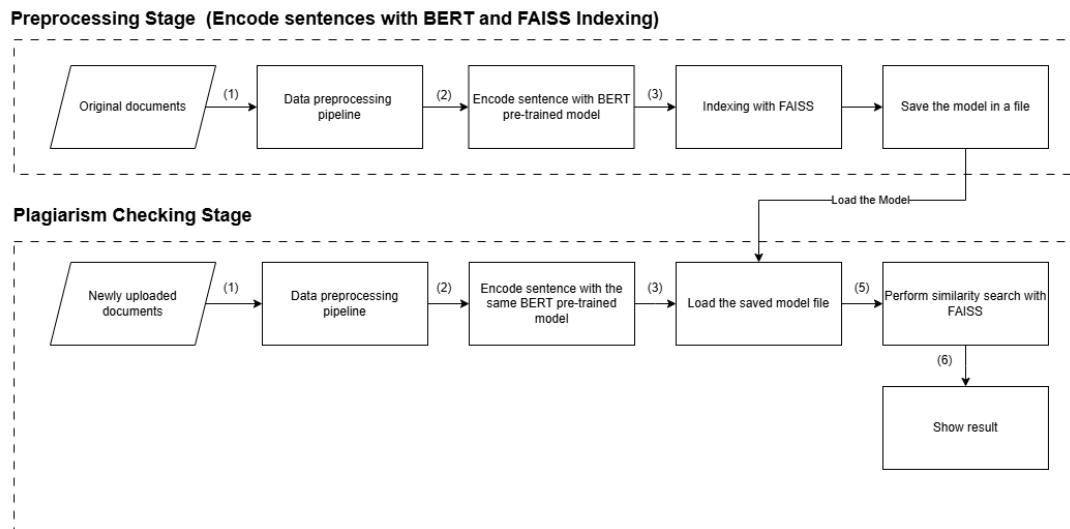


Figure 14: BERT and FAISS Pipeline Implementation

4.3.4.3 Technical Implementation

For this implementation, we use sentence transformers function from the SentenceTransformer library directly to encode the sentence to a dense vector in our Python code and FAISS library for indexing. This section outlines the encoding process using a pre-trained model from BERT and FAISS indexing.

1. Each sentence is converted into a high-dimensional dense vector using a pre-trained multilingual model called paraphrase-multilingual-MiniLM-L12-v2 from transformer, which allows the system to detect some exact matches as well as paraphrased content.

```
from sentence_transformers import SentenceTransformer

# Utilize paraphrase-multilingual-MiniLM-L12-v2 to support for Khmer language
model_MiniLM = SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')

# Combine all sentences and encode with the model
embeddings = model_MiniLM.encode(sentences, convert_to_numpy=True)
```

2. These vectors are added to a FAISS index to support fast and scalable search. This index allows fast comparison between new sentences and the existing dataset by vector distance. For this implementation, we can call the FAISS library directly and add the embeddings to the index.

```
import faiss
import numpy as np

vector_dimension = embeddings.shape[1]
index = faiss.IndexFlatL2(vector_dimension)
index.add(embeddings)
```

4.3.4.4 Results and Limitation

This approach does not always return all matching sentences from the original dataset, even though they are completely identical or highly similar. This limitation is largely due to the way semantic similarity is computed, which is based on approximation rather than exact sentence matching. Another key limitation lies in the method's scalability, as frequently adding new documents to the dataset requires re-encoding the entire corpus, which becomes increasingly time-consuming as the dataset grows. Additionally, this method relies on a GPU to process high-dimensional embeddings efficiently. However, such hardware is not currently available on the Ministry of Education's server.

4.3.5 Elasticsearch-Based Plagiarism Detection Method

This section outlines the fourth approach to plagiarism detection using Elasticsearch. Elasticsearch is widely used due to its search capabilities, making it a suitable candidate for identifying potential plagiarized content.

4.3.5.1 Conceptual Overview

Elasticsearch is a search engine that supports fast and efficient full-text search. It is designed to store, index, and search large volumes of text data effectively. Like the N-gram and Jaccard similarity method described earlier, Elasticsearch also uses an inverted index—a data structure that maps each term in the dataset to the list of documents where it appears [15]. For ranking results, Elasticsearch uses the BM25 (Best Matching 25) algorithm, which builds upon TF-IDF by also considering document length. This slightly favors shorter documents and helps prevent longer documents from gaining an unfair advantage simply by repeating terms more frequently [16].

4.3.5.2 Process Flow with Elasticsearch

To implement our plagiarism detection tool with Elasticsearch, we divide the process into two main stages: the original document preprocessing stage and the plagiarism checking stage.

1. In the preprocessing stage, each sentence from the original documents is cleaned and tokenized using our data preprocessing pipeline. These processed sentences are then stored in an Elasticsearch index, enabling efficient retrieval and full-text search capabilities during plagiarism detection.
2. In the plagiarism checking stage, when a user uploads a new document, it will go through the same cleaning and tokenization process using our data preprocessing pipeline. The system then queries the Elasticsearch index to retrieve sentences that are textually similar to the uploaded content. If any matching results are above similarity threshold, they are flagged and presented as potential plagiarism cases.

