Real-time/Field-Based Research Project Report On PlagCheck: A Plagiarism Detection Tool

A dissertation submitted to the Jawaharlal Nehru Technological University, Hyderabad
in partial fulfilment of the requirement for the award of a degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by

Shaik Faizan Ahmed (23B81A05L3)

AVS Mohan Kumar (23B81A05M6)

Banala Vikas Rao (23B81A05R0)

Department of Computer Science and Engineering

CVR COLLEGE OF ENGINEERING

(An UGC Autonomous Institution, Affiliated to JNTUH, Accredited by NBA, and NAAC)
Vastunagar, Mangalpalli (V), Ibrahimpatnam (M), Ranga Reddy (Dist.) - 501510,
Telangana State.

2024-25

CVR COLLEGE OF ENGINEERING

(An UGC Autonomous Institution, Affiliated to JNTUH,
Accredited by NBA, and NAAC)
Vastunagar, Mangalpalli (V), Ibrahimpatnam (M),
Ranga Reddy (Dist.) - 501510, Telangana State.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project work entitled **"PlagCheck: A plagiarism detection tool"** is being submitted by Shaik Faizan Ahmed (23B81A05L3), AVS Mohan Kumar (23B81A05M6), Banala Vikas Rao (23B81A05R0) in partial fulfilment of the requirement for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering,** during the academic year 2024-2025.

Signature of the Section Coordinator **Signature of the HoD**

DECLARATION

We hereby declare that this project report titled **"PlagCheck: A Plagiarism Detection Tool"** submitted to the Department of Computer Science and Engineering, CVR College of Engineering, is a record of original work done by us. The information and data given in the report is authentic to the best of our knowledge. This Real Time/Field-Based Research Project report is not submitted to any other university or institution for the award of any degree or diploma or published at any time before.

Shaik Faizan Ahmed – 23B81A05L3
AVS Mohan Kumar – 23B81A05M6
Banala Vikas Rao – 23B81A05R0

Date:

Place:

<u>**ABSTRACT**</u>

Plagiarism is a growing concern in academic, research, and professional fields, as it undermines originality and intellectual integrity. To combat this issue, various plagiarism detection tools have been developed, utilizing techniques such as text comparison, keyword matching, and machine learning-based similarity analysis. These tools help users identify duplicated content by comparing it with existing sources, including online databases and published materials. However, many traditional detection systems have limitations, such as restricted access to external sources, high costs, and complex interfaces that make them difficult to use.

PlagCheck: A Plagiarism Detection Tool overcomes these limitations by providing an efficient, free, and user-friendly solution for detecting textual similarities. Unlike conventional tools, PlagCheck performs real-time file comparison for uploaded documents, calculating similarity scores without relying on external databases. It leverages Natural Language Processing (NLP) and cosine similarity analysis to ensure precise detection by directly comparing multiple files for similarities. Additionally, it offers an interactive graphical representation of similarity scores, allowing users to interpret results more easily.

PlagCheck is built using Flask, NLTK, Scikit-learn, and Plotly, ensuring a seamless user experience. The system allows users to upload files in .txt, .pdf, and .docx formats, or paste text directly for analysis. It processes these inputs using NLP to extract and compare sentences, calculates similarity using cosine similarity, and provides visual results such as heatmaps and similarity tables. The platform also features a **Quick Access** option for viewing previous plagiarism checks, streamlining workflow and offering convenience to users.

This project provides an accessible, effective, and automated way to verify content originality, making it ideal for students, educators, and professionals who require quick, accurate results without complex interfaces or costly services.

<u>TABLE OF CONTENTS</u>

CHAPTER 1

INTRODUCTION

1.1 Motivation

Plagiarism has become a widespread issue in the digital age, where vast amounts of information are easily accessible. With just a few clicks, individuals can copy content from various sources without proper attribution, leading to ethical concerns in research, journalism, and content creation. The availability of online materials has made it easier for people to replicate and reuse content without acknowledging the original authors, undermining creativity and intellectual honesty.

In academic settings, plagiarism is a serious problem, as students and researchers sometimes copy work from existing sources rather than producing original content. Many students resort to copying assignments, research papers, or reports from the internet to save time or achieve better grades. This not only hampers their learning process but also affects the credibility of educational institutions. Traditional plagiarism detection methods, such as manual checking by educators, are time-consuming and inefficient, making it difficult to ensure academic integrity.

Detecting plagiarism manually is challenging, especially with the increasing volume of digital content. Many existing plagiarism detection tools have limitations, such as requiring subscriptions, providing inaccurate results, or lacking support for multiple file formats. These challenges motivated us to develop PlagCheck: A Plagiarism Detection Tool, a system designed to efficiently detect plagiarism by analyzing text using Natural Language Processing (NLP) and advanced similarity algorithms. Unlike conventional tools, PlagCheck directly compares uploaded files and checks for similarity without relying on external sources. It utilizes NLP techniques and cosine similarity analysis for accurate detection, providing real-time results in a user-friendly interface.

Our tool aims to provide a reliable, user-friendly, and effective solution for detecting plagiarized content across various formats, including .txt, .pdf, and .docx files. By leveraging Flask for seamless web interaction, PlagCheck offers a more accessible and scalable approach than traditional plagiarism checkers. The system's design makes it ideal for both individual users and educational institutions seeking a more efficient and automated way to maintain academic integrity.

1.2 Problem Statement

Plagiarism is a growing issue in academia, research, and content creation, where individuals often copy text from online sources without proper attribution. The increasing ease of accessing and duplicating digital content has made it difficult to ensure originality, leading to ethical concerns and academic dishonesty. Educational institutions, publishers, and organizations struggle to identify plagiarized content effectively, as manual detection is time-consuming and prone to errors.

Existing plagiarism detection tools often come with significant limitations. Many require costly subscriptions, offer restricted access to online sources, or produce inaccurate similarity results. Additionally, some tools lack support for multiple file formats, making it difficult for users to check documents in various formats like TXT, PDF, and DOCX. The absence of a reliable, accessible, and accurate plagiarism detection system makes it challenging for individuals and institutions to maintain content originality.

To address these challenges, we propose **PlagCheck: A Plagiarism Detection Tool**, which leverages **Natural Language Processing (NLP)** and **advanced similarity algorithms** to effectively detect plagiarism. The tool is designed to compare text from multiple uploaded documents, analyze their similarity using cosine similarity and other NLP techniques, and generate a detailed plagiarism report. Unlike existing systems, **PlagCheck** does not rely on external databases for plagiarism checks, instead focusing on direct content comparison, which ensures a more accurate and real-time analysis.

By offering a **Flask-based web interface**, **multi-format support**, and a user-friendly design, **PlagCheck** aims to enhance the accuracy, efficiency, and accessibility of plagiarism detection. This system makes it easier for both individuals and institutions to ensure content originality and academic integrity.

1.3 Project Objectives

The primary objective of **PlagCheck: A Plagiarism Detection Tool** is to develop an efficient, accessible, and user-friendly system for identifying plagiarized content. The tool is designed to help students, educators, researchers, and content creators ensure originality by analyzing text similarity between uploaded documents and online sources.

The key objectives of this project are:

1. **To develop an effective plagiarism detection system** that analyzes text using **Natural Language Processing (NLP)** and similarity algorithms, such as cosine similarity, to accurately detect plagiarism.
2. **To support multiple input formats**, including TXT, PDF, and DOCX, enabling users to check documents in different file types seamlessly.
3. **To compare text with online sources** and retrieve relevant content for analysis. By leveraging search engine integration, the system identifies potential matches across the web in real-time.
4. **To generate detailed plagiarism reports** that highlight matching sources, similarity percentages, and instances of copied content. This includes presenting results in an easy-to-understand format for users.
5. **To provide a user-friendly web interface** using **Flask**, offering an interactive experience that is accessible on multiple devices. The interface will facilitate document uploads, result display, and access to past plagiarism checks.
6. **To visualize plagiarism results effectively**, through graphical representations like heatmaps, similarity graphs, and charts, helping users interpret similarity scores and understand content overlaps more intuitively.

By achieving these objectives, **PlagCheck** aims to provide a reliable, efficient, and accessible solution for plagiarism detection, enhancing content integrity and promoting academic and professional honesty.

1.4 Project Report Organization

This report is structured to provide a detailed understanding of **PlagCheck: A Plagiarism Detection Tool**, covering its motivation, problem statement, methodology, implementation, and results. The report is divided into the following chapters:

- **Chapter 1 – Introduction**: This chapter introduces the issue of plagiarism and the need for plagiarism detection tools. It discusses the motivation behind this project, the problem statement, project objectives, and an overview of the report structure.
- **Chapter 2 – Literature Review**: This chapter examines existing plagiarism detection tools and techniques, highlighting their strengths and limitations. It provides a comparative analysis to justify the need for **PlagCheck**, exploring gaps in current solutions and the technologies that can be leveraged for better performance.
- **Chapter 3 – Requirement Analysis**: This chapter outlines the software, hardware, and user requirements necessary for the successful development and deployment of **PlagCheck**, including any dependencies and technologies used (such as **Flask** and NLP-based algorithms).
- **Chapter 4 – System Design**: This chapter details the proposed system architecture, including the backend structure with **Flask**, the methods and algorithms used (like cosine similarity), and different system design models, such as Class Diagrams, Use Case Diagrams, Activity Diagrams, and Sequence Diagrams. It also describes the datasets and technology stack used for development, including libraries like **NLTK**, **Scikit-learn**, and **Plotly**.
- **Chapter 5 – Implementation**: This chapter provides an overview of the actual development process of **PlagCheck**, including screenshots of the user interface, detailed descriptions of the result generation process, testing procedures, and validation of the system's performance. It also discusses how the user interface was created using **Flask** to ensure smooth interaction.
- **Chapter 6 – Conclusions**: This chapter summarizes the findings of the project, presents the conclusions, and discusses the future scope for enhancements, such as integrating AI-based detection techniques or expanding the system's database of sources. It also highlights potential improvements to the user experience and new features for future releases.
- **References**: This section lists all the sources, research papers, and materials referenced throughout the report.
- **Appendix**: This section includes any additional materials, such as published papers, source code, or data used in the development and testing of the tool.

CHAPTER 2

LITERATURE REVIEW

2.1 Existing Work

Plagiarism detection has been an essential area of research due to the increasing availability of digital content and the ease with which it can be copied. Over the years, various tools and techniques have been developed to identify textual plagiarism through methods such as string matching, fingerprinting, and Natural Language Processing (NLP)-based similarity analysis.
Several widely used plagiarism detection tools include:

- **Turnitin**: A well-known academic plagiarism detection tool that compares submitted text against an extensive proprietary database of student papers, academic journals, and internet sources. While effective, Turnitin is subscription-based and often inaccessible to individual users due to high licensing costs.
- **Grammarly Plagiarism Checker**: Integrated into Grammarly’s writing assistant, this tool checks documents against online web pages. However, it lacks access to scholarly databases and does not offer deep semantic comparison, making it less effective for academic purposes.
- **Copyscape**: Designed primarily for web content plagiarism detection, Copyscape is widely used by content creators and website managers. It scans publicly available websites but is not suitable for academic or technical document comparison.
- **Plagscan**: This tool allows for content comparison against both online sources and internal document repositories. While it supports multiple formats, its effectiveness is limited by the size and comprehensiveness of its reference database.

While each of these tools provides valuable services, they suffer from certain limitations:

- Restricted or no access to academic papers or proprietary databases.
- High subscription costs that limit accessibility for students or independent researchers.
- Limited accuracy in detecting paraphrased or AI-generated content.
- Poor support for comparing multiple uploaded documents directly.

These limitations point to the need for a more accessible and accurate tool like **PlagCheck**. It leverages NLP and cosine similarity to detect plagiarism in **TXT, PDF, and DOCX** files without relying on expensive databases. Instead, it compares files directly and presents results through intuitive visual reports, making it efficient and affordable for academic and professional use.

2.2 Limitations of Existing Work

Despite the availability of plagiarism detection tools, existing systems face several challenges that limit their effectiveness. Some of the key limitations include:

1. **Inability to Detect Paraphrased Content** – Many tools rely on exact text matching, making them ineffective against paraphrased or rewritten content that retains the original meaning but uses different wording.
2. **Limited Database Access** – Tools like Turnitin and Grammarly compare text only against their own databases, excluding private academic papers, paid research journals, and offline sources, leading to incomplete plagiarism detection.
3. **False Positives and Improper Citation Handling** – Some tools flag common phrases, properly cited references, or legally reused content as plagiarism, leading to inaccurate results.
4. **Inefficiency in Detecting AI-Generated and Hybrid Plagiarism** – With the rise of AI-based content generation, existing tools struggle to detect contextually similar but syntactically different content, making them less effective against modern plagiarism techniques.
5. **Inability to Handle Obfuscated Source Code** – Code plagiarism detection tools like MOSS and JPlag fail to identify structurally similar but syntactically modified code, allowing plagiarized programs to bypass detection.
6. **High Cost and Accessibility Issues** – Many advanced plagiarism detection tools require paid subscriptions, making them inaccessible to students, researchers, and small organizations.

Due to these limitations, there is a need for a more efficient, accessible, and context-aware plagiarism detection system, which PlagCheck aims to provide.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 Software Requirements

The PlagCheck plagiarism detection tool requires the following software components for its development and execution:

- **Programming Language:** Python
- **Framework:** Flask or any other lightweight Python web framework (depending on your final choice)
- **Libraries and Dependencies:**
    - **NLTK** – For text preprocessing and tokenization
    - **BeautifulSoup** – For web scraping
    - **Requests** – For fetching web content
    - **SerpAPI** – For Google Search API integration
    - **Scikit-learn** – For cosine similarity calculations
    - **Pandas** – For text data handling and manipulation
    - **Plotly / Seaborn / Matplotlib** – For data visualization (e.g., heatmaps of similarity)
    - **docx2txt** and **PyPDF2** – For extracting text from .docx and .pdf files
- **Frontend Technologies (if applicable):** HTML, CSS, and optionally JavaScript (for interactive elements)
- **Development Environment:** Visual Studio Code or any Python-compatible IDE
- **Operating System:** Windows, macOS, or Linux

3.2 Hardware Requirements

To ensure smooth execution, **PlagCheck** requires the following minimum hardware specifications:

- **Processor** – Intel Core i3 (or equivalent) and above
- **RAM** – Minimum 4GB (8GB recommended for better performance)
- **Storage** – Minimum 500MB of free disk space
- **Internet Connection** – Required for online plagiarism detection (Google Search API)
- **GPU (Optional)** – Not necessary, but can improve performance in large-scale processing

3.3 User Requirements

The PlagCheck tool is designed to be intuitive and accessible for a wide range of users, including students, educators, and professionals. The following user requirements apply:

- **Basic Computer Skills:** Users should be comfortable with uploading documents and reading generated plagiarism reports.
- **Internet Access:** An active internet connection is required for performing real-time web searches through SerpAPI.
- **Supported File Formats:** Users should upload content in one of the supported formats: .txt, .pdf, or .docx.
- **Web-Based Interface:** The tool operates via a simple browser-based interface, requiring no software installation or advanced technical knowledge.

CHAPTER 4

SYSTEM DESIGN

4.1 Proposed System Architecture

The PlagCheck plagiarism detection tool follows a modular and layered architecture designed for efficient processing, real-time similarity analysis, and clear result visualization. The system comprises the following key components:

1. User Interface Layer

- **Developed using a custom web framework** (e.g., Flask or Django) to provide an intuitive and responsive frontend.
- Enables users to:
    - Upload documents in .txt, .pdf, or .docx formats, or manually enter text.
    - Initiate the plagiarism detection process with a single click.
    - View detailed results and visualizations through an organized web dashboard.

2. Preprocessing Layer

- Extracts text from uploaded files using docx2txt and PyPDF2.
- Splits and tokenizes content using NLTK.
- Cleans and normalizes text for consistency and better accuracy in detection.

3. Plagiarism Detection Layer

- Sends queries to the **Google Search API** via **SerpAPI** to find online content matching sentence-level input.
- Retrieves and scrapes content from matched webpages using BeautifulSoup.
- Calculates textual similarity using CountVectorizer and **cosine similarity** from Scikit-learn.
- Flags potential plagiarism based on similarity thresholds.

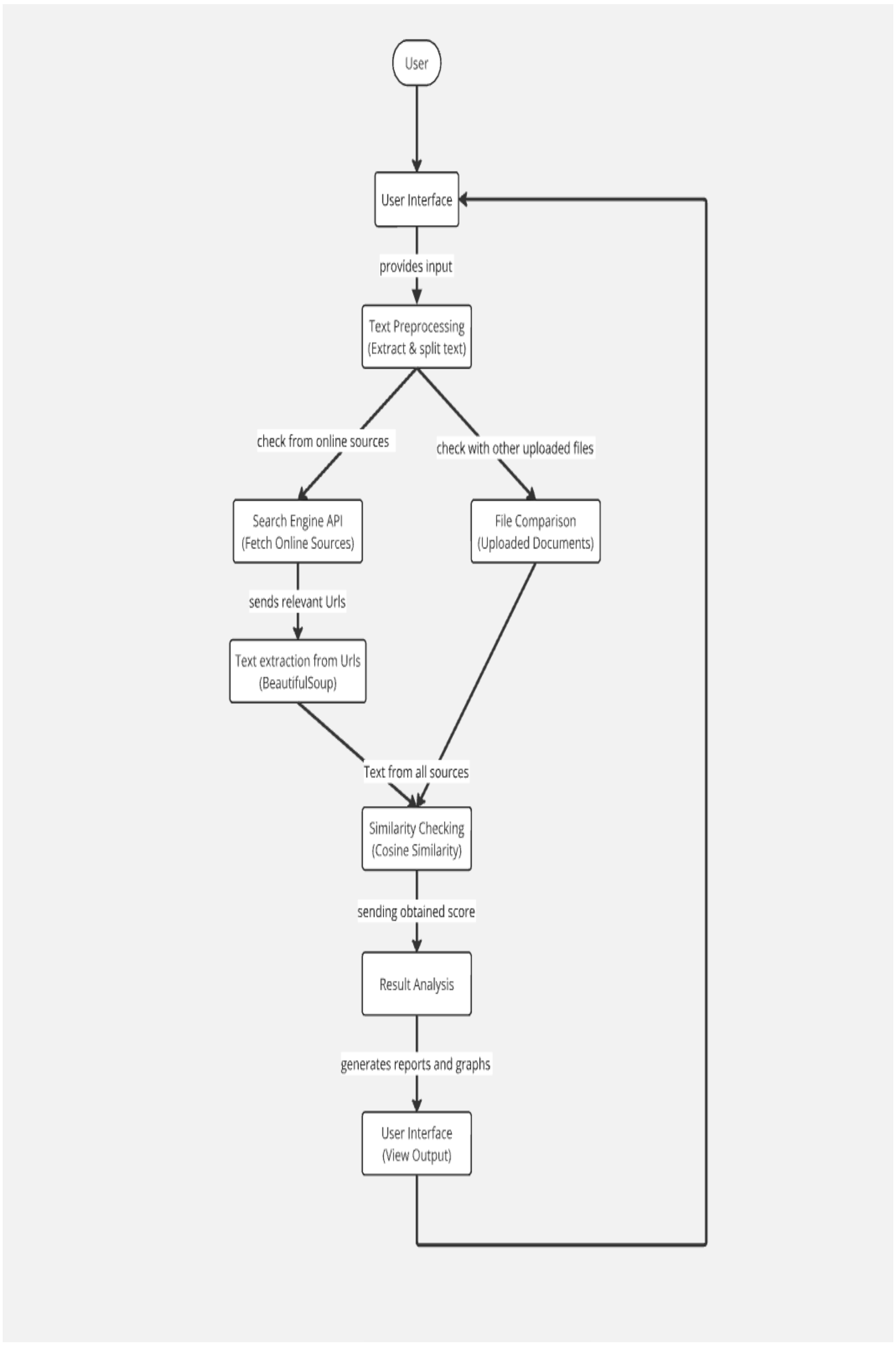
4. Analysis & Visualization Layer

- Highlights sentence-level similarity scores.
- Generates interactive visualizations (e.g., heatmaps, bar charts) using **Plotly**, enabling users to quickly identify plagiarized sections.
- Displays match confidence and overlap with online content.

5. Output & Report Generation Layer

- Presents results in a dynamic, filterable table with clickable source links.
- Summarizes plagiarism statistics for the entire document.
- *Future Enhancement:* Export/download functionality for complete plagiarism reports.

4.1.1 Architecture Diagram


User
User Interface
provides input
Text Preprocessing (Extract & split text)
check from online sources
check with other uploaded files
Search Engine API (Fetch Online Sources)
File Comparison (Uploaded Documents)
sends relevant Urls
Text extraction from Urls (BeautifulSoup)
Text from all sources
Similarity Checking (Cosine Similarity)
sending obtained score
Result Analysis
generates reports and graphs
User Interface (View Output)

4.2 Proposed Methods or Algorithms

PlagCheck: A Plagiarism Detection Tool employs a combination of **Natural Language Processing (NLP)**, **Web Scraping**, and **Machine Learning** techniques to effectively detect plagiarism. The key methods and algorithms used in the system are:

1. Text Preprocessing and Tokenization

- **Library Used:** NLTK (Natural Language Toolkit)
- **Method:**
    - Extracts text from uploaded files (e.g., .txt, .pdf, .docx) using extraction tools (docx2txt, PyPDF2).
    - Splits the extracted text into individual sentences using NLTK's sentence tokenizer.
    - Cleanses the text by removing unnecessary spaces, special characters, and formatting inconsistencies to improve text comparison accuracy.

2. Web Search-Based Plagiarism Detection

- **Library Used:** SerpAPI (Google Search API), BeautifulSoup
- **Method:**
    - Each sentence from the input text is queried via the Google Search API (SerpAPI) to find similar content online.
    - The top result URLs for each sentence are retrieved.
    - The content from these top URLs is scraped using **BeautifulSoup** for further analysis.

3. Text Similarity Measurement Using Cosine Similarity

- **Library Used:** Scikit-learn (CountVectorizer, Cosine Similarity)
- **Algorithm:** Cosine Similarity
- **Method:**
    - The extracted text from the user's document and the scraped online content are converted into word vectors using **CountVectorizer** from Scikit-learn.
    - **Cosine similarity** is then calculated to measure the degree of similarity between the input text and the content found online.
    - If the calculated similarity score exceeds a predefined threshold, the content is flagged as potential plagiarism.

4. Visualization of Plagiarism Results

- **Library Used:** Plotly
- **Method:**
    - The results are presented in interactive visual graphs, including:
        - **Scatter Plot:** Displays the similarity percentage for different sentences in the document.
        - **Bar Chart:** Highlights the most plagiarized sections, making it easy to spot problematic areas.
        - **Line Graph:** Shows trends in plagiarism detection across sentences, allowing users to analyze content overlap over the course of the document.

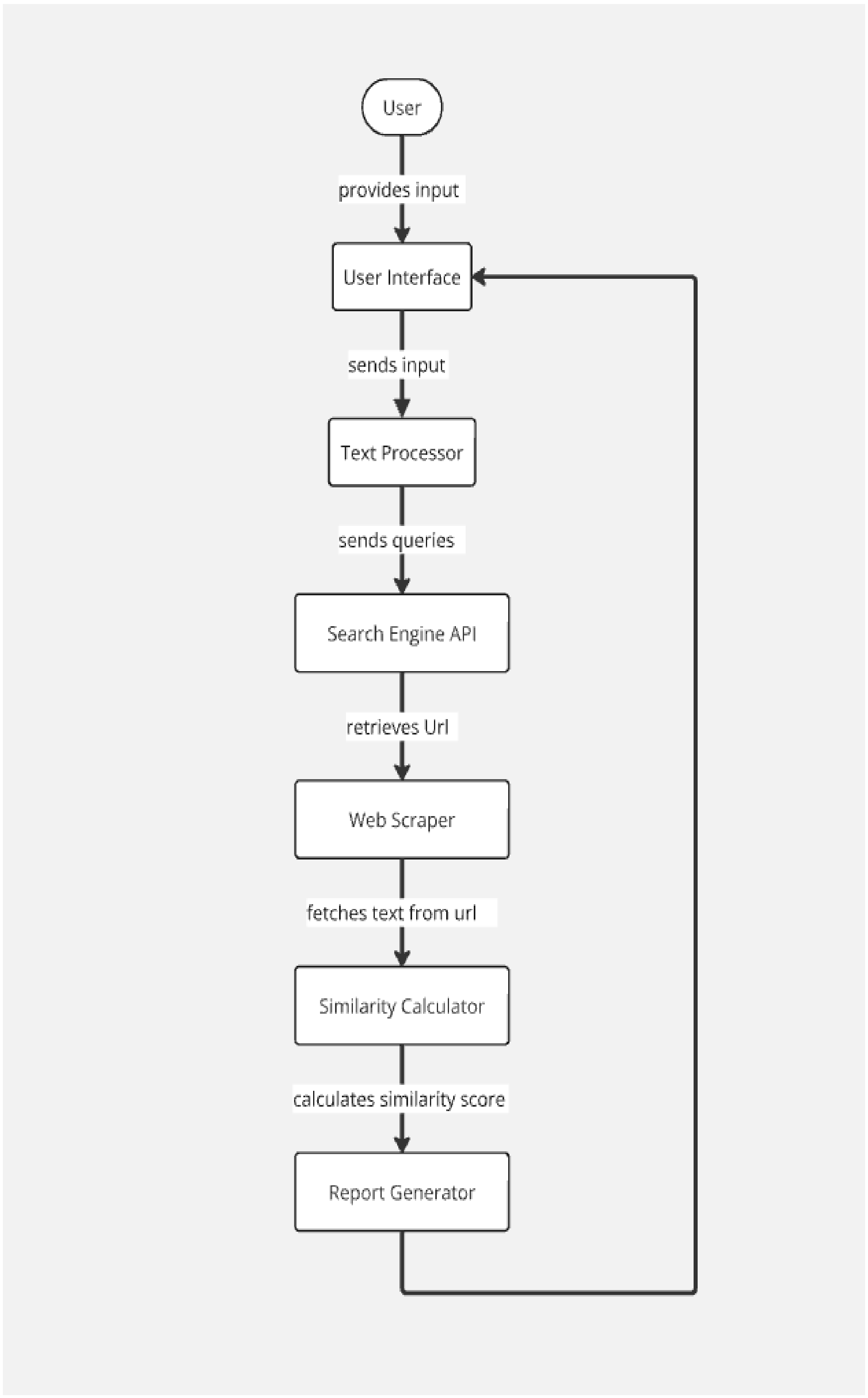
5. Plagiarism Report Generation

- **Method:**
    - The results are presented in a **table format**, with clickable URLs linking to the detected sources, allowing users to view the original content.
    - *Future Enhancement:* The system will include an option to **download plagiarism reports**, allowing users to keep a record for reference.

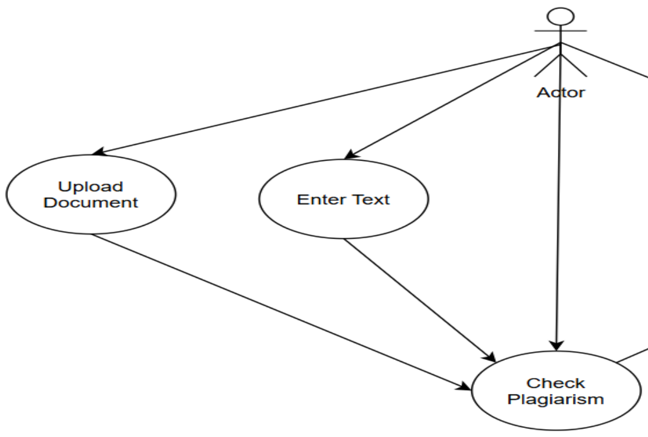
4.3 Class / Use Case / Activity / Sequence Diagrams

In this section, we describe the various diagrams used to represent the **PlagCheck: A Plagiarism Detection Tool** system. These diagrams provide a visual representation of the system's structure, functionalities, and interactions between different components.

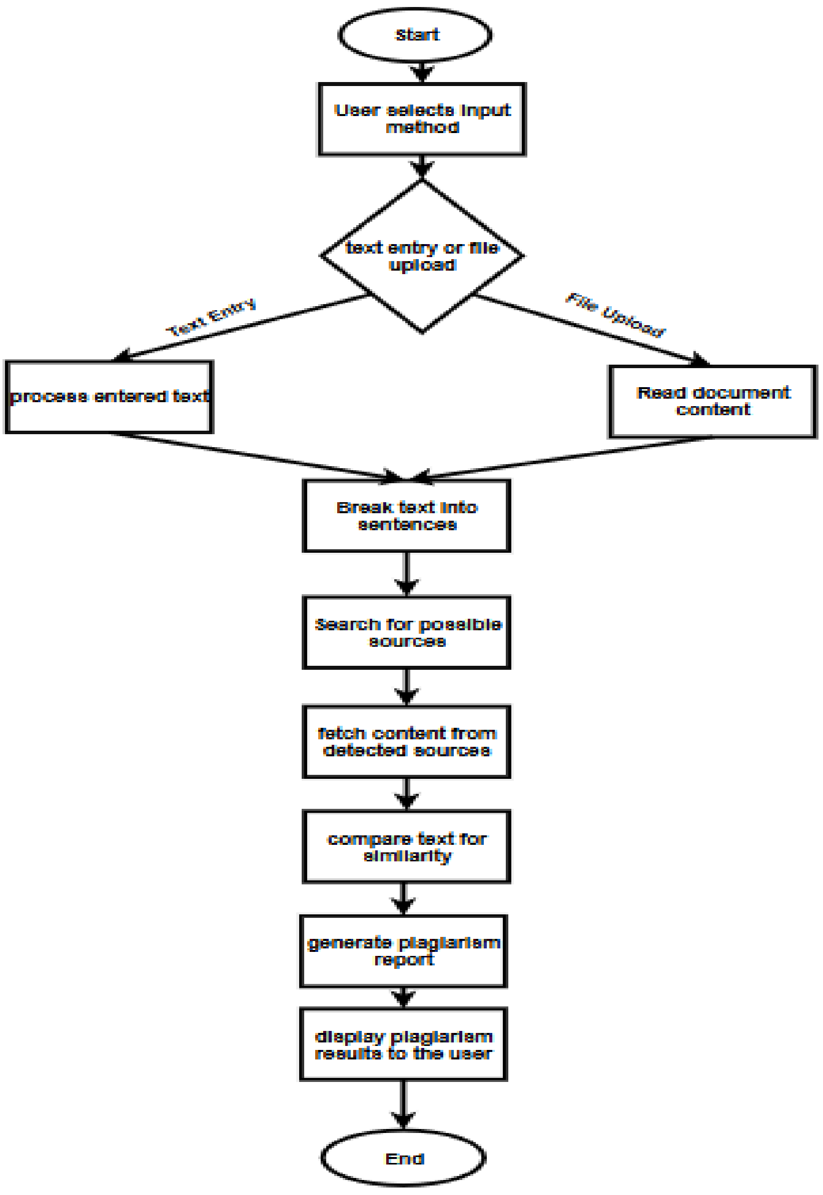
- **Class Diagram**: Represents the structure of the system, showing the relationships between different classes such as PlagiarismChecker, TextProcessor, SearchEngineAPI, and SimilarityCalculator.



