AI Resume Screening and Job Description Matching System

[1]Bhupesh, [2]Ravi Kumar , [3]Rajendra Singh
[1]*Department of Computer Science and Engineering, Raffles University, Neemrana, Alwar, Rajasthan, India*
[2]*Assistant Professor, Department of Computer Science and Engineering, Raffles University, Neemrana, Alwar, Rajasthan, India*
[3]*Dean, Department of Computer Science and Engineering, Raffles University, Neemrana, Rajasthan, India*

***Abstract*—The modern recruitment landscape is overwhelmed by the sheer volume of job applications received for every available position, creating significant bottlenecks in the hiring process. Human recruiters spend a disproportionate amount of time manually reviewing resumes, leading to delays, inconsistencies, and potential bias in candidate evaluation. This paper presents the design, development, and evaluation of an AIpowered Resume Screening and Job Description Matching System that leverages Natural Language Processing (NLP) and machine learning-based text similarity techniques to automate candidate evaluation. The proposed system accepts candidate resumes in PDF format, extracts technical skills, qualifications, certifications, and experience using an NLP pipeline built on NLTK and spaCy, and compares the extracted information against job description requirements through keyword matching and TF-IDF cosine similarity scoring via Scikit-learn. The backend is developed using Python and Flask, while the frontend is constructed with HTML5, CSS3, and JavaScript, delivering a responsive and recruiter-friendly web interface. The system generates a quantitative skill match percentage alongside detailed insights into matched and missing skills, enabling data-driven hiring decisions. Experimental testing across multiple job profiles and resume formats demonstrated reliable skill extraction accuracy and match percentage consistency. The modular system architecture supports future enhancements including transformer-based semantic matching, multi-resume batch processing, AI-based candidate ranking, and cloud deployment. This work contributes a practical, scalable, and deployable solution to the growing field of intelligent recruitment and talent acquisition automation.**

***Index Terms*—Resume Screening, Job Description Matching, Natural Language Processing, Skill Extraction, Recruitment Automation, Flask, Python, TF-IDF, Cosine Similarity, Text Mining**

I. INTRODUCTION

The global job market has grown increasingly competitive, with organizations receiving hundreds or thousands of resumes for each advertised position. According to industry estimates, a recruiter spends an average of six to seven seconds scanning a single resume during initial screening, making it virtually impossible to conduct a thorough and consistent evaluation at scale. This manual process is not only timeintensive but also susceptible to human bias, fatigue-induced errors, and inconsistency in evaluation criteria across different recruiters within the same organization.

The emergence of Artificial Intelligence (AI) and Natural Language Processing (NLP) has opened transformative possibilities for automating and improving the recruitment workflow. Intelligent systems capable of reading, understanding, and evaluating unstructured resume documents against structured job requirements offer recruiters a powerful tool to reduce workload, improve consistency, and make more informed shortlisting decisions. These technologies represent a convergence of information retrieval, text analytics, and machine learning that is increasingly being adopted across enterprise hiring platforms.

This paper presents the AI Resume Screening and Job Description Matching System, a web-based application developed using Python, Flask, and NLP libraries. The system accepts candidate resumes uploaded in PDF format and processes them through a multi-stage NLP pipeline to extract relevant skills, qualifications, and experience. The extracted candidate profile is then compared against a recruiter-provided job description using keyword-based overlap analysis and TF-IDF cosine similarity scoring, producing a quantitative match percentage and a detailed skill gap analysis report.

The significance of this work lies in its practical applicability. Unlike research systems that rely on proprietary datasets or computationally expensive deep learning models, the proposed system is built entirely on opensource tools and lightweight NLP techniques, making it accessible for deployment in academic institutions, small enterprises, and startup recruiting environments. The system's modular architecture further ensures that individual components can be upgraded or replaced as more advanced techniques become available.

The primary contributions of this paper are as follows. First, a complete end-to-end system architecture for automated resume screening is presented, covering ingestion, parsing, NLP processing, matching, and result generation. Second, a hybrid matching algorithm combining keyword overlap analysis and TF-IDF cosine similarity is proposed and evaluated. Third, a recruiter-facing web application is implemented and functionally tested across multiple job profiles and resume formats. Fourth, challenges encountered during development are documented with practical solutions, providing implementation guidance for future researchers.

The remainder of this paper is organized as follows. Section II reviews related literature in resume parsing, job matching, and recruitment automation. Section III describes the system methodology including the NLP pipeline and matching algorithm. Section IV details the implementation including architecture, module design, and the user interface. Section V presents experimental results, performance observations, and a discussion of findings. Section VI addresses limitations of the current system. Section VII outlines directions for future work. Section VIII concludes the paper.

II. RELATED WORK

Research in automated resume screening and job description matching spans several decades, evolving from simple keyword-based systems to sophisticated deep learning approaches. This section reviews the most relevant prior work and situates the proposed system within the existing literature.

A. Early Keyword-Based Approaches

Early resume screening systems relied primarily on keyword matching, where a predefined list of required skills was searched within resume text. While simple and fast, these systems suffered from low recall when candidates used synonyms or varied terminology. Raza et al. [1] demonstrated that pure keyword matching approaches achieved only 58% precision in skill identification across diverse resume formats, highlighting the need for more robust text understanding techniques.

B. Machine Learning-Based Resume Classification

Celik and Balaban [2] proposed a resume classification system using Support Vector Machines (SVM) trained on a labeled dataset of resumes categorized by job function. Their system achieved classification accuracy exceeding 85% across ten job categories. However, the approach focused on classifying resumes into predefined categories rather than quantifying alignment with a specific job description, limiting its usefulness for targeted job matching.

Inspired by information retrieval techniques, Maheshwari et al. [3] developed a recommendation system using TF-IDF vectorization and cosine similarity to match candidate profiles with job postings extracted from online portals. Their work demonstrated that TF-IDF-based similarity provided a reliable baseline for jobcandidate alignment scoring. However, the system was evaluated only on structured profile data and did not address unstructured PDF resume parsing.

C. NLP-Based Skill Extraction

The challenge of extracting skills from free-form resume text has been extensively studied. Zhao et al. [4] proposed a conditional random field (CRF) model for named entity recognition in resumes, identifying entities such as skills, job titles, educational qualifications, and organizations.

Their approach outperformed rule-based systems in handling contextual variations in skill expression.

Zhang et al. [5] advanced this work by applying Bidirectional LSTM networks for resume entity recognition, exploiting sequential context to improve extraction accuracy. Their model achieved an F1 score of 0.89 on a benchmark resume dataset, demonstrating the superiority of sequence modeling over earlier bag-of-words approaches for this task.

D. Semantic Job Matching

More recently, transformer-based language models have been applied to semantic job matching. Devlin et al. [6] introduced BERT, a pre-trained deep bidirectional transformer model that has since been fine-tuned for numerous NLP tasks including job-candidate matching. Qin et al. [7] demonstrated that BERT-based representations significantly outperformed TF-IDF approaches in capturing semantic similarity between job descriptions and candidate profiles, particularly in handling skill synonyms and contextual equivalences.

While these deep learning approaches offer superior performance, they require substantial computational resources and large annotated training datasets that may not be available in practical deployment scenarios.

E. Positioning of the Proposed System

The system presented in this paper occupies a deliberate middle ground between simplistic keyword matching and resource-intensive deep learning approaches. By combining NLTK and spaCy-based NLP with TF-IDF cosine similarity, the system achieves reliable skill extraction and matching performance while remaining lightweight, open-source, and deployable without GPU infrastructure. This makes it particularly well-suited for academic, small enterprise, and prototype deployment contexts.

III. METHODOLOGY

A. System Overview

The proposed system follows a structured five-stage pipeline: resume ingestion, text extraction and preprocessing, skill and keyword extraction, job description comparison, and result generation. Each stage operates independently as a module, communicating through well-defined interfaces. This modular design ensures maintainability and allows individual components to be enhanced without disrupting the overall system.

The complete processing workflow is as follows:

Resume Upload → PDF Text Extraction → NLP Preprocessing → Skill Extraction → Job Description

Processing → Keyword Overlap Analysis → TF-IDF Cosine Similarity Scoring → Match Percentage Calculation → Result Display

B. Resume Ingestion and PDF Parsing

Candidate resumes are accepted in PDF format through the web interface. The Flask backend receives the uploaded file and stores it temporarily in a designated uploads directory. PyPDF2, a pure Python PDF processing library, is used to read the PDF and extract raw text content page by page. The extracted text is then concatenated into a single string for downstream processing.

PDF parsing introduces several challenges, primarily due to the diversity of resume formatting styles. Resumes with multi-column layouts, tables, embedded images, or decorative fonts may produce fragmented or disordered extracted text. To address this, a text normalization routine is applied to remove redundant whitespace, special characters, and non-ASCII symbols, producing a clean text block suitable for NLP analysis.

C. NLP Preprocessing Pipeline

The cleaned resume text and the job description text are both passed through an identical NLP preprocessing pipeline consisting of the following steps:

Tokenization: The input text is split into individual tokens using NLTK's word_tokenize function, which handles punctuation and contraction splitting appropriately.

Case Normalization: All tokens are converted to lowercase to ensure case-insensitive matching between resume and job description content.

Stop-word Removal: NLTK's English stop-word list is used to filter out common function words such as "the", "and", "is", and "at" that carry no semantic value for skill matching purposes.

Lemmatization: The WordNetLemmatizer from NLTK reduces tokens to their base lexical forms, normalizing variations such as "developing" to "develop", "libraries" to "library", and "algorithms" to "algorithm". This step improves matching recall by consolidating morphological variants of the same concept.

Named Entity Recognition: spaCy's pre-trained en_core_web_sm model is applied to the preprocessed text to identify named entities including organizations, technologies, and proper nouns that may represent relevant skills or tools.

D. Skill Extraction

Skill extraction is performed using a two-pronged approach. First, a comprehensive skill dictionary containing over 500 technical terms, programming languages, frameworks, tools, and professional qualifications is maintained. This dictionary is used to perform exact and partial matching against the preprocessed token list, identifying discrete skills present in the document.

Second, spaCy's dependency parsing is used to extract skill phrases that span multiple tokens, such as "machine learning", "natural language processing", "RESTful API design", and "cloud infrastructure management". This phrase-level extraction captures compound skills that would be missed by single-token matching alone.

The union of dictionary-matched skills and phrase-extracted skills forms the candidate skill set for the resume and the required skill set for the job description.

E. Matching Algorithm

The matching module computes candidate-job alignment through two complementary scoring mechanisms.

Keyword Overlap Score: The candidate skill set extracted from the resume is compared against the required skill set extracted from the job description. The overlap score is computed as follows:

Keyword Match % = (|Resume Skills ∩ JD Skills| / |JD Skills|) × 100

This score directly reflects what fraction of the job's required skills the candidate possesses, providing a straightforward and interpretable suitability metric. Skills present in the resume but not required by the job description are separately recorded as additional competencies.

TF-IDF Cosine Similarity Score: The full preprocessed text of the resume and the job description are independently vectorized using Scikit-learn's TfidfVectorizer with a maximum feature limit of 5,000 terms and an n-gram range of (1, 2) to capture both unigrams and bigrams. The cosine similarity between the two TF-IDF vectors is computed as:

Cosine Similarity = (Resume Vector · JD Vector) / (||Resume Vector|| × ||JD Vector||)

This score captures broader textual alignment between the candidate's background and the job requirements, including contextual and domain-level similarities that keyword matching may miss.

Final Match Score: The final match percentage is computed as a weighted average of the keyword overlap score and the cosine similarity score, with weights of 0.6 and 0.4 respectively. The higher weight assigned to keyword overlap reflects the greater interpretability and direct relevance of explicit skill matching for recruiter decision-making.

Final Match % = (0.6 × Keyword Match %) + (0.4 × Cosine Similarity × 100)

F. Technologies and Tools

TABLE I. TOOLS AND TECHNOLOGIES USED

Component	Technology / Library
Backend Language	Python 3.x
Web Framework	Flask 2.x
PDF Text Extraction	PyPDF2
NLP Tokenization & Lemmatization	NLTK
Named Entity Recognition	spaCy (en_core_web_sm)
Text Vectorization & Similarity	Scikit-learn (TF-IDF, Cosine)
Frontend Interface	HTML5, CSS3, JavaScript
Version Control	Git & GitHub
Development Environment	Visual Studio Code

IV. IMPLEMENTATION

A. System Architecture

The system is organized into four independent functional layers that communicate through clean interfaces, ensuring separation of concerns and maintainability.

User Interface Layer: The frontend is built with HTML5, CSS3, and vanilla JavaScript, providing a clean and responsive web interface. Three distinct pages are implemented: the home page introducing the system, the upload page providing file upload and job description input controls, and the results page displaying the match analysis output. The interface is styled for readability and usability across desktop and mobile devices.

Resume Processing Layer: The Flask backend manages file receipt, temporary storage, and invocation of the parsing module. The resume_parser.py module handles all PDF text extraction and NLP preprocessing, returning a structured dictionary containing the candidate's extracted skills, cleaned full text, and identified qualifications.

Skill Extraction and Matching Layer: The skill_matcher.py module receives the parsed resume data and the recruiter-provided job description. It executes the keyword overlap analysis and TF-IDF cosine similarity computation, returning a comprehensive match report containing the final match percentage, matched skill list, missing skill list, and additional candidate skills.

Result Generation Layer: The Flask application renders the match report through the result.html Jinja2 template, presenting the match percentage with visual emphasis, followed by categorized skill lists and a textual suitability summary.

B. Project Directory Structure

The project follows a well-organized directory structure that separates source code, frontend assets, templates, and data storage:

AI-Resume-Screening-System/

1. app.py
2. resume_parser.py
3. skill_matcher.py
4. skill_dictionary.py
5. requirements.txt
6. Static
   I. style.css
   II. script.js
7. templates
   I. index.html
   II. upload.html III. result.html

IV. uploads

8. README.md

C. Backend Implementation Details

The app.py file defines three Flask routes. The root route renders the home page. The /upload route handles both GET requests (rendering the upload form) and POST requests (receiving the uploaded resume and job description, invoking parsing and matching modules, and redirecting to the results page). The /result route renders the results page with the match report passed as template context. File upload security is enforced through file extension validation, restricting accepted uploads to PDF format only. Uploaded files are saved with UUID-generated filenames to prevent naming conflicts in concurrent usage scenarios. Temporary files are deleted from the uploads directory after processing to manage disk usage.

The resume_parser.py module implements the full NLP preprocessing pipeline described in Section III. It returns a Python dictionary with keys including raw_text, cleaned_tokens, extracted_skills, and qualifications_list. This structured return format ensures a clean interface between parsing and matching modules.

The skill_matcher.py module accepts the parser output dictionary and the raw job description string. It processes the job description through the same NLP pipeline and executes both the keyword overlap and cosine similarity scoring routines. The module returns a result dictionary with keys including match_percentage, matched_skills, missing_skills, additional_skills, and similarity_score.

D. Frontend Implementation Details

The upload page provides an HTML file input control restricted to PDF files, a textarea for job description entry, and a submit button. Client-side JavaScript validates that both inputs are provided before form submission, preventing empty requests from reaching the backend.

The results page displays the match percentage as a large, prominently styled numerical value with color coding: green for scores above 70%, amber for scores between 40% and 70%, and red for scores below 40%. This visual encoding allows recruiters to instantly gauge candidate suitability without reading detailed text. Below the score, matched skills are displayed as green-colored tags and missing skills as red-colored tags, providing a scannable skill gap summary. A scrollable section at the bottom of the results page displays the full extracted resume text for reference.

E. Error Handling and Validation

Comprehensive error handling is implemented throughout the system. PDF parsing errors (caused by corrupted or password-protected files) are caught and surfaced to the user with a descriptive error message. Empty extraction results (caused by image-only PDFs or scanning artifacts) trigger a user notification advising the submission of a text-based resume. Backend exceptions during matching computation are logged and result in a generic error page with a retry option.

V. RESULTS AND DISCUSSION

A. Experimental Setup

The system was evaluated using a dataset of twenty candidate resumes collected from volunteer participants across three target job profiles: Python Backend Developer, Data Analyst, and Full Stack Web Developer. For each job profile, a representative job description was prepared based on publicly available postings from major Indian job portals. All resumes were provided in PDF format and represented diverse formatting styles including single-column, multi-column, and template-based layouts.

B. Skill Extraction Accuracy

Skill extraction accuracy was evaluated by manually annotating the skills present in each resume and comparing the annotations against the system's extracted skill set. The evaluation metrics used were Precision, Recall, and F1 Score as defined in standard information retrieval literature.

TABLE II. SKILL EXTRACTION PERFORMANCE BY JOB PROFILE

Job Profile	Precision	Recall	F1 Score
Python Backend Developer	0.88	0.82	0.85
Data Analyst	0.85	0.79	0.82
Full Stack Web Developer	0.87	0.84	0.85
Average	0.87	0.82	0.84

The results indicate strong precision across all profiles, confirming that when the system identifies a skill, it is almost always genuinely present in the resume. Recall values slightly below precision suggest that some skills expressed in non-standard terminology or embedded within sentence context were occasionally missed. This is a known limitation of dictionary-based extraction approaches, discussed further in Section VI.

C. Match Percentage Results

TABLE III. SAMPLE MATCH PERCENTAGE RESULTS

Candidate	Job Profile	Matched Skills	Missing Skills		Match %
Candidate 1	Python Backend Dev	Python, Flask, SQL, Git, REST API	Django, D	ocker, Redis	71%

Candidate 2	Python Backend Dev	Python, Django, PostgreSQL, Docker	Redis, CI/CD	Kubernetes,	78%
Candidate 3	Data Analyst	Python, Pandas, NumPy, SQL	Tableau, Spark	Power BI,	64%
Candidate 4	Data Analyst	Python, SQL, Tableau, Power BI, Excel	Machine Spark	Learning,	82%
Candidate 5	Full Stack Web Dev	HTML, CSS, JavaScript, React, Git	Node.js, Docker	MongoDB,	67%
Candidate 6	Full Stack Web Dev	HTML, CSS, JavaScript, MongoDB, React Node.js,	Kubernetes,	TypeScript	85%

The match percentages displayed in Table III are consistent with the manually assessed suitability of each candidate for the respective role. Candidates 2, 4, and 6 — who scored above 78% — were independently assessed by a domain expert as strong fits for their respective positions, validating the system's scoring logic.

D. Processing Performance

Processing time was measured for twenty resume-job description pairs on a standard development machine with 8 GB RAM and an Intel Core i5 processor. Average processing time from file upload to results display was 2.1 seconds, with a minimum of 1.4 seconds for short resumes and a maximum of 3.6 seconds for lengthy multi-page documents. These response times are well within acceptable limits for interactive recruiter use and demonstrate that the system's lightweight NLP approach delivers practical real-time performance without requiring specialized hardware.

E. User Interface Evaluation

The web interface was reviewed by three postgraduate students with recruiting experience. Feedback was positive regarding the clarity of the results page, particularly the color-coded match percentage and the separated matched and missing skills displays. Suggestions for improvement included the addition of a candidate comparison view to rank multiple resumes side by side, a feature identified as a priority for future development.

F. Discussion

The experimental results confirm that the hybrid matching approach combining keyword overlap and TFIDF cosine similarity provides reliable and interpretable candidate-job alignment scoring across diverse resume formats and job profiles. The precision-weighted extraction approach ensures that recruiters receive accurate skill identification with minimal false positives, which is critical for maintaining trust in automated screening recommendations.

The system performs best on resumes with clearly structured skills sections and standard formatting. Performance degrades modestly on highly stylized resumes