DESIGN AND IMPLEMENTATION OF PLAGIARISM CHECKER

A Project report submitted in partial fulfilment of the requirements

For the award of the Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

Seerapu Harsha Nandini

(320136410099)

Patnala Harika

(320136410136)

Pujari Krishna Vamsi

(320136410089)

Palasetti Jayanth Kumar

(320136410076)

Under the esteemed guidance of

Mr.Syed Mujib Rahaman

Professor

Department of Computer Science Engineering

Dr. L. BULLAYYA COLLEGE OF ENGINEERING

(Permanently Affiliated to Andhra University, Visakhapatnam)

New Resapuvanipalem, Visakhapatnam

Year of submission:2024

Dr. L. BULLAYYA COLLEGE OF ENGINEERING

New Resapuvanipalem, Visakhapatnam-530013

Department of Computer Science Engineering

Bonafide Certificate

This is to certify Ms.Seerapu Harsha Nandini, Ms.Patnala Harika, Mr.Pujari Krishna Vamsi and Mr.Palasetti Jayanth Kumar bearing register numbers 320136410099, 320136410136, 320136410089, 320136410076 students of Final year B. Tech in Computer Science Engineering, has carried out the project work titled "Design and Implementation of Plagiarism Checker " at Dr. L. Bullayya College Of Engineering, Visakhapatnam during the academic year 2023-24.

Project Supervisor
Mr.Syed Mujib Rahaman
Associate Professor
Dept. of Computer
Science Engineering

Head of the Department
Dr. D. Madhavi
Professor
Dept. of Computer
Science Engineering

ABSTRACT

Plagiarism is a type of cheating that involves the use of another person's ideas, words etc without giving them proper credit. Intellectual property rights protect the original work of creators, ensuring they receive credit and recognition for their efforts.

While a lot of text data is generated everyday, the originality of the content could be validated using a plagiarism checker. A plagiarism checker works by comparing a given text with a vast database of sources. It analyzes the text for similarities and identifies any matching content.

Through the implementation of advanced algorithms, our plagiarism checker will provide accurate results by detecting potential instances of plagiarism, highlighting the specific sections that may have been copied. It provides a percentage to indicate the level of similarity between the given text and other sources.

Using a plagiarism checker can save you from unintentional plagiarism, which can have serious consequences in academic and professional settings. It helps you avoid accidental duplication by highlighting any similarities between your work and existing sources. By detecting copied content, a plagiarism checker helps maintain the authenticity and originality of your work. It also promotes ethical writing practices and encourages you to properly cite and reference your sources. Additionally, using a plagiarism checker can enhance your writing skills by making you more aware of proper attribution and citation techniques.

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to Prof D. Deepak Chowdary, Principal, Dr. L. Bullayya College of Engineering for giving us the opportunity to do this project.

We would like to thank Dr. D. Madhavi, HOD, Department of CSE, Dr. L. Bullayya College of Engineering, Visakhapatnam, for her guidance in producing this work.

We are deeply indebted the head of research cluster "Privacy Preserving Network Security in Data Science and Deep Learning" and project guide Mr.Syed Mujib Rahaman Associate Professor, Department of Computer Science Engineering, Dr. Lankapalli Bullayya College of Engineering,Visakhapatnam, for guiding us throughout the project in spite of his busy schedule.

Apart from the efforts of our, the success of this project depends largely on the encouragement of other faculty of CSE, Dr. Lankapalli Bullayya College of Engineering, Visakhapatnam. We take this opportunity to express my gratitude to the entire faculty who has been instrumental in the successful completion of this project.

Also deserving of thanks for our family and friends, for their support for their confidence in our achievements.

Seerapu Harsha Nandini
Patnala Harika
Pujari Krishna Vamsi
Palasetti Jayanth Kumar

DECLARATION

This is to declare that the Project work entitled “Design and Implementation of Plagiarism Checker” is a bonafide work done by us under the research cluster group “Privacy Preserving Network Security Data Science” with the esteemed guidance of Mr.Syed Mujib Rahaman, Associate Professor, Department of CSE, Dr.L.Bullayya College of Engineering. This project report is being submitted in the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science Engineering during the academic year 2023-2024. This project possesses originality as it is not extracted from any source and it has not been submitted to any other institutions and university.

Seerapu Harsha Nandini
Reg.No: 320136410099

Patnala Harika
Reg.No: 320136410136

Pujari Krishna Vamsi
Reg.No: 320136410089

Jayanth Kumar
Reg.No: 320136410076

Visakhapatnam
Date:

LIST OF TABLES

S.No.	TABLE	Page No.
1	SRS table	
2	Test cases table	

LIST OF FIGURES

S.No.	FIGURE	Page No.
1	Use case diagram	
2	Class diagram	
3	Object diagram	
4	Sequence diagram	
5	Activity diagram	
6	State chart diagram	

TABLE OF CONTENTS

Chapter No.	Title	Page No.
	Abstract Acknowledgement Declaration	
1	Introduction 1.1 Background and Motivation 1.2 Problem Statement	
2	Requirement Elicitation and Analysis 2.1 Existing System 2.2 Proposed System 2.3 Feasibility Study 2.4 System Requirements 2.4.1 Functional Requirements 2.4.2 SRS table 2.4.3 Non Functional Requirements	
3	System Design 3.1 Object Oriented Analysis and Design 3.1.1 Use case Diagram 3.1.2 Sequence Diagram 3.1.3 Class Diagram 3.1.4 Activity Diagram	
4	Implementation Details 4.1 Software Environment 4.1.1 About Algorithm 4.1.2 Software Technologies	
5	Testing 5.1 Test cases table	
6	Conclusion	
	References Appendix A- Input/Output Screens B- Sample Code	

1. INTRODUCTION

Plagiarism can have some pretty serious consequences. It can result in academic penalties like failing grades, suspension, or even expulsion. In professional settings, it can damage your reputation and career prospects. In some cases, legal actions can be taken against those who engage in plagiarism. That's why it's so important to use tools like plagiarism checkers to ensure the originality and integrity of your work.

A plagiarism checker is a tool that helps identify any copied or unoriginal content in a document or piece of writing. It's a valuable tool for academic institutions and writers to ensure the authenticity and originality of their work. By comparing the text with a database of existing content, the plagiarism checker can highlight any similarities or matches found.

A plagiarism checker is crucial because it helps maintain academic integrity and ensures the originality of work. By using a plagiarism checker, you can avoid unintentional plagiarism and properly attribute sources. It also promotes ethical writing practices and upholds the credibility of your work. Additionally, plagiarism checkers save time by quickly identifying any potential instances of copied content. It's an essential tool for students, researchers, and writers to ensure their work is authentic and respects intellectual property.

1.1 Background and Motivation:

Plagiarism is a serious offence that involves using someone else's ideas, words, or work without proper attribution. It undermines the credibility of the author and violates ethical standards.

Plagiarism checkers were developed to address this issue by providing a tool to detect instances of plagiarism. They help educators, researchers, and content creators identify any copied or unoriginal content in a document. By comparing the text with a vast database of sources or with a document provided by the user, plagiarism checkers can flag potential matches and highlight areas that may require further investigation.

The motivation behind using a plagiarism checker is to promote honesty, integrity, and originality in academic and professional writing. It encourages individuals to properly attribute ideas and give credit where it is due. Additionally, plagiarism checkers help educators in evaluating students' work and ensuring that they have not engaged in any form of plagiarism.

By using a plagiarism checker, you can ensure that your work is original, properly cited, and free from any unintentional instances of plagiarism. It helps maintain the quality and credibility of your writing.

1.2 Problem Statement:

Develop a plagiarism checker that can accurately detect similarities between texts to ensure academic integrity and prevent plagiarism in academic and professional settings.

So, the problem we are addressing with the plagiarism checker is the prevalence of plagiarism in academic and professional settings. Plagiarism undermines the integrity of work and can have serious consequences for individuals. The challenge is to develop a reliable and user-friendly tool that accurately detects instances of plagiarism, helping users maintain academic integrity and ensure the authenticity of their work. By creating this plagiarism checker, we aim to promote originality and ethical writing practices.

2. Requirements Elicitation and Analysis

2.1 Existing System:

We currently rely on manual comparison or limited database searches to detect plagiarism. The problem with these methods is that they can be time-consuming and may not provide comprehensive results. So, what we need is an automated and efficient system that can handle large volumes of text and accurately identify instances of plagiarism. The current system also has limitations like the lack of real-time detection, limited coverage of online sources, and potential false positives or false negatives. By developing a new and improved plagiarism checker, we can overcome these challenges and provide a more effective solution.

2.2 Proposed System:

Comparing every word of input text against all the source documents is an expensive process. Hence we propose a two-step approach to do plagiarism score calculation in an efficient way. An information retrieval process is performed as the first step. In this step, the input text and source documents are tokenized into sentences. These sentences in input text as well as sentences in source documents are vectorized using a pre-trained Sentence Transformers model which was originally trained on a huge corpus. The sentences in source documents that are very close to the input text sentences are retrieved by calculating cosine similarity of vectors. By the end of the information retrieval step, we will have only a limited number of source sentences that are similar to input text sentences, which makes further processing simpler in a short time.

In the second step, the input text and the retrieved source sentences are tokenized into words. Words are preprocessed by converting into lower case, removing punctuations, cleaning extra white spaces and removing stop words. N-grams of three word sets are obtained from the preprocessed words. These n-grams are vectorized using TF-IDF vectorizer. The vectors of input text n-grams are compared with the vectors of retrieved sentences n-grams for exact match. The indices of matched n-grams are collected for both input text and source sentences.

Based on the number of words in an input text that are exactly matching with source documents, a plagiarism score is assigned. For all the input sentences, the plagiarism scores are accumulated and an originality score is assigned. For reporting, details of the source documents that are used in plagiarism are collected using the indices. A final report of originality score as well as source documents used in plagiarism are created in a human-readable format.

Algorithms used in the implemented system are as follows: Sentence Tokenization and Word Tokenization using NLTK library, Stopword removal using NLTK library, Punctuation removal using string and regex libraries, vectorization using Sentence Transformers pretrained model from HuggingFace, Cosine similarity scoring using util module of sentence-transformers library, TF-IDF vectorization using SciKit-Learn library, N-gram collection using SciKit-Learn library, Efficient exact word matching using SciKit-Learn library, Deployment of project and writing UI page using Streamlit library.

2.3 Feasibility Study:

The feasibility study assesses various aspects of the system to determine if it is feasible to develop and implement. Here are some key areas to consider:

1. Technical Feasibility: Evaluate the technical requirements and capabilities needed to develop the plagiarism checker system. Considering factors such as the availability of suitable technologies, software development expertise, and any potential technical challenges that may arise.

2. Economic Feasibility: Assess the financial feasibility of developing and maintaining the plagiarism checker system. Considering the costs associated with software development, infrastructure, ongoing maintenance, and potential revenue streams or cost-saving benefits that the system may bring.

3.Operational Feasibility: Evaluate the practicality of implementing and operating the plagiarism checker system within your intended environment. Consider factors such as user acceptance, integration with existing systems or platforms, and any potential operational challenges that may arise.

2.4 System Requirements

The system requirements or software requirements is a listing of what software programs or hardware devices are required to operate the program or game properly. System requirements is a statement that identifies the functionality that is needed by a system in order to satisfy the user's requirements. They are the first and foremost important part of any project, because if the system requirements are not fulfilled, then the project is not complete. A software requirement is a document that captures a complete description about how the system is expected to perform. It is usually signed off at the end of requirements engineering phase.

A software requirement can be of 2 types:
1. Functional Requirements
2. Non-functional Requirements

2.4.1 Functional Requirements:

The functional requirements for a system describe what the system should do. Those requirements depend on the type of software being developed, the expected users of the software. These are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.

1.Text Input:
Accept text input in various formats, such as plain text, documents (e.g., DOCX, PDF), and URLs.

2.Document Comparison:
Compare the submitted text against a database of sources, including academic papers, articles, books and other documents.

3.Multiple File Support:
Allow users to check multiple documents simultaneously for plagiarism, either by uploading multiple files or entering multiple URLs.

4.Real-time Feedback:

Provide real-time feedback during the checking process, highlighting potential instances of plagiarism and indicating the percentage of similarity.

5.Detailed Reports:
Generate detailed plagiarism reports that identify the matched sources, highlight the plagiarised sections, and provide links to the original sources.

6.Originality Score:
Calculate an originality score or similarity index for the submitted content, indicating the degree of similarity to other sources.

7.Source Retrieval:
Offer the ability to retrieve and display the original sources that match the submitted content.

8.Batch Processing:
Allow users to submit a batch of documents for plagiarism checking in a single operation.

2.4.2 SOFTWARE REQUIREMENTS SPECIFICATION:

Functional requirements table

S.no	**Requirements**	**Requirements Number**	**Essential(or) desirable**	**Description**
1.	Comparing Documents	RS1	Essential	The System shall be able to compare two documents for similarity.
2.	Support Files	RS2	Essential	The System shall support various file formats like URL of PDF, PDF ,text.
3.	Provide Percentage	RS3	Essential	The System shall provide percentage of similarity between documents.
4.	Handling documents	RS4	Essential	The System shall handle large documents efficiently.
5.	Detailed Reports	RS5	Essential	The System shall be able to provide the plagiarized content.

2.4.3 Non-functional Requirements

Non-Functional Requirement (NFR) specifies the quality attribute of a software system. Non Functional requirements in Software Engineering allows you to impose constraints or restrictions on the design of the system

1. Performance: The plagiarism checker should provide fast and efficient results, even when processing large amounts of data.

2. Accuracy: The checker should have a high level of accuracy in detecting plagiarism and minimising false positives.

3. Scalability: The system should be able to handle a growing number of users and an increasing volume of content without compromising performance.

4. Security: The checker should ensure the privacy and security of user-submitted content and data.

5. User-friendly interface: The interface should be intuitive, easy to navigate, and visually appealing for a positive user experience

3. SYSTEM DESIGN

System Design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. In System design, developers:

● Define design goals of the project
● Decompose the system into smaller sub systems
● Design hardware/software strategies
● Design persistent data management strategies
● Design global control flow strategies
● Design access control policies and
● Design strategies for handling boundary conditions.
System design is not algorithmic. It is decomposed of several activities. They are: ● Identify Design Goals
● Design the initial subsystem decomposition
● Refine the subsystem decomposition to address the design goals. System Design is the transformation of an analysis model into a system design model. Developers define the design goals of the project and decompose the system into smaller subsystems that can be realised by individual teams. Developers also select strategies for building the system, such as the hardware/software platform on which the system will run, the persistent data management strategy, the goal control flow, the access control policy and the handling of boundary conditions. The result of the system design is a model that includes a clear description of each of these strategies.

3.1 Object Oriented Analysis and Design

"Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain". Object–Oriented Analysis (OOA) is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises interacting objects. The main difference between object oriented analysis and other forms of analysis is that in object-oriented approach, requirements are organised around objects, which integrate both data and functions. They are modelled after real-world objects that the system interacts with. In traditional analysis methodologies, the two aspects - functions and data - are considered separately.

The primary tasks in object-oriented analysis (OOA) are:
- Identifying objects
- Organising the objects by creating object model diagram
- Defining the internals of the objects, or object attributes
- Defining the behaviour of the objects, i.e., object actions
- Describing how the objects interact

The common models used in OOA are use cases and object models.

3.1.1 Scenarios

Scenario 1

Scenario Name	Upload Source
Participating Actor	User
Pre-Condition	None
Flow of Events	User uploads a source document
Post Condition	System will ask user to upload query document

Scenario 2

Scenario Name	Upload Query
Participating Actor	User
Pre-Condition	User uploaded source document
Flow of Events	User uploads a Query document
Post Condition	Comparison between source and query would be done.

Scenario 3

Scenario Name	Comparison & document
Participating Actor	System
Pre-Condition	User uploaded a source & Query document
Flow of Events	System compares the given source & query documents.
Post Condition	Provide percentage

Scenario 4

Scenario Name	Provide percentage
Participating Actor	System
Pre-Condition	Documents compared
Flow of Events	System provides a percentage of similarity with detailed reports.
Post Condition	Provide plagiarised content

Scenario 5

Scenario Name	Provide plagiarised content
Participating Actor	System
Pre-Condition	Provide percentage
Flow of Events	System provides the plagiarised content.
Post condition	Plagiarised content will be displayed on the webpage.

UML Diagrams

A UML diagram is a diagram based on the UML (Unified Modelling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artefacts or classes, in order to better understand, alter, maintain, or document information about the system.

