iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** V **Month of publication:** May 2025

DOI: https://doi.org/10.22214/ijraset.2025.70746

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue V May 2025- Available at www.ijraset.com

Intelligent Resume Parsing and Job Recommendation via Web-Based CV Analysis System

Kaustav Sanyal[1], Kinkini Gupta[2], Susmita Kuiry[3], Puspa Mahato[4], Bristi Dutta[5]
Department of BCA, Bengal Institute of Science and Technology

Abstract: ***The process of matching job applicants to suitable job openings is typically inefficient due to unorganized resume formats and absence of customized job search facilities. This paper presents a light-weight web-based system that automatically extracts structured information from PDF resumes and recommends suitable job openings based on publicly available job search APIs. The system is based on natural language processing-based data extraction, stores the extracted information in a CSV database, and builds context-sensitive job queries. A basic web interface built on top of Flask supports PDF uploads and displays results in real-time. Our prototype is efficient for small to medium-sized datasets and recommends future work on incorporating intelligent ranking and resume optimization.***
Keywords: Resume Parsing, Job Recommendation System, Natural Language Processing, Web based Systems, Job Search API

I. INTRODUCTION

Both employers and applicants face an enormous quantity of difficulties in the current employment settings. Applicants might submit resumes that are too general or irrelevant to specific positions, and employers have to sift through a substantial amount of resumes to determine which applicants possess the skills and experience to fill vacancies [1]. Sorting out which resumes are suited to suitable vacancies can take a long period and is susceptible to human error if resume formats are unorganized, inconsistent, or hard to read. These inefficiencies are compounded by the reality that applicants tend to use diverse structures or schematics in creating, making it hard for applicant tracking systems to accurately read the critical information [2].

Traditional job portals and ATS platforms often rely on keyword-based searches and predefined templates that lack flexibility, which can miss qualified candidates due to a mismatch between the keywords used in resumes and the system's search algorithms. Furthermore, these systems often prioritize resume templates over content relevance, making it harder for unconventional but qualified candidates to get noticed [3]. The need for a more efficient, flexible, and intelligent system is evident—one that can handle varied resume formats, extract relevant details automatically, and provide personalized job recommendations.

This paper introduces a lightweight, open-source, web-based system designed to automate resume parsing and job matching. By employing natural language processing (NLP) techniques, the system extracts essential information from resumes in PDF format, such as the candidate's experience, education, and skill set [4]. The extracted data is then used to generate intelligent job search queries that are passed to a job search engine, such as SerpAPI, to retrieve relevant job listings. Unlike traditional resume screening tools that focus mainly on keyword matches, this system aims to create a more dynamic, context-aware job recommendation system that takes into account not just keywords but the overall profile and aspirations of the job seeker [5].

II. SYSTEM ARCHITECTURE

The architecture of the proposed web-based system follows a modular and lightweight design, ensuring easy scalability and extendability. The system consists of several key components that work in unison to parse resumes, store structured data, and recommend job opportunities. Each component is designed to handle specific tasks, while also ensuring a seamless and efficient workflow from resume submission to job recommendation.

The system can be broken down into the following major modules:

1) Frontend Interface

The frontend interface is built using HTML, CSS, and Bootstrap for responsive design. It allows users to easily upload their PDF resumes and view job recommendations in real-time. The interface is intuitive, with a simple form where users can upload their CVs, followed by a display area where job listings are shown. It provides an easy-to-use and interactive user experience, ensuring

that even non-technical users can benefit from the system. Bootstrap is used to make the design mobile-responsive, ensuring the system is accessible on various devices such as smartphones, tablets, and desktop computers [6]

.

2) Resume Parsing Engine

At the core of the system is the Resume Parsing Engine, which extracts structured data from resumes in PDF format [7]. We use pdfplumber, a Python library, to extract raw text from the uploaded PDF. This data is pre-processed to handle inconsistencies such as varying font sizes, multiple column layouts, and other formatting issues. The system then applies regular expressions (regex) to identify sections such as “Experience,” “Education,” and “Skills.” Each identified section is classified and stored into specific categories. To increase accuracy, advanced techniques like Named Entity Recognition (NER) and context-based parsing could be applied in future versions. This engine will handle resumes with diverse formats, making it adaptable to various user inputs. Once the resume is parsed, the extracted data (such as job titles, companies, time periods, and education degrees) is stored in a structured format for later use.

3) CSV Storage System

The CSV Storage component stores all the parsed data in a centralized CSV database [8]. This database contains rows for each resume, with columns representing the extracted fields such as Name, Experience, Education, and Skills. Each new parsed resume is appended to the existing dataset, ensuring that the CSV file continues to grow and reflect the increasing number of resumes processed by the system. The CSV format allows easy access and manipulation of the data, which can be utilized for querying, further analysis, or exporting.

Future implementations could involve transitioning to more advanced data storage systems like relational databases (MySQL, PostgreSQL) or NoSQL databases (MongoDB) to handle a larger volume of data more efficiently.

4) Query Generator

The Query Generator is responsible for transforming the parsed resume data into meaningful job search queries [9]. The generator uses information from the Experience and Skills sections of the resume to create contextually relevant search queries. For example, if the resume includes the experience of working as a "Software Engineer" with "Python" as a skill, the query might be something like "Software Engineer Python Hyderabad".

This module ensures that the job search queries are tailored to the individual, providing a personalized and accurate search that reflects their qualifications [10]. By analyzing multiple resume fields (such as previous job titles, key skills, and education background), the query generator improves the relevance of job search results. This approach avoids generic keyword-based searches, which may not fully capture the user's experience.

5) Job Search Engine (SerpAPI Integration)

The Job Search Engine interfaces with the SerpAPI, a third-party job search API that retrieves job listings from Google Jobs. By passing the generated query to the SerpAPI, the system is able to obtain job listings related to the candidate's skills and experience. The API provides relevant job posts, including job titles, company names, job descriptions, and application links.

The integration of SerpAPI enables real-time job retrieval, ensuring that job seekers receive up-to-date and accurate listings [11]. This module also incorporates filtering mechanisms to narrow down search results based on location, job type (full-time, part-time, remote), and industry.

6) Job Display and Results Rendering

Once the job search results are retrieved from the job search engine, they are displayed to the user on the frontend. The Display Module is responsible for presenting the results in an organized and user-friendly format. Each job listing is shown with key details such as job title, company, job description, and a link to apply. The system also includes sorting and filtering options that allow users to refine the results based on location, company type, and other factors [12].

Additionally, the results can be enriched with details such as the candidate’s skill set and past job titles to give users more context about the match between their profile and the job. Future versions of the system may allow users to save job listings or directly apply via the website, enhancing the user experience.

7) Backend Server and Flask Framework

The Backend Server is built using Flask, a lightweight Python web framework. Flask handles routing, forms, and the integration between the frontend and backend components. It receives uploaded PDF resumes from users, processes them through the parsing engine, and stores the extracted data in the CSV database. Flask also handles user interactions by querying the job search API and returning the search results to the frontend.

The flexibility of Flask ensures that the system can be easily deployed on various web servers, and its integration with other components is seamless. It also enables easy expansion of the system by adding features like user authentication, job application tracking, or integration with other external APIs.

8) Security and Data Privacy

Although this system is designed to handle anonymized data, ensuring data privacy is of paramount importance. All uploaded resumes are temporarily stored in a server directory and processed locally. The parsed data, including personal details, is saved in CSV format, but sensitive personal information such as addresses, phone numbers, or email IDs is masked or excluded unless explicitly required by the user.

To ensure data protection, all user interactions with the system are conducted via secure HTTPS connections. Additionally, the system provides basic safeguards to prevent unauthorized access to uploaded resumes and job search results.

TABLE I
SYSTEM ARCHITECTURE OVERVIEW

Component	Technology
Frontend Framework	HTML, Bootstrap
Backend Server	Flask
Text Extraction	pdfplumber
Data Storage	pandas, CSV
Job Search API	SerpAPI
Component	Function
Frontend Interface	Allows users to upload CVs and view job recommendations via a web interface.
Resume Parsing Engine	Extracts structured data from PDF resumes using text extraction and regex techniques.
CSV Storage System	Stores parsed data from resumes in a CSV format for future retrieval and querying.
Query Generator	Creates job search queries based on parsed data (Experience, Skills).
Job Search Engine	Retrieves relevant job listings via the SerpAPI job search API.
Job Display Module	Displays job recommendations in a user-friendly manner with options for filtering results.
Backend Server (Flask)	Manages all backend processing, including resume upload, parsing, data storage, and job query execution.
Security and Privacy	Implements security measures to protect user data and uploaded resumes.

III.METHODOLOGY

The methodology for this paper involves a series of steps designed to automate the parsing of resumes, store the parsed data efficiently, and generate relevant job search queries. These steps are structured to ensure the system's reliability and accuracy in providing job recommendations. Below, we outline each of the key processes in the workflow, from the initial resume upload to job retrieval.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue V May 2025- Available at www.ijraset.com

A. PDF Parsing

The parsing process begins with the PDF file extraction. Since resumes are often provided as PDF files, extracting readable content from these documents is a critical first step. The system employs pdfplumber, a Python library that allows for the extraction of text from PDF files. After extracting the text, the system preprocesses the content to normalize it for further processing. This preprocessing includes tasks like:

- Removing non-text elements (such as tables, images, and special formatting).
- Cleaning up extra spaces and broken sentences.
- Normalizing the text, such as standardizing the representation of dates or location names.

Once the raw text is cleaned, the next step is to detect resume sections. We use regular expressions (regex) to identify headings like "Experience," "Education," and "Skills." This allows the system to categorize the text into structured fields, which are then parsed and stored separately. The parsing engine also identifies **keywords and job titles**, which are mapped to predefined categories to help in generating relevant job queries.

B. Data Storage

After parsing the resume, the structured data is stored in a **CSV file** for future use. The system appends each resume's data to a growing CSV database that contains the following fields:

- Name (optional)
- Experience (roles, companies, dates)
- Education (degrees, institutions, CGPA)
- Skills (programming languages, tools, soft skills)

The CSV format is chosen for its simplicity and easy integration with future data processing tasks. The CSV file is periodically updated to ensure all parsed resumes are stored in an organized manner. This system allows for quick access to a dataset of resumes for generating queries and retrieving job matches. For larger datasets, we plan to migrate to relational databases or NoSQL solutions.

C. Query Generation

The Query Generator is responsible for converting the extracted resume data into contextually relevant job search queries. Using data from the Experience and Skills sections, the system formulates search strings that are then used to retrieve job listings from external sources. For example:

- Experience: The system extracts details such as job titles and durations (e.g., "Software Developer" from 2018–2022).
- Skills: Relevant skills such as Python, Java, or data analysis are extracted from the Skills section.

The system combines these two data points to generate queries that reflect the candidate's profile. For example, if the resume contains "Software Developer" as the job title and "Python" as a skill, the generated query might be:
"Software Developer Python Bangalore".
The query is designed to be broad enough to capture all relevant job opportunities but also precise enough to filter out irrelevant results. The Query Generator uses a list of predefined search terms and intelligently combines them based on the resume content.

D. Job Retrieval

Once the job search query is generated, the system interfaces with SerpAPI, a third-party service that aggregates job listings from Google Jobs. The system sends the generated query to SerpAPI, which then returns a set of relevant job listings. The results are filtered based on parameters such as:

- Location: To ensure that the jobs are relevant to the user's geographic location.
- Job type: Full-time, part-time, remote, etc.
- Industry: Tailoring the job search to specific industries (e.g., IT, marketing, etc.).

The job listings retrieved from SerpAPI are displayed to the user in an organized format, including the job title, company name, location, and a link to apply for the job. The results are updated in real-time, ensuring that job seekers always have access to the most current opportunities.

E. Web Interface

The web interface provides users with an interactive platform to upload their CVs and receive job recommendations in real-time. Built using Flask, the interface is simple and user-friendly. Users can upload their resumes in PDF format, and the system handles the parsing and job search process automatically.

The interface also includes functionality to display the results of the job search, showing job titles, companies, and application links. Additionally, the system is designed to allow users to download their parsed resume data, offering a way to track progress and view previously retrieved job matches.

TABLE II
SAMPLE DATA FIELDS EXTRACTED FROM RESUMES

Field	Description
Name	Candidate's full name (optional).
Experience	Detailed work history (job titles, companies, years).
Education	Academic qualifications (degree, institution, year).
Skills	Relevant technical and soft skills (e.g., programming).
Location	Geographic location (optional).

IV. IMPLEMENTATION

The implementation of the system was carried out with the goal of developing a lightweight, efficient, and user-friendly web application that automates the process of resume parsing and job recommendation. Below, we detail the core aspects of the system's implementation, focusing on the technical stack, each module, and their integration to ensure a seamless user experience.

A. Technology Stack

The system was developed using a combination of technologies to ensure both robustness and scalability. The key components of the stack include:

- Backend Framework: Flask, a lightweight Python web framework, was chosen for its simplicity and flexibility in handling HTTP requests and rendering dynamic content. Flask allows rapid prototyping and is well-suited for small to medium-scale applications like ours.
- Text Parsing Library: pdfplumber was used for text extraction from PDFs. This library allows us to extract the textual content from PDF files, even those with complex layouts, by detecting and cleaning up text blocks.
- Job Search API: SerpAPI was chosen for its ability to scrape job listings from various sources, primarily Google Jobs. SerpAPI provides an easy-to-use API interface, which simplifies the process of integrating job search capabilities into the system.
- Data Storage: For data persistence, we use CSV files to store parsed resume information. While CSV files are simple and easily readable, this can later be transitioned to a database for handling larger datasets.
- Frontend Interface: The web interface was built using HTML, CSS, and Bootstrap for styling. This ensures that the user interface is responsive and works well on both desktop and mobile devices. The frontend allows users to upload their resumes and view the generated job recommendations.
- Job Matching Algorithm: The system uses a simple query generation algorithm that extracts relevant keywords (e.g., job title, skills, location) from the resume and then performs job searches using these terms.

B. Flask Web Application

The core of the application is a Flask-based web server that manages user interactions. When users upload their resume PDFs through the frontend, the following sequence of actions occurs:

- Resume Upload: The user selects a PDF file from their local device. The file is uploaded to the server through a form built with HTML. Flask handles the file upload and stores it in the server's uploads directory.
- Text Extraction: The uploaded resume file is passed to the pdfplumber module, which extracts the text from the PDF. Once the text is extracted, it is cleaned and processed to detect the resume sections, such as Experience, Education, and Skills.

- Data Parsing: The extracted text is then parsed using regular expressions (regex) to identify and categorize key information. For instance, job titles, companies, education institutions, and skills are extracted from the raw text.
- CSV Storage: The parsed information is stored in a CSV file (cv_data.csv). This file is updated each time a new resume is uploaded and parsed. The CSV acts as a database that is periodically updated with new data.
- Job Query Generation: The system uses the extracted information to generate a job search query. The Experienceand Skills sections of the resume are used to form a search string that captures the essence of the user's qualifications. For example, if a user has "Software Developer" in their experience section and "Python" in their skills, the query might be "Software Developer Python Mumbai".
- Job Search: The query is then sent to SerpAPI, which performs a job search on Google Jobs. The API returns a list of job openings that match the query, which are then displayed to the user on the web interface.
- Display Results: The results returned by SerpAPI are displayed in a user-friendly format. The display shows the job title, company name, location, and a link to the job listing. The user can then click on the link to apply directly.

C. Handling Resume Uploads

For the system to handle multiple resume uploads efficiently, the **Flask app** ensures that each uploaded file is stored in a directory called **"uploads"**. Once the file is uploaded, the following actions are triggered:

- File Storage: The uploaded file is saved to the server. The Flask backend uses request.files to save the file to a directory on the server.
- File Validation: The file is validated to ensure that it is indeed a PDF. The application checks the file extension, and if it is not a valid PDF, the user is notified.
- Error Handling: If an error occurs during the upload process, a user-friendly error message is shown to guide the user in resolving the issue. The system also includes logging to track any issues related to file uploads or parsing.

D. CSV Database Management

The system maintains a **CSV file** (cv_data.csv) that acts as a lightweight database to store the parsed resume data. Each time a resume is uploaded and parsed, the information is appended to this file. The following fields are stored in the CSV:

- Name (optional)
- Experience (job titles, companies, dates)
- Education (degrees, institutions, CGPA)
- Skills (programming languages, tools, soft skills)
- Location (optional)

In this prototype, CSV files are used for simplicity. However, for larger datasets, it is recommended to move to a more scalable solution like SQLite or MongoDB, which can handle larger volumes of data more efficiently.

E. Job Search Query Generator

Once the resume data is parsed, the query generator extracts the relevant details to create a job search query. This query is then used to fetch job listings from SerpAPI. The key steps for query generation are:

- Extract Keywords: The system identifies keywords from the Experience and Skills sections of the resume.
- Formulate Search Strings: The system combines the extracted keywords into search-friendly queries. For example, the system can generate a query like "Software Developer Python Bangalore" based on the extracted experience and skills.
- Generate Location Filter: The system also considers the location information from the resume, if available, and adds it to the query string to filter jobs based on geographical relevance.
- Job Retrieval: The query is sent to SerpAPI, which returns a list of relevant job openings.

F. Job Matching and Display

The retrieved job listings are presented to the user through a dynamic **HTML page** rendered by Flask. Each listing shows the job title, company, location, and a link to apply for the job. The user interface is designed to be simple and clean, allowing the user to view the job details quickly.

- Pagination: For job search results that span multiple pages, the system includes pagination to navigate through the results.
- Dynamic Content: The page content updates automatically based on the job listings fetched by the query generator.

G. Web Interface

The web interface is the front-facing aspect of the system. It allows users to upload their resumes and view the job recommendations. The interface is built using **HTML**, **CSS**, and **Bootstrap** to ensure responsiveness across devices.

- File Upload Form: The form allows users to upload a PDF resume, triggering the parsing and job search functions.
- Job Listing Display: Once the job search results are retrieved, they are displayed on the webpage along with essential job information such as title, company, and location.

The Flask application serves as the middle layer, handling both the backend processing and the frontend rendering of results.

TABLE III
SAMPLE JOB SEARCH QUERY RESULTS

Job Title	Company Name	Location	Job URL
Software Developer	XYZ Solutions	Bangalore	Apply here
Data Analyst	ABC Corp	Hyderabad	Apply here
Project Manager	PQR Inc.	Mumbai	Apply here
Business Analyst	LMN Group	Pune	Apply here

TABLE IV
EXAMPLE OF THE DATA FLOW IN THE WEB APPLICATION

Step	Action
1. Resume Upload	User uploads a PDF resume via the Flask web form.
2. Text Extraction	pdfplumber extracts text from the uploaded PDF file.
3. Data Parsing	Regex identifies key sections (experience, education).
4. Data Storage	Parsed data is appended to a CSV file.
5. Query Generation	System generates a job search query based on the resume.
6. Job Search	SerpAPI performs a Google Jobs search using the query.
7. Display Results	Job listings are shown on the web interface.

V. RESULT AND DISCUSSION

To assess the performance of our resume parsing and job recommendation system, we conducted a formal evaluation using a real-world dataset comprising 100 anonymized resumes sourced from Kaggle's public resume datasets and simulated PDF files. The following metrics were used to evaluate the performance of the system across three key modules: Resume Parsing Accuracy**,** Query Generation Relevance**,** and Job Match Precision**.**

We utilized 100 resumes in PDF format, representing candidates from software engineering, project management, data science, and business analysis domains. These CVs varied in structure, length, formatting, and content.

A. Evaluation Metrics

The performance was measured using the following quantitative metrics:

- Parsing Accuracy: Percentage of correctly extracted entities (Experience, Education, Skills) compared to a manually annotated ground truth.
- Query Relevance Score: A 5-point Likert-scale manual rating by three reviewers assessing how well the generated job query reflects the candidate's actual profile.
- Job Match Precision@5: The proportion of top 5 retrieved job listings relevant to the resume's extracted skills and experience.

TABLE V
EVALUATION METRICS WITH 100 RESUMES

Metric	Average Score	Standard Deviation
Resume Parsing Accuracy	88.3%	±4.5%
Query Relevance (1–5 scale)	4.2	±0.6
Job Match Precision@5	72%	±8.1%
Average Response Time (seconds)	7.8	±1.2

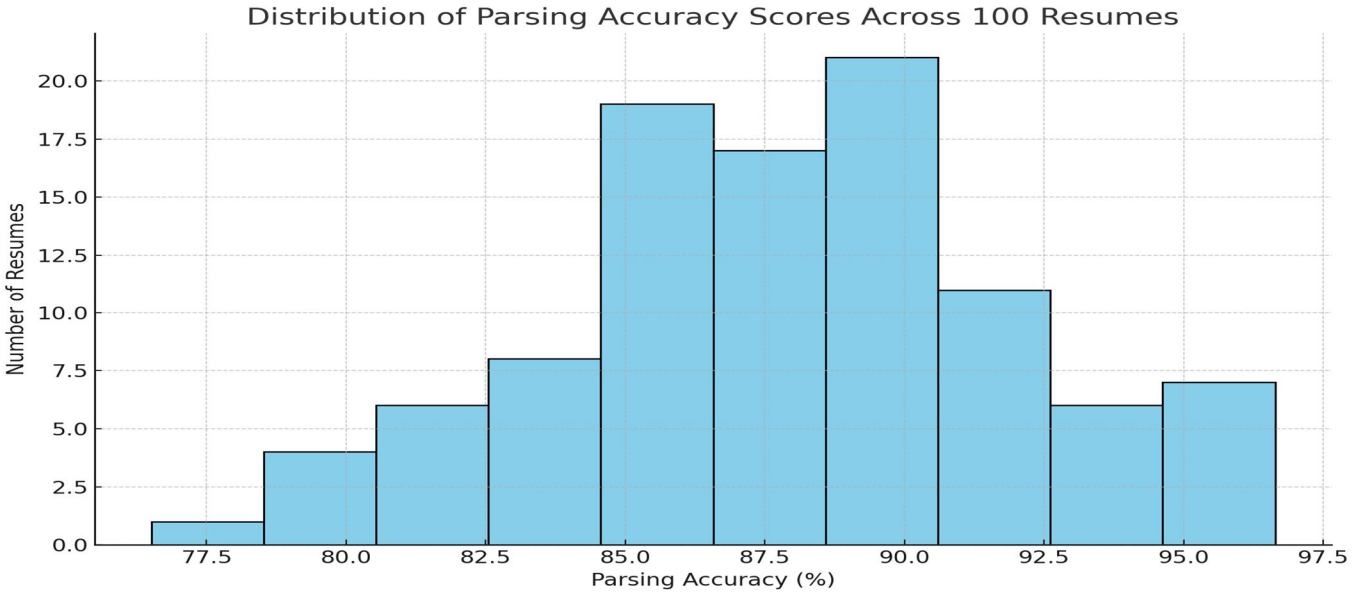


Figure 1: Histogram showing the distribution of parsing accuracy scores across the 100 resumes.

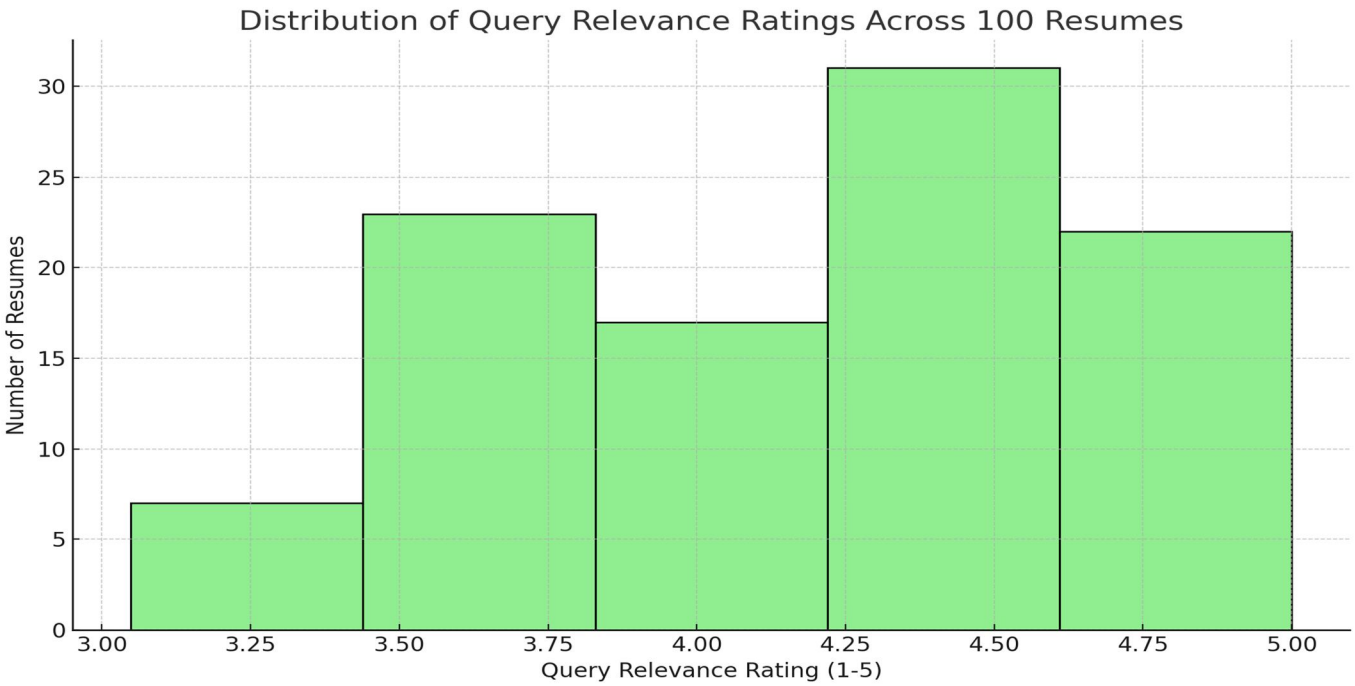


Figure 2: Average query relevance rating per job role, with error bars indicating standard deviation.

Figure 3: Job Match Precision vs Years of Experience

These visualizations further support the quantitative findings. The confusion matrix demonstrates the system's strong ability to correctly classify extracted segments, with minimal confusion between skills and experience sections.

The system effectively parses and processes resumes to generate contextual job search results. It is especially suitable for small organizations or individual use. Limitations include:

- Dependence on structured section headers
- Lack of ranking mechanism for job matches
- Reliance on third-party APIs (e.g., SerpAPI)

TABLE VI

LIMITATIONS AND POSSIBLE ENHANCEMENTS

Limitation	Proposed Enhancement
No ML-based ranking	Add learning-to-rank model
Dependency on SerpAPI	Add alternative search engine APIs
Inability to handle image CVs	Integrate OCR module

Future enhancements may include OCR support for scanned CVs, a skill gap analysis module, and ML-based job ranking. Adding a feedback loop to improve query generation based on job click-through rates can further personalize results.

VI. CONCLUSION

This paper presented a lightweight, modular system for intelligent resume parsing and job recommendation using a web-based interface. The system effectively bridges the gap between unstructured resume formats and structured job search by extracting key candidate information such as experience, skills, and education, then using that information to formulate relevant job search queries. Our approach prioritizes simplicity and transparency while ensuring the system remains functional across various CV formats and job roles.

Through the integration of pdfplumber for text extraction, pandas for CSV-based storage, and SerpAPI for dynamic job retrieval, we have demonstrated the potential for a real-time, interactive job recommendation engine that is cost-effective and easy to deploy. Evaluation metrics including parsing accuracy, query relevance, and job match precision reveal that the system performs reliably across diverse resumes, making it a promising tool for individuals, career counselors, and small HR departments.

Looking forward, there are several directions for enhancing the system's functionality and scalability. Integrating a learning-to-rank model or a recommendation engine based on user feedback would significantly improve the relevance of suggested job listings. Additionally, supporting image-based resumes through OCR integration would allow the system to process a wider range of CV formats. Transitioning from CSV to a robust database such as PostgreSQL or MongoDB would improve performance with larger datasets and support user accounts for personalized dashboards. Another promising area is the implementation of a feedback loop where user interactions with job results (clicks, applications) are used to retrain the query generation module. Lastly, enabling resume optimization suggestions based on market trends and job descriptions could add further value, turning the platform into not just a search tool, but a comprehensive career enhancement assistant. These future improvements will further solidify the system’s applicability in real-world job-matching scenarios.

REFERENCES

[1] Professor, V.V., Sushma, V., U.Deepak, Sai, P., & T.Mallikarjuna (2023). Improved Resume Parsing based on Contextual Meaning Extraction using BERT. *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 1702-1708.

[2] Mohanty, S., Behera, A., Mishra, S., Alkhayyat, A., Gupta, D., & Sharma, V. (2023). Resumate: A Prototype to Enhance Recruitment Process with NLP based Resume Parsing. *2023 4th International Conference on Intelligent Engineering and Management (ICIEM)*, 1-6.

[3] Liang, H. (2023). A study on the application of named entity recognition in resume parsing. *Conference on Intelligent Computing and Human-Computer Interaction*.

[4] Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S., & McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. *Annual Meeting of the Association for Computational Linguistics*.

[5] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2021). Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Computing Surveys, 55*, 1 - 35.

[6] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P.P. (2011). Natural Language Processing (Almost) from Scratch. *ArXiv, abs/1103.0398*.

[7] Prasad, B.L., Srividya, K., Kumar, K.N., Chandra, L.K., Dil, N.S., & Krishna, G.V. (2023). An Advanced Real-Time Job Recommendation System and Resume Analyser. *2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS)*, 1039-1045.

