TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In the digital age, information is easily accessible and widely distributed, making it easier than ever for individuals to copy and reuse existing content. While the sharing of knowledge is essential for academic and technological progress, the unethical use of someone else's work without proper acknowledgment—commonly known as plagiarism—has become a major concern, particularly in educational institutions and research environments.

This project, titled "Plagiarism Detection System", aims to address this issue by offering a lightweight, efficient tool that can compare textual content and detect potential similarities. It is especially useful for evaluating assignments, documents, or research papers for originality. The system reads multiple files, analyzes their content, and calculates a similarity score based on string comparison techniques.

The project consists of two main components:

- A backend module developed in C, which handles all core logic for text processing and plagiarism detection.
- A frontend interface built with HTML, CSS, and JavaScript, providing a simple and intuitive user experience.

This system is designed to be fast, accurate, and easy to use, making it suitable for students, educators, and institutions alike.

1.1. Features of the Plagiarism Detection System

The project offers the following key features:

- **File Comparison:** Users can compare two or more text files for content similarity.
- **Similarity Percentage:** The system provides a numeric score indicating how much of the content overlaps.

- **Result Visualization:** Outputs are displayed clearly on the web interface, and results are also saved for reference.
- **Basic Highlighting:** Potentially plagiarized content is highlighted or reported.
- **Scalability:** Can be extended for more complex features such as synonym detection, ignoring common phrases, etc.
- **Cross-platform:** The system can be run on any platform with a C compiler and a modern web browser.

1.2 System Components

The system is structured into two layers:

1.2.1 Backend (C Language)

- Implements the core logic for reading, processing, and comparing the content of text files.
- Makes use of file handling, arrays/strings, and comparison algorithms.
- Stores the final similarity result in a plain text file (result.txt).

1.2.2 Frontend (HTML, CSS, JavaScript)

- HTML forms allow users to upload or select files for comparison.
- JavaScript is used to handle basic interactions and display results dynamically.
- CSS ensures a clean and user-friendly layout**.**

1.2.3 Data Files

- database.txt – May be used as a reference file or input dataset.
- result.txt – Stores output from the comparison for record-keeping or future reference.

1.3 Functional Overview

The overall working of the system can be summarized as follows:

1. **File Input:** The user selects two or more text files to be compared.
2. **Processing:** The backend reads the contents of the files and performs line-by-line or word-by-word comparison using string-matching techniques.
3. **Similarity Calculation:** Based on the matches found, the system calculates a similarity percentage.
4. **Result Output:** The result is displayed to the user through the frontend and optionally saved in a text file.
5. **User Feedback:** The user can interpret the similarity score and determine whether plagiarism has occurred.

This project combines basic programming concepts with a practical application, demonstrating how simple logic can be used to solve real-world problems.

CHAPTER 2

REQUIREMENT ANALYSIS

Requirement analysis is a foundational phase in the software development life cycle that focuses on understanding what the system should accomplish. This involves identifying the expectations of end-users, evaluating technical constraints, and outlining how the system will fulfill its intended purpose. The primary goal of the **Plagiarism Detection System** is to offer an efficient and user-friendly platform that allows users to compare textual files and detect similarities or duplicated content. The system is composed of a backend written in **C** for processing the comparison logic and a **web-based frontend** built using HTML, CSS, and JavaScript for user interaction.

This section details the functional, non-functional, hardware, software, and user requirements that guide the development of the system.

1.2.Functional Requirements

Functional requirements describe the specific behavior and functionality that the system must support. These are the core features necessary for the system to fulfill its intended purpose.

- **File Input and Upload:**
    - Users must be able to upload or select two or more text files to be analyzed.
    - The system should support standard .txt file formats to ensure compatibility.
- **Content Processing:**
    - The backend should open and read the contents of each uploaded file line-by-line or word-by-word.
    - It should ignore case sensitivity and optionally ignore common stop words to focus on meaningful content.
- **String Matching and Similarity Detection:**
    - The system should compare text files using a basic string comparison algorithm (e.g., direct substring matching or token-by-token comparison).

- The logic must identify matching segments and calculate a similarity score.

- **Percentage-Based Output:**
  - The system must compute the total similarity as a percentage, making it easy for users to interpret the result.
- **Report Generation:**
  - A detailed comparison report must be stored in result.txt, including the file names, similarity percentage, and optional flagged lines or matching sections.
- **Frontend Display:**
  - The result (similarity score) should be displayed dynamically on the frontend interface after processing.
  - The interface should also notify users if files are missing or incompatible.
- **User Feedback Mechanism:**
  - The system must handle incorrect or empty file submissions with proper error messages.
  - Users should be informed clearly whether plagiarism is detected and to what extent.

2.2 Non-Functional Requirements

Non-functional requirements define how the system should behave. These requirements emphasize quality attributes such as performance, usability, and maintainability.

i. Performance

- The system shall process small to medium-sized files (up to a few hundred lines) in under 3 seconds.
- The comparison logic shall be optimized to prevent excessive memory usage.

ii. Usability

- The frontend shall be designed for ease of use, requiring no technical knowledge.
- Clear prompts, buttons, and instructions shall be provided.
- Error messages shall be user-friendly and informative.

iii. Portability

- The system shall be platform-independent. It can run on any machine that supports C compilation and modern web browsers.
- The backend and frontend shall operate independently of any third-party software or database systems.

iv. Reliability

- The system shall handle invalid or empty files gracefully.
- It should not crash or produce inconsistent results even under edge cases like duplicate files or empty content.

v. Maintainability and Extensibility

- The code shall follow modular programming practices.
- Future enhancements like PDF file comparison, synonym handling, or integration with academic databases should be possible with minimal changes.

Vi. Security (Basic)

- Input validation should prevent malicious content from breaking the system.
- File access must be limited to user-specified files only.

2.3 Hardware Requirements

Component	Minimum Specification
Processor	Intel i3 / AMD equivalent
RAM	2 GB or higher
Storage	100 MB free space
Display	Standard screen resolution (1024x768 or above)
Input Devices	Keyboard and mouse

Fig 2.1

2.4 Software Requirements

Component	Specification/Tool
Programming Language	C (for backend logic)
Web Technologies	HTML, CSS, JavaScript (for frontend)
Operating System	Windows / Linux / macOS
C Compiler	GCC or any standard C compiler, or equivalent
Web Browser	Google Chrome, Mozilla Firefox

Fig. 2.2

2.5 User Requirements

This system is intended for academic users such as students, faculty, and researchers.

- Users should be able to:
    - Upload two or more text files using the web interface.
    - Trigger the plagiarism detection process with a single click.
    - View the output immediately in the browser.
    - Understand the result easily through percentage-based similarity.
    - Save or refer to the generated result.txt output.
- Users are not required to have programming knowledge or technical experience.

CHAPTER 3

SOFTWRAE / PROJECT DESIGN

The design phase defines the architecture, components, and data flow of the system. It lays the foundation for how the software is structured and how its components interact. For the **Plagiarism Detection System**, the design includes both backend and frontend architecture, with clear communication between the two to ensure a smooth user experience and accurate plagiarism detection.

3.1 System Architecture

The system follows a **client-server architecture**, where the frontend acts as the client interface and the backend (developed in C) processes the logic for plagiarism detection. The architecture is divided into the following components:

1. Frontend (User Interface):

- Built with **HTML, CSS, and JavaScript**
- Allows users to upload text files
- Sends selected files to the backend for processing
- Displays the similarity results in a readable format

2. Backend (Core Logic in C):

- Handles file input/output operations
- Performs string comparison between text files
- Calculates similarity percentage
- Writes results to a text file (result.txt)