UML is a modern approach to modelling and documenting software. In fact, it’s one of the most popular business process modelling techniques. It is based on diagrammatic representations of software components. As the old proverb says: “a picture is worth a thousand words”. By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

Mainly UML has been used as a general-purpose modelling language in the field of software engineering. However, it has now found its way into the documentation of several business processes or workflows. For example, activity diagrams, a type of UML diagram, can be used as a replacement for flowcharts. They provide both a more standardised way of modelling workflows as well as a wider range of features to improve readability.

UML is a visual language rather than a programming language. To depict the behaviour and structure of a system, we use UML diagrams. Software engineers, businesspeople, and system architects can benefit from UML's modelling, design, and analysis tools. Unified Modeling Language was made a standard by the Object Management Group (OMG) in 1997. Since then, OMG has been in charge of it. UML was issued as a recognised standard by the International Organization for Standardization (ISO) in 2005. UML has undergone numerous revisions over time and is regularly examined.

3.1.2 Use Case

In the Unified Modelling Language (UML), a use case diagram can summarise the details of your system's users (also known as actors) and their interactions with the system.An effective use case diagram can help the team discuss and represent:

- Scenarios in which system or application interacts with people, organisations, or external systems.
- Goals that your system or application helps those entities (known as actors) achieve.
- The scope of the system.

UML use case diagrams are ideal for:

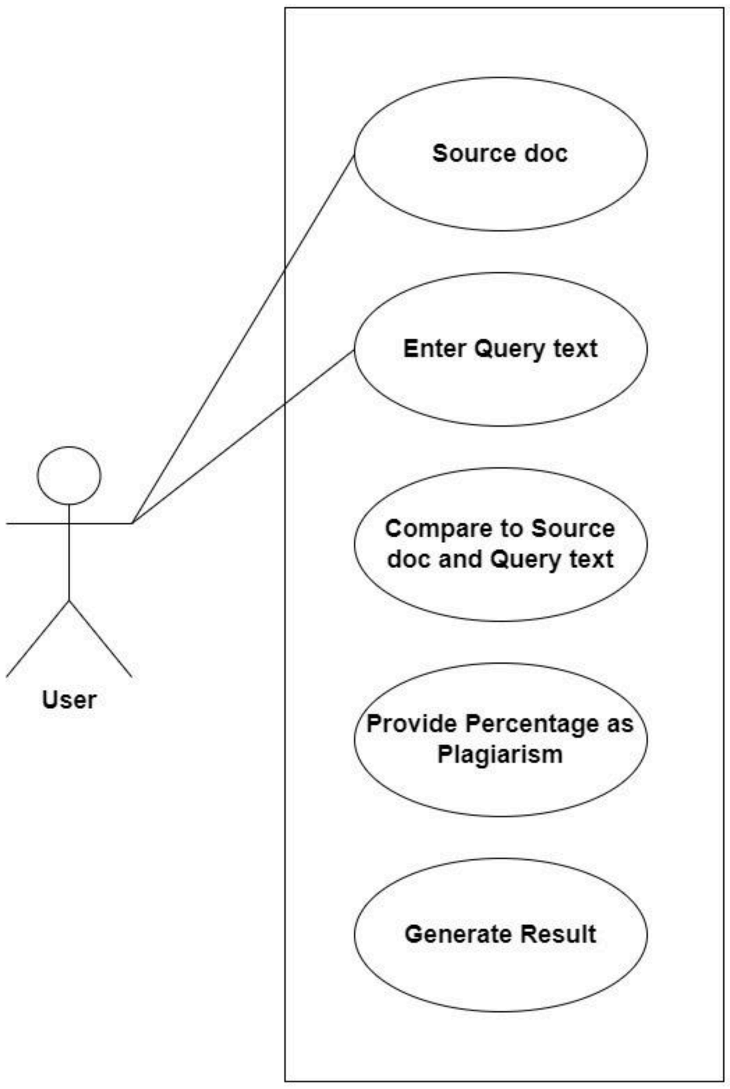
- Representing the goals of system-user interactions.
- Defining and organising functional requirements in a system.
- Specifying the context and requirements of a system.
- Modelling the basic flow of events in a use case

Relationships in use case diagrams:
There are five types of relationships in use case diagrams.

They are:

- Association between an actor and a use case
- Generalisation of an actor
- Extend relationship between two use cases
- Include relationship between two use cases

Use-case Diagram


Source doc
Enter Query text
Compare to Source doc and Query text
Provide Percentage as Plagiarism
Generate Result
User

1.Source document upload to the user. And also enter the query text.

2.In this diagram,the User interacts with the Plagiarism Checker by submitting text.
3.The plagiarism checker then checks for similarity and notifies the user of the result.
4.System comparison to the source document text and query text,similarity checked by using cosine similarity algorithm.
5.The Result use case represents whether the text is flagged for plagiarism or not.
6.Threshold value met plagiarism percentage displayed on the webpage.

3.1.3 Activity Diagram:

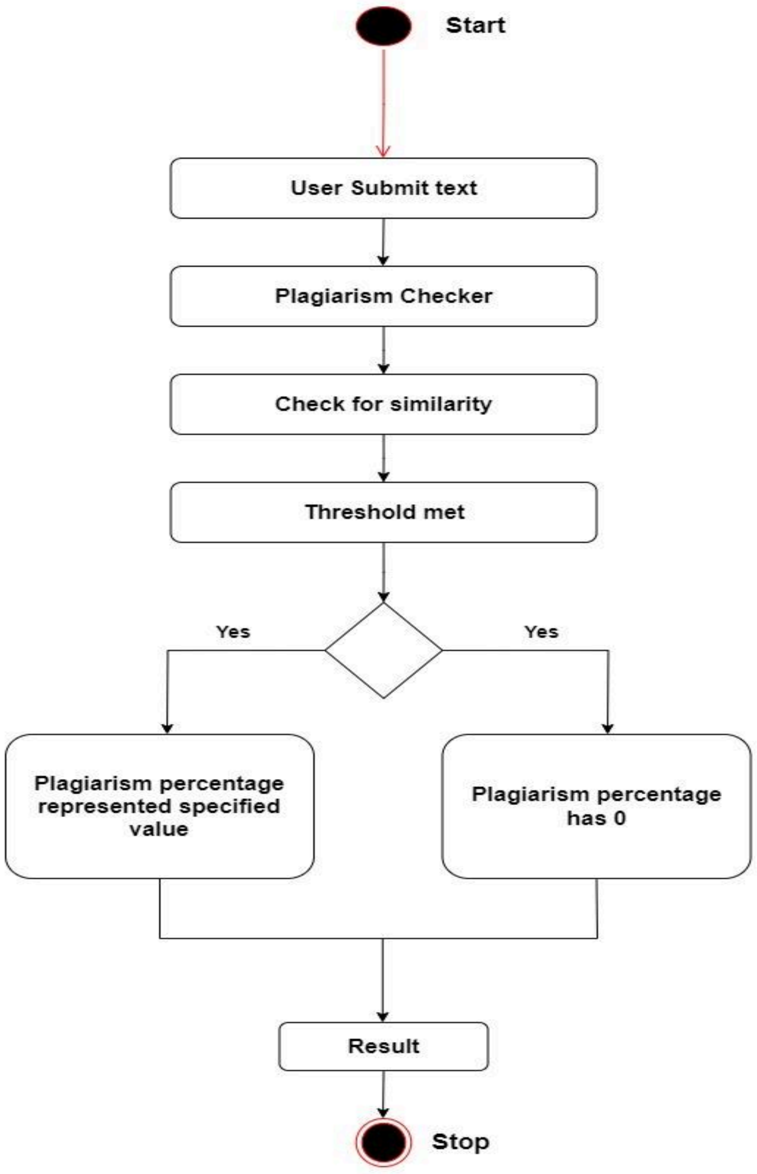
Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration. In the Unified Modeling Language, activity diagrams are intended to model both computational and organisational processes (i.e., workflows), as well as the data flows intersecting with the related activities. Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data.

- **Initial Node:** Represents the Starting point of an activity.
- **Activity State:** Represents the activities within the process.
- **Action:** Represents the executable sub-areas of an activity.
- **Control Flow:** Represents the Flow of control from one action to another.
- **Activity Final node:** Represents the end of single control flow.

Benefits:

1.Shows the progress of workflow amongst the users, and the system.
2.Demonstrates the logic of an algorithm.
3.Simplifies the majority of the UML processes by clarifying complicated use cases.