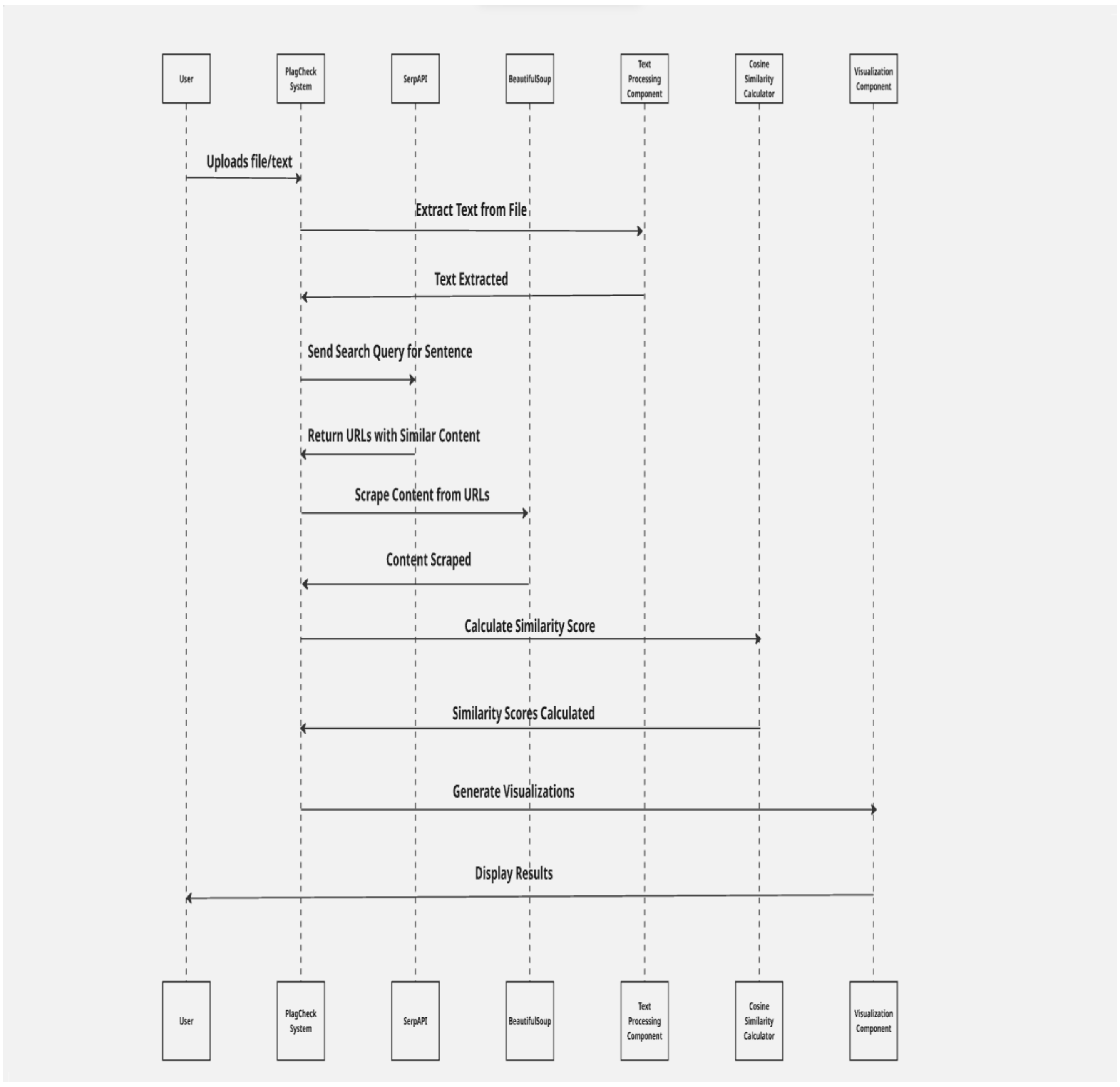
- **Use Case Diagram:** Illustrates the interactions between the user and the system, highlighting key functionalities such as text input, file upload, plagiarism detection, and similarity report generation.



- **Activity Diagram:** Represents the step-by-step workflow, from **text input** to **result generation**, ensuring clarity in process execution.


Start
User selects input method
text entry or file upload
Text Entry
File Upload
process entered text
Read document content
Break text into sentences
Search for possible sources
fetch content from detected sources
compare text for similarity
generate plagiarism report
display plagiarism results to the user
End

- **Sequence Diagram**: Shows the step-by-step interaction between different system components, including how the system fetches URLs, extracts text, and calculates similarity.



4.4 Datasets and Technology Stack

4.4.1 Datasets

PlagCheck leverages multiple data sources to enable accurate plagiarism detection, comparison, and reporting. The primary datasets used include:

- **Uploaded Document Dataset:**
  Stores text extracted from user-uploaded files (.txt, .pdf, .docx), used as input for plagiarism analysis.
- **Online Web Content Dataset:**
  Consists of data scraped from web pages returned by Google Search (via SerpAPI), used for sentence-level comparison with input text.
- **Similarity Scores Dataset:**
  Maintains computed cosine similarity scores between input sentences and matched web content, helping identify potentially plagiarized sections.
- **User Interaction Logs:**
  Tracks user actions such as file uploads, plagiarism checks performed, and report generation history, aiding in improving user experience.
- **Visualization and Reporting Dataset:**
  Stores results prepared for graphical representations and tabular report generation, including URLs of matched content and plagiarism percentages.

4.4.2 Technology Stack

PlagCheck is built using a robust and scalable technology stack that supports natural language processing, web scraping, and interactive visualizations. The stack includes:

- **Frontend:**
  HTML5, CSS3 (Tailwind CSS), JavaScript – for responsive and visually appealing UI.
- **Backend:**
  Flask (Python-based web framework) – handles routing, processing logic, and interaction between modules.
- **File Handling & Text Extraction:**
  docx2txt, PyPDF2 – for extracting raw text from uploaded .docx and .pdf files.

- **NLP & Text Processing:**
  NLTK – used for sentence tokenization and text preprocessing.
  Scikit-learn – used for vectorization (CountVectorizer) and cosine similarity computation.
- **Web Search & Scraping:**
  SerpAPI – for querying Google Search programmatically.
  BeautifulSoup – for extracting textual content from web pages.
- **Visualization:**
  Plotly – generates interactive scatter plots, bar charts, and line graphs for plagiarism analysis.
- **Authentication & Security:**
  Flask-Login or Flask-Security (optional) – for user authentication and session management (in future upgrades).
- **Database (optional/future scope):**
  SQLite or MongoDB – for storing user history, reports, and interaction logs.
- **Version Control & Deployment:**
  Git, GitHub – for version control.
  Docker (optional) – for containerized deployment.
  Deployment via PythonAnywhere, Heroku, or AWS (based on scalability needs).

CHAPTER 5

IMPLEMENTATION

PlagCheck uses a modular architecture for text extraction, similarity analysis, and visualization. It supports file-to-file and web-based comparison with a Flask backend and Tailwind CSS frontend.

System Architecture Overview

The implementation is divided into the following key modules:

1. **Frontend Interface:**
   - Built using HTML, JavaScript, and Tailwind CSS.
   - Provides user-friendly forms for:
     - Direct text input.
     - Single file upload (.txt, .pdf, .docx).
     - Multiple file comparison (similarity check).
   - Includes interactive modals for help, instructions, and plagiarism history ("Quick Access").
   - Displays output in tabular and graphical formats.
2. **Backend Server (Flask):**
   - Acts as the bridge between frontend and processing logic.
   - Handles user requests and routes them to the appropriate detection method.
   - Provides APIs for similarity checking, file uploads, and report generation.
3. **File & Text Processing:**
   - Extracts and cleans raw text from uploaded files using docx2txt and PyPDF2.
   - Utilizes NLTK for:
     - Sentence tokenization.
     - Preprocessing (removing special characters, stop words, etc.).
4. **Web-Based Plagiarism Detection:**
   - Integrates SerpAPI to fetch Google search results for each sentence.
   - Scrapes top URLs using BeautifulSoup to retrieve relevant textual content.
   - Compares user content with scraped content using cosine similarity.
5. **Similarity Checking (File-to-File):**
   - Enables users to upload multiple files.
   - Compares content pairwise using CountVectorizer and cosine similarity from Scikit-learn.

    - Results are rendered as a heatmap and similarity matrix.
6. **Visualization Module:**
    - Implements interactive charts via Plotly:
        - **Bar Chart:** Shows highly plagiarized sentences.
        - **Line Graph:** Displays plagiarism trends across the document.
        - **Heatmap:** Visualizes similarity across multiple documents.
7. **Plagiarism Report:**
    - Generates detailed tabular reports with:
        - Sentence-level similarity scores.
        - Highlighted plagiarized content.
        - Clickable source URLs.
    - Plans for downloadable PDF report feature (future enhancement).

5.1 Code Implementation

The implementation of the **PlagCheck: A Plagiarism Detection Tool** is divided into multiple modular components to ensure clarity, maintainability, and scalability. Each module is responsible for handling specific tasks such as text extraction, preprocessing, plagiarism detection, similarity measurement, and result visualization. The application is developed using Python with supporting libraries and tools like Flask, Scikit-learn, NLTK, Plotly, SerpAPI, and BeautifulSoup.

5.1.1 Text Extraction and Preprocessing

```
def extract_text(file):
    if file.filename.endswith(".pdf"):
        reader = PyPDF2.PdfReader(file)
        return " ".join(page.extract_text() for page in reader.pages)
    elif file.filename.endswith(".docx"):
        return docx2txt.process(file)
    elif file.filename.endswith(".txt"):
        return file.read().decode("utf-8")
    return ""
```

5.1.2 Sentence Tokenization and Cleaning

```
def preprocess_text(text):
    sentences = nltk.sent_tokenize(text)
    cleaned_sentences = [re.sub(r'\s+', ' ', s.strip()) for s in sentences]
    return cleaned_sentences
```

5.1.3 Web Search-Based Plagiarism Detection (Using SerpAPI)

```
def fetch_search_results(sentence):
    params = {
        "engine": "google",
        "q": sentence,
        "api_key": SERPAPI_KEY
    }
    search = GoogleSearch(params)
    results = search.get_dict()
    return [res['link'] for res in results.get("organic_results", [])[:3]]
```

5.1.4 Similarity Measurement Using Cosine Similarity

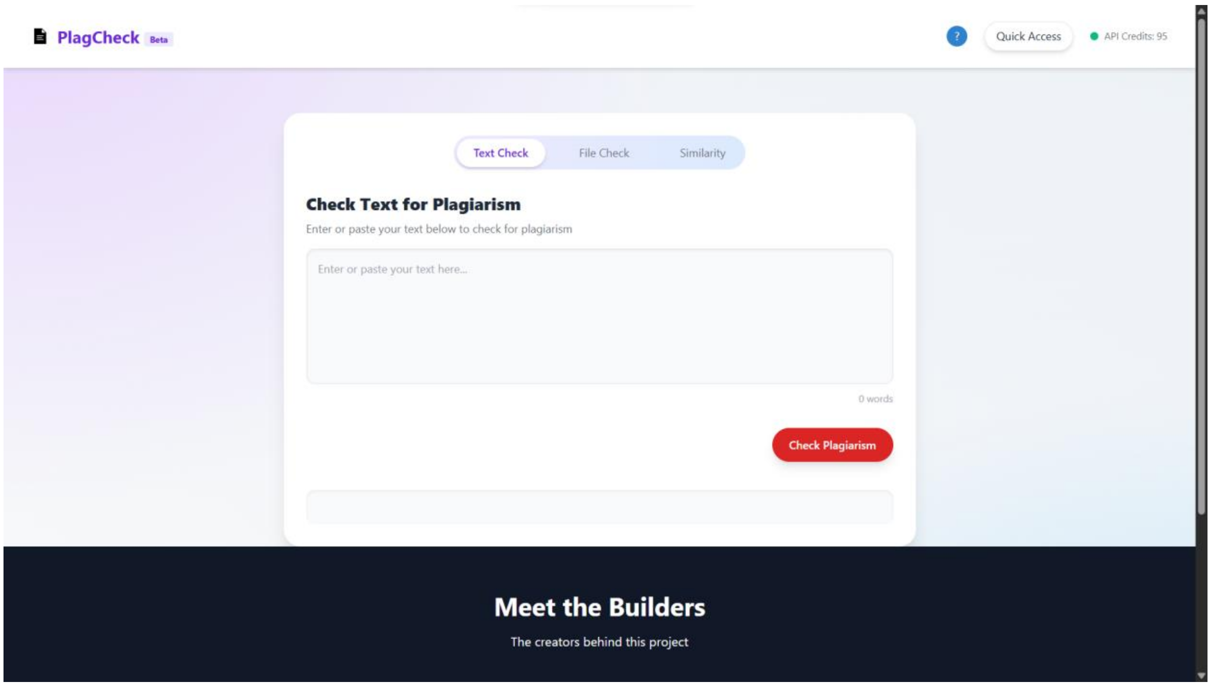
```
def calculate_similarity(input_text, web_texts):
    all_texts = [input_text] + web_texts
    vectorizer = CountVectorizer().fit_transform(all_texts)
    similarity_matrix = cosine_similarity(vectorizer)
    return max(similarity_matrix[0][1:])  # Ignore self-similarity
```

5.1.5 Visualization of Results

```
def plot_similarity_graph(similarity_scores):
    fig = go.Figure(data=go.Scatter(
        x=list(range(1, len(similarity_scores)+1)),
        y=similarity_scores,
        mode='lines+markers',
        name='Similarity'
    ))
    fig.update_layout(title='Sentence-wise Similarity Score',
                      xaxis_title='Sentence Number',
                      yaxis_title='Similarity Score')
    fig.show()
```

5.2 Testing

Initial Look



Check Text for Plagiarism

Enter or paste your text below to check for plagiarism

Enter or paste your text here...

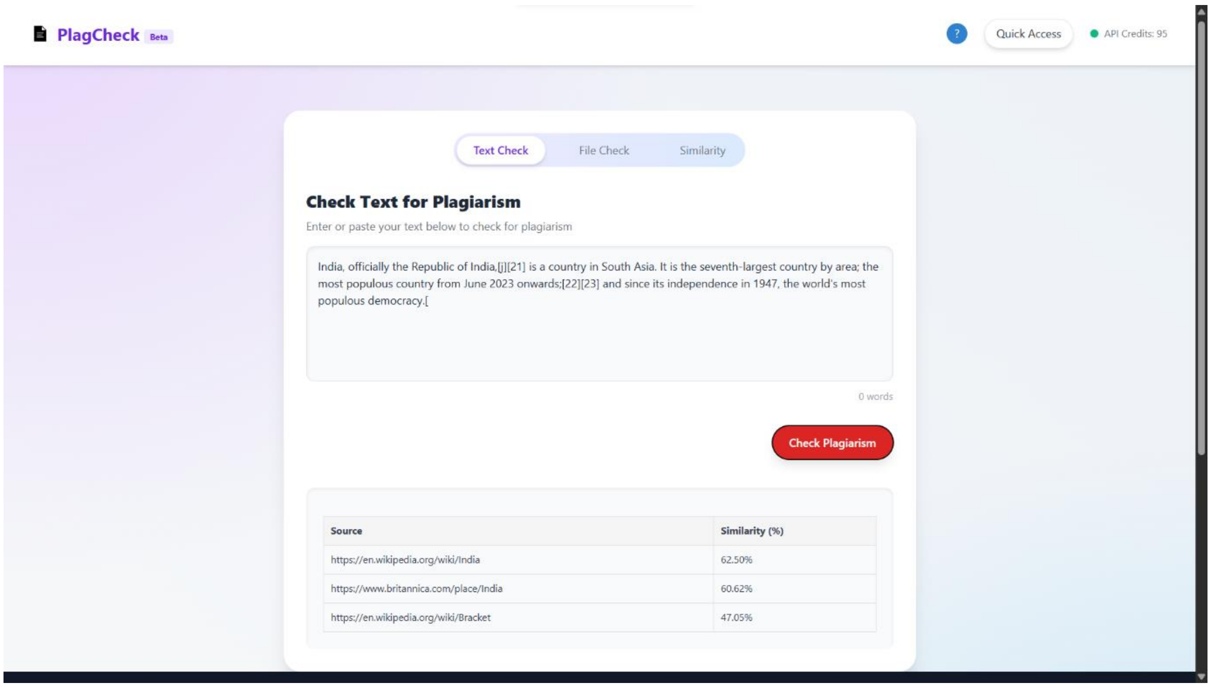
0 words

Check Plagiarism

Meet the Builders

The creators behind this project

After User performed Text Check



Check Text for Plagiarism

Enter or paste your text below to check for plagiarism

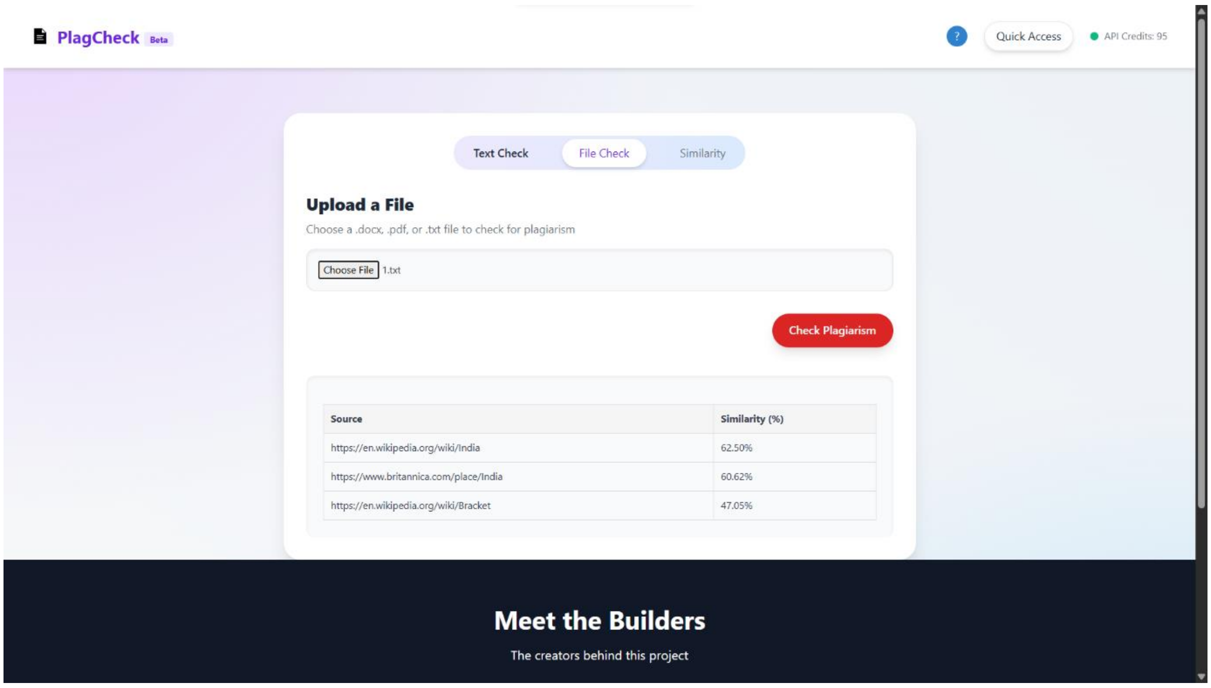
India, officially the Republic of India,[j][21] is a country in South Asia. It is the seventh-largest country by area; the most populous country from June 2023 onwards;[22][23] and since its independence in 1947, the world's most populous democracy.[

0 words

Check Plagiarism

Source	Similarity (%)
https://en.wikipedia.org/wiki/India	62.50%
https://www.britannica.com/place/India	60.62%
https://en.wikipedia.org/wiki/Bracket	47.05%

After User performed File Check



Upload a File

Choose a .docx, .pdf, or .txt file to check for plagiarism

Choose File 1.txt

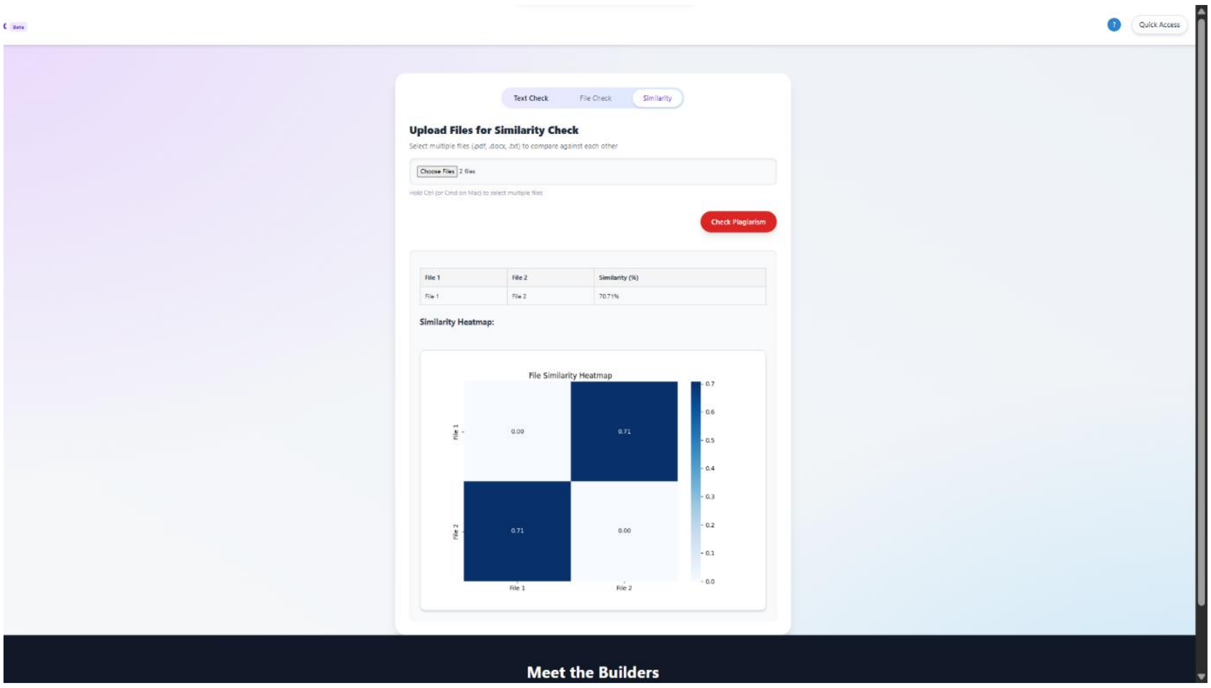
Check Plagiarism

Source	Similarity (%)
https://en.wikipedia.org/wiki/India	62.50%
https://www.britannica.com/place/India	60.62%
https://en.wikipedia.org/wiki/Bracket	47.05%

Meet the Builders

The creators behind this project

After user performed similarity check



Upload Files for Similarity Check

Select multiple files (.pdf, .docx, .txt) to compare against each other

Choose Files 2 files

Hold Ctrl (or Cmd on Mac) to select multiple files

Check Plagiarism

File 1	File 2	Similarity (%)
File 1	File 2	70.71%

Similarity Heatmap:

Meet the Builders

CHAPTER 6

CONCLUSION

6.1 Conclusion

PlagCheck addresses growing issue of plagiarism in academic and professional settings by offering a comprehensive and user-friendly platform for plagiarism detection and content comparison. By integrating Natural Language Processing (NLP), web scraping via SerpAPI, and machine learning-based similarity analysis, the tool enhances content originality checks for students, educators, and researchers.

The project has demonstrated its ability to accurately detect potential plagiarism, compare multiple documents, and provide insightful visualizations for in-depth analysis. Its clean user interface, modular backend built with Flask, and real-time processing features make it a practical and scalable solution for academic integrity enforcement.

Through rigorous testing and validation, PlagCheck has proven effective in meeting its core objectives of promoting original content, improving academic transparency, and aiding in plagiarism prevention. The platform represents a proactive step toward fostering ethical writing practices and safeguarding intellectual property in the digital age.

6.2 Future Scope

PlagCheck holds significant potential for enhancement and broader application in the coming phases of development. Key areas for future improvements include:

1. **Downloadable Plagiarism Reports**: Introducing the functionality to generate and download detailed plagiarism reports for record-keeping and academic submissions.
2. **Multilingual Plagiarism Detection**: Expanding the system to support multiple languages, enabling cross-language plagiarism detection and wider global usability.
3. **Improved Machine Learning Models**: Integrating more advanced models like BERT or other transformers to improve semantic similarity detection.
4. **Mobile App Integration**: Developing a mobile-friendly version or dedicated mobile application for on-the-go plagiarism detection and quick access to past checks.
5. **Institutional Collaboration**: Partnering with educational institutions to integrate PlagCheck into academic workflows, including LMS platforms and grading systems.

6. **Real-Time Plagiarism Alerts**: Implementing background plagiarism monitoring with real-time alerts for educational content creators, bloggers, and researchers.
7. **Version History and Change Tracking**: Adding a version comparison tool that shows how documents evolve and whether edits reduce or increase similarity scores.
8. **Secure Cloud-Based Storage**: Offering encrypted cloud storage for past plagiarism reports, enabling users to manage and revisit their content history securely.

REFERENCES

Books

[1] Ranjan, J. (2020). *Artificial Intelligence in Research: A Guide to Smart Research and Publication*. Springer.

[2] Silver, D. (2018). *Machine Learning for Research and Development*. MIT Press.

[3] Holt, D.H. (1997). *Management Principles and Practices*. Prentice-Hall.

Articles

[1] Naik, Ramesh R., Maheshkumar B. Landge, and C. Namrata Mahender. "A review on plagiarism detection tools." *International Journal of Computer Applications* 125.11 (2015): 16-22.

[2] Hage, J., Rademaker, P. and Van Vugt, N., 2010. A comparison of plagiarism detection tools. *Utrecht University. Utrecht, The Netherlands*, *28*(1).

[3] Foltýnek, T., Meuschke, N. and Gipp, B., 2019. Academic plagiarism detection: a systematic literature review. *ACM Computing Surveys (CSUR)*, *52*(6), pp.1-42.

Websites

[1] Google Scholar: https://scholar.google.com/
[2] GitHub: https://github.com/
[3] SerpAPI: https://serpapi.com/
[4] NLTK Documentation: https://www.nltk.org/
[5] Scikit-learn: https://scikit-learn.org/
[6] Plotly: https://plotly.com/
[7] Python PyPI: https://pypi.org/
[8] Flask Documentation: https://flask.palletsprojects.com/

APPENDIX:(SOURCE CODE)

app.py

```
from flask import Flask, render_template, request, jsonify
from werkzeug.utils import secure_filename
import os, requests, json
import matplotlib; matplotlib.use('Agg')

from file_processing import read_file
from text_processing import get_sentences
from plagiarism_checker import check_plagiarism, get_similarity_between_files
from visualization import plot_similarity_graphs, plot_similarity_heatmap

app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
HISTORY_FILE = 'history.json'
API_KEY =
"452ad616874c729414e8e83a112a68c8c20c4bcd927b5b9f6db24ed45dbc37d1"

def get_api_credits():
    try:
        data =
requests.get(f"https://serpapi.com/account.json?api_key={API_KEY}").json()
        return data.get("remaining_searches", 0)
    except Exception as e:
        print(f"Error: {e}")
        return None

def load_history():
    if os.path.exists(HISTORY_FILE):
        with open(HISTORY_FILE, 'r') as f:
            try: return json.load(f)
            except json.JSONDecodeError: return []
    return []

def save_history(history):
    with open(HISTORY_FILE, 'w') as f:
        json.dump(history, f)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/check', methods=['POST'])
def check_plagiarism_route():
    option = request.form.get('option')
    history = load_history()
    try:
        if option == 'text':
            text = request.form.get('text')
            if not text: return jsonify({"error": "No text provided."}), 400
            df = check_plagiarism(get_sentences(text), text)
            if df.empty: return jsonify({"data": "No plagiarism found."})
            result = df.to_dict(orient="records")
            history.append({'type': 'text', 'result': result})
            save_history(history)
            return jsonify({"data": result})

        elif option == 'file':
            uploaded_file = request.files.get('file')
            if not uploaded_file or not uploaded_file.filename:
                return jsonify({"error": "No file uploaded."}), 400
            filename = secure_filename(uploaded_file.filename)
            filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            uploaded_file.save(filepath)
            text = read_file(filepath)
            if not text: return jsonify({"error": "Could not extract text."}), 400
            df = check_plagiarism(get_sentences(text), text)
            if df.empty: return jsonify({"data": "No plagiarism found."})
            result = df.to_dict(orient="records")
            history.append({'type': 'file', 'filename': filename, 'result': result})
            save_history(history)
            return jsonify({"data": result})

        elif option == 'similarity':
            uploaded_files = request.files.getlist('files')
            saved_files = []
            for file in uploaded_files:
                if file and file.filename:
                    filename = secure_filename(file.filename)
                    filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
                    file.save(filepath)
                    saved_files.append(filepath)
            if not saved_files:
                return jsonify({"error": "No valid files uploaded."}), 400
            files = [read_file(f) for f in saved_files]
            similarity_list = get_similarity_between_files(files)
            heatmap_path = plot_similarity_heatmap(similarity_list, len(files))
            history.append({'type': 'similarity', 'files': [f.filename for f in uploaded_files],
'similarity': similarity_list})
            save_history(history)
            return jsonify({"data": similarity_list, "heatmap": heatmap_path})

        return jsonify({"error": "Invalid option selected."}), 400
    except Exception as e:
        return jsonify({"error": f"An error occurred: {str(e)}"}), 500

@app.route('/history', methods=['GET'])
```

```
def get_history():
    return jsonify({"history": load_history()})

@app.route('/api/credits', methods=['GET'])
def api_credits():
    credits = get_api_credits()
    return jsonify({"credits": credits}) if credits is not None else jsonify({"error": "Unable
to fetch credits."}), 500

if __name__ == '__main__':
    app.run(debug=True, host="0.0.0.0", port=int(os.environ.get("PORT", 5000)))
```

file_processing.py

```
import docx2txt
from PyPDF2 import PdfReader
import os

def read_file(file):
    """Reads content from a file path or uploaded file and returns it as text."""
    content = ""

    if file is None:
        return content

    try:
        # If file is a string (i.e., file path)
        if isinstance(file, str):
            ext = os.path.splitext(file)[1].lower()

            if ext == '.txt':
                with open(file, 'r', encoding='utf-8') as f:
                    content = f.read()

            elif ext == '.pdf':
                with open(file, 'rb') as f:
                    pdf_reader = PdfReader(f)
                    for page in pdf_reader.pages:
                        text = page.extract_text()
                        if text:
                            content += text

            elif ext == '.docx':
                content = docx2txt.process(file)

        # If file is an uploaded FileStorage object (e.g., from Flask)
        else:
            if file.mimetype == "text/plain":
                content = file.read().decode("utf-8")

            elif file.mimetype == "application/pdf":
```

```
            pdf_reader = PdfReader(file)
            for page in pdf_reader.pages:
                text = page.extract_text()
                if text:
                    content += text

        elif file.mimetype == "application/vnd.openxmlformats-
officedocument.wordprocessingml.document":
            content = docx2txt.process(file)

    except Exception as e:
        print(f"Error reading file: {e}")

    return content
```

plagiarism_checker.py

```
import requests
import random
import pandas as pd
from bs4 import BeautifulSoup
from serpapi import GoogleSearch
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from urllib.parse import urlparse

# Random User-Agent Headers
USER_AGENTS = [
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/74.0.3729.169 Safari/537.36',
    'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/78.0.3904.97 Safari/537.36',
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/80.0.3987.132 Safari/537.36',
]

def get_random_headers():
    """Returns a random header to make HTTP requests."""
    return {'User-Agent': random.choice(USER_AGENTS)}

def get_url(sentence):
    """Fetches the first URL result for the sentence from Google Search."""
    params = {
        "q": sentence,
        "api_key":
"452ad616874c729414e8e83a112a68c8c20c4bcd927b5b9f6db24ed45dbc37d1",
        "engine": "google",
    }
    try:
        search = GoogleSearch(params)
        results = search.get_dict()
        if 'organic_results' in results:
            return results['organic_results'][0].get('link', None)
    except Exception as e:
        print(f"Error fetching URL: {e}")
    return None

def get_text_from_url(url):
    """Fetches and returns the textual content from a URL."""
    try:
        if any(domain in url for domain in ["facebook.com", "instagram.com"]):
            print(f"Skipping unsupported URL: {url}")
            return ""
        response = requests.get(url, headers=get_random_headers(), timeout=10)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, 'html.parser')
        return ' '.join([p.text for p in soup.find_all('p')])
    except requests.exceptions.RequestException as e:
        print(f"Failed to fetch from {url}: {e}")
        return ""

def get_similarity(text1, text2):
    """Calculates cosine similarity between two texts."""
    if not text1 or not text2:
        return 0.0
    vectorizer = CountVectorizer().fit_transform([text1, text2])
    return cosine_similarity(vectorizer[0:1], vectorizer[1:2])[0][0]

def get_similarity_between_files(file_texts):
    """Calculates cosine similarity between multiple file texts."""
    similarity_list = []
    for i in range(len(file_texts)):
        for j in range(i + 1, len(file_texts)):
            similarity = get_similarity(file_texts[i], file_texts[j])
            similarity_list.append({
                'File 1': f"File {i + 1}",
                'File 2': f"File {j + 1}",
                'Similarity': similarity
            })
    return similarity_list

def check_plagiarism(sentences, text):
    """Checks for plagiarism in given sentences."""
    similarity_list = []
    urls = []

    for sentence in sentences:
        url = get_url(sentence)
        urls.append(url)

    for idx, url in enumerate(urls):
        if url:
            try:
                text_from_url = get_text_from_url(url)
                if text_from_url.strip():  # Avoid comparing with empty content
                    similarity = get_similarity(text, text_from_url)
                    similarity_list.append({
                        'Sentence': sentences[idx],
                        'Source': url,
                        'Similarity': similarity
                    })
            except Exception as e:
                print(f"Error processing URL at index {idx}: {e}")

    df = pd.DataFrame(similarity_list).sort_values(by='Similarity',
ascending=False).reset_index(drop=True)

    # Clean display: Show "Source1", "Source2", etc. with links
    df['Source'] = df['Source'].apply(
    lambda x: x if x else 'No URL found'
)
    return df
```

text_processing.py

```
import nltk
from nltk.tokenize import sent_tokenize

# Make sure necessary data is available
nltk.download('punkt')

def get_sentences(text):
    """Splits text into sentences."""
    if not text:
        return []
    return sent_tokenize(text)
```

visualization.py

```
import matplotlib.pyplot as plt
import seaborn as sns
import io
import base64
import pandas as pd
import plotly.express as px

def plot_similarity_heatmap(similarity_list, file_count):
    """Generates a base64-encoded heatmap image of file similarities."""
    matrix = [[0] * file_count for _ in range(file_count)]
    for sim in similarity_list:
        i = int(sim['File 1'].split()[-1]) - 1
        j = int(sim['File 2'].split()[-1]) - 1
        matrix[i][j] = sim['Similarity']
        matrix[j][i] = sim['Similarity']

    df = pd.DataFrame(matrix,
                      columns=[f'File {i+1}' for i in range(file_count)],
                      index=[f'File {i+1}' for i in range(file_count)])

    plt.figure(figsize=(8, 6))
    sns.heatmap(df, annot=True, cmap="Blues", fmt=".2f")
    plt.title("File Similarity Heatmap")

    buf = io.BytesIO()
    plt.savefig(buf, format="png")
    buf.seek(0)
    encoded_img = base64.b64encode(buf.read()).decode('utf-8')
    plt.close()

    return encoded_img

def plot_similarity_graphs(similarity_list):
    """Displays similarity scatter, line, and bar charts using Plotly."""
    df = pd.DataFrame(similarity_list)

    fig1 = px.scatter(df, x='File 1', y='File 2', color='Similarity', title='Similarity Scatter
Plot')
    fig1.show()

    fig2 = px.line(df, x='File 1', y='Similarity', color='File 2', title='Similarity Line Chart')
    fig2.show()

    fig3 = px.bar(df, x='File 1', y='Similarity', color='File 2', title='Similarity Bar Chart')
    fig3.show()
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>PlagCheck</title>
  <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css"
rel="stylesheet">
  <style>
    body {
      background: radial-gradient(at top left, #e9d8fd 0%, transparent 40%),
                  radial-gradient(at bottom right, #bee3f8 0%, transparent 40%),
                  linear-gradient(to top right, #f8f9fa, #edf2f7);
    }

    .info-button {
      width: 30px;
      height: 30px;
      border-radius: 50%;
      background-color: #3182ce;
      color: white;
      font-size: 16px;
      display: flex;
      align-items: center;
      justify-content: center;
      cursor: pointer;
      transition: background-color 0.3s ease;
    }

    .info-button:hover {
      background-color: #2b6cb0;
    }

    .tab-content { display: none; }
    .tab-content.active { display: block; }

    table {
      width: 100%;
      margin-top: 1rem;
      border-collapse: collapse;
    }

    table th, table td {
      padding: 10px;
      border: 1px solid #e0e0e0;
      text-align: left;
    }

    table th {
      background-color: #f5f5f5;
    }

    table tr:hover {
      background-color: #f0f0f0;
    }

    /* Custom Styling */
    .history-item {
      font-size: 14px;
      margin-bottom: 8px;
    }
    .history-item strong {
      color: #4a4a4a;
    }
    .history-item em {
      color: #999;
    }
  </style>
```

```
</head>

<body class="min-h-screen font-sans text-gray-800">

  <!-- Usage Modal -->
  <div id="usageModal" class="fixed inset-0 bg-black bg-opacity-50 z-50 flex items-
center justify-center hidden">
    <div class="bg-white p-6 rounded-2xl shadow-xl w-11/12 max-w-lg relative">
      <button id="closeModal" class="absolute top-2 right-2 text-gray-500 hover:text-red-
500 text-2xl">&times;</button>
      <h2 class="text-2xl font-bold text-purple-700 mb-4">How to Use PlagCheck</h2>
      <ul class="list-disc list-inside text-gray-700 space-y-2 text-sm">
        <li><strong>Text Check:</strong> Paste your content in the text box and click
"Check Plagiarism".</li>
        <li><strong>File Check:</strong> Upload a single file (.pdf, .docx, or .txt) to
analyze for plagiarism.</li>
        <li><strong>Similarity:</strong> Upload multiple files to detect similarities among
them. A heatmap and similarity scores will be shown.</li>
        <li>Use the <span class="text-green-600">API Credits</span> indicator to track
usage if integrated with APIs.</li>
      </ul>
    </div>
  </div>

  <!-- Quick Access Modal -->
  <div id="quickAccessModal" class="fixed inset-0 bg-black bg-opacity-50 z-50 flex
items-center justify-center hidden">
    <div class="bg-white p-6 rounded-2xl shadow-xl w-11/12 max-w-lg relative">
      <button id="closeQuickAccess" class="absolute top-2 right-2 text-gray-500
hover:text-red-500 text-2xl">&times;</button>
      <h2 class="text-2xl font-bold text-purple-700 mb-4">Plagiarism Check History</h2>
      <div id="historyList" class="text-gray-700 space-y-2 text-sm max-h-64 overflow-y-
auto">
        <!-- Dynamic History Items will go here -->
      </div>
    </div>
  </div>

  <!-- Header -->
  <header class="flex items-center justify-between px-10 py-6 bg-white shadow-md">
    <div class="flex items-center space-x-3">
      <img src="https://img.icons8.com/ios-filled/50/000000/document--v1.png" class="h-
6 w-6" alt="logo" />
      <h1 class="text-2xl font-bold text-purple-700">PlagCheck
        <span class="text-xs bg-purple-100 text-purple-700 px-2 py-0.5
rounded">Beta</span>
      </h1>
    </div>

    <div class="flex items-center space-x-6">
      <button id="openModal" class="info-button">?</button>
      <button id="openQuickAccess" class="px-4 py-2 rounded-full shadow-md border border-orange-400">Quick Access</button>
      <div class="flex items-center space-x-2">
        <div class="w-3 h-3 rounded-full bg-green-500"></div>
        <span class="text-sm text-gray-500">API Credits: 95</span>
      </div>
    </div>
  </header>

  <!-- Main Section -->
  <main class="max-w-4xl mx-auto mt-16 p-8 bg-white rounded-3xl shadow-xl backdrop-blur-md">

    <div class="flex justify-center mb-8">
      <div class="flex space-x-6 bg-gradient-to-r from-purple-100 to-blue-100 p-1 rounded-full">
        <button class="tab-button bg-white text-purple-700 font-semibold py-2 px-6 rounded-full shadow-md" data-tab="text">Text Check</button>
        <button class="tab-button text-gray-500 py-2 px-6 rounded-full hover:bg-white hover:text-purple-700" data-tab="file">File Check</button>
        <button class="tab-button text-gray-500 py-2 px-6 rounded-full hover:bg-white hover:text-purple-700" data-tab="similarity">Similarity</button>
      </div>
    </div>

    <!-- Tabs Content -->
    <div id="text" class="tab-content active">
      <h2 class="text-2xl font-extrabold text-gray-800 mb-2">Check Text for Plagiarism</h2>
      <p class="text-gray-500 mb-4">Enter or paste your text below to check for plagiarism</p>
      <textarea class="w-full h-48 p-4 border border-gray-200 rounded-xl focus:outline-none focus:ring-2 focus:ring-purple-400 bg-gray-50 shadow-inner resize-none" placeholder="Enter or paste your text here..."></textarea>
      <div class="text-right text-sm text-gray-400 mt-1">0 words</div>
    </div>

    <div id="file" class="tab-content">
      <h2 class="text-2xl font-extrabold text-gray-800 mb-2">Upload a File</h2>
      <p class="text-gray-500 mb-4">Choose a .docx, .pdf, or .txt file to check for plagiarism</p>
      <input type="file" accept=".pdf,.docx,.txt" class="block w-full text-sm text-gray-700 bg-gray-50 border border-gray-200 rounded-xl p-4 shadow-inner focus:outline-none focus:ring-2 focus:ring-purple-400"/>
    </div>

    <div id="similarity" class="tab-content">
      <h2 class="text-2xl font-extrabold text-gray-800 mb-2">Upload Files for Similarity Check</h2>
      <p class="text-gray-500 mb-4">Select multiple files (.pdf, .docx, .txt) to compare against each other</p>
      <input type="file" accept=".pdf,.docx,.txt" multiple class="block w-full text-sm text-gray-700 bg-gray-50 border border-gray-200 rounded-xl p-4 shadow-inner focus:outline-none focus:ring-2 focus:ring-purple-400"/>
      <p class="text-sm text-gray-400 mt-2">Hold Ctrl (or Cmd on Mac) to select multiple files</p>
    </div>

    <!-- Form Submission -->
    <form id="plag-form" enctype="multipart/form-data">
      <input type="hidden" id="option" name="option" value="text">
      <div class="flex justify-end mt-8">
        <button type="submit" class="px-6 py-3 rounded-full shadow-lg font-semibold bg-red-600 hover:bg-red-700 text-white">Check Plagiarism</button>
      </div>
    </form>

    <!-- Results -->
    <div id="result" class="mt-10 bg-gray-50 p-6 rounded-xl shadow-inner text-sm text-gray-700 whitespace-pre-wrap"></div>

    <!-- Script -->
    <script>
      // Tab Navigation
      const tabs = document.querySelectorAll('.tab-button');
      const contents = document.querySelectorAll('.tab-content');
      const optionInput = document.getElementById('option');
      const historyList = document.getElementById('historyList');

      tabs.forEach(tab => {
        tab.addEventListener('click', () => {
          const target = tab.dataset.tab;
          optionInput.value = target;

          tabs.forEach(btn => btn.classList.remove('bg-white', 'text-purple-700', 'shadow-md'));
          tab.classList.add('bg-white', 'text-purple-700', 'shadow-md');

          contents.forEach(c => c.classList.remove('active'));
          document.getElementById(target).classList.add('active');
        });
      });

      // Word Count Functionality
      const textarea = document.querySelector('#text textarea');
      const wordCount = document.querySelector('#text + .text-right');

      textarea.addEventListener('input', () => {
        const words = textarea.value.trim().split(/\s+/).filter(Boolean);
        wordCount.textContent = `${words.length} words`;
      });
      // Form Submission Handling
      document.getElementById('plag-form').addEventListener('submit', async (e) => {
        e.preventDefault();
        const option = optionInput.value;
        const formData = new FormData();
        formData.append('option', option);

        if (option === 'text') {
          const textarea = document.querySelector('#text textarea');
          formData.append('text', textarea.value);
        } else if (option === 'file') {
          const file = document.querySelector('#file input[type="file"]').files[0];
          if (!file) return alert('Please upload a file');
          formData.append('file', file);
        } else if (option === 'similarity') {
          const files = document.querySelector('#similarity input[type="file"]').files;
          if (!files.length) return alert('Please upload at least two files');
          for (const file of files) {
            formData.append('files', file);
          }
        }

        const resultDiv = document.getElementById('result');
        resultDiv.innerHTML = 'Checking...';

        try {
          const response = await fetch('/check', { method: 'POST', body: formData });
          const data = await response.json();

          if (option === 'similarity') {
            let tableHTML = `<table><thead><tr><th>File 1</th><th>File
2</th><th>Similarity (%)</th></tr></thead><tbody>`;
            data.data.forEach(result => {
              tableHTML += `<tr><td>${result["File 1"]}</td><td>${result["File
2"]}</td><td>${(result["Similarity"] * 100).toFixed(2)}%</td></tr>`;
            });
            tableHTML += '</tbody></table>';
            if (data.heatmap) {
              tableHTML += `<h3 class="mt-6 font-bold text-lg">Similarity Heatmap:</h3>
                    <img class="mt-2 rounded-xl border shadow-md"
src="data:image/png;base64,${data.heatmap}" alt="Similarity Heatmap" />`;
            }
            resultDiv.innerHTML = tableHTML;
            addHistory('Similarity', `Compared ${data.data.length} files`, new
Date().toLocaleString());
          } else {
            if (Array.isArray(data.data)) {
              let tableHTML = `<table><thead><tr><th>Source</th><th>Similarity
(%)</th></tr></thead><tbody>`;
              data.data.forEach(result => {
```

```
            tableHTML += `<tr><td><a href="${result.Source}"
target="_blank">${result.Source}</a></td><td>${(result.Similarity *
100).toFixed(2)}%</td></tr>`;
          });
          tableHTML += '</tbody></table>';
          resultDiv.innerHTML = tableHTML;
        } else {
          resultDiv.innerHTML = data.data || 'No plagiarism found.';
        }
        addHistory(option.charAt(0).toUpperCase() + option.slice(1), data.data || 'No
plagiarism found.', new Date().toLocaleString());
      }
    } catch (err) {
      console.error(err);
      resultDiv.innerHTML = 'Error occurred. Please try again.';
    }
  });

  // History Management
  function addHistory(type, description, date) {
    const item = { type, description, date };

    // Save to DOM
    const historyItem = document.createElement('div');
    historyItem.classList.add('history-item');
    historyItem.innerHTML = `<strong>${type}</strong> - ${description}
<em>${date}</em>`;
    historyList.appendChild(historyItem);

    // Save to localStorage
    const existingHistory = JSON.parse(localStorage.getItem('plagHistory')) || [];
    existingHistory.push(item);
    localStorage.setItem('plagHistory', JSON.stringify(existingHistory));
  }

  function loadHistory() {
    const savedHistory = JSON.parse(localStorage.getItem('plagHistory')) || [];
    savedHistory.forEach(item => {
      const historyItem = document.createElement('div');
      historyItem.classList.add('history-item');
      historyItem.innerHTML = `<strong>${item.type}</strong> - ${item.description}
<em>${item.date}</em>`;
      historyList.appendChild(historyItem);
    });
  }

  loadHistory();

  // Modal Toggling for Instructions and History
  document.getElementById('openModal').addEventListener('click', () => {
    document.getElementById('usageModal').classList.remove('hidden');
```

```
      });

      document.getElementById('closeModal').addEventListener('click', () => {
        document.getElementById('usageModal').classList.add('hidden');
      });

      document.getElementById('openQuickAccess').addEventListener('click', () => {
        document.getElementById('quickAccessModal').classList.remove('hidden');
      });

      document.getElementById('closeQuickAccess').addEventListener('click', () => {
        document.getElementById('quickAccessModal').classList.add('hidden');
      });
    </script>
  </main>

  <!-- Meet the Builders Section -->
<section class="bg-gray-900 text-white py-16 px-6">
  <h2 class="text-4xl font-bold text-center mb-4">Meet the Builders</h2>
  <p class="text-center text-lg mb-12">The creators behind this project</p>
  <div class="flex flex-wrap justify-center gap-8">
    <!-- Faizan's Card -->
    <div class="bg-gray-800 rounded-2xl p-6 w-72 text-center">
      <h3 class="text-xl font-semibold mb-2">Shaik Faizan Ahmed</h3>
      <p class="text-sm">Roll No: 23B81A05L3</p>
      <p class="text-sm">2nd Year, 2nd Semester</p>
      <p class="text-sm">CSE-D, CVR College of Engineering</p>
      <p class="text-sm mt-2">Plagiarism Detection Module Developer</p>
    </div>
    <!-- Mohan's Card -->
    <div class="bg-gray-800 rounded-2xl p-6 w-72 text-center">
      <h3 class="text-xl font-semibold mb-2">AVS Mohan Kumar</h3>
      <p class="text-sm">Roll No: 23B81A05M6</p>
      <p class="text-sm">2nd Year, 2nd Semester</p>
      <p class="text-sm">CSE-D, CVR College of Engineering</p>
      <p class="text-sm mt-2">Frontend Developer</p>
    </div>
    <!-- Vikas's Card -->
    <div class="bg-gray-800 rounded-2xl p-6 w-72 text-center">
      <h3 class="text-xl font-semibold mb-2">Banala Vikas Rao</h3>
      <p class="text-sm">Roll No: 23B81A05B5</p>
      <p class="text-sm">2nd Year, 2nd Semester</p>
      <p class="text-sm">CSE-D, CVR College of Engineering</p>
      <p class="text-sm mt-2">Backend & File Comparison Developer</p>
    </div>
  </div>
</section>

</body>
</html>
```

ABBREVIATIONS

Abbreviation	Full Form
NLP	Natural Language Processing
API	Application Programming Interface
UI	User Interface
PDF	Portable Document Format
DOCX	Microsoft Word Open XML Document
TXT	Plain Text File
SerpAPI	Search Engine Results Page Application Programming Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
NLTK	Natural Language Toolkit
ML	Machine Learning
CPU	Central Processing Unit
RAM	Random Access Memory
VS Code	Visual Studio Code
URL	Uniform Resource Locator
CSV	Comma-Separated Values