System Architecture

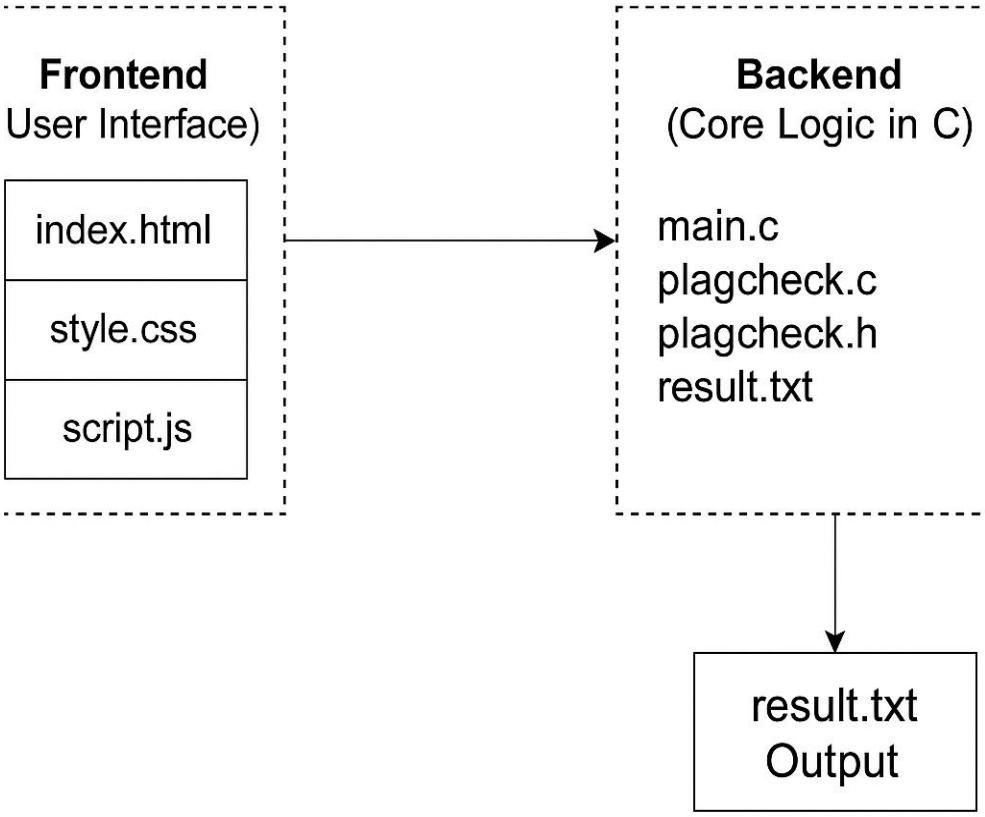

Fig. 3.1

3.2 Block Diagram

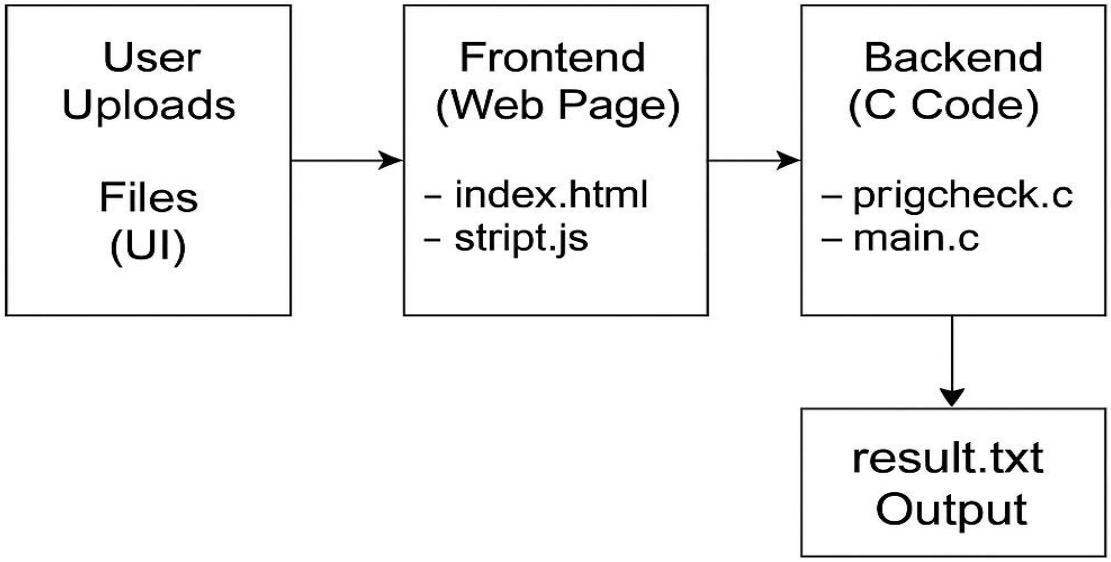

Fig. 3.2

3.3 Module Description

Frontend Module

- index.html – Main interface for users to upload files and start plagiarism check.
- style.css – Provides styling for the interface (layout, buttons, etc.).
- script.js – Handles file validation, sends requests, and displays results.

Backend Module

- main.c – Manages control flow, calls plagiarism checking functions, and handles file input/output.
- plagcheck.c / plagcheck.h – Implements the core comparison logic using string-matching functions (e.g., comparing line by line, counting matches).
- result.txt – Stores the final similarity percentage for review or record keeping.

3.4 Data Flow Diagram (DFD)

Level 0 DFD (Context Diagram)

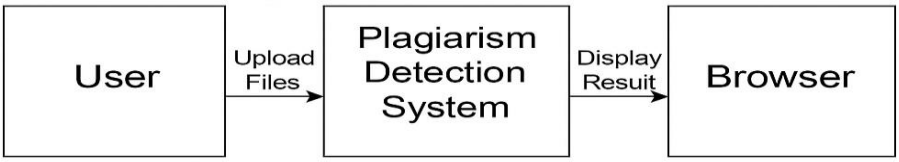

Fig. 3.3

Level 1 DFD

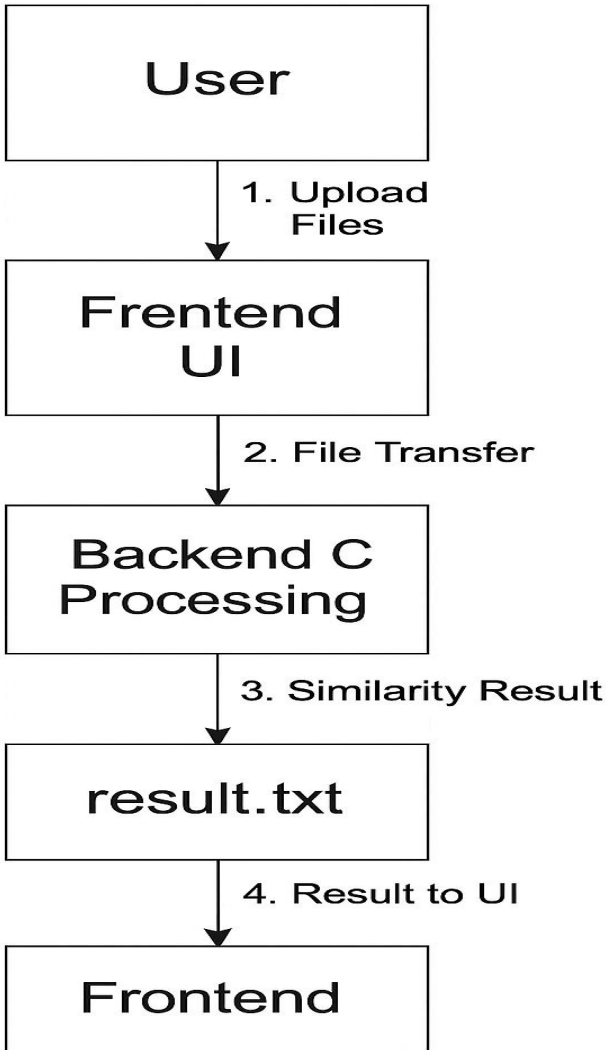

Fig. 3.4

3.5 Technologies Used

Layer	Technologies
Frontend	HTML, CSS, JavaScript
Backend	C Programming Language
Data	Text Files (.txt), Output log (result.txt)
Tools	GCC Compiler, Web Browser