Preprocessing Stage (Elasticsearch Setup and Indexing)

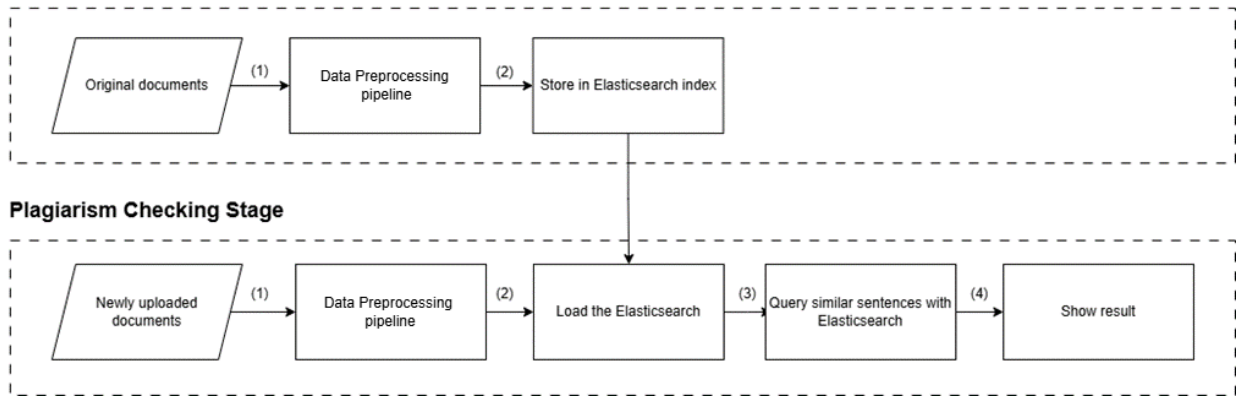


Figure 15: Elasticsearch Pipeline Implementation

4.3.5.3 Technical Implementation

The Elasticsearch method involves configuring a full-text search engine that can index and query large volumes of sentence-level data. This section outlines how we setup the Elasticsearch, and use for indexing and querying.

1. Docker was used to deploy Elasticsearch, which was configured to run on port 9200 and made accessible via its RESTful API. The application utilized the official Python Elasticsearch client to communicate with the containerized instance for both indexing and query operations.

```
GET http://localhost:9200
Body
JSON
1  {
2    "name": "d7420bedf989",
3    "cluster_name": "docker-cluster",
4    "cluster_uuid": "b6tu19DqQeib8zVl3U3IHA",
5    "version": {
6      "number": "8.13.0",
7      "build_flavor": "default",
8      "build_type": "docker",
9      "build_hash": "09df99393193b2c53d92899662a8b8b3c55b45cd",
10     "build_date": "2024-03-22T03:35:46.757803203Z",
11     "build_snapshot": false,
12     "lucene_version": "9.10.0",
13     "minimum_wire_compatibility_version": "7.17.0",
14     "minimum_index_compatibility_version": "7.0.0"
15   },
16   "tagline": "You Know, for Search"
17 }
```

The screenshot shows a REST client interface with a GET request to `http://localhost:9200`. The response is a JSON object representing the Elasticsearch status, including details like name, cluster_name, cluster_uuid, version (number, build_flavor, build_type, build_hash, build_date, build_snapshot, lucene_version, minimum_wire_compatibility_version, minimum_index_compatibility_version), and tagline. The status is 200 OK.

- To enable efficient search, a custom mapping was first defined in Elasticsearch to specify the structure of the indexed documents. This mapping included fields such as title, authors, publication year, university, and sentence. Once the index was created with the defined mapping, documents adhering to this structure were inserted into the system.

```
# Define mapping with all required fields
mapping = {
    "mappings": {
        "properties": {
            "title": {"type": "text"},
            "authors": {"type": "text"},
            "publish_year": {"type": "integer"},
            "university": {"type": "keyword"},
            "sentence": {
                "type": "text",
                "fields": {
                    "keyword": {"type": "keyword"} # Enables exact search
                }
            }
        }
    }
}

# Create the index with the mapping
response = requests.put(
    f"{ELASTICSEARCH_URL}/{INDEX_NAME}",
    headers={"Content-Type": "application/json"},
    data=json.dumps(mapping)
)
```

- Elasticsearch analyzes the text from our queries and returns the most relevant matches based on scoring result.

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:9200/documents/_search`. The request body is a JSON object with a query matching the sentence field. The response is a 200 OK status with a JSON body containing search results.

```
POST http://127.0.0.1:9200/documents/_search

{
  "query": {
    "match": {
      "sentence": "ការងារ វិទ្យា មាន ពីរ ប្រភេទ ការងារ ប្រភេទ ទី ១ គឺជា ការងារ វិទ្យា ជា ការងារ ត្រូវ ចាប់អារម្មណ៍ ផ្តល់ គួរ ប្រាកដ ចំណូល"
    }
  }
}
```

Response (200 OK, 68 ms):

```
{
  "max_score": 83.59089,
  "hits": [
    {
      "_index": "cdde_books",
      "_id": "2vZe_pQ83HcTW9otSkop",
      "_score": 83.59089,
      "_source": {
        "university": "សាកលវិទ្យាល័យភូមិន្ទភ្នំពេញ",
        "book": "មូលដ្ឋានគ្រឹះសង្គមវិទ្យា",
        "sentence": "ការងារ វិទ្យា មាន ពីរ ប្រភេទ ការងារ ប្រភេទ ទី ១ គឺជា ការងារ វិទ្យា ជា ការងារ ត្រូវ ចាប់អារម្មណ៍ ផ្តល់ គួរ ប្រាកដ ចំណូល ឬ អត្ថប្រយោជន៍ នឹង សង្គម ការងារ"
      }
    }
  ]
}
```

4.3.5.4 Results and Limitation

Elasticsearch performed well both speed and accuracy. It can find all the identical sentences in the database in short period of time. Even when the input sentences had small changes, such as one or two extra or missing words, Elasticsearch still returned similar results. This shows that it can handle both exact matches and slightly different sentences, making it a strong choice for our plagiarism detection tool requirement.

4.4 Methods Comparison and Discussion

This section presents a comparative analysis of the plagiarism detection methods implemented in the system. The goal is to identify the most effective approach based on detection accuracy, execution time, memory usage, and scalability. The first subsection outlines the evaluation methodology and testing criteria, while the second provides a justification for the final method selection based on observed results.

4.4.1 Evaluation Methodology

To evaluate the effectiveness of each plagiarism detection method, we tested whether the system could successfully return all expected matches for a set of known plagiarized sentences. We randomly selected 600 sentences from the 551 indexed documents: 300 were exact matches, while the remaining 300 were slightly modified by inserting, deleting, or substituting, while still preserving the original content and sentence structure.

Each method—TF-IDF with cosine similarity, N-gram with Jaccard similarity, Elasticsearch, and BERT embeddings with FAISS was evaluated based on its ability to accurately retrieve all 600 sentences. The comparison focuses on accuracy, execution time, memory usage, and scalability. The results are summarized in the table below.

Methods	Score		Time Execution (second)	Peak Memory Usage (MB)	Scalability
	Exact Sentence	Similar Sentence			
TF-IDF & Cosine	1.00	0.87	103.90	291.19	Inefficient for frequent data insertions, deletions, or updates due to the need to recompute the entire TF-IDF matrix.
Ngrams and Jaccard (PostgreSQL inverted index)	1.00	0.77	> 1000	15.88	Not scalable for large datasets, as n-gram growth significantly increases memory and slows down query performance.
paraphrase-multilingual-miniLM-L12-v2 BERT*	0.71	0.23	83.834	0.21	Frequent data updates require intensive memory and computation to rebuild.
Elasticsearch	1.00	0.86	2.1424	9.69	Elasticsearch allows parallel search and indexing for large datasets.

Table 1: Methods Comparison

* GPU is required for this operation

4.4.2 Final Method Selection

Based on the evaluation results, Elasticsearch offers the best overall balance between accuracy, performance, memory efficiency, and scalability. It successfully returned all exact matches and performed nearly as well as TF-IDF on similar sentences, while executing significantly faster and using less memory. Unlike methods such as TF-IDF and BERT, which require costly recomputation or GPU support, Elasticsearch supports incremental updates and parallel indexing, making it well-suited for large-scale and continuously growing datasets. Given these advantages, Elasticsearch was selected as the final method for our plagiarism detection system.

5. System Architecture and Workflow

This section outlines the overall architecture of the proposed plagiarism detection system, with a focus on how different technology components interact to support both uploading document to the database and plagiarism detection. It describes the technical components involved in the backend infrastructure and explains the key workflows from system input to output. The following subsections provide an overview of the system's core components and describe two key backend workflows: one for indexing documents and another for detecting plagiarism.

5.1 Overview of System Components

The proposed system is composed of multiple technology components, including React.js, Flask API, Redis, MinIO file storage, Google Tesseract OCR, and Python scripts for the data preprocessing pipeline. These components work together to prepare documents for indexing and querying in Elasticsearch. Figure 16 illustrates how these components interact throughout the system's workflow.

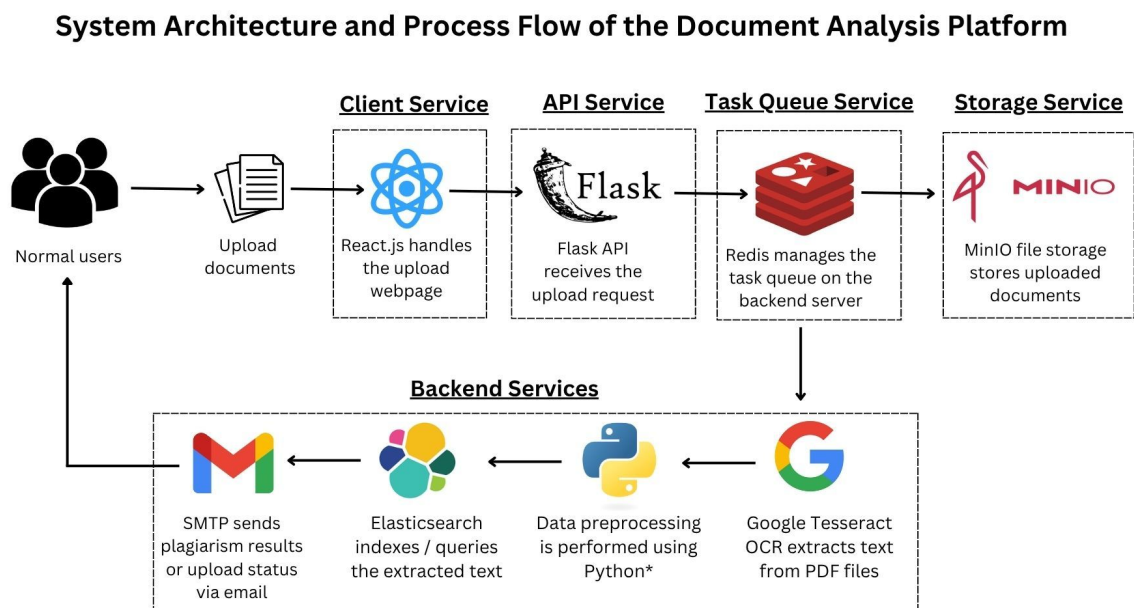


Figure 16: System Architecture

The following steps describe the interaction flow in detail, corresponding to the numbered sequence shown in the diagram:

1. User upload documents (Step 1–3): Normal users begin by uploading documents through a React.js web interface (Step 2). React.js will send the uploaded file to a Flask-based backend API (Step 3), which handles the upload request.
2. Task queuing and storage (Step 4): After receiving the request, the Flask API enqueues the processing task into a Redis-backed task queue [17] (Step 4). If the users want to upload the documents to the system, this API will forward the uploaded document to MinIO, an object storage system [18], before proceeding to text extraction (Step 5). If the users want to only check for plagiarism, it will then proceed to text extraction.
3. Text extraction (Step 6–7): If the document is a PDF, it will first convert the pdf to images in a folder for Google Tesseract OCR to extract raw text from each image (Step 6). The extracted text is then passed to a custom Python-based preprocessing pipeline (Step 7), which handles tokenization, cleanup, and segmentation.
4. Elasticsearch execution (Step 8): The preprocessed sentences are then either indexed into or queried from Elasticsearch (Step 8), depending on whether the operation is for plagiarism checking or uploading documents to the system.
5. Result Delivery (Step 9–10): Once the operation is complete, the results—including matched sentences and similarity scores—are compiled in an excel sheet and sent to the user via email using an SMTP client (Step 9). The user will receive a notification containing the result (Step 10).

This architecture supports asynchronous processing through Redis, efficient file management via MinIO, OCR text extraction, text preprocessing with Python, and scalable similarity search through Elasticsearch. The design enables the system to scale with increasing document and supports the future integration of additional detection methods.

5.2 Document Indexing Workflow

This section outlines the process of uploading more documents to MinIO file storage system and indexing their content in Elasticsearch. The workflow handles both PDF and Microsoft Word files by extracting their text, preprocessing the content, and storing the results in an Elasticsearch index. The diagram below illustrates the process starting from document datasets are inserted until users receive its status.

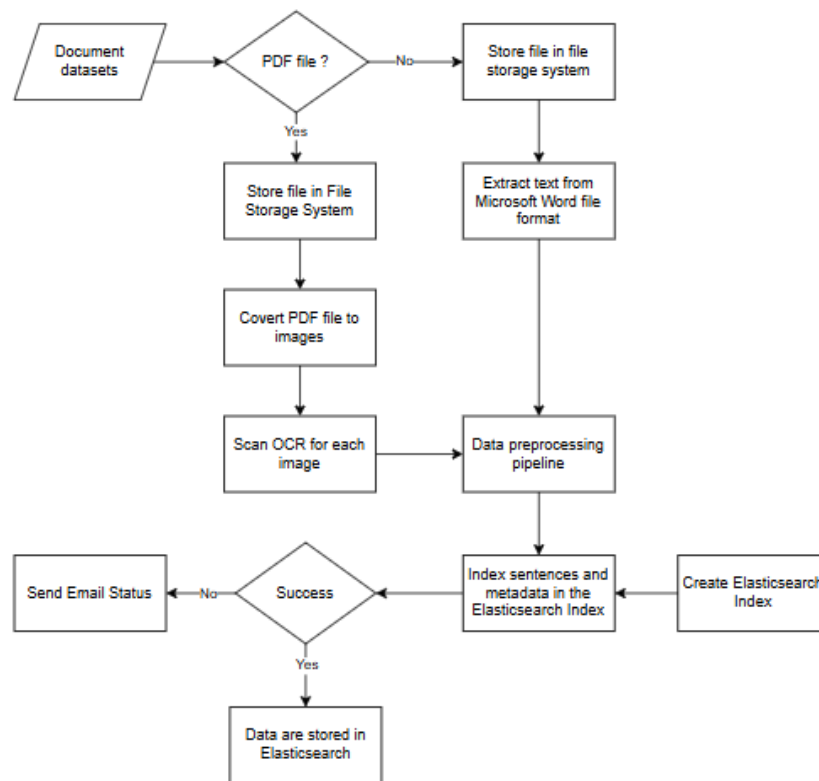


Figure 17: Documents Indexing Workflow

As shown in the diagram above, the workflow begins with the input of document datasets. Depending on the file type, the system either extracts text from Microsoft Word files directly or applies OCR to PDF files after converting them into images. The extracted text is then passed through a data preprocessing pipeline before being indexed into Elasticsearch. Once indexing is successful, the system stores the processed data and sends a status notification via email.

5.3 Plagiarism Detection Workflow

This section outlines the workflow used to detect plagiarism in user-submitted documents by comparing their content against previously indexed documents in Elasticsearch. The system supports both PDF and Microsoft Word file formats, ensuring flexibility and compatibility with common academic materials. After extracting and preprocessing the text, each sentence is matched against the existing Elasticsearch index. Sentences with a similarity score above a predefined threshold are considered potential matches and compiled into a report.

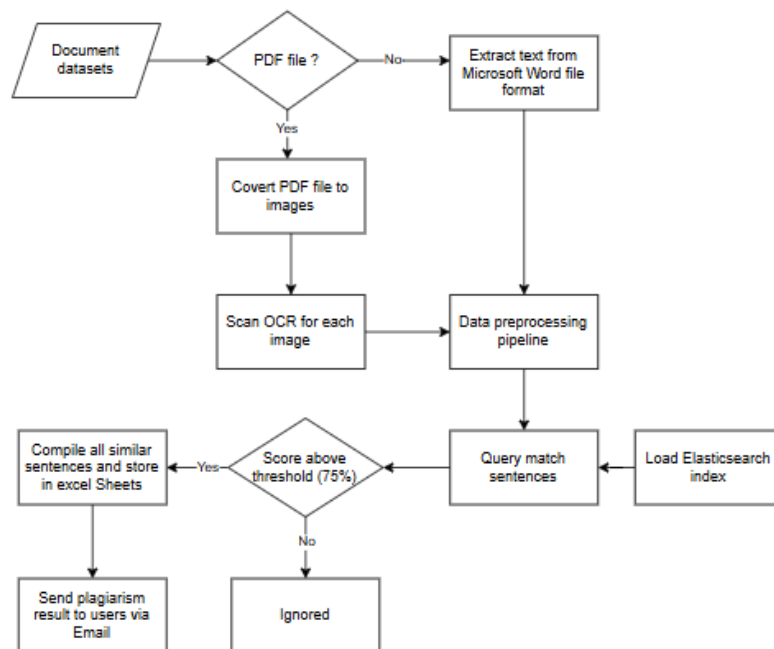


Figure 18: Plagiarism Detection Workflow

As shown in the diagram above, the plagiarism detection process begins with the upload of a document dataset. The system first checks whether the file is in PDF format. If it is, the PDF is converted into images and processed through OCR to extract text. If the file is a Microsoft Word document, the text is extracted directly. The raw text is then passed through a data preprocessing pipeline. Once preprocessed, the sentences are queried against the Elasticsearch index. If the similarity score is above the threshold (75%), the matched sentence is retained. All matched sentences are compiled into an Excel report and send to the users via email, completing the plagiarism detection process.

6. Web Application Implementation

This section outlines the implementation of the web application, which was developed using Flask for the backend API and React for the frontend interface. The system enables users to interact easily with the plagiarism detection tool through a user interface. The application supports three core functionalities: plagiarism checking, indexing new documents into MinIO file storage system and Elasticsearch, and viewing and managing stored documents.

6.1 Plagiarism Detection Webpage

The plagiarism detection webpage allows users to drag and drop multiple documents for analysis, supporting two file formats: PDF and Microsoft Word (.docx). Users are required to enter their email address to receive the plagiarism detection results once processing is complete.

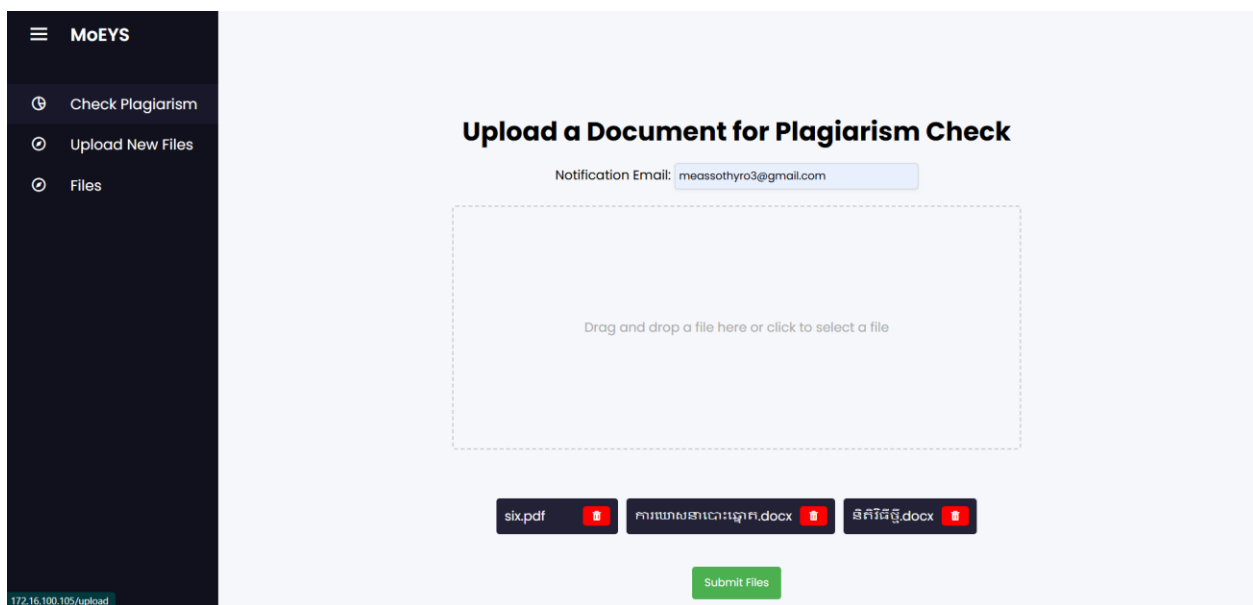


Figure 19: Plagiarism Detection Webpage

After users upload the files they want to check for plagiarism, they can click the "Upload" button. The data is then packaged into a FormData object and sent to the Flask backend server via an HTTPS POST request using the Fetch API. The backend receives the files and email input from users for asynchronous processing.

To handle high-volume uploads during peak hours, the backend is integrated with a Redis server to queue document processing tasks. Each job is queued and processed one at a time. A Redis dashboard, accessible via port 9181, allows administrators to monitor the job queue, track processing status, and identify failed tasks, as shown in the following figure.

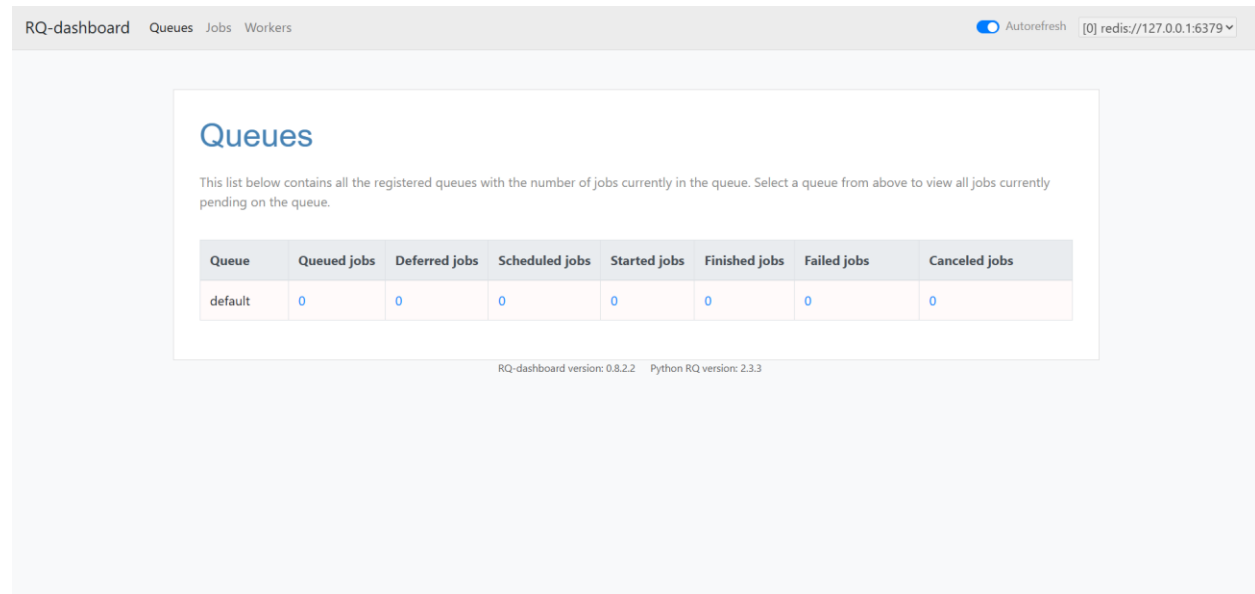


Figure 20: Redis Queue Dashboard Webpage

Once a job reaches the front of the queue, the plagiarism detection process begins by identifying the file type. If the file is a PDF, it is concurrently converted into images and then processed with OCR to extract text. The extracted text undergoes preprocessing, as described in Section 4.2.3. After preprocessing, each sentence in the cleaned data is queried against the Elasticsearch index. If the similarity score exceeds a defined threshold, the sentence is retained and compiled into an Excel report using the ExcelWriter Python library. The report includes columns such as the sentence from the uploaded document, the matched sentence found in the database, the university name of the original source, the name of the original document that was plagiarized, and the overall similarity score.

After excel report and certificate are generated. They are emailed to the user using the smtpplib, EmailMessage, and MIMEText from Python libraries.

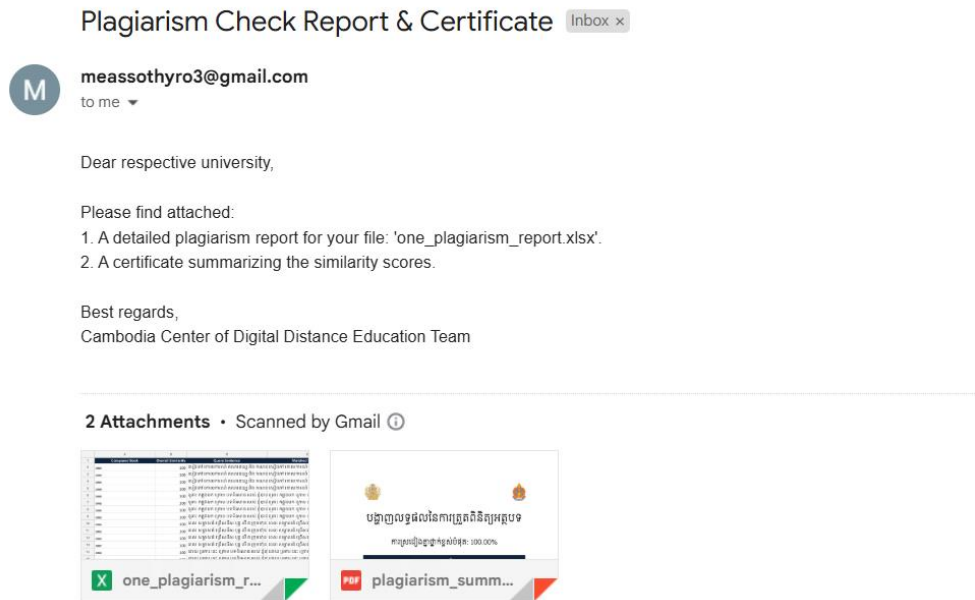


Figure 23: Plagiarism Detection Results Sent Via Email

6.2 Indexing and Uploading Documents Webpage

This webpage allows users to drag and drop multiple documents for uploading into both the Elasticsearch index and the file storage system. This functionality supports improved detection accuracy over time as the dataset grows. The drag-and-drop area supports two file formats: PDF and Microsoft Word (.docx). Users are required to select the university they want to upload the documents to, and also enter their email address to receive a status update once the upload process to both MinIO and Elasticsearch is complete.

Figure 24: Indexing and Uploading Documents Webpage

After selecting the target university and enter their email, users can drop the files into the designated area and click the "Upload" button to initiate the upload. The selected files and user input are then packaged into a `FormData` object and sent to the Flask backend server via an `HTTPS POST` request using the `Fetch API`. Once received, the backend uses the `MinIO` client library—along with the configured hostname and access credentials—to store the files in the appropriate bucket and folder path. These uploaded files can be viewed via the `MinIO` web dashboard, accessible on port 9001, as shown in the following figure.

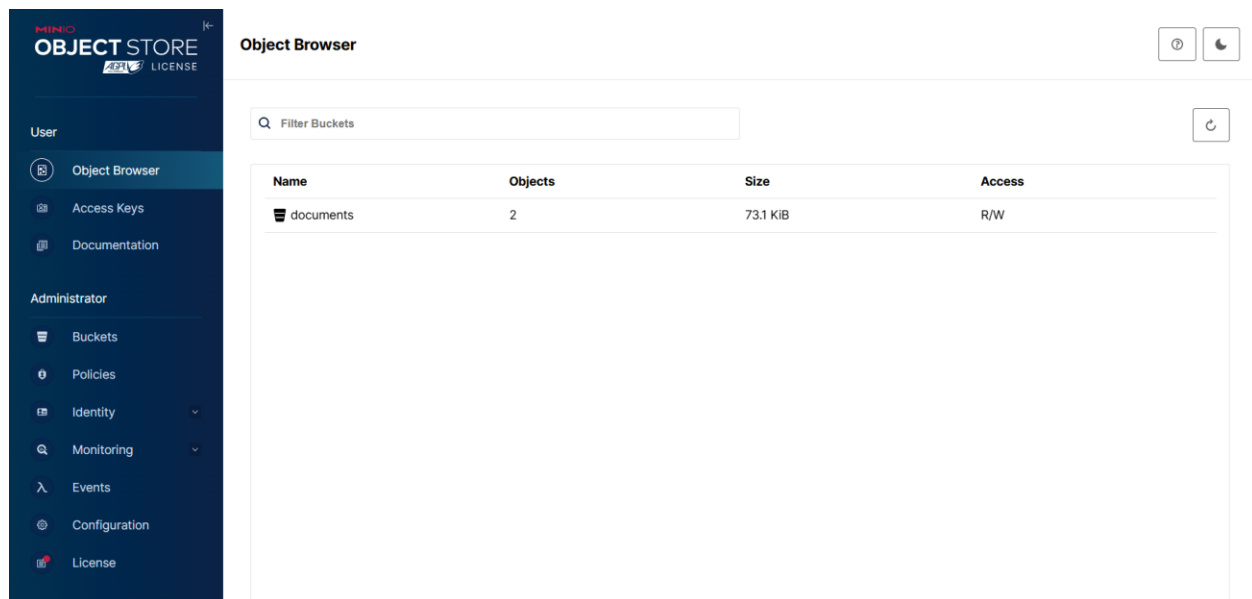


Figure 25: MinIO Webpage

To index the data in Elasticsearch, the backend preprocesses the extracted text and formats it according to the predefined index mapping. The *bulk()* function from the Elasticsearch library is then used to efficiently insert the new data. After both Elasticsearch indexing and MinIO storage are successfully completed, the *smtplib* library is triggered. The *EmailMessage* and *MIMEText* Python libraries are used to customize the email subject, body, and content, which are then sent to notify the user whether the upload was successful.

6.3 File Storage System Webpage

This webpage allows users to view all documents stored in the file storage system and delete them without needing to access the MinIO dashboard directly. It displays all available storage locations where documents have been uploaded. Additionally, users can download a document by clicking on its name or delete it by clicking the delete button, which removes the file not only from MinIO but also from the corresponding Elasticsearch index.

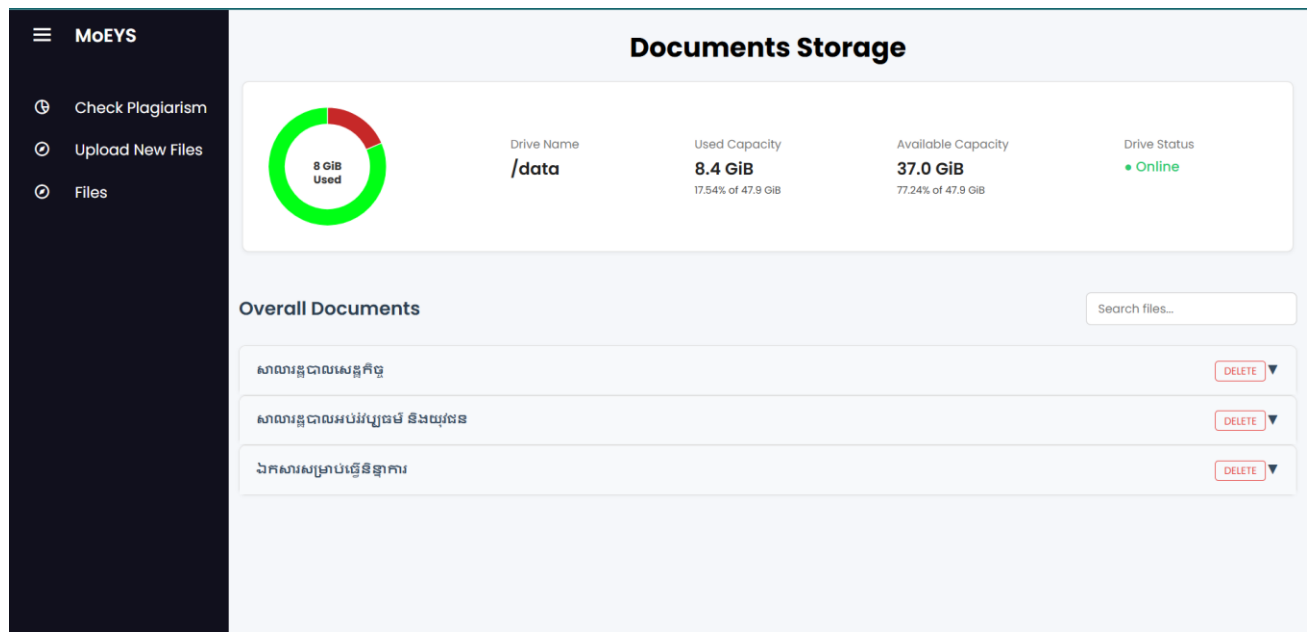


Figure 26: MinIO Custom Storage Webpage

7. Current Results and Limitations

The current application is designed to perform database-based plagiarism detection, allowing users to upload thousands of documents and check for similarities against content stored in the system's large internal database. It is capable of retrieving all exact matches as well as detecting some sentences that have been slightly modified.

One significant limitation lies in the OCR extraction process for the Khmer language. The accuracy of existing OCR tools remains relatively low for low-resolution documents, which directly affects the quality of extracted text. Since the platform relies heavily on OCR to process PDF files, improving Khmer OCR accuracy is essential to enhancing the system's overall reliability and detection performance.

8. User Feedback

During testing and review, staff at the Ministry of Education, Youth and Sport commented that *"The web application is efficient and fast. It allows us to upload documents and find similar sentences easily."* However, they also observed that *"However, uploading large PDF files format for either plagiarism detection process or into a file storage system takes noticeably longer compared to Microsoft Word documents file format."* This feedback highlights a key performance difference due to the need for Optical Character Recognition (OCR) when handling PDF files, whereas Microsoft Word files provide faster results.

9. Future Plan

For future development, we aim to improve the accuracy of Khmer OCR, which remains a critical challenge for reliable text extraction. We plan to conduct further research into OCR optimization techniques, such as custom model training, image preprocessing, and post-OCR correction. Enhancing OCR performance will significantly strengthen the quality of extracted text and, in turn, the accuracy of plagiarism detection.

Additionally, we plan to extend the system's capabilities to support plagiarism detection from external online sources. This will involve enabling users to query and detect plagiarized content directly from publicly available websites and internet resources, thereby expanding coverage beyond the existing internal database and improving the overall effectiveness of the detection system.

10. Conclusion

This project presents the design, and the implementation of a Khmer-language plagiarism detection system aimed at addressing the lack of digital tools for academic integrity in Cambodia. This project explores multiple detection methods—including TF-IDF with cosine similarity, N-gram with Jaccard similarity stored in PostgreSQL inverted index, BERT-based semantic matching, and Elasticsearch-based retrieval—to compare uploaded documents against a growing database of academic texts. Each method was evaluated for accuracy, scalability, and performance. The final system selects Elasticsearch to integrate with the web application, allowing users to interact easily.

The application allows users to upload Microsoft Word and PDF files, processes them through a preprocessing pipeline, and performs similarity checks using a web-based interface. Feedback from the Ministry of Education, Youth and Sport confirmed the system's effectiveness in identifying matching content, although OCR processing time and accuracy for PDF files remains a limitation. The system is currently deployed on a self-hosted server within the Ministry's infrastructure, enabling secure access for internal users.

In conclusion, this tool represents an important step toward modernizing academic quality control in Cambodia. While the current implementation focuses on database-based detection, future development will explore extending the system to query external web content, further improving its ability to detect paraphrased and copied material from online sources.

Bibliography

- [1] T. Phallavattana, "Plagiarism Among University Students: Causes, Consequences, and Recommendation," Cambodia Education Forum, 18 September 2023. [Online]. Available: <https://cefcambodia.com/2023/09/18/plagiarism-among-university-students-causes-consequences-and-recommendations/>.
- [2] A. M. T. G. a. C. D. Karen Avetisyan, "A Simple and Effective Method of Cross - Lingual Plagiarism Detection," *Research Square*, 2023.
- [3] N. Thuon, "Khmer Semantic Search Engine," *Digital Information and Document Retrieval*, 2024`.
- [4] G. Developer, "Tesseract Open Source OCR Engine (main repository)," Google, 2005. [Online]. Available: <https://github.com/tesseract-ocr/tessdata/blob/main/khm.traineddata>.
- [5] S. Ratanak, "How to Khmer OCR with Tesseract in Python," Khmerlang, 1 June 2022. [Online]. Available: <https://www.khmerlang.com/posts/5>.
- [6] S. Ratanak, "Khmer Analysis Plugin for Elasticsearch," Khmerlang, 28 August 2022. [Online]. Available: <https://www.khmerlang.com/posts/6>.
- [7] M. Durdin, "khmer normalizer," Github, 12 August 2024. [Online]. Available: <https://github.com/sillsdev/khmer-normalizer>.
- [8] Seanghay, "A fast Khmer word segmentation toolkit," pypi.org, 10 February 2025. [Online]. Available: <https://pypi.org/project/khmercut/>.
- [9] S. Ratanak, "Khmer spellcheck correction," Khmerlang, 2022. [Online]. Available: <https://khmerlang.com/tools/khmer-spelling-check>.
- [10] T. Nimol, "KSWv2-Khmer-Stop-Word-based-Dictionary-for-Keyword-Extraction," Github, 19 October 2024. [Online]. Available: <https://github.com/back-kh/KSWv2-Khmer-Stop-Word-based-Dictionary-for-Keyword-Extraction>.
- [11] M. U. Hutabarat, "Comparing Text Documents Using TF-IDF and Cosine Similarity in Python," Medium, 17 December 2023. [Online]. Available: <https://medium.com/@mifthulyn07/comparing-text-documents-using-tf-idf-and-cosine-similarity-in-python-311863c74b2c>.
- [12] P. Sharma, "How to Save and Load Machine Learning Models in Python Using Joblib library," [Online]. Available: <https://www.analyticsvidhya.com/blog/2023/02/how-to-save-and-load-machine-learning-models-in-python-using-joblib-library/>. [Accessed 04 April 2025].
- [13] N. E. Diana, "Measuring Performance of N-Gram and Jaccard Similarity Metrics in Document Plagiarism Application," *Researchgate*, 2019.

- [14] Pinecone, "Introduction to Facebook AI Similarity Search," Pinecone, 2022. [Online]. Available: <https://www.pinecone.io/learn/series/faiss/faiss-tutorial/>.
- [15] A. Brasetvik, "Elasticsearch from the Bottom Up," 17 September 2023. [Online]. Available: <https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up>.
- [16] S. Connelly, "Practical BM25 - Part2: The BM25 Algorithm and its variables," 19 April 2018. [Online]. Available: <https://www.elastic.co/blog/practical-bm25-part-2-the-bm25-algorithm-and-its-variables>.
- [17] C. Le, "Setting Up Redis as a Message Queue: A Step-by-Step Guide," DEV community, 10 September 2024. [Online]. Available: https://dev.to/chanh_le/setting-up-redis-as-a-message-queue-a-step-by-step-guide-5gj0.
- [18] M. Documentation, "Install and Deploy MinIO," 2022. [Online]. Available: <https://min.io/docs/minio/linux/operations/installation.html>.
- [19] L. P. Sirivath, "Culture ministry warns against plagiarism as Berne Convention set to come into force," Phnom Penh Post, 25 February 2022. [Online]. Available: <https://www.phnompenhpost.com/national/culture-ministry-warns-against-plagiarism-berne-convention-set-come-force>.
- [20] M. Potthast, "Cross-Language plagiarism detection," *Language Resources and Evaluation*, 2011.