Activity Diagram:



1.User submits text:
The activity starts with the user submitting text for plagiarism checking.
2.System processes text:
The system processes the submitted text, including any necessary preprocessing.
3.System compares text:
The system compares the processed text with existing documents using a text comparison algorithm
4.If similarity > threshold:
If the similarity between the submitted text and any existing document is above a specified threshold:
-The system generates a plagiarism report.
-The report is displayed to the user.
5.If similarity <= threshold:
-If the similarity is below or equal to the threshold:
-The user is notified that no plagiarism was detected.

3.1.4 Class Diagram

Class diagrams are one of the most useful types of diagrams in UML as they clearly map out the structure of a particular system by modelling its classes, attributes, operations, and relationships between objects. It is a static diagram that represents the static view of an application. Class diagram is not only used for visualising, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

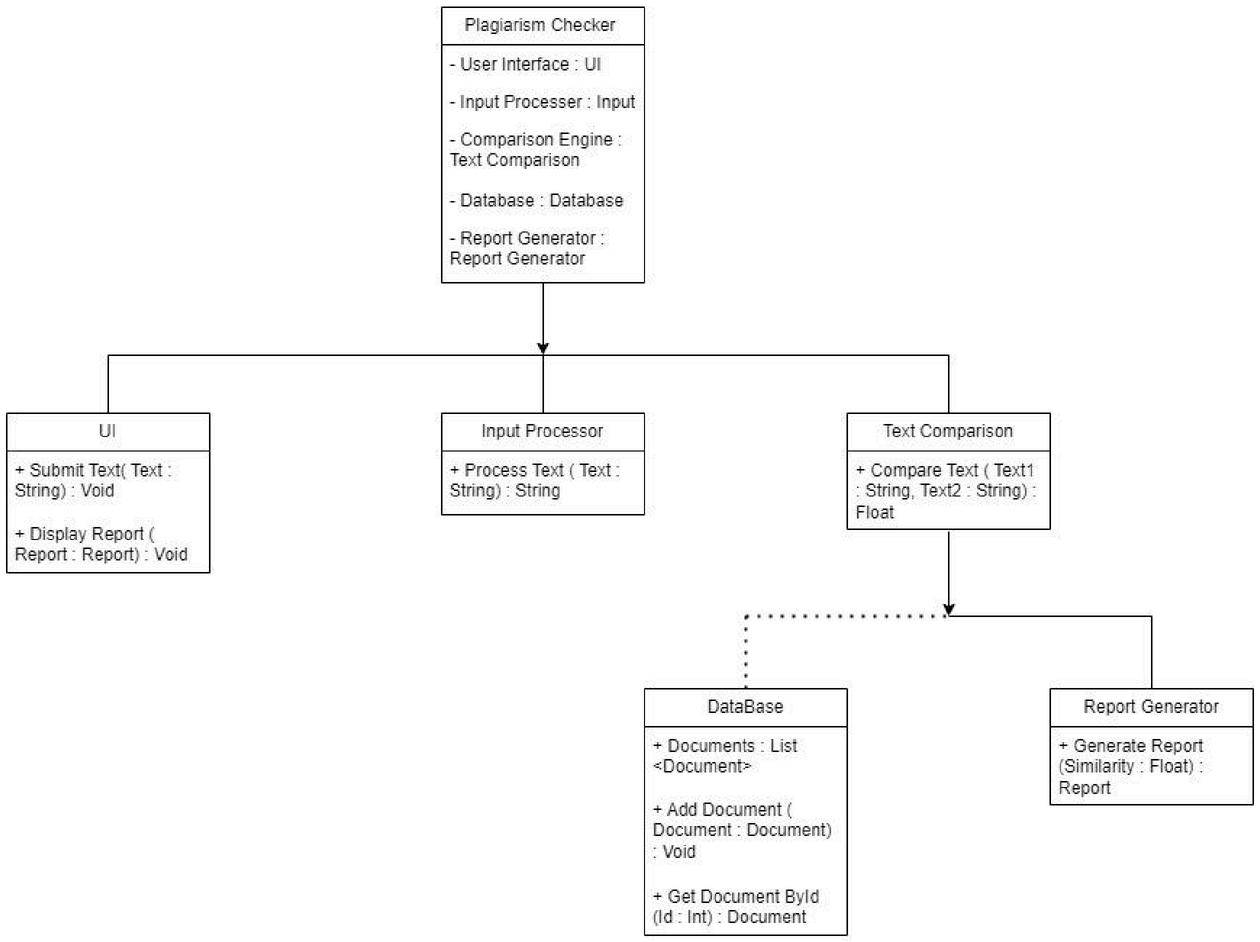
The class shape itself consists of a rectangle with three rows. The top row contains the name of the class, the middle row contains the attributes of the class, and the bottom section expresses the methods or operations that the class may use. Classes and subclasses are grouped together to show the static relationship between each object.

1. Class Name: The name of the class appears in the first partition.

2. Class Attributes :Attributes are shown in the second partition.The attribute type is shown after the colon.Attributes map onto member variables (data members) in code.

3. Class Operations (Methods) :Operations are shown in the third partition.They are services the class provides.The return type of a method is shown after the colon at the end of the method signature.

The purpose of the class diagram can be summarised as:
● Analysis and design of the static view of an application.
● Describe responsibilities of a system.
● Base for component and deployment diagrams.
● Forward and reverse engineering.


Plagiarism Checker
- User Interface : UI
- Input Processer : Input
- Comparison Engine : Text Comparison
- Database : Database
- Report Generator : Report Generator
UI
+ Submit Text(Text : String) : Void
+ Display Report (Report : Report) : Void
Input Processor
+ Process Text (Text : String) : String
Text Comparison
+ Compare Text (Text1 : String, Text2 : String) : Float
DataBase
+ Documents : List <Document>
+ Add Document (Document : Document) : Void
+ Get Document ById (Id : Int) : Document
Report Generator
+ Generate Report (Similarity : Float) : Report

1.**PlagiarismChecker**: The main object coordinating various components.
2.**UserInterface**: Manages user interactions, submitting text, and displaying reports.
3.**InputProcessor**: Processes the submitted text.
4.**TextComparison**: Compares text for similarity.
5.**Database**: Stores documents for comparison.
6.**ReportGenerator**: Generates plagiarism reports.

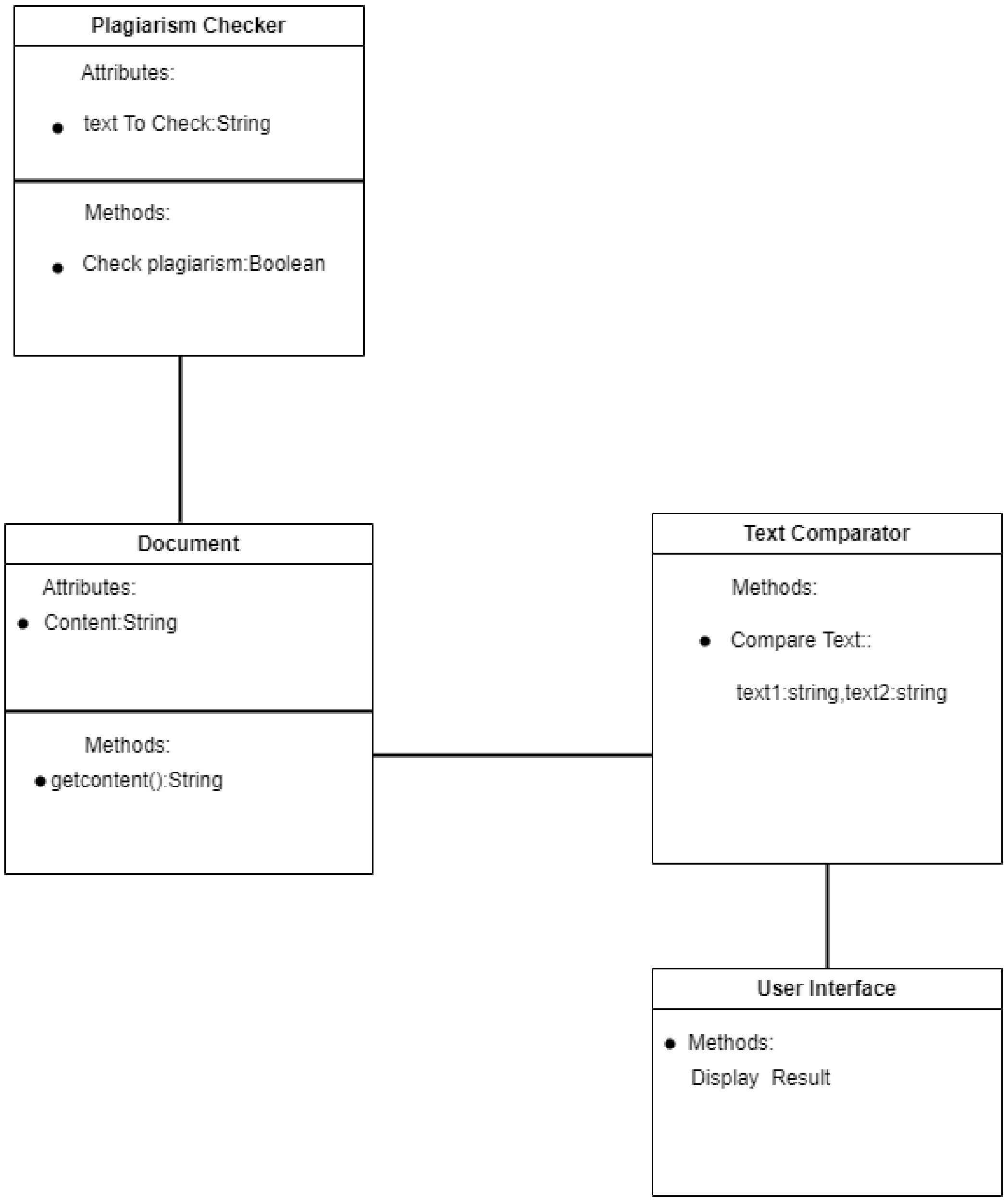
3.1.5 Object Diagram

Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.

Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.

Object diagrams are used to render a set of objects and their relationships as an instance.

1. All the objects present in the system should be examined before starting drawing the object diagram.
2. Before creating the object diagram, the relation between the objects must be acknowledged.
3. The association relationship among the entities must be cleared already.
4. To represent the functionality of an object, a proper meaningful name should be assigned.
5. The objects are to be examined to understand its functionality.


Plagiarism Checker
Attributes:
● text To Check:String
Methods:
● Check plagiarism:Boolean
Document
Attributes:
● Content:String
Methods:
● getcontent():String
Text Comparator
Methods:
● Compare Text::
text1:string,text2:string
User Interface
● Methods:
Display Result

3.1.6 Sequence Diagram

Sequence diagrams, commonly used by developers, model the interactions between the objects to the single use case. They illustrate how the different parts of a system interact with each other to carry outa function and the order in which the interactions occur when a particular use case is executed in simple words, a sequence diagram shows different parts of a system in sequence to get something done.

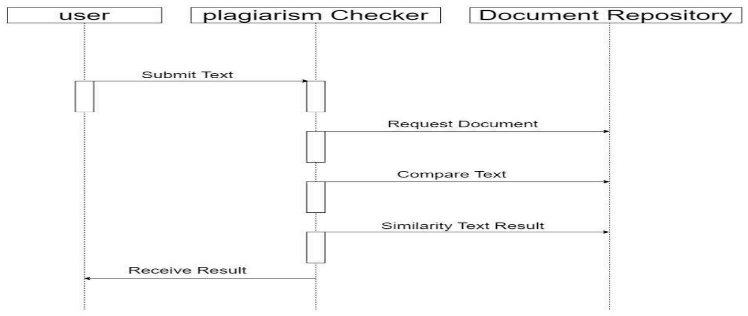
A sequence diagram describes an interaction among a set of objects participated and arranged in a chronological order. It shows the objects participating in the interaction and the messages that they send to each other.

A sequence diagram simply depicts interaction between objects in a sequential order i.e., the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

Uses of sequence diagrams –

- Used to model and visualise the logic behind a sophisticated function,operation or procedure.
- They are also used to show details of UML use case diagrams.
- Used to understand the detailed functionality of current or future systems.
- Visualise how messages and tasks move between objects or components in a system.

Sequence diagram:



1.The User initiates the process by submitting text to the Plagiarism Checker.2.The Plagiarism Checker requests documents from the Document Repository to compare the submitted text against.
3.The plagiarism Checker compares the submitted text with the documents in the repository to check for similarity.
4.The plagiarism Checker provides the similarity result based on the comparison.
5.The User receives the result of the plagiarism check.

The sequence diagram is an interaction between the user ,the plagiarism checker,and the document repository during the process of checking for plagiarism on text.

4.Implementation Details

The objective of the coding or programming phase is to translate the design of the system produced during the design phase into code in a given programming language, which can be executed by a computer and that performs the computation specified by the design.

The coding phase affects both testing and maintenance. The goal of coding is not to reduce the implementation cost, but the goal should be to reduce the cost of later phases.
Coding Approach:
There are two major approaches for coding any software system.
They are Top-Down approach and Bottom-up approach.

Bottom-up Approach can best suit for developing the object-oriented systems. During the system design phase, we decompose the system into an appropriate number of subsystems, for which objects can be modelled independently. These objects exhibit the way the subsystems perform their operations.

Once objects have been modelled, they are implemented by means of coding. Even though related to the same system as the objects are implemented by each other, the Bottom-Up approach is more suitable for coding these objects. In this approach, we first do the coding of objects independently and then we integrate these modules into one system to which they belong.

To provide as much flexibility as possible, the display application is implemented as multiple nearly- independent modules. Each module is responsible for the display of a particular kind of data and is implemented.

4.1 Software Environment

Python programming language is used due to its extensive libraries like NLTK for tokenizing sentences and words, string and regex libraries for punctuation removal, sentence transformers for vectorization, util module for scoring cosine similarity, SciKit-Learn library for TF-IDF vectorization and for N-gram collection. And deployment of projects and writing UI pages using Streamlit library.

4.1.1. Software Technologies

NLTK library

The Natural Language Toolkit (NLTK) is a popular library used for natural language processing tasks in Python. NLTK provides various functionalities, including sentence tokenization and word tokenization.

Sentence tokenization is the process of splitting a text into individual sentences. NLTK provides a method called sent_tokenize() that can be used to perform sentence tokenization. It uses pre-trained models and rules to identify sentence boundaries accurately. Word tokenization, on the other hand, is the process of splitting a sentence or a piece of text into individual words or tokens. NLTK provides a method called word_tokenize() that can be used for word tokenization.

String and regex libraries

The string library in Python provides a set of useful functions and constants for working with strings. One of the functions that can be used for punctuation removal is the translate() function. This function allows you to create a translation table that maps specific characters to None, effectively removing them from the string. By creating a translation table that includes all the punctuation marks you want to remove, you can easily eliminate them from your text.

The regex (regular expression) library in Python, often referred to as re, provides powerful tools for pattern matching and text manipulation. Using regex, you can define patterns that match specific characters or character classes, including punctuation marks. The re library provides functions like sub() and subn() that allow you to replace matches with a specified string or remove them altogether. By constructing a regex pattern that matches punctuation marks, you can remove them from your text using these functions.

Sentence Transformers library

Sentence transformers are a powerful tool in natural language processing that can convert sentences or text into numerical representations, also known as vectors. These vectors capture the semantic meaning of the sentences, enabling various downstream tasks like text classification, clustering, and similarity matching.

Sentence transformers use pre-trained models that have been trained on large corpora of text data. These models learn to encode the meaning of sentences into dense vectors. One popular pre-trained model for sentence transformers is called BERT (Bidirectional Encoder Representations from Transformers). BERT is a transformer-based model that captures contextual information and produces high-quality sentence embeddings.

Scikit-learn library

The scikit-learn library is a powerful tool for machine learning in Python. It provides a wide range of algorithms and utilities for tasks like classification, regression, clustering, and more. It's a popular choice among data scientists and machine learning practitioners because of its ease of use and extensive documentation.

Streamlit library

It is a library for building interactive web applications with Python. With Streamlit, you can create beautiful and intuitive user interfaces for your data science projects or any other Python application.

Streamlit makes it super easy to create interactive components like sliders, dropdowns, and buttons, and you can update the UI in real-time as users interact with it.

4.2 Algorithm

Sentence transformers

SentenceTransformers is a Python framework for state-of-the-art sentence, text and image embeddings.

You can use this framework to compute sentence / text embeddings for more than 100 languages. These embeddings can then be compared e.g. with cosine-similarity to find sentences with a similar meaning. This can be useful for semantic textual similarity, semantic search, or paraphrase mining.

The framework is based on PyTorch and Transformers and offers a large collection of pre-trained models tuned for various tasks. Further, it is easy to fine-tune your own models.

Cosine Similarity

Term Frequency - Inverse Document Frequency (TF-IDF) is a widely used statistical method in natural language processing and information retrieval. It measures how important a term is within a document relative to a collection of documents (i.e., relative to a corpus).

Words within a text document are transformed into importance numbers by a text vectorization process. There are many different text vectorization scoring schemes, with TF-IDF being one of the most common.

5.TESTING

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrast to the actual requirements. The main purpose of testing is to discover errors and it is a process to check the functionality of components. It ensures that software systems meet its requirements and user expectations without fault . Software Testing is an important element of quality assurance and represents the ultimate view of specification and design. Testing can also be defined as – A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

When to Start Testing?

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.

It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC −
During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
Reviewing the design in the design phase with the intent to improve the design is also considered as testing Testing performed by a developer on completion of the code is also categorized as testing.

When to Stop Testing?
It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that a software is 100% tested. The following aspects are to be considered for stopping the testing process
- Testing Deadlines
- Completion of test case execution
- Completion of functional and code coverage to a certain point.

White Box Testing:
White box testing is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. Implementation and impact of the code are tested.
It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
-Code implementation is necessary for white box testing.
-It is mostly done by software developers.
-Knowledge of implementation is required.
-It is the inner or the internal software testing.
-It is the Structural test of the software.
-This type of testing is software is started after detail design document.

Black Box Testing:
Black box testing is a software testing method in which the internal structure/design/implementationof the item being tested is not known to the tester. Only the external design and structure are tested.
It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.
-Implementation of code is not needed for black box testing.

-It is mostly done by software testers.
-No knowledge of implementation is needed.
-It can be referred to as outer or external software testing.
-It is a functional test of the software.
-This testing can be initiated on the basis of requirement specifications document.

5.1 Test cases table:

S.No	TestCase Number	Requirement with description	Essential (or) Desirable	Expected Output	Actual Output	Result
1.	TC1	Provide two identical download as source & query.	Essential	The plagiarism checker should detect 100% similarity b/w documents.	The plagiarism checker should detect 100% similarity b/w documents.	SUCCESS

2.	TC2	Provide documents in different file formats such as pdf,URL or pdf,text etc.	Essential	The plagiarism checker should be able to handle & analyze different file formats accurately.	The plagiarism checker should be able to handle & analyze different file formats accurately.	SUCCESS
3.	TC3	Provide two documents of different lengths,one shorter or longer.	Essential	The plagiarism checker should accurately detect similarities, regardless of their length.	The plagiarism checker should accurately detect similarities, regardless of their length.	SUCCESS
4.	TC4	Give two documents where different words with similar meanings are used.	Essential	The plagiarism checker should be able to identify similarities b/w documents, even if different Synonyms are used.	The plagiarism checker should be able to identify similarities b/w documents, even if different Synonyms are used.	SUCCESS

5.	TC5	Provide two documents where only a portion of 1 document is copied into the other.	Essential	The plagiarism checker should be able to identify copied portion and provide plagiarism content.	The plagiarism checker should be able to identify copied portion and provide plagiarism content.	SUCCESS

6. CONCLUSION

The main objective of our plagiarism checker system is to detect and prevent instances of plagiarism in academic writing. It aims to promote academic integrity and ensure that originality and proper citation practices are upheld.

The significance of our research on a plagiarism checker lies in its contribution to the academic community. By developing an effective plagiarism detection system, we are helping to maintain the integrity of scholarly work. Our research can empower educators to identify instances of plagiarism and educate students on proper citation and originality. Ultimately, our work can foster a culture of academic honesty and ensure that knowledge is built upon trustworthy foundations.

Plagiarism checker is a valuable tool for promoting academic integrity and ensuring originality in writing. By detecting instances of plagiarism, it helps maintain the highest standards of ethics and credibility.

Remember to use it responsibly and encourage others to do the same.

Future Scope

No end for plagiarism checker, but research is ongoing and can often provide still more substantial improvements. It would be better if there is a still more easier way for the user to compare the document with a vast database of sources.

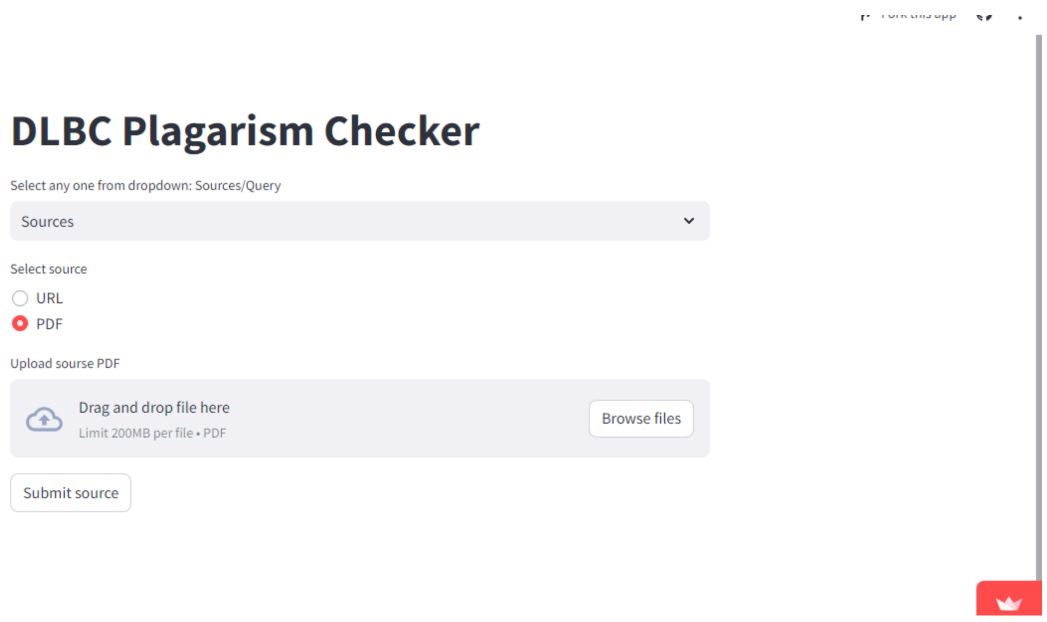
Not comparing only through the limited features but also including all the possible features for comparision.

References

1. https://streamlit.io/
2. https://www.nltk.org/
3. https://huggingface.co/sentence-transformers
4. https://www.geeksforgeeks.org/cosine-similarity/

Appendix

A – Input/Output Screens



DLBC Plagarism Checker

Select any one from dropdown: Sources/Query

Sources

Select source

URL [✓ PDF]

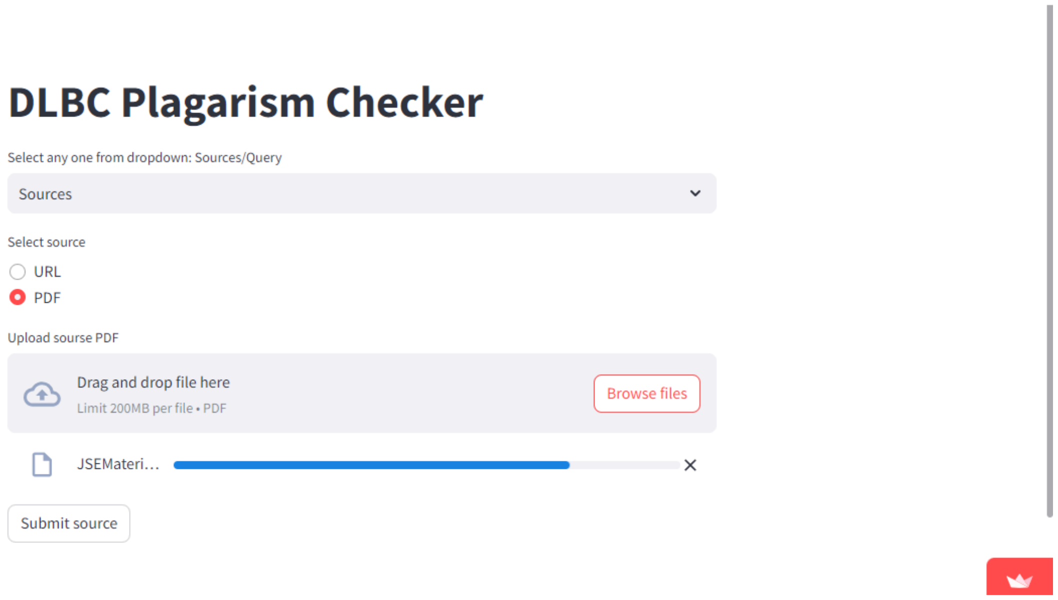
Upload sourse PDF

Drag and drop file here

Limit 200MB per file • PDF

Browse files

Submit source



DLBC Plagarism Checker

Select any one from dropdown: Sources/Query

Sources

Select source

URL [✓ PDF]

Upload sourse PDF

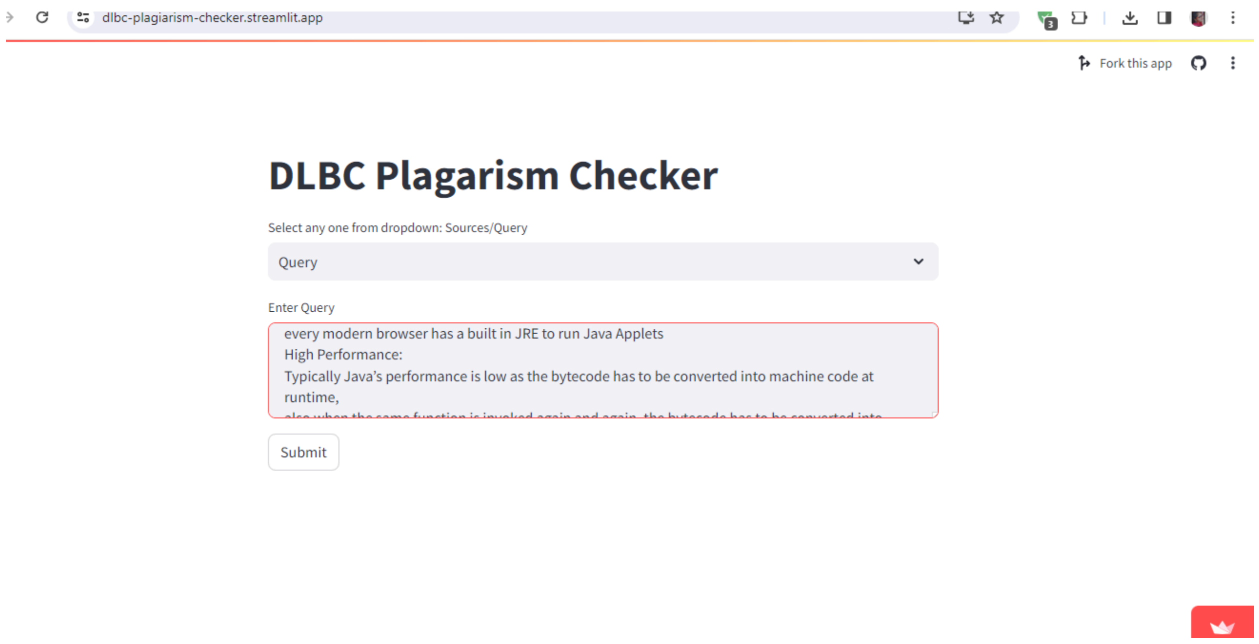
Drag and drop file here

Limit 200MB per file • PDF

Browse files

JSEMateri…

Submit source



dlbc-plagiarism-checker.streamlit.app

Fork this app

DLBC Plagarism Checker

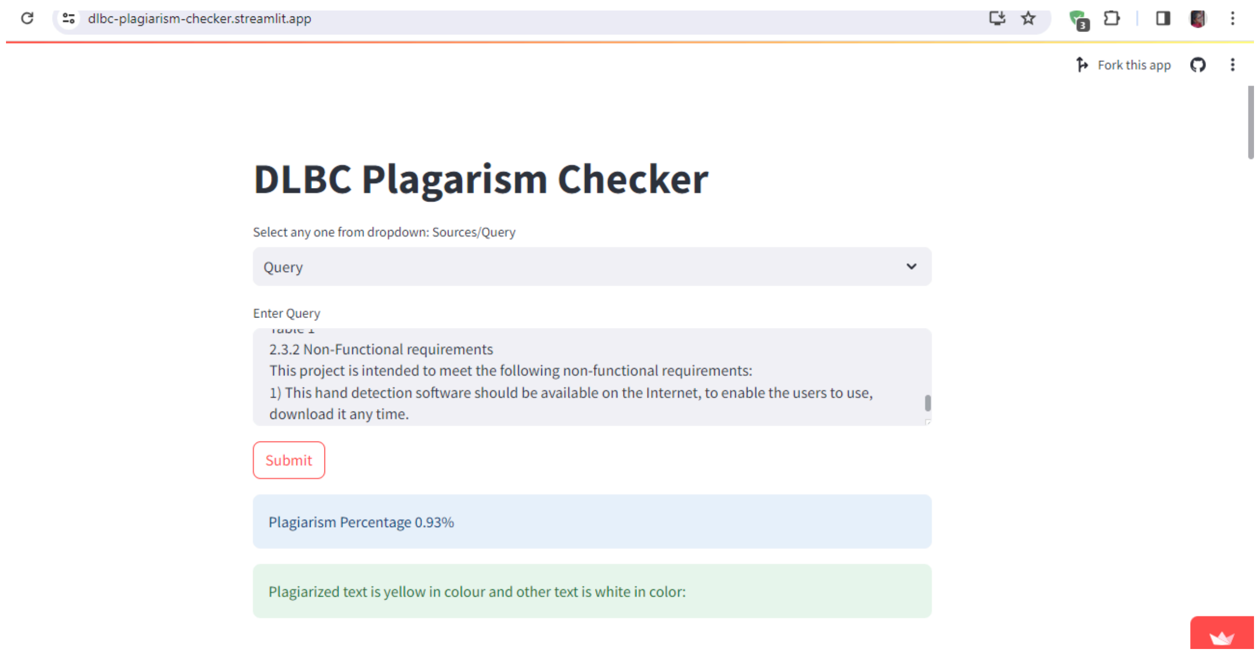
Select any one from dropdown: Sources/Query

Query

Enter Query

every modern browser has a built in JRE to run Java Applets
High Performance:
Typically Java's performance is low as the bytecode has to be converted into machine code at runtime,

Submit



dlbc-plagiarism-checker.streamlit.app

Fork this app

DLBC Plagarism Checker

Select any one from dropdown: Sources/Query

Query

Enter Query

2.3.2 Non-Functional requirements
This project is intended to meet the following non-functional requirements:
1) This hand detection software should be available on the Internet, to enable the users to use, download it any time.

Submit

Plagiarism Percentage 0.93%

Plagiarized text is yellow in colour and other text is white in color:

Sample Code

```
import sys
import subprocess
import download
from pypdf import PdfReader
import torch
import numpy as np
from sentence_transformers import SentenceTransformer, util
from nltk.tokenize import sent_tokenize
from tqdm.auto import tqdm
from sklearn.feature_extraction.text import TfidfVectorizer
import streamlit as st
import os
from pathlib import Path
import json
import pickle
import string
import re
import nltk
DIRNAME = Path(os.getcwd())

if not os.path.exists(DIRNAME/'nltk_data'):
    os.mkdir(DIRNAME/'nltk_data')
    nltk.download('punkt', DIRNAME/'nltk_data')
nltk.data.path.append((DIRNAME/'nltk_data').__str__())

if not os.path.exists(DIRNAME/'sources'):
    os.mkdir(DIRNAME/'sources')

if not os.path.exists(DIRNAME/'meta.json'):
    data = {
        'processed_sources':[],
        'source_sentences':0,
        'source_lengths':[],
        'source_indices':[]
    }
    with open(DIRNAME/'meta.json','w') as f:
        json.dump(data,f)
```

```
# load embedding model
embedding_model = SentenceTransformer((DIRNAME/ 'st_model').__str__())

st.title("DLBC Plagarism Checker")
# load source data
def extract_pdf(filepath):
    # a pdf file that is locally available in system
    try:
        book = PdfReader(filepath)
        total = ''
        for page_num in tqdm(range(book.pages.__len__())):
            extracted = book.pages[page_num].extract_text()
            total += '/n'+extracted
        return total
    except Exception as e:
        print(filepath)
        print('Error:',e)
        return ''

def extract_txt(filepath):
    # a txt file that is locally available
    try:
        with open(filepath, 'r') as f:
            extracted = f.read()
        return extracted
    except Exception as e:
        print(filepath)
        print('Error:',e)
        return ''

def download_file(file_url):
    # download pdf/txt file with proper url
    try:
        filepath = DIRNAME /'sources'/ file_url.split('/')[-1]
        print(filepath)
        download.download(file_url, filepath)
        return filepath
    except Exception as e:
        print(filepath)
        print('Error:',e)
```

```
def add_source_embedding():
    source_files = os.listdir(DIRNAME/'sources')
    meta_json = json.load(open(DIRNAME/'meta.json','r'))
    sources = []
    for file in source_files:
        if file not in meta_json['processed_sources']:
            meta_json['processed_sources'].append(file)
            print('Processing:',file)
            if file.endswith('.pdf'):
                sources.append(extract_pdf(DIRNAME/'sources'/file))
            elif file.endswith('.txt'):
                sources.append(extract_txt(DIRNAME/'sources'/file))
    if sources:
        # text
        source_lengths = meta_json['source_lengths']
        source_indices = meta_json['source_indices']
        source_sentences = []
        base_length = len(source_lengths)
        # sentence tokenize documents
        for doc_id, doc in enumerate(sources):
            sentences = sent_tokenize(doc)
            source_lengths.append(len(sentences))
            for sent_id, sentence in enumerate(sentences):
                source_indices.append((doc_id+base_length, sent_id))
                source_sentences.append(sentence)
        meta_json['source_sentences'] = len(source_indices)
        meta_json['source_indices'] = source_indices

        sources_embedding_list = []
        BATCH_SIZE = 10
        for i in tqdm(range(1+len(source_sentences)//BATCH_SIZE)):
            tmp = source_sentences[i*BATCH_SIZE:(i+1)*BATCH_SIZE]
            if tmp:
                with torch.no_grad():
                    emb = embedding_model.encode(tmp)
                    sources_embedding_list.append(emb)
        sources_embedding = np.vstack(sources_embedding_list)
        if os.path.exists(DIRNAME/'source_sentences.pkl'):
            with open(DIRNAME/'source_sentences.pkl','rb') as f:
                tmp_sents = pickle.load(f)
            source_sentences = tmp_sents + source_sentences
        with open(DIRNAME/'source_sentences.pkl','wb') as f:
                pickle.dump(source_sentences, f)
        if os.path.exists(DIRNAME/'source_embedding.npy'):
            tmp_array = np.load(DIRNAME/'source_embedding.npy')
            sources_embedding = np.vstack((tmp_array,sources_embedding))
        np.save(DIRNAME/'source_embedding.npy',sources_embedding)
        print('embeddings updated in file')
        with open(DIRNAME/'meta.json','w') as f:
            json.dump(meta_json,f)
def most_similar(query_embedding, source_embedding, topk=50):
    scores = util.dot_score(query_embedding, source_embedding).max(dim=0).values
    print('dot score calculation completed')
    top_indices = torch.argsort(scores,descending=True)[:topk]
    print('top indices selected')
    return top_indices

def preprocess_sentences(sentences):
    maketrans = str.maketrans('','',string.punctuation)
    preprocessed = []
    for s in sentences:
        # convert to lowercase
        s = s.lower()
        # remove punctuations
        s = s.translate(maketrans)
        # remove unncessary spaces, newline characters
        s = re.sub('[\t\n\s]+',' ',s).strip()

        preprocessed.append(s)
    return preprocessed



def highlight_indices(index, sentence):
    grams = [vocab[ii] for ii in index]
    results = []
    for gram in grams:
        loc = sentence.lower().find(gram.lower())
        if loc>=0:
            results.append((loc, loc+len(gram)))
    results.sort()
    consolidated = []
    start, stop = results[0]
    for res in results[1:]:
        aa,bb = res
        if aa<=stop:
            stop = bb
        else:
            consolidated.append((start,stop))
            start,stop = res
    if (start,stop) not in consolidated:
        consolidated.append((start,stop))
    return consolidated


# load query

# query = """In the laboratory we like to take things apart, look under the hood,
# poke and prod, hook up our diagnostic tools and check out what
# is really going on. Today, we're investigating JavaScript's type
# system and we've found a little diagnostic tool called typeof to
# examine variables. Put your lab coat and safety goggles on, and
# come on in and join us."""

def colored_markdown(text):
    print('color',text)
    return f'<span style="color:yellow; font-size: 16px;">{text}</span>'
def normal_markdown(text):
    print('normal',text)
    return f'<span style="color:white; font-size: 16px;">{text}</span>'



select  =st.selectbox("Select any one from dropdown: Sources/Query",["Sources","Query"])
if select == "Query":
     query = st.text_area("Enter Query")
    button = st.button("Submit")
    if button:
        try:
              source_embedding = np.load(DIRNAME/'source_embedding.npy')
```

```
  with open(DIRNAME/'source_sentences.pkl','rb') as f:
    source_sentences = pickle.load(f)
print('source embedding',source_embedding.shape)
print('source sentences',len(source_sentences))
meta_json = json.load(open(DIRNAME/'meta.json','r'))

query_sentences = sent_tokenize(query)
print(len(query_sentences))
query_embedding_list = []
BATCH_SIZE = 20
for i in tqdm(range(1+len(query_sentences)//BATCH_SIZE)):
    tmp =query_sentences[i*BATCH_SIZE:(i+1):BATCH_SIZE]
    if tmp:
        with torch.no_grad():
            emb = embedding_model.encode(tmp)
            query_embedding_list.append(emb)
query_embedding = np.vstack(query_embedding_list)
print('query embedding',query_embedding.shape)
idx = most_similar(query_embedding, source_embedding,topk=50)
source_lengths = meta_json['source_lengths']
source_indices = meta_json['source_indices']
#print('top indices',idx)
print('source indices',len(source_indices))
topk_sentences = np.array(source_sentences)[idx]
topk_indices = np.array(source_indices)[idx]


query_processed = preprocess_sentences(query_sentences)
source_processed = preprocess_sentences(topk_sentences)

# ngram modeling
ngram_vectorizer = TfidfVectorizer(ngram_range=(3,3),)

source_vec = ngram_vectorizer.fit_transform(source_processed)
vocab = {v:k for k,v in ngram_vectorizer.vocabulary_.items()}



  query_vec = ngram_vectorizer.transform(query_processed)

  colored_query = []
```

```
            total_length = 0
            colored_length = 0
            for qv,sent in zip(query_vec,query_processed):
                ind = qv.indices
                total_length+=len(sent)
                if ind.any():
                    print(ind)
                    print([vocab[ii] for ii in ind])
                    cons = highlight_indices(ind,sent)
                    print(cons)
                    current = 0
                    for aa,bb in sorted(cons):
                        colored_length += bb-aa
                        if aa>current:
                            # add normal string
                            colored_query.append(normal_markdown(sent[current:aa]))
                        # add highlighted string
                        colored_query.append(colored_markdown(sent[aa:bb]))
                        current = bb
                    if current < len(sent):
                        # add normal string
                        colored_query.append(normal_markdown(sent[current:]))

                else:
                    # add normal string
                    colored_query.append(normal_markdown(sent))
            percent = f'Plagiarism Percentage {round(colored_length/total_length*100,2)}%'
            st.info(percent)
            st.success('Plagiarized text is yellow in colour and other text is white in color:')
            for out in colored_query:
                st.markdown(f'<p>{out}</p>',unsafe_allow_html=True)
        except Exception as e:
            st.success(e)
else:
    input_type = st.radio("Select source",["URL","PDF"])
    if input_type == "URL":
        url = st.text_input("Enter source URL")
        button = st.button("Submit source")
        if button:
            file_name = download_file(url)
        st.info(f"File downloaded...")
        add_source_embedding()
else:
    file = st.file_uploader("Upload sourse PDF",type= ["pdf"])
    button = st.button("Submit source")
    if button:
        with open(DIRNAME/"sources"/file.name,'wb') as f:
            f.write(file.getbuffer())
        st.success(f"file saved...")
        add_source_embedding()
```