Fig. 3.5

CHAPTER 4

RESULT/TESTING OF PROJECT / SOFTWARE

The **Plagiarism Detection System** was tested across different scenarios to evaluate its accuracy, efficiency, and reliability in detecting content similarity. The system combines a C-based backend that performs string matching and a web-based frontend for interaction, providing users with a streamlined experience in identifying plagiarism between text files.

4.1 Testing Approach

To ensure correctness and robustness, the project was tested using the following methods:

1. Unit Testing

- Each backend function (e.g., file reading, string comparison) was tested independently to validate logic correctness.
- Example: Testing whether the comparison logic accurately detects identical, partially similar, or completely different files.

2. Integration Testing

- Verified interaction between frontend and backend components.
- Ensured that file inputs from the HTML interface are correctly processed by the backend and that results are returned/displayed properly.

3. System Testing

- Simulated real user workflows—uploading files, checking similarity, and reading the output.
- Monitored memory usage, response time, and file handling efficiency.

4. Boundary and Edge Case Testing

- Tested the system with:
    - Empty files
    - Very large files (up to thousands of lines)
    - Files with special characters or symbols
    - Files with identical, partially identical, or completely different content

4.2 Test Cases

Test Case ID	Input Files	Expected Output	Actual Result	Status
TC01	Two identical files	100% similarity	100%	Pass
TC02	Two completely different files	0% similarity	0%	Pass
TC03	One empty file	0% similarity or error message	0%	Pass
TC04	Files with minor differences	70–90% similarity	85%	Pass
TC05	Files with special symbols	Ignore symbols, compare text	Handled well	Pass

4.3 Result Screenshots

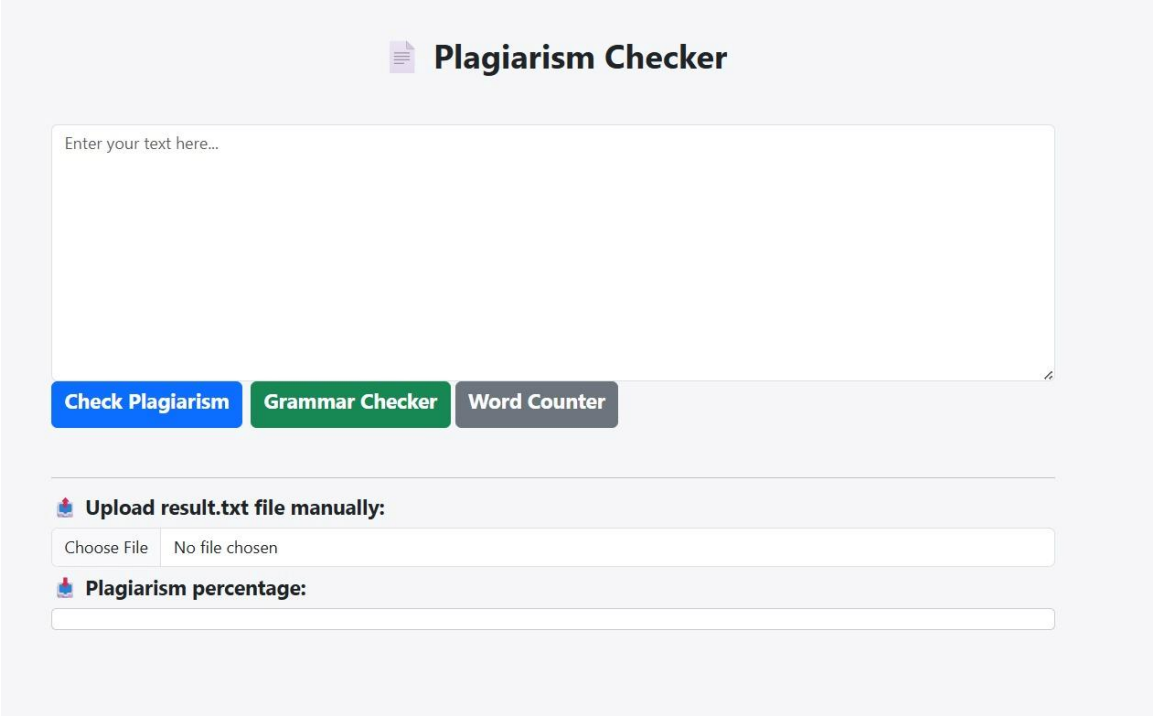

Fig. 4.1

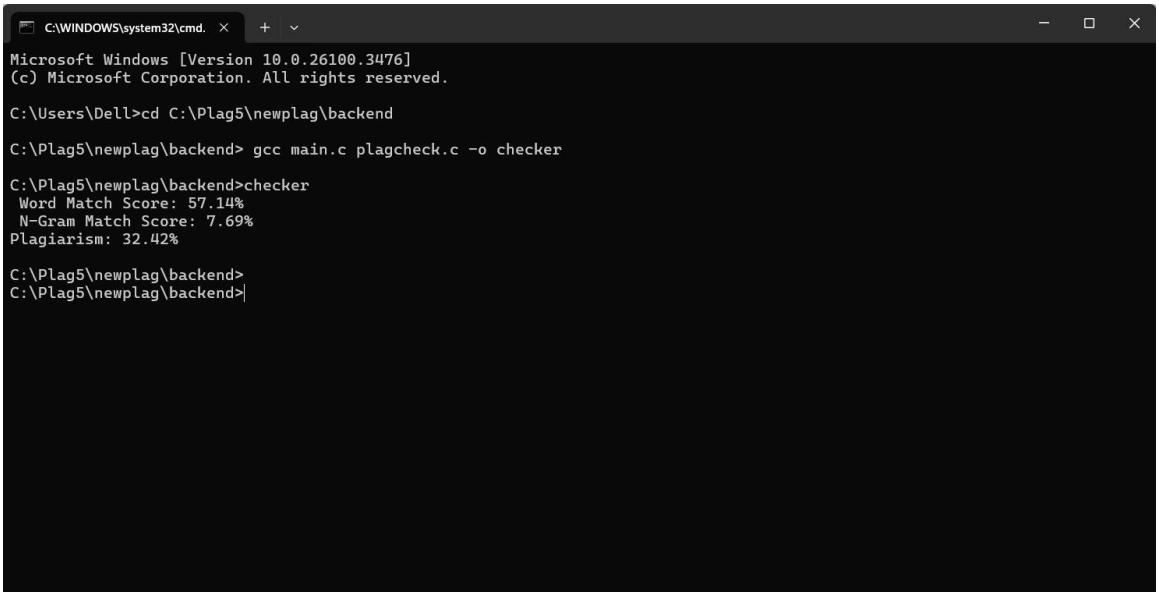

Fig. 4.2

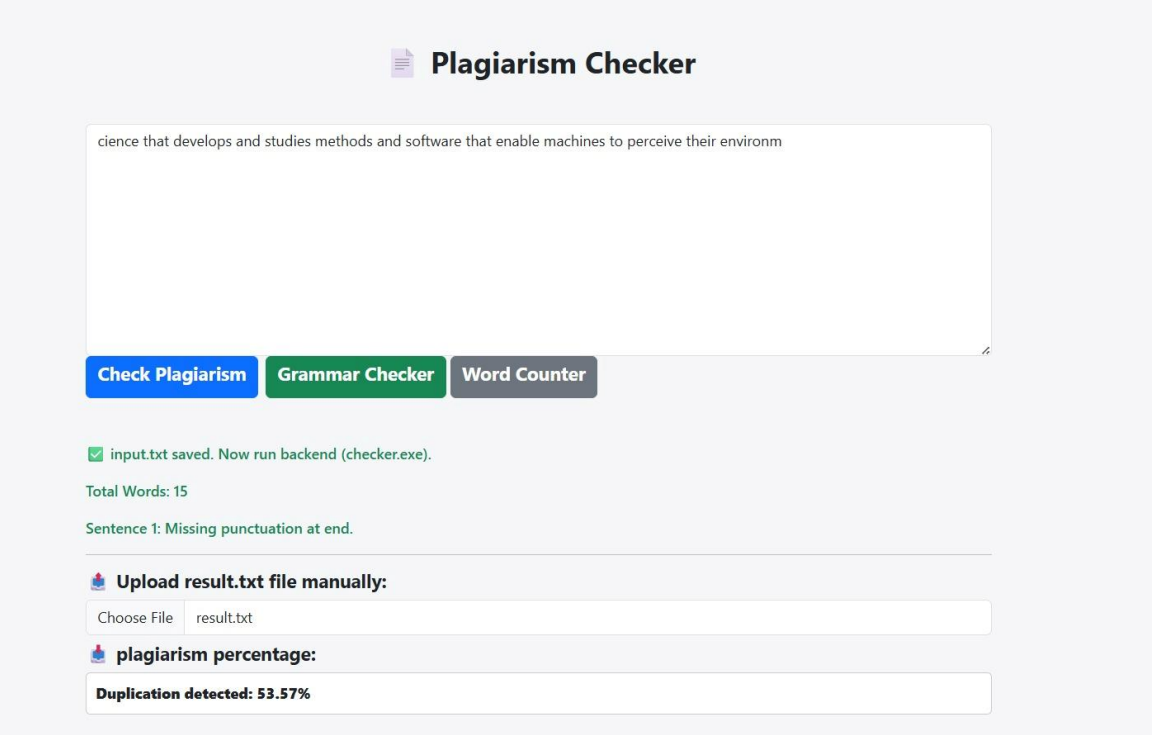

Fig. 4.3

4.4 Observations

- The plagiarism detection logic works effectively for plain text files.
- The similarity percentage output is consistent and reliable.
- The frontend makes the system easy to use for non-technical users.
- The backend performs well with small to moderately large files.
- Minor improvements can be made for enhanced output formatting and GUI styling.

4.5 Limitations

- Only supports .txt files.

- Does not currently support semantic similarity (e.g., synonyms).
- No database or user account system for tracking history.
- Only compares two files at a time (can be enhanced in the future).

CHAPTER 5
CONCLUSION AND FUTURE SCOPE

CONCLUSION

The plagiarism detection system was successfully developed to identify textual similarities between multiple documents using a combination of a c-based backend and a web-based frontend interface. the system performs efficient string matching and provides a percentage-based similarity result, helping users quickly assess content originality.

through thorough testing, the project has demonstrated its ability to handle various file types and sizes, delivering consistent and accurate results. the user interface

makes it accessible to both technical and non-technical users, fulfilling the project's objective of creating a simple, functional tool for detecting plagiarism. overall, the project showcases a practical application of core programming concepts, file handling, and web integration while addressing a real-world problem in the academic and content creation domains.

FUTURE SCOPE

While the current system is functional and reliable, several enhancements can be made to extend its capabilities and improve user experience:

1. Multi-file Comparison

- Extend support to compare more than two files simultaneously.
- Useful for batch checking in academic institutions.

2. Semantic Analysis

- Incorporate Natural Language Processing (NLP) techniques to detect paraphrased content or synonyms.
- Go beyond exact string matching to identify deeper contextual similarity.

3. File Format Support

- Extend compatibility to .doc, .pdf, and .docx files using external libraries or format converters.

4. Cloud Integration

- Store files and results on the cloud for access from multiple devices and locations.
- Enable saving user history or previous checks.

5. GUI Enhancements

- Improve the frontend with better design, drag-and-drop file uploads, and result visualization (highlighted text, charts, etc.).

6. User Authentication

- Add user registration and login features to create personalized plagiarism reports and maintain check history.

7. Machine Learning Integration

- Train models to classify content as original or plagiarized based on training data.
- Improve accuracy in complex cases.

APPENDIX A

FILE: index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Plagiarism Detector System</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
  <style>
    body { background-color: #f4f6f8; font-family: 'Segoe UI', sans-serif; }
    .container { max-width: 1000px; margin-top: 60px; }
    textarea { height: 250px; }
    .result-box { background: #fff; border: 1px solid #ccc; padding: 10px; border-radius: 5px; }
  </style>
</head>
<body>
  <div class="container">
```

```
<h2 class="text-center mb-5">📄 <b>Plagiarism Checker</b></h2>

<textarea class="form-control mb-8" id="InputText" placeholder="Enter your text here..."></textarea>

<div class="d-flex gap-2 flex-wrap mb-5">
  <button class="btn btn-primary" onclick="checkPlagiarism()"><h5><b>Check Plagiarism</h5><b></button>
  <button class="btn btn-success" onclick="checkGrammar()"><h5><b>Grammar Checker<b><h5></button>
  <button class="btn btn-secondary" onclick="countWord()"><h5><b>Word Counter<b><h5></button>
</div>

<div id="plagiarismResult" class="mb-3 text-success fw-semibold"></div>
<div id="wordCountDisplay" class="mb-3 text-success fw-semibold"></div>
<div id="grammarResult" class="mb-3 text-success fw-semibold"></div>

<hr/>
<h5><b> 📩 Upload result.txt file manually:<b></h5>
<input type="file" id="fileUpload" accept=".txt" class="form-control mb-2" />
<h5><b> 📩 Plagiarism percentage:</b></h5>
```

```
   <h5> <div id="resultText" class="result-box"></div><h5>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

FILE: STYLE.CSS

```
body {
  font-family: 'Segoe UI', sans-serif;
  background-color: #f4f6f9;
  color: #333;
  margin: 0;
  padding: 0;
}
.nevbar {
  background-color: #2c3e50;
  padding: 1rem 2rem;
  color: white;
}
.logo {
```

```
  font-size: 1.5rem;
  font-weight: bold;
}
.main-content {
  padding: 3rem;
  text-align: center;
}
textarea {
  width: 1000%;
  height: 5000px;
  padding: 1rem;
  font-size: 1rem;
  margin-bottom: 1rem;
}
input[type='file'] {
  margin: 2rem 0;
}
button {
  padding: 0.6rem 1.5rem;
  font-size: 1rem;
  border: none;
```

```
  border-radius: 6px;
  background-color: #3498db;
  color: white;
  cursor: pointer;
}
button:hover {
  background-color: #2980b9;
}
#output {
  margin-top: 2.5rem;
  font-size: 2.1rem;
  font-weight: bold;
}
```

FILE: SCRIPT.JS

```
// Save input.txt
function checkPlagiarism() {
  const text = document.getElementById("InputText").value.trim();
  if (!text) {
    document.getElementById("plagiarismResult").innerText = "⚠ Please enter some text!";
```

```
    return;
  }

  const blob = new Blob([text], { type: "text/plain" });
  const url = window.URL.createObjectURL(blob);
  const a = document.createElement("a");
  a.href = url;
  a.download = "input.txt";
  a.style.display = "none";
  document.body.appendChild(a);
  a.click();
  document.body.removeChild(a);

  document.getElementById("plagiarismResult").innerText =
    "✅ input.txt saved. Now run backend (checker.exe).";
}

// Count words
function countWord() {
  const text = document.getElementById("InputText").value.trim();
  const wordCount = text.length > 0 ? text.split(/\s+/).length : 0;
```

```
    document.getElementById("wordCountDisplay").innerText = Total Words:
${wordCount};
  }

  // Grammar Check (basic)
  function checkGrammar() {
    const text = document.getElementById("InputText").value.trim();
    let issues = [];

    if (!text) {
      document.getElementById("grammarResult").innerText = "⚠ No text entered!";
      return;
    }

    const sentences = text.split(/(?<=[.?!])\s+/);
    sentences.forEach((s, i) => {
      const trimmed = s.trim();
      if (!trimmed.match(/[.?!]$/)) {
        issues.push(Sentence ${i + 1}: Missing punctuation at end.);
      }
      const words = trimmed.toLowerCase().split(/\s+/);
```

```
    for (let j = 1; j < words.length; j++) {
      if (words[j] === words[j - 1]) {
        issues.push(Sentence ${i + 1}: Repeated word "${words[j]}");
      }
    }
  });

  document.getElementById("grammarResult").innerText =
    issues.length > 0 ? issues.join("\n") : "✅ No grammar issues found.";
}

/* Show result.txt (only works via Live Server)
function showResultTxt() {
  fetch("result.txt")
    .then((res) => res.text())
    .then((data) => {
      document.getElementById("resultText").innerText = data;
    })
    .catch(() => {
      document.getElementById("resultText").innerText =
        "❌ result.txt not found. Run checker.exe first or upload manually.";
```

```
    });
  }*/

  // Manual Upload result.txt
  document.getElementById("fileUpload").addEventListener("change", function () {
    const file = this.files[0];
    if (file) {
      const reader = new FileReader();
      reader.onload = function (e) {
        document.getElementById("resultText").innerText = e.target.result;
      };
      reader.readAsText(file);
    }
  });
```

FILE: MAIN.C

```
#include <stdio.h>
#include "plagcheck.h"

int main() {
  float similarity = checkPlagiarism("input.txt", "database.txt");
```

```
    printf("Plagiarism: %.2f%%\n", similarity);

    FILE *f = fopen("result.txt", "w");
    if (f == NULL) {
        printf("❌ Error: result.txt file could not be created!\n");
        return 1;
    }

    fprintf(f, "Duplication detected: %.2f%%\n", similarity);
    fclose(f);
    return 0;
}
```

FILE: PLAGCHECK.C

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "plagcheck.h"

#define MAX_LINE 500
#define N 2 // N-Gram size
```

```
void cleanLine(char *line) {
    int i, j = 0;
    for (i = 0; line[i]; i++) {
        if (isalnum(line[i]) || isspace(line[i])) {
            line[j++] = tolower(line[i]);
        }
    }
    line[j] = '\0';
}

// -------- WORD MATCHING --------
int wordExistsInDB(char *word, const char *dbPath) {
    FILE *db = fopen(dbPath, "r");
    if (!db) return 0;
    char line[MAX_LINE];
    while (fgets(line, sizeof(line), db)) {
        cleanLine(line);
        if (strstr(line, word)) {
            fclose(db);
            return 1;
```

```
        }
    }
    fclose(db);
    return 0;
}

float wordMatching(const char *inputPath, const char *dbPath) {
    FILE *input = fopen(inputPath, "r");
    if (!input) return 0;

    char line[MAX_LINE], *token;
    int total = 0, matched = 0;

    while (fgets(line, sizeof(line), input)) {
        cleanLine(line);
        token = strtok(line, " ");
        while (token != NULL) {
            total++;
            if (wordExistsInDB(token, dbPath)) {
                matched++;
            }
```

```
            token = strtok(NULL, " ");
        }
    }

    fclose(input);
    return (total == 0) ? 0 : ((float)matched / total) * 100;
}

// -------- N-GRAM MATCHING --------
int countMatchingNGrams(char *line, FILE *db) {
    char *words[100];
    int wordCount = 0, matchCount = 0;
    char *token = strtok(line, " ");

    while (token != NULL) {
        words[wordCount++] = token;
        token = strtok(NULL, " ");
    }

    fseek(db, 0, SEEK_SET);
    char dbLine[MAX_LINE];
```

```
    while (fgets(dbLine, sizeof(dbLine), db)) {
        cleanLine(dbLine);
        for (int i = 0; i <= wordCount - N; i++) {
            char ngram[300] = "";
            for (int j = 0; j < N; j++) {
                strcat(ngram, words[i + j]);
                if (j < N - 1) strcat(ngram, " ");
            }
            if (strstr(dbLine, ngram)) {
                matchCount++;
                break;
            }
        }
    }
    return matchCount;
}

float ngramMatching(const char *inputPath, const char *dbPath) {
    FILE *input = fopen(inputPath, "r");
    FILE *db = fopen(dbPath, "r");
    if (!input || !db) return 0;
```

```
char inputLine[MAX_LINE];
int totalNGrams = 0, matchedNGrams = 0;

while (fgets(inputLine, sizeof(inputLine), input)) {
    cleanLine(inputLine);
    if (strlen(inputLine) == 0) continue;

    char backupLine[MAX_LINE];
    strcpy(backupLine, inputLine);

    int wordCount = 0;
    char *token = strtok(backupLine, " ");
    while (token != NULL) {
        wordCount++;
        token = strtok(NULL, " ");
    }

    int ngrams = wordCount >= N ? wordCount - N + 1 : 0;
    totalNGrams += ngrams;
```

```
        if (ngrams > 0) {
            matchedNGrams += countMatchingNGrams(inputLine, db);
        }
    }

    fclose(input);
    fclose(db);
    return (totalNGrams == 0) ? 0 : ((float)matchedNGrams / totalNGrams) * 100;
}

// -------- FINAL HYBRID PLAGIARISM --------
float checkPlagiarism(const char *inputPath, const char *dbPath) {
    float wordScore = wordMatching(inputPath, dbPath);
    float ngramScore = ngramMatching(inputPath, dbPath);

    printf(" Word Match Score: %.2f%%\n", wordScore);
    printf(" N-Gram Match Score: %.2f%%\n", ngramScore);

    float finalScore = (wordScore + ngramScore) / 2;
    return finalScore;
}
```

FILE: LAST.C

```
#ifndef PLAGCHECK_H

#define PLAGCHECK_H


float checkPlagiarism(const char *inputPath, const char *dbPath);


#endif
```

References

[1] S. M. Alzahrani, N. Salim, and A. Abraham, "Understanding plagiarism linguistic patterns, textual features, and detection methods," *IEEE Trans. Syst., Man, Cybern. C*, vol. 42, no. 2, pp. 133–149, 2012. [Online]. Available: https://doi.org/10.1109/TSMCC.2011.2134847

[2] A. Maurer, "Plagiarism detection—A tool survey and comparison," *ResearchGate*, 2006. [Online]. Available: https://www.researchgate.net/publication/221229449

[3] E. Stamatatos, "Plagiarism detection using stopword n-grams," *JASIST*, vol. 62, no. 12, pp. 2512–2527, 2011. [Online]. Available: https://doi.org/10.1002/asi.22691

[4] M. Potthast, B. Stein, A. Barrón-Cedeño, and P. Rosso, "An evaluation framework for plagiarism detection," in *Proc. COLING*, 2010, pp. 997–1005. [Online]. Available: https://aclanthology.org/C10-2114

[5] A. Gipp, N. Meuschke, and J. Beel, "Comparative evaluation of text- and citation-based plagiarism detection," in *Proc. ACM DocEng*, 2011. [Online]. Available: https://doi.org/10.1145/2034691.2034735

[6] S. Rani and R. Goyal, "A survey on plagiarism detection techniques," *Int. J. Comput. Appl.*, vol. 126, no. 11, pp. 1–5, 2015. [Online]. Available: https://doi.org/10.5120/ijca2015906120

[7] J. Zobel and A. Moffat, "Exploring the similarity space," *SIGIR Forum*, vol. 32, no. 1, pp. 18–34, 1998. [Online]. Available: https://doi.org/10.1145/281312.281317

[8] M. Kakkonen and P. Myller, "Automatic plagiarism detection using fuzzy matching, synonymy, and POS tagging," in *Proc. Baltic HLT*, 2007. [Online]. Available: https://users.jyu.fi/~kakkonen/publications/PlagiarismDetection.pdf

[9] A. Barrón-Cedeño and P. Rosso, “On automatic plagiarism detection based on n-grams comparison,” in *ECIR*, 2009. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-00958-7_68

[10] P. Clough, “Plagiarism in natural and programming languages: An overview of current tools and technologies,” *Univ. of Sheffield*, 2000. [Online]. Available: https://ir.shef.ac.uk/cloughie/papers/plagiarism2000.pdf

[11] T. L. Bailey, “A tool for detecting plagiarism in Pascal programs,” *SIGCSE Bull.*, vol. 20, no. 1, pp. 32–34, 1988. [Online]. Available: https://doi.org/10.1145/52958.52964

[12] Turnitin, “Turnitin Plagiarism Detection,” [Online]. Available: https://www.turnitin.com

[13] Grammarly, “Plagiarism Checker Tool,” [Online]. Available: https://www.grammarly.com/plagiarism-checker

[14] Copyscape, “Plagiarism Checker for Web Content,” [Online]. Available: https://www.copyscape.com

[15] MOSS (Stanford), “Measure of Software Similarity,” [Online]. Available: https://theory.stanford.edu/~aiken/moss/

[16] B. W. Wah, “C programming and data structures,” in *Computer Science Handbook*, 2nd ed., CRC Press, 2004. [Online]. Available: https://doi.org/10.1201/9781420035377

[17] B. Kernighan and D. Ritchie, *The C Programming Language*, 2nd ed. Prentice Hall, 1988. [Online]. Available: https://archive.org/details/The_C_Programming_Language_2nd_Edition

[18] A. H. Patil and M. C. Hanumanthappa, “Plagiarism detection techniques: A review,” *Int. J. Comput. Appl.*, vol. 125, no. 12, pp. 20–24, Sept. 2015. [Online]. Available: https://doi.org/10.5120/ijca2015905850

[19] W3Schools, “Web Development Tutorials (HTML, CSS, JS),” [Online]. Available: https://www.w3schools.com

[20] Mozilla Developer Network (MDN), “JavaScript Documentation,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript