Table of Content

1. INTRODUCTION

1.1 Overview

Inrecent years,the hiring process has seen a massive shift toward digitalization. As the number of job applicants increases, HR departments are overwhelmed by the volume of resumes they must screen for each job opening. Traditional resume screening is manual, labor-intensive, prone to human bias, and often inconsistent. Recruiters may overlook well-qualified candidates simply due to fatigue, time constraints, or subjective preferences.

To address these challenges, our project proposes an **AI-powered Resume Screening and Ranking System**, built using **Python**, **Streamlit**, and **Natural Language Processing (NLP)**. This application takes in a predefined **Job Description (JD)** and multiple resumes in **PDF format**, evaluates how well each resume matches the JD, ranks them, and highlights potential gaps in candidate skills.

It not only automates the matching process using **TF-IDF vectorization** and **cosine similarity**, but also extracts relevant information such as **candidate name**, **email ID**, and **keywords that are missing** from the resumes. It provides a simple yet powerful user interface for uploading resumes and downloading results in **CSV format**, which can then be used by HR teams or applicant tracking systems (ATS).

This system demonstrates the power of NLP in solving real-world HR challenges, enhancing fairness and efficiency in hiring.

1.2 Objective

The primary objectives of the project are outlined below:

a) Automate Resume Screening:

To eliminate the need for manual resume review by leveraging text processing and semantic analysis to automatically screen resumes against a job description.

b) Rank Candidates:

To generate a **match score** for each candidate based on textual similarity and rank them from highest to lowest.

c) Extract Key Information:

To extract vital details like **candidate name** and **email ID** from resumes to present a concise and informative overview.

d) Recommend Improvements:

To identify and list missing keywords in the candidate's resume, helping them understand where their resume lacks alignment with the JD.

e) Provide Usable Output:

To allow HR professionals to download a comprehensive **CSV report** of all candidates including their scores, name, email, and feedback.

f) Build a User-Friendly Interface:

To create an easy-to-use web interface using **Streamlit**, allowing non-technical users to use the application without complex setup.

1.3 Significance of the Project

Thisproject holds high relevanceinthe domain of Human Resources (HR) and talent acquisition. Here's why:

a) Time-Saving:

Screening hundreds of resumes manually can take hours. With this tool, the same can be accomplished in minutes with better accuracy.

b) Consistency:

Unlike humans, the algorithm does not get tired or make subjective judgments. This ensures **fair evaluation criteria** for all resumes.

c) Objectivity:

The matching process is purely based on textual relevance to the JD, thereby eliminating unconscious biases (e.g., based on names or backgrounds).

d) Easy Integration:

The CSV output makes it easy to plug this tool into existing HRMS or ATS platforms.

e) Insightful Feedback:

Candidates and HR can use the feedback to **improve resume quality** or adjust the job description for clarity.

f)Scalable for Large Recruitments:

Companies conducting **mass recruitment drives or campus placements** can screen thousands of resumes in bulk.

g) Real-World Impact:

This system has direct applicability in startups, MNCs, recruitment agencies, and academic institutions.

1.4 Features of the Resume Screener (Expanded)

Feature	Description
Upload Resumes	Upload multiple resumes in PDF format via drag-and-drop or file picker.
Text Extraction	Uses pdfminer to extract clean text content from resumes.
Text Cleaning	Applies preprocessing (lowercasing, removing punctuation, etc.).
TF-IDF Vectorization	Converts text into numeric form using TfidfVectorizer for similarity scoring.
Cosine Similarity	Calculates similarity score between JD and resume text.
Score Generation	Assigns a percentage score to each resume based on semantic match.
Resume Ranking	Sorts and ranks resumes from best match to worst.
Name & Email Extraction	Extracts candidate's name and email using regex and heuristics.
Missing Keyword Finder	Lists 5 most important keywords missing from low-scoring resumes.
Results Display	Shows score, name, email, and recommendations for each resume.
CSV Download	Exports full ranked results to a downloadable CSV file.
Streamlit UI	Interactive web UI for uploading resumes and viewing results.
Lightweight and Fast	Requires no heavy models; fast even with many resumes.

2. HARDWARE AND SOFTWARE REQUIREMENTS

In order to build, deploy, and run the Resume Screener and Ranking System effectively, both **hardware** and **software components** are required. The system is lightweight and can run on a personal laptop or cloud-based environment with minimal specifications.

This section outlines the **minimum and recommended requirements** for both development and deployment environments.

2.1 Hardware Requirements

The hardware requirements are divided into two categories: **Development Machine Requirements** (for building and testing the application) and **User/Client-Side Requirements** (for running the deployed web app).

A. Development Machine Requirements

Component	Minimum Requirement	Recommended Requirement
Processor	Intel Core i3 / AMD Ryzen 3	Intel Core i5 or higher
RAM	4 GB	8 GB or higher
Storage	2 GB free space	SSD with 5 GB free space
Operating System	Windows 10 / Linux Ubuntu 18.04+	Windows 11 / Ubuntu 20.04+
Display	1280 x 720 resolution	Full HD (1920 x 1080)
Internet	Required for installing dependencies	Required for deployment and GitHub

B. User (Client-Side) Requirements

Since the front-end is built using **Streamlit**, the application is served via a web interface. Users do not need any special hardware or software other than a basic device with a modern web browser.

Component	Requirement
Device	Desktop, Laptop, or Mobile
Browser	Chrome, Firefox, Edge (latest version)
Internet Speed	Minimum 1 Mbps for smooth interaction
Storage	Ability to download CSV files

2.2 Software Requirements

The software stack consists of the **operating system**, **programming language**, **frameworks**, and **libraries** needed to develop and run the application.

A. Programming Environment

Software	Version/Tool Used
Operating System	Windows 10/11 or Linux Ubuntu 20.04+
IDE / Editor	Visual Studio Code
Programming Language	Python 3.8 or higher
Python Environment	venv or conda virtual environment
Command Line Tool	Windows CMD / PowerShell / Terminal

B. Required Python Libraries

The following Python libraries are essential to run the Resume Screener app:

Library	Purpose
streamlit	Frontend interface and file upload UI
pandas	Handling tabular data and CSV generation
pdfminer.six	Extracting text content from PDF resumes
scikit-learn	TF-IDF vectorization and similarity scoring
re (built-in)	Regular expressions for email/name extraction
os (built-in)	File handling for temporary PDF storage

C. Recommended Python Packages Installation Command

bash

CopyEdit

pip install streamlit pandas pdfminer.six scikit-learn

2.3 Third-Party Tools & Services (Optional)

Tool/Service	Purpose
GitHub	Version control and source code management
Streamlit Cloud	Online deployment of the app (optional)
Google Colab	Prototyping and notebook experimentation
Heroku/Vercel	Alternative cloud deployment platforms
Docker	For containerizing the app (advanced)

2.4 Software Configuration

Belowisanexampleofsettingupthe virtual environment and dependencies:

python -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate

pip install -r requirements.txt

Your requirements.txt should include:

nginx

streamlit

pandas

scikit-learn

pdfminer.six

2.5 Deployment Environment (Optional)

If youplan to deploy thisapp online forusers (e.g., HR teams or students), you can use:

- **Streamlit Cloud** – easiest for small apps
- **Render / Heroku** – for scalable cloud hosting
- **Localhost** – for internal use in companies or colleges

Example deployment command:

bash

CopyEdit

streamlit run app.py

3. MODULES

The Resume Screening and Ranking System is divided into several functional modules that work together to form the complete system. Each module is responsible for a specific task, and they interact seamlessly to process, evaluate, and present the resumes in a user-friendly way.

This modular design ensures scalability, maintainability, and clarity in the development and enhancement of the application.

3.1 Module Overview

Module Name	Purpose
1. Resume Upload Module	Upload and handle multiple PDF resumes via the UI
2. Text Extraction Module	Extract raw text from uploaded PDF resumes
3. Text Cleaning Module	Clean and preprocess extracted text
4. Job Description Loader	Load and clean the job description from a text file
5. Feature Extraction Module	Convert text into TF-IDF vectors for analysis
6. Similarity Matching Module	Calculate similarity scores between resumes and the job description
7. Keyword Gap Detector	Identify key terms missing from resumes
8. Candidate Info Extractor	Extract candidate name and email using heuristics
9. Result Display Module	Display scores, feedback, and progress bars on Streamlit UI
10. CSV Report Generator	Export ranked resume results to a downloadable CSV file

3.2 Module Descriptions

Module 1: Resume Upload Module

Responsibilities:

• Accepts one or more resumes in PDF format
• Performs basic validation (file type, size)
• Temporarily stores them for processing

Key Functions:

- st.file_uploader() (Streamlit function)

Module 2: Text Extraction Module

Responsibilities:

• Extracts raw text from each PDF using pdfminer.six
• Handles both file objects (from UI) and file paths (for testing)

Key Function in utils.py:

python

CopyEdit

def extract_text_from_pdf(path):

Module 3: Text Cleaning Module

Responsibilities:

• Converts text to lowercase
• Removes newlines, special characters, and symbols
• Makes text consistent for NLP analysis

Function:

python

CopyEdit

def clean_text(text):

Module 4: Job Description Loader

Responsibilities:

• Loads job description from a .txt file
• Preprocesses it the same way as resumes

Key Code:

python

CopyEdit

with open("job_description.txt", "r") as f:

jd_raw = f.read()

jd_clean = clean_text(jd_raw)

Module 5: Feature Extraction Module

Responsibilities:

• Transforms resume and JD texts into numerical vectors using **TF-IDF**
• Captures term importance for comparison

Tools Used:

- TfidfVectorizer from sklearn.feature_extraction.text

Module 6: Similarity Matching Module

Responsibilities:

• Computes cosine similarity between resume and JD vectors
• Generates a match score (0–100%)

Function:

python

CopyEdit

defcalculate_similarity(resume_text, jd_text):

Module 7: Keyword Gap Detector

Responsibilities:

• Compares cleaned resume with cleaned JD
• Identifies keywords missing from resume
• Shows top 5 missing terms if score is low

Code Snippet:

python

CopyEdit

```
if score < 70:
    missing_keywords = [...]
```

Module 8: Candidate Info Extractor

Responsibilities:

• Uses regular expressions to extract email ID
• Applies heuristics to find probable candidate name (from first few lines of resume)

Function:

python

CopyEdit

```
def extract_name_and_email(text):
    ...
```

Module 9: Result Display Module

Responsibilities:

• Shows each resume's name, score, recommendation
• Displays progress bar, missing keywords, and ranking

UI Elements Used:

- st.write, st.progress, st.markdown, etc.

Module 10: CSV Report Generator

Responsibilities:

• Collects all resume results
• Converts them to a Pandas DataFrame
• Generates a downloadable CSV file with rank, name, email, score, etc.

Key Code:

python

CopyEdit

df = pd.DataFrame(results)

df.to_csv("results.csv", index=False)

⬅ END Summary

This modular breakdown shows how the project follows **Separation of Concerns** — each module is isolated to perform one specific task. This makes debugging easier, enhances code reusability, and simplifies future upgrades.

---

4. SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

The Software Requirements Specification (SRS) provides a structured and detailed description of the functionalities, features, and constraints of the system. It serves as a contract between the developer and the end-user and ensures that all stakeholders have a common understanding of the system's behavior and performance.

4.1 Introduction

4.1.1 Purpose

The purpose of this document is to outline the complete software requirements of the **AI-Powered Resume Screener and Ranking System**. It is intended for use by developers, testers, project managers, and end users (e.g., HR professionals, placement officers).

4.1.2 Scope The system allows users to:
• Upload multiple PDF resumes
• Automatically extract text from resumes
• Compare each resume with a given job description
• Score each resume based on its similarity with the JD
• Rank the resumes and export results as a CSV
• Highlight key missing terms and suggest improvements This project helps automate the tedious and time-consuming task of manual resume screening, improving efficiency and objectivity in candidate shortlisting.

4.2 Overall Description

4.2.1 Product Perspective The Resume Screener system is a **standalone web application** developed using **Python, Streamlit, and NLP techniques**. It can be run locally or hosted on the cloud. It is an enhancement over traditional manual screening systems. **4.2.2 Product Features**

•Resume text extraction from PDF
•Cleaning and preprocessing of text data
•TF-IDF-based similarity matching
•Cosine similarity scoring
•Keyword gap analysis
•Resume ranking based on match score
•Candidate name and email extraction
•CSV download of results
•Easy-to-use web interface

4.2.3 User Characteristics

- **HR Professionals / Recruiters**: Non-technical users looking to shortlist candidates efficiently.
- **Developers**: Users who want to extend or modify the system.
- **Students / Colleges**: For campus recruitment simulations.

4.2.4 Constraints

•Input format limited to PDF files
•Job description must be in text format (.txt)
•Resume must contain extractable text (scanned images not supported)
•Works best for English-language resumes

4.2.5 Assumptions & Dependencies

- Python 3.8+ must be installed
- Internet required to install dependencies or deploy on cloud
- Streamlit must be installed to run UI
- Resume format must be readable by pdfminer.six

4.3 Specific Requirements

4.3.1FunctionalRequirements

ID	Requirement Description
FR-01	System shall allow the user to upload multiple PDF resumes
FR-02	System shall allow the user to load a job description text file
FR-03	System shall extract raw text from each resume
FR-04	System shall clean and preprocess both resume and job description
FR-05	System shall compute similarity score using TF-IDF and cosine similarity
FR-06	System shall display match percentage for each resume
FR-07	System shall highlight missing important keywords from the resume
FR-08	System shall extract candidate name and email address from the resume text
FR-09	System shall display recommendations (e.g., Strong Match or Needs Improvement)
FR-10	~~System shall allow users to download results as a CSV file including rank, score,~~ name, etc.

4.3.2 Non-Functional Requirements

ID	Non-Functional Requirement Description
NFR-01	The system shall have a user-friendly interface
NFR-02	The application shall process resumes in under 5 seconds each
NFR-03	The system shall support upload of at least 20 resumes at once
NFR-04	The application shall ensure that no personal data is stored permanently
NFR-05	The application shall work on major browsers (Chrome, Firefox, Edge)

4.4 System Models

4.4.1 Use Case: Resume Screening

Actors: HR Recruiter (Primary), System (Secondary)
Preconditions: Job description uploaded; resumes selected
Postconditions: Ranked resume list generated with insights

Flow:

1. User uploads job description text file 2. User uploads multiple resume PDFs 3. System extracts, cleans, and analyzes resumes 4. System calculates match scores 5. System displays scores and ranking 6. User downloads CSV report

4.4.2 State Diagram

(Will be shown in Section 10)

4.4.3 Activity Diagram

(Will be shown in Section 10)

5. DATA DICTIONARY

The data dictionary provides detailed descriptions of all data elements used throughout the application. This includes data collected from user input (e.g., resumes), intermediate data generated during processing, and output data like CSV files and screen results.

It serves as a valuable reference for developers, testers, and maintainers to understand the format, structure, and meaning of various data components.

5.1 Data Elements Table

Name	Type	Source/Location	Description
uploaded_files	List[File]	Streamlit FileUploader	List of resumes uploaded in PDF format by the user
resume_text	String	extract_text_from_pdf()	Raw text extracted from the resume PDF
jd_raw	String	job_description.txt	Original text content of the job description
jd_clean	String	clean_text(jd_raw)	Cleaned job description

Name	Type	Source/Location	Description
			used for similarity comparison
resume_clean	String	clean_text(resume_text)	Cleaned resume content used for similarity comparison
match_score	Float	calculate_similarity()	Cosine similarity score between resume and JD (0–100%)
missing_keywords	List[String]	Generated dynamically	List of important words from JD not found in resume
name	String	extract_name_and_email(resume_text)	Candidate's name, heuristically extracted from resume
email	String	extract_name_and_email(resume_text)	Candidate's email ID extracted using regex

Name	Type	Source/Location	Description
result	Dict	App Logic	Dictionary holding details of each resume analysis
results	List[Dict]	App Logic	List of all resume analysis dictionaries
df	pandas.DataFrame	pd.DataFrame(results)	Structured tabular data for all resume analysis (used for CSV export)

5.2 Data Structures

result Dictionary (One per resume)

```
result = {
    "Resume": "rahul_kumar_resume.pdf",
    "Candidate Name": "Rahul Kumar",
    "Email": "rahul@example.com",
    "Score": 87.45,
    "Recommendation": "✅ Strong Match",
    "Hint": "python, sql, powerbi",
    "Rank": 1
}
```

results List

```
results = [
    {...},  #  resume  1
    {...},  #  resume  2
    ...
]
```

df DataFrame (for CSV export)

Resume	Candidate Name	Email	Score	Recommendation	Hint	Rank
rahul_kumar_resume.pdf	Rahul Kumar	rahul@example.com	87.45 ✅	Strong Match	python, sql, powerbi analytics,	1
sample_resume_2.pdf	Ravi Singh	ravi@abc.com	65.78 ⚠️	Needs Improvement	pandas	2

5.3 File Dependencies

File	Type	Purpose
app.py	Python file	Main Streamlit app file for UI and logic
utils.py	Python file	All backend functions (text extraction, cleaning)
job_description.txt	Text file	Input job description used for resume comparison
results.csv	CSV	Downloadable file containing ranked resume results

5.4 Regular Expressions Used

Use	Regex Pattern	Purpose
Extract email	[\w\.-]+@[\w\.-]+	Captures standard email formats
Clean text content	[^a-zA-Z0-9\s]	Removes special characters and symbols

Summary: This data dictionary outlines all critical variables and structures used in the application, serving as a blueprint for understanding the program's inner workings. It ensures consistency in development, debugging, testing, and maintenance.

6. DATA FLOW DIAGRAMS (DFD)

Data Flow Diagrams (DFDs) visually represent how data moves through a system. It shows the system's functional perspective by identifying processes, data stores, data sources, and data destinations.

We will include:

• Level 0 DFD (Context Diagram)
• Level 1 DFD (Decomposition of Main Process)

6.1 DFD Level 0 (Context Diagram)

Purpose:
Shows the entire system as a single process and its interaction with external entities (users).

External Entities:

• **User (HR/Recruiter)** – Uploads resumes and job descriptions, downloads CSV output.

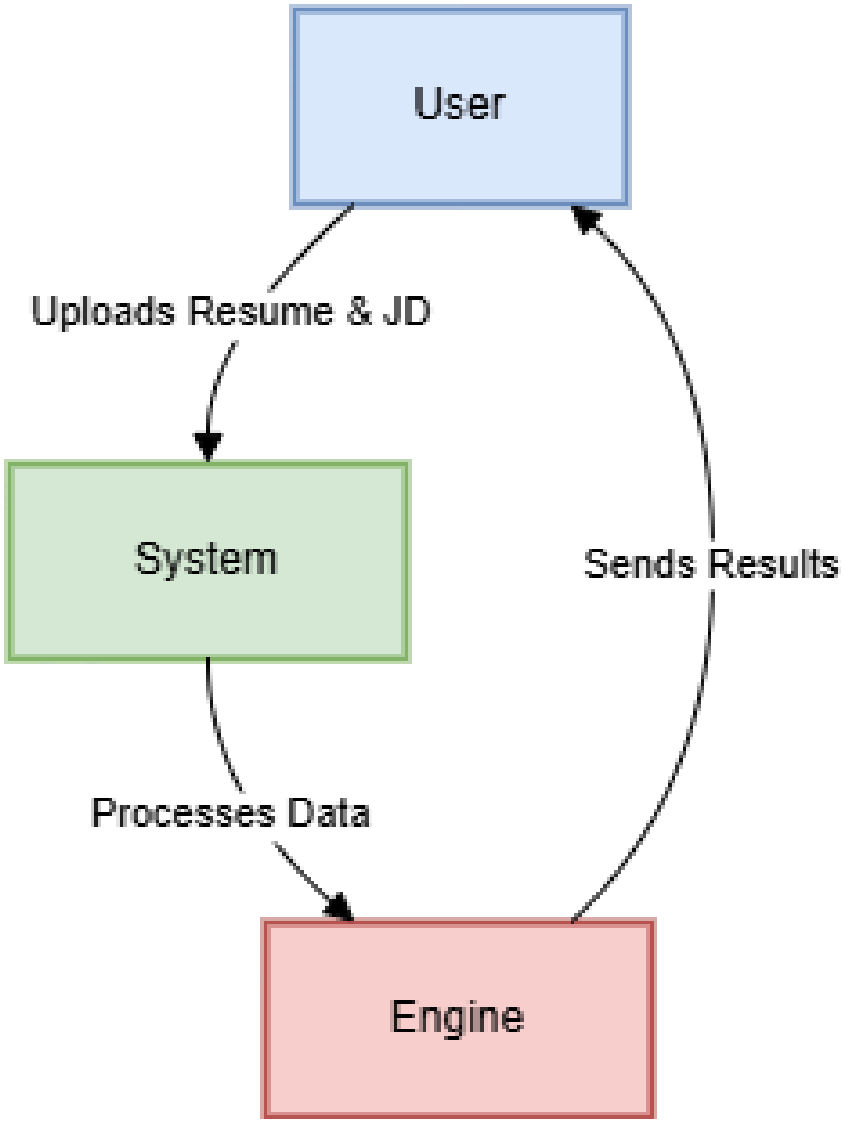
Processes:

• **Resume Screener System** – Central system performing text extraction, comparison, ranking, and output.

Data Flows:

•Job Description File (from user → system)
•Resume PDFs (from user → system)
•Screening Results (from system → user)

📈 DFDDiagram Level 0



6.2 DFD Level 1 (Functional Decomposition)

Purpose:
Breaks down the central system into sub-processes to show internal data movement.

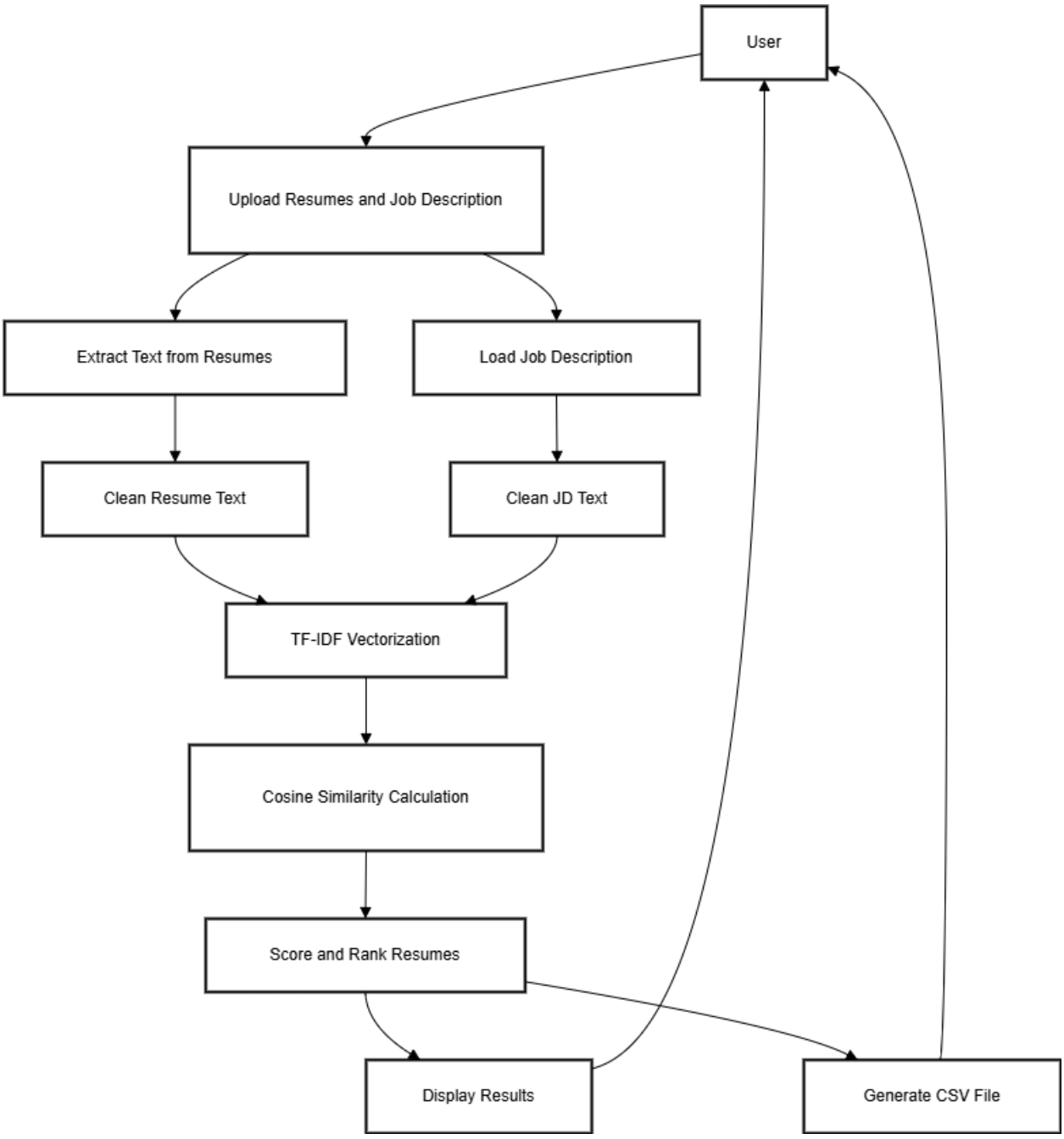
Processes:

1. **1.0 Upload Job Description**
2. **2.0 Upload Resumes**
3. **3.0 Extract and Clean Text**
4. **4.0 Compare with JD & Score**
5. **5.0 Rank and Generate Report**
6. **6.0 Download CSV Output**

Data Stores:

•Job Description File
•Resume Files
•Processed Results

📈 DFD Diagram Level 1



6.3 Data Descriptions

Data Flow	Source	Destination	Description
Job Description File	User	Upload JD Module	Plain text job description
Resume Files	User	Upload Resume Module	Multiple resumes in PDF format
Cleaned Data	Extract Module	Scoring Module	Preprocessed text ready for analysis
Match Scores	Scoring Module	Ranking Module	Similarity scores with feedback
CSV Report	Ranking Module	User	Downloadable file with results and recommendations

END Summary

The DFDs outline how the system handles user input, processes the data, and produces results. These diagrams form the basis for understanding the data interactions and flow architecture in the system.

8. SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

8.1 Introduction

TheSoftwareRequirements Specification (SRS) defines the functionality, performance, design constraints, and attributes of the **AI-Based Resume Screening and Ranking System**. It acts as a reference for developers, testers, stakeholders, and end-users to understand what the software is expected to do.

8.2 Purpose of the System

Thepurpose of this system is to automate the initial phase of recruitment by screening multiple resumes against a job description using Natural Language Processing (NLP). It evaluates and ranks candidates based on relevance and allows recruiters to download the results in CSV format.

8.3 Scope of the System

Thissoftware will:

- Allow uploading of multiple resumes (PDF format)
- Extract and clean text using NLP techniques
- Match resumes against a provided job description
- Rank resumes based on similarity scores
- Identify missing keywords
- Display recommendations
- Generate downloadable CSV reports

8.4 Functional Requirements

ID	Requirement Description
FR1	The system shall allow the user to upload multiple PDF resumes FR2
The	system shall allow the user to upload or provide a job description FR3
The	system shall extract text from resumes and clean it using NLP FR4 The
system	shall compare resume content with the job description FR5 The
system	shall calculate similarity scores using cosine similarity FR6 The
system	shall identify and display missing keywords FR7 The system shall
rank	the resumes based on scores FR8 The system shall allow
downloading	of results in CSV format FR9 The system shall extract
candidate	names and email addresses

8.5 Non-Functional Requirements

Category	Requirement Description
Performance	The system should process and rank resumes within seconds per file
Usability	The UI should be simple, intuitive, and accessible via browser
Reliability	The system should process valid resumes without crashing
Portability	Should run on Windows, Linux, and Mac OS using Python & Streamlit
Scalability	Should support screening of at least 50 resumes per session
Maintainability	Codebase should follow modular design with clear documentation
Security	Uploaded files should not be stored permanently

8.6 Hardware Requirements

Component	Minimum Requirement
Processor	Intel Core i3 or higher
RAM	4 GB or more
Disk Space	500 MB free
Display	Minimum 1024x768 resolution
Internet	Required only for installing Python dependencies

8.7 Software Requirements

Software	Version	Purpose
Python	3.8 or higher	Programming language
Streamlit	1.0 or higher	Web app frontend
scikit-learn	Latest	TF-IDF, cosine similarity
pdfminer.six	Latest	PDF text extraction
pandas	Latest	Data handling & CSV generation
re (regex)	Built-in	Text processing and extraction
OS (Windows/Linux/Mac)	Any	Local deployment environment

8.8 Assumptions and Dependencies

- Assumes resumesarein Englishandformatted textually.
- Requires job descriptions to be uploaded as .txt files.
- Depends on Python libraries being properly installed. No
- database integration is required in the current version.

Summary

This SRS section outlines all functional and non-functional expectations of the system. It helps align development with business goals and sets the foundation for implementation, testing, and maintenance.

10. CLASS DIAGRAM

10.1 Purpose of the Class Diagram

The Class Diagram describesthe **object-oriented structure** of the Resume Screening System. It shows the main classes used in the system, their attributes, methods, and the relationships between them. This helps in designing a modular, maintainable, and extensible codebase.

10.2 Key Classes in the System

The majorclassesidentified inthisproject are:

1. **ResumeProcessor**
2. **JobDescription**
3. **SimilarityCalculator**
4. **Result**
5. **CSVExporter**
6. **UserInterface (StreamlitApp)**

10.3 Class Descriptions

Class: ResumeProcessor

Attribute	Type	Description
resume_path	String	Path to the uploaded resume
text	String	Extracted and cleaned resume content

Attribute	Type	Description
candidate_name	String	Extracted candidate name (from text)
candidate_email	String	Extracted candidate email (from text)

Method	Return Type	Description
extract_text()	String String	Extracts raw text from the PDF file
clean_text()		Cleans and processes the extracted text
extract_name_email()	Tuple	Extracts candidate name and email

Class: JobDescription

Attribute	Type	Description
jd_text	String	Original job description text
clean_text	String	Preprocessed job description

Method	Return Type	Description
load_from_file()	String	Loads text from a file
clean()	String	Cleans the JD text using regex/NLP

Class: SimilarityCalculator

Attribute	Type	Description
tfidf_vectorizer	Object	TF-IDF vectorizer object

Method	Return Type	Description
compute_similarity(resume_text, jd_text)	Float	Computes cosine similarity percentage

Class: Result

Attribute	Type	Description
resume_name	String	Name of the uploaded resume
candidate_name	String	Extracted candidate name
candidate_email	String	Extracted email
score	Float	Matching percentage
recommendation	String	Status (Strong Match/Needs Improvement)
keyword_hint	String	Missing important keywords

Class: CSVExporter

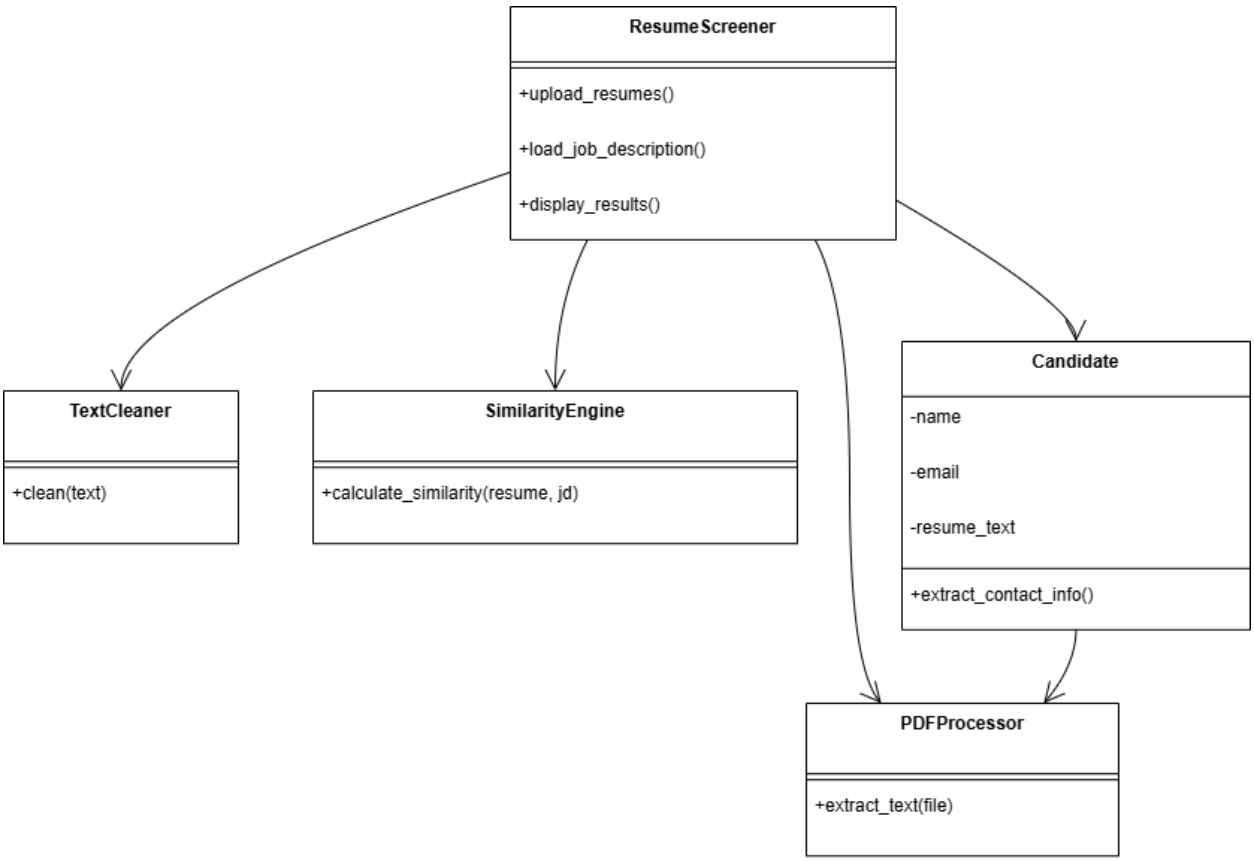
Method	Return Type	Description
export_to_csv(results)	CSV File	Exports list of results to CSV

Class: UserInterface (StreamlitApp)

Responsibility	Description
Render UI	Handles all Streamlit UI components
Upload files	Receives resume and JD uploads from the user
Trigger processing	Calls methods from processing classes
Display results	Renders progress bars, recommendations, and tables

Responsibility	Description
Handle download	Generates and serves the CSV file

10.4 Class Diagram (Text-Based UML Style)



Summary

The class diagram provides a clear modular structure for the entire application. It enables easy maintenance and facilitates potential feature expansions such as database integration, authentication, or more advanced NLP models.

11. USE CASE DIAGRAM

11.1 Purpose of the Use Case Diagram

The **UseCase Diagram** visually representsthe interactions between users (actors) and the system. It identifies the primary functionalities provided by the system and how end-users (typically HR recruiters or hiring managers) will interact with it.

This diagram is useful for understanding the **functional scope** of the project from a user's perspective.

11.2 Primary Actor

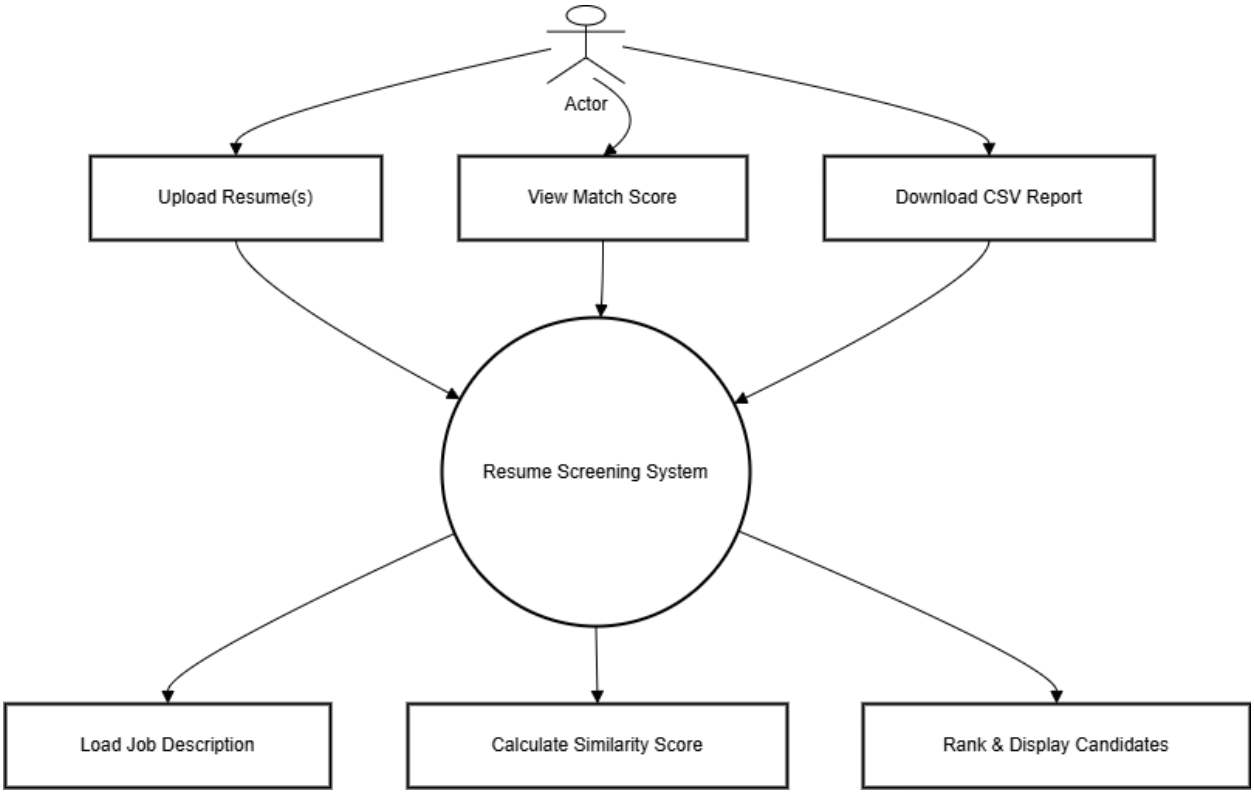
- **Recruiter (HR/User):** The person responsible for uploading resumes and job descriptions, reviewing match scores, downloading reports, and making hiring decisions based on results.

11.3 Use Cases

Here arethemajor use cases of the system:

Use Case ID	Use Case Name	Description
UC1	Upload Job Description	Recruiter uploads a .txt file containing job requirements
UC2	Upload Resumes	Recruiter uploads multiple resumes in PDF format
UC3	Extract and Clean Resume Text	System extracts text from resumes and processes it
UC4	Compare Resume with JD	System uses NLP to compare resume content with job description
UC5	Calculate Match Score	System computes a score based on similarity between resume and job description
UC6	Highlight Missing Keywords	System suggests important keywords missing in resumes
UC7	Display Recommendations	System displays recommendations based on match score
UC8	Export CSV Report	Recruiter can download a CSV file of ranked resumes with candidate details

11.4 Use Case Diagram



11.5 Use Case Descriptions (Detailed)

UC1: Upload Job Description

- **Actor**: Recruiter
- **Description**: Uploads a .txt file containing job role requirements.
- **Precondition**: Job description file exists
- **Postcondition**: JD is stored in memory and ready for comparison

UC2: Upload Resumes

- **Actor**: Recruiter
- **Description**: Uploads multiple resumes in PDF format
- **Precondition**: Resumes must be in .pdf format
- **Postcondition**: Resumes are ready for processing

UC3: Extract and Clean Resume Text

- **Actor**: System
- **Description**: Extracts text from resumes and cleans unnecessary formatting
- **Tools Used**: pdfminer.six, regex

UC4: Compare Resume with JD

- **Actor**: System
- **Description**: Preprocessed resumes and JD are compared using TF-IDF + Cosine Similarity

---

UC5: Calculate Match Score

- **Actor**: System
- **Description**: Score is calculated as a percentage indicating resume relevance

---

UC6: Highlight Missing Keywords

- **Actor**: System
- **Description**: System shows keywords present in JD but missing from resume

---

UC7: Display Recommendations

- **Actor**: System

- **Description**: Based on score, resume is tagged as "Strong Match" or "Needs Improvement"

UC8: Export CSV Report

- **Actor**: Recruiter
- **Description**: Enables downloading of all results in CSV with name, email, score, and feedback

Summary

The Use Case Diagram provides a user-centered view of the system's key features. It highlights the functionalities the recruiter interacts with, and how the backend system automates the screening process.

12. CODING

12.1 Overview of the Codebase

The codingsectiondocumentsand explains the actual implementation of the project — the **AI-powered Resume Screening and Ranking System**. The system is built using **Python**, **NLP techniques (TF-IDF + Cosine Similarity)**, and **Streamlit** for the web interface.

The codebase includes the following key components:

app.py – Main Streamlit application logic

utils.py – Contains helper functions for text extraction, cleaning, similarity computation

job_description.txt – Contains the job requirements

Resume PDF files – Uploaded by the user

Generated CSV file – Final output

12.2 Project Structure

```
resume-screener-nlp-main/
|   ├── app.py                  # Main application file
├── utils.py                    # Utility functions
├── job_description.txt         # Sample job description file
├── requirements.txt            # Required Python packages
└── assets/                     # (Optional) Store sample resumes or UI assets
    └── output/                 # (Generated) Downloadable CSVs
```

12.3 Code: utils.py

```
python
CopyEdit
import os
import re
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from pdfminer.high_level import extract_text


defextract_text_from_pdf(path):
    ifhasattr(path, 'read'):
        withopen("temp_resume.pdf", "wb") as f:
            f.write(path.read())
        text=extract_text("temp_resume.pdf")
        os.remove("temp_resume.pdf")
        returntext
    else:
        returnextract_text(path)


defclean_text(text):
text=text.lower()
text=re.sub(r'\n', ' ', text)
text=re.sub(r'[^a-zA-Z0-9\s]', '', text)
returntext


defcalculate_similarity(resume_text, jd_text):
```

```
    vectorizer = TfidfVectorizer()
    vectors = vectorizer.fit_transform([resume_text, jd_text])
    score = cosine_similarity(vectors[0:1], vectors[1:2])
    return round(float(score[0][0]) * 100, 2)

defextract_name_and_email(text):
    email = re.search(r'[\w\.-]+@[\w\.-]+', text)
    email = email.group(0) if email else 'Not found'
    name = text.strip().split('\n')[0].strip().title()
    iflen(name.split()) > 6 or not name.replace(" ", "").isalpha():
        name = 'Not found'
    return name, email

defexport_to_csv(results, filename="resume_results.csv"):
    df= pd.DataFrame(results)
    df.to_csv(filename, index=False)
    return filename
```

12.4 Code: app.py

```
python
CopyEdit
import streamlit as st
from utils import extract_text_from_pdf, clean_text, calculate_similarity,
extract_name_and_email, export_to_csv
import base64

st.set_page_config(page_title="Resume Screener", layout="wide")
```

```
#Sidebar
with st.sidebar:
    st.title("🔧 Options")
    st.markdown("Built with ❤️ using NLP & Streamlit")
    st.markdown("[GitHub Repo](https://github.com/rahulk1812)")

#Title & Description
st.title("📄 AI Resume Screener using NLP")
st.markdown("""
This app compares uploaded resumes against a job description and gives a **matching score**.
It also highlights resumes that may need improvements based on missing skills or keywords.
""")

#Load Job Description
try:
    with open("job_description.txt", "r", encoding='utf-8') as f:
        jd_raw = f.read()
    jd_clean = clean_text(jd_raw)
except FileNotFoundError:
    st.error("🚫 Job description file not found. Please ensure 'job_description.txt' is in the app folder.")
    st.stop()

st.subheader("📌 Job Description")
```

```
st.info(jd_raw)


#Resume Upload
st.subheader("📤 Upload Resumes")
uploaded_files = st.file_uploader("Upload multiple resumes (PDF)", type=["pdf"],
accept_multiple_files=True)

if uploaded_files:
    st.subheader("📊 Resume Match Results")
    results = []

    foruploaded_fileinuploaded_files:
        resume_name=uploaded_file.name
        resume_text=extract_text_from_pdf(uploaded_file)
        resume_clean=clean_text(resume_text)
        match_score=calculate_similarity(resume_clean, jd_clean)
        candidate_name,candidate_email = extract_name_and_email(resume_text)


        #Basicmissingkeyword analysis
        missing_keywords= []
        forwordinjd_clean.split():
            ifwordnotinresume_clean and len(word) > 5:
                missing_keywords.append(word)
        keyword_hint=",".join(missing_keywords[:5]) if match_score < 70 else ""


        result ={
            "Resume":resume_name,
```

```
            "Name": candidate_name,
            "Email": candidate_email,
            "Score": match_score,
            "Recommendation": "✅ Strong Match" if match_score >= 70 else " ⚠️ Needs Improvement",
            "Hint": keyword_hint
        }
        results.append(result)

        #Show result in UI
        st.write(f"**{resume_name}** - Match Score: **{match_score}%** → {result['Recommendation']}")
        st.progress(int(match_score))
        ifkeyword_hint:
            st.markdown(f"🔍 Might be missing important terms: `{keyword_hint}`")
        st.markdown("---")

    #Export as CSV
    csv_filename = export_to_csv(results)
    withopen(csv_filename, "rb") as f:
        b64 = base64.b64encode(f.read()).decode()
        href = f'<a href="data:file/csv;base64,{b64}" download="{csv_filename}">📥 Download Results as CSV</a>'
        st.markdown(href, unsafe_allow_html=True)

else:
    st.warning("Please upload at least one resume to begin screening.")
```

12.5 Code Explanation (High-Level Summary)

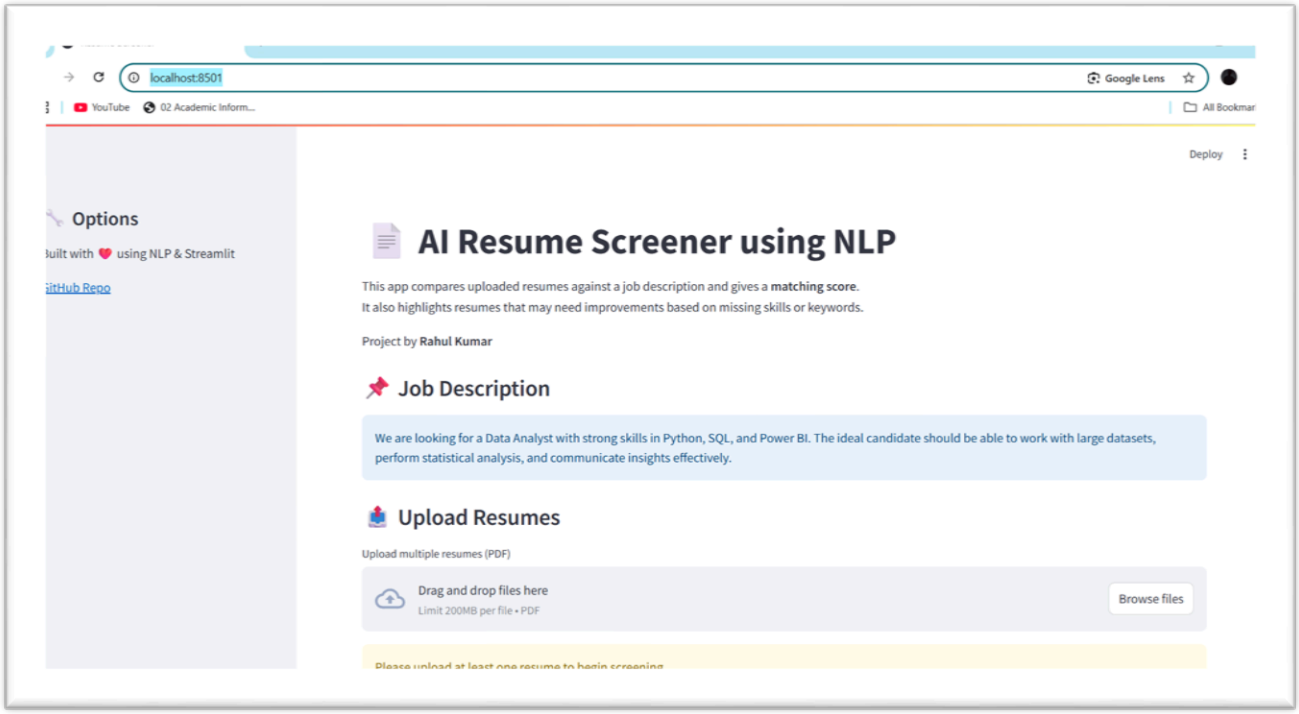
- app.pyhandles:
  - Streamlit UI
  - Uploading resumes
  - Displaying match score
  - Showing hints and export link
- utils.py handles:
  - Text extraction and preprocessing
  - Similarity calculation using TF-IDF
  - Candidate info extraction
  - Export to CSV

13. OUTPUT SCREENS (SCREENSHOTS + DESCRIPTIONS)

This section showcases the various screens that appear during the use of the AI Resume Screening and Ranking System. Each screen is explained with its **purpose**, **interaction flow**, and **expected output**.

Application Launch Screen

Screenshot: (screenshot of the app on initial load)



Description:

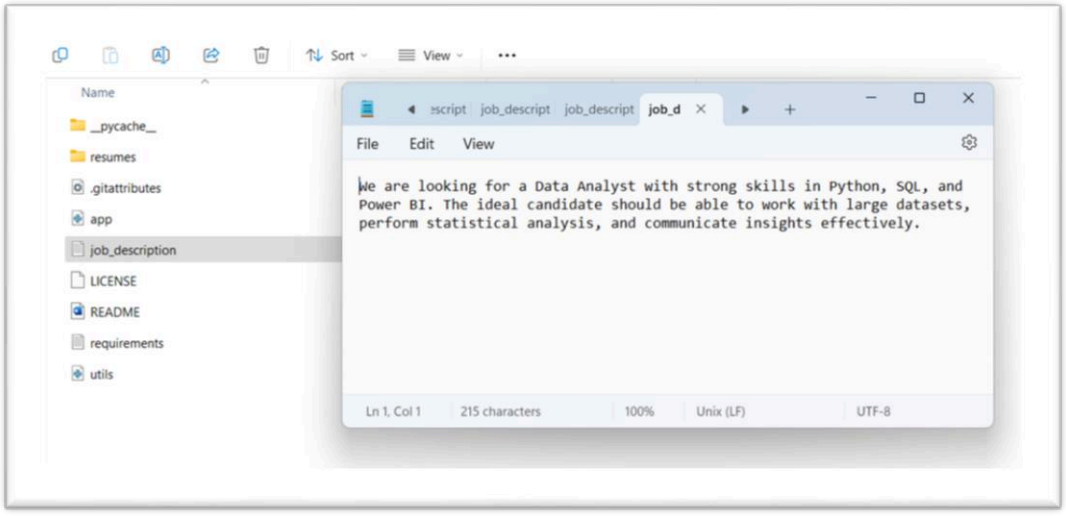
- This is the homepage of the Resume Screener.
- The title, project description, and sidebar are visible.
- Sidebar includes GitHub link and basic project metadata.

Components:

- Project Title
- Brief Introduction
- Sidebar Options

13.2 Job Description Display

Screenshot: (Screenshot of the loaded Job Description section)



Description:

- Once the app loads, it automatically reads job_description.txt.

- This content is displayed in an info box for the user to verify.

Expected Behavior:

- If job_description.txt is missing, an error message is shown.
- Text is cleaned and used for backend matching.

13.3 Resume Upload Panel

Screenshot: (Insert screenshot showing PDF file upload area)

Description:

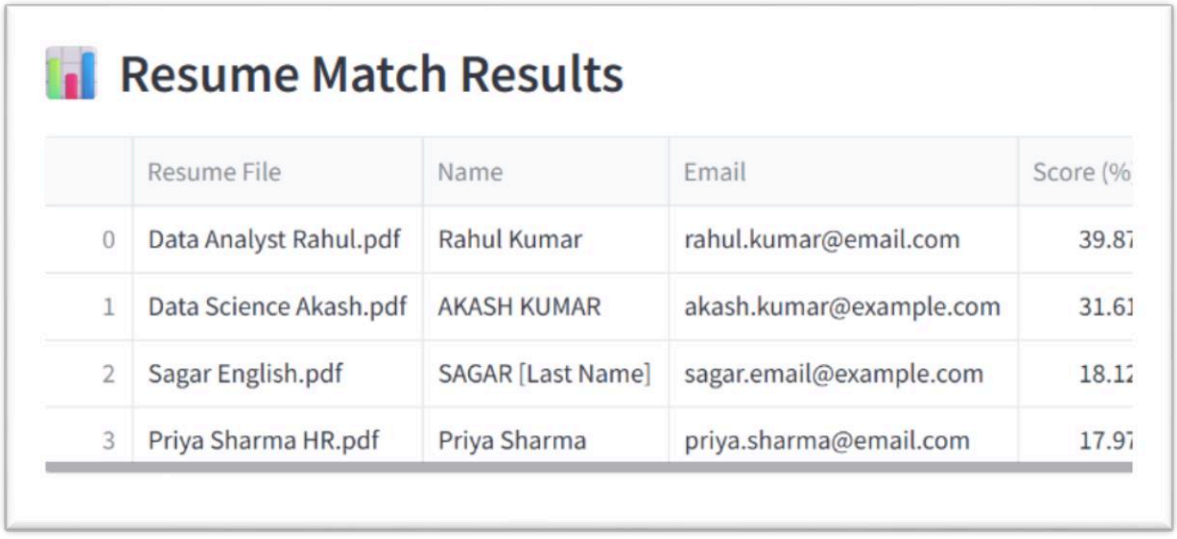
- Users can upload multiple resumes in .pdf format.
- Streamlit's file uploader allows batch processing of resumes.

User Actions:

- Click on "Browse files"
- Select multiple .pdf resumes

13.4 Resume Match Results Screen

Screenshot: (Screenshot of result display with progress scores)



Resume Match Results

	Resume File	Name	Email	Score (%
0	Data Analyst Rahul.pdf	Rahul Kumar	rahul.kumar@email.com	39.87
1	Data Science Akash.pdf	AKASH KUMAR	akash.kumar@example.com	31.61
2	Sagar English.pdf	SAGAR [Last Name]	sagar.email@example.com	18.12
3	Priya Sharma HR.pdf	Priya Sharma	priya.sharma@email.com	17.97

Description:

- Displays each uploaded resume along with:
  • Resume file name
  • Candidate's name and email

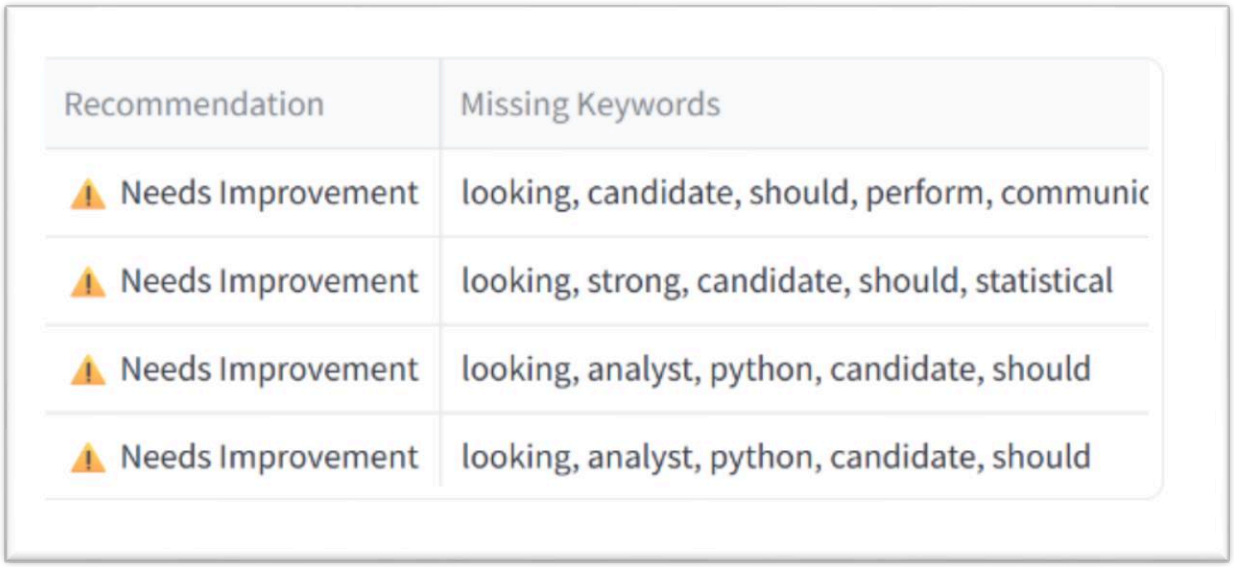
- Match score (out of 100%)
- Recommendation (Strong Match / Needs Improvement)
- Hints if resume is weak

Features:

- Text-based feedback and keyword hints

13.5 Missing Keyword Hints

Screenshot: (Screenshot of keyword hints)



Recommendation	Missing Keywords
⚠ Needs Improvement	looking, candidate, should, perform, communic
⚠ Needs Improvement	looking, strong, candidate, should, statistical
⚠ Needs Improvement	looking, analyst, python, candidate, should
⚠ Needs Improvement	looking, analyst, python, candidate, should

Description:

- Shows terms that appear in the JD but are missing in the resume.
- Helps candidate improve their resume by adding relevant skills.

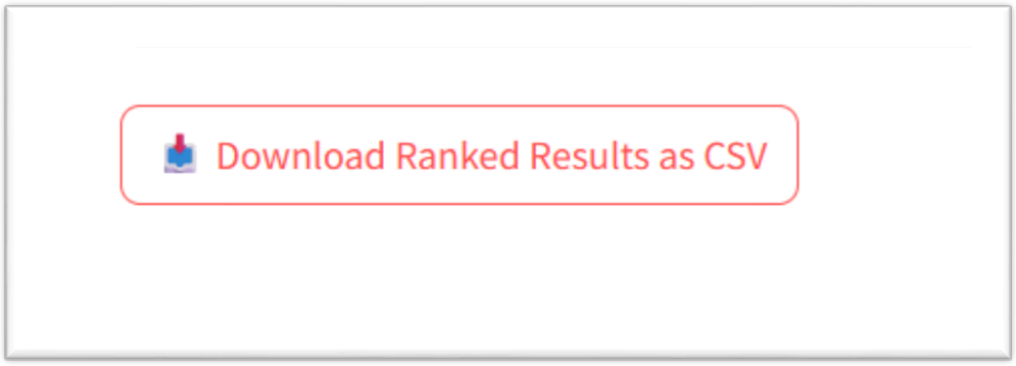
Logic:

- Keywords longer than 5 characters are checked.

- Top 5 missing words are shown if score < 70.

13.6 Export CSV Option

Screenshot: (Screenshot of download link)



Description:

- At the end of results, the system provides a button to download a .csv report.
- File includes: Resume filename, Candidate Name, Email, Match Score, Recommendation, and Keyword Hint.

CSV Format Example:

Resume	Name	Email	Score	Recommendation	Hint
Rahul_Kumar.pdf	Rahul Kumar	rahul@example.com	85.25	✅ Strong Match	
Soni_Resume.pdf	Soni Gupta	soni@domain.com	59.11	⚠️ Needs Improvement	development, analysis, tools

13.7 Error Screen: Missing JD File

Description:

- If job_description.txt is not found in the root directory, an error is shown.
- App halts further execution until a valid JD file is provided.

Message:

Job description file not found. Please ensure 'job_description.txt' is in the app folder.

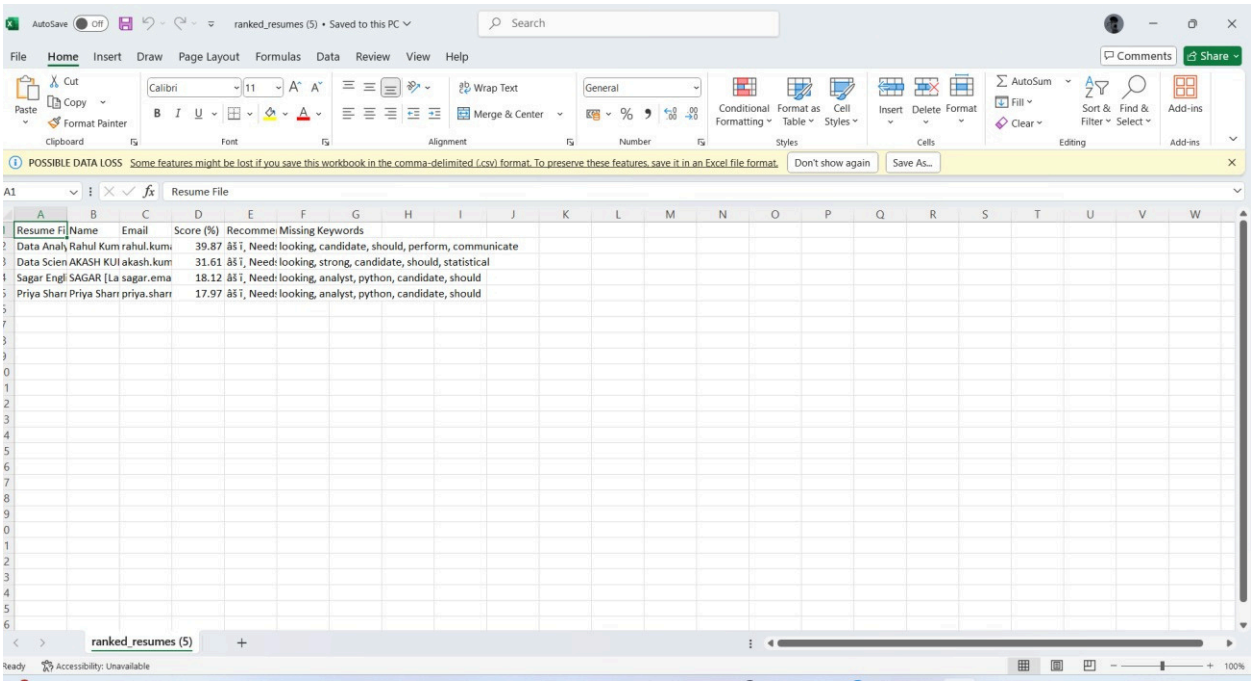
13.8 Error Screen: Invalid Resume Format

Description:

- If user uploads non-PDF files or corrupted PDFs, app shows an error (Streamlit default behavior).
- Only .pdf extensions are accepted.

13.9 Downloaded CSV Report (Sample View)

Screenshot: (Screenshot of opened CSV file in Excel)



Description:

- The downloaded report can be opened in Excel, Google Sheets, etc.
- It helps recruiters quickly review, sort, and filter candidates based on scores.

Summary of Output Screens:

Screen No.	Screen Title	Purpose
13.1	App Launch Screen	Starting point of interaction
13.2	Job Description View	Confirms loaded JD from text file
13.3	Resume Upload Panel	Allows uploading multiple PDF resumes
13.4	Resume Match Results	Displays score, name, recommendation
13.5	Keyword Hint Section	Identifies missing keywords from resumes
13.6	CSV Export Option	Allows downloading all results
13.7	Missing JD File Error	Error when JD file not found
13.9	CSV Sample Output	Preview of downloaded CSV in spreadsheet

14. LIMITATIONS

Eventhough the **AIResume Screening and Ranking System** demonstrates significant capabilities in automating resume filtering and analysis, it is essential to acknowledge its current limitations. These limitations point toward areas that may require future enhancements for improved efficiency, fairness, and scalability.

14.1 Accuracy of Similarity Score

- **TF-IDF Based Matching** is lexical and sensitive to wording differences.
- Two resumes with similar skillsets but different vocabulary may score differently.
- Semantic meaning is not deeply understood (no context awareness like BERT).

Example:

- A candidate writing "software design patterns" may score lower if the JD says "architecture principles", though both refer to related skills.

14.2 Limited Name and Email Extraction

- Name is extracted from the first line of the resume, which may not always contain the candidate's name.
- Email extraction relies on regex and may miss obfuscated formats (e.g., rahul [at] example [dot] com).
- Multiple email addresses are not handled efficiently.

14.3 No Section-Wise Analysis

- The system does not differentiate between resume sections such as **Experience**, **Skills**, **Projects**, etc.
- All content is treated as plain text.
- Weightage cannot be assigned based on resume structure or priority of keywords.

14.4 Language and File Type Constraints

- OnlyPDF filesare supported.
- Resumes in image format (e.g., scanned PDFs, JPEGs) cannot be processed.
- Non-English resumes are not supported (limited to basic English tokenizer).

14.5 No Real-Time Learning or Feedback

- Thesystemdoesn't learn fromrecruiterpreferences or outcomes.
- Machine learning model is static; it doesn't adapt based on real-world hiring results or changes in job market trends.

14.6 No Bias Mitigation Strategy

- Thesystemdoesn't currentlyaddress issues of bias (gender, location, age) in resume screening.
- It may unintentionally favor certain resume formats or terminologies common to specific regions or institutions.

14.7 UI/UX Limitations

- WhileStreamlit provides a quick UI, it lacks advanced design customization.
- Not suited for high-volume enterprise usage or mobile-optimized enterprise dashboarding.

14.8 No Resume Parser or Structured Output

- Unlike professionalresume parsers(e.g.,Sovren, Affinda), this system does not extract structured data like:
  • Work history
  • Education timeline
  • Certifications
- Lacks timeline analysis or date sorting.

14.9 No Cloud Storage or Database

- Uploadedresumesand generated results are not stored permanently.
- App is session-based; data is lost after reload.
- Not suitable for collaboration or cloud analytics without additional integration.

14.10 Job Description Must Be Static

- OnlyoneJD canbeused at atime.
- No support for multi-role job descriptions or bulk comparisons across job categories.

Summary Table of Limitations

No.	Limitation Area	Description
1 2 3 4 5 6 7 8 9 10	Similarity Model	Purely lexical, no semantic understanding
	Name/Email Detection	May be inaccurate for poorly formatted resumes
	No Section Awareness	Treats entire resume as unstructured text
	Language Support	Only English, no multilingual or OCR support
	Learning/Adaptability	Does not learn from recruiter feedback
	Bias Detection	No fairness filters or diversity analysis
	UI Flexibility	Limited design and customization via Streamlit
	Structured Parsing	No structured data like experience, education timeline
	Data Storage	No database or file system integration for persistence
		Supports only one job description at a time

15. FUTURE SCOPE

The AIResume Screeningand Ranking System has strong foundational capabilities, but there are several opportunities for enhancement to make it more powerful, scalable, and intelligent. The following future scope outlines possible directions for growth and innovation:

15.1 Integration with Deep Learning Models (BERT, GPT)

- Replace or enhance the TF-IDF model with **transformer-based models** like BERT or RoBERTa.
- These models understand context, semantics, and sentence meaning.
- Would significantly improve the accuracy of resume–JD matching.

Benefit:

- Better handling of synonyms, related concepts, and variations in phrasing.

15.2 Named Entity Recognition (NER)

- Use NLP techniques to **extract structured data** such as:
  - Name
  - Email
  - Skills
  - Certifications
  - Work Experience
  - Education
- Libraries like spaCy or transformers can be used for this.

Benefit:

- Enables deeper analytics, better filtering, and comparison between candidates.

15.3 Resume Parser Integration

- Integrate third-party resume parsing tools such as:
  - **Sovren API**
  - **Affinda Resume Parser**
  - **Rchilli**
- These tools provide structured parsing from PDFs, including timelines, job roles, and tech stacks.

Benefit:

- Improves accuracy and usability in enterprise HR environments.

15.4 Multi-JD Comparison

- Allowrecruitersto upload**multiple job descriptions**.
- Compare and rank resumes **against each JD separately**.
- Could be visualized via dropdowns or a dashboard-style interface.

Use Case:

- A company hiring for several roles can filter resumes accordingly.

15.5 Integration with Applicant Tracking Systems (ATS)

- Connect the tool with popular ATS platforms such as:
  - Zoho Recruit
  - Greenhouse
  - Workday
  - Lever

Benefit:

- Automates the entire candidate intake, ranking, and shortlisting workflow.

15.6 Dashboard and Analytics (Power BI / Streamlit Charts)

- Adddashboards to show:
  - Average match scores
  - Candidate ranking lists
  - Trends in skills gaps
- Tools: Plotly, Seaborn, Power BI embedding

15.7 Cloud Deployment (AWS / Azure / Heroku)

- Hostthe system oncloud platforms.
- Enable multiple recruiters to access the tool simultaneously.
- Secure file upload and database integration (MongoDB, PostgreSQL)

Benefit:

- Makes the system accessible from anywhere and more scalable.

15.8 OCR for Image-Based Resumes

- IntegrateOptical CharacterRecognition (OCR) using Tesseract or EasyOCR.
- Allows processing of scanned resumes and image-based content.

15.9 Resume Format Scoring

- Introduce rules forscoring: • Resume formatting • Use of bullet points • Length of sections

  Encourage professional presentation via score feedback.

15.10 Bias Mitigation and Fairness Filters

- UseAI FairnessToolkits to:
  - Check for gender bias in wording
  - Detect overused corporate phrases
  - Ensure diversity in selection

Ethical Use:

- Helps companies stay compliant and inclusive in hiring.

Summary Table of Future Enhancements

No.	Feature	Purpose
1 2 3 4 5 6 7 8 9 10	Transformer Models	Improve accuracy with semantic analysis
Mitigation Toolkit	Named Entity Recognition	Extract structured candidate data
	Resume Parser API	Provide timeline-based insights
	Multi-JD Support	Screen candidates for multiple job roles
	ATS Integration	Seamless connection with HR tools
	Data Visualization Dashboard	Recruiter-friendly insights and filters
	Cloud Deployment	Remote access, security, scalability
	OCR Integration	Support for image/scanned resume formats
	Resume Formatting Score	Improve resume design and presentation
		Promote fair, ethical, and inclusive hiring

16. REFERENCES

Thissection lists the resources, research papers, tools, libraries, and technologies referenced during the development of the **AI Resume Screening and Ranking System**.

16.1 Research Papers and Articles

ASurvey on Resume Screening with Machine Learning Techniques"
Journal of Artificial Intelligence Research and Advances, 2021.

• **"Contextualized Word Representations for Document Matching"** –
Devlin et al., BERT Research Paper, arXiv:1810.04805.

• **"Recruitment Automation using Artificial Intelligence"** –
Whitepaper by Deloitte, 2020.

• **TF-IDF and Similarity Measures** –
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

16.2 Python Libraries Used

• **Streamlit** – For building interactive web UI
https://streamlit.io/

• **Scikit-learn** – For TF-IDF vectorization and cosine similarity
https://scikit-learn.org/

• **PDFMiner** – For extracting text from PDF documents
https://pdfminersix.readthedocs.io/

• **re (Regex)** – For extracting email IDs and cleaning text
https://docs.python.org/3/library/re.html

• **Pandas** – For managing and exporting tabular data
https://pandas.pydata.org/

• **OS** – Standard Python library for file system management
https://docs.python.org/3/library/os.html

16.3 Tools and Platforms

•**VSCode** –Code editing and debugging
•**GitHub** – Version control and code repository
•**Google Colab / Jupyter Notebooks** – Testing NLP code snippets
•**Canva / Draw.io** – For creating diagrams like DFD, ER, and UML
•**Figma** – Optional prototyping of UI layout

16.4 Other Useful References

•**spaCyandNLTK Documentation**– For advanced NLP in future enhancements
https://spacy.io/
https://www.nltk.org/

• **Tesseract OCR** – For extracting text from scanned PDFs (future scope)
https://github.com/tesseract-ocr/tesseract

• **Hugging Face Transformers** – For context-based NLP models like BERT
https://huggingface.co/

• **OpenAI GPT APIs** – For intelligent resume summarization and scoring
https://openai.com/api/

16.5 Community and Forums

•**StackOverflow**–Problem-solvingfor bugs and integration issues
https://stackoverflow.com/

•**Medium Blogs** – Resume Screening AI Projects and ML/NLP insights
•**Reddit - r/MachineLearning / r/LanguageTechnology** – Use cases and ideas

Summary

These references have played a crucial role in the research, development, and documentation of the project. Future improvements can build upon these resources and integrate more advanced technologies for real-world enterprise readiness.