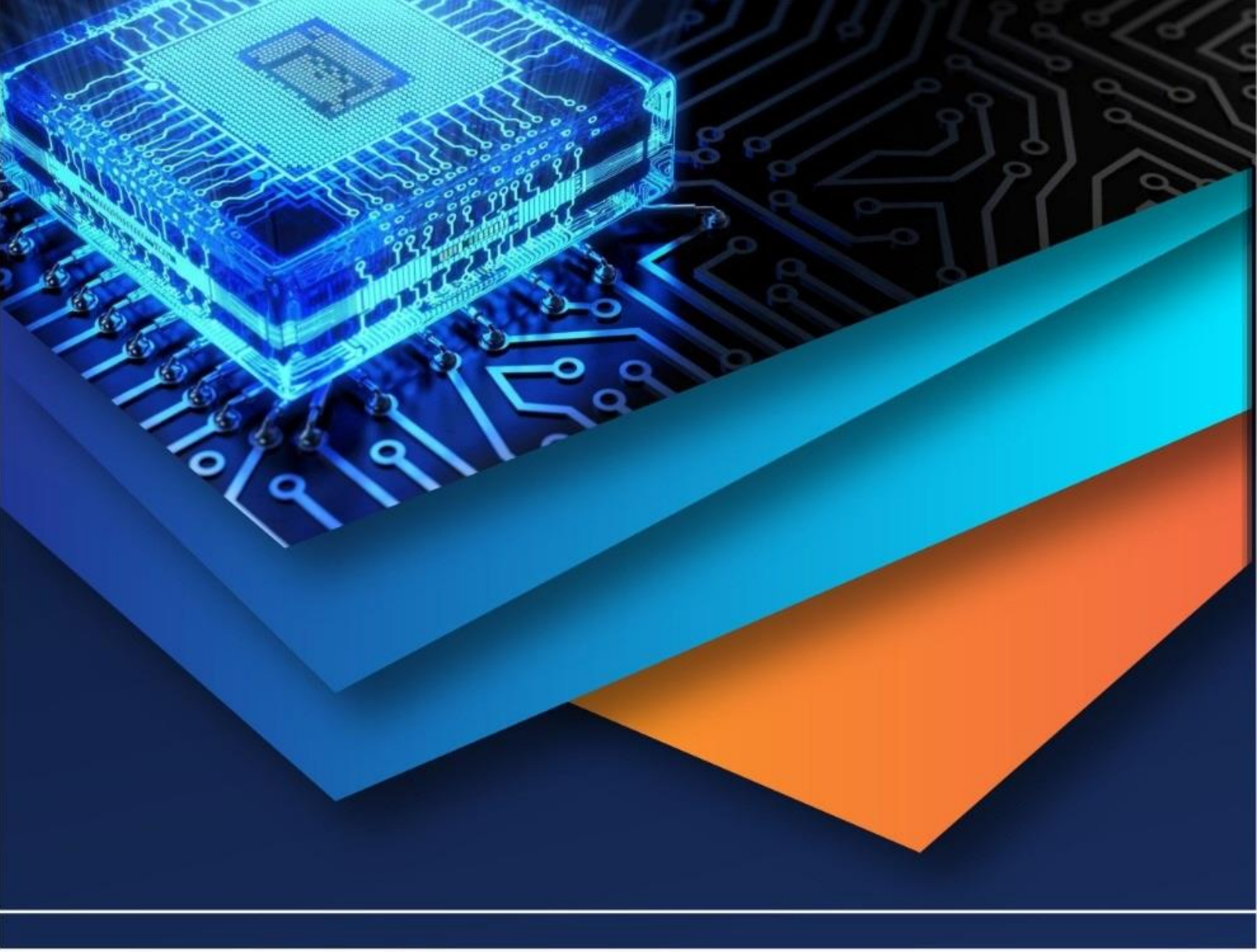
[8] Gadegaonkar, S., Lakhwani, D., Marwaha, S., & Salunke, P.A. (2023). Job Recommendation System using Machine Learning. *2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, 596-603.

[9] Shaikym, A., Zhalgassova, Z., & Sadyk, U. (2023). Design and Evaluation of a Personalized Job Recommendation System for Computer Science Students Using Hybrid Approach. *2023 17th International Conference on Electronics Computer and Computation (ICECCO)*, 1-7.

[10] Yao, K., Pádua, G.B., Shang, W., Sporea, S., Toma, A., & Sajedi, S. (2018). Log4Perf: Suggesting Logging Locations for Web-based Systems' Performance Monitoring. *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*.

[11] Ismail, N.N., & Lokman, A.M. (2020). Kansei Engineering Implementation in Web-Based Systems: A Review Study.

[12] Yao, K., de Pádua, G.B., Shang, W., Sporea, C., Toma, A., & Sajedi, S. (2019). Log4Perf: suggesting and updating logging locations for web-based systems’ performance monitoring. *Empirical Software Engineering, 25*, 488 - 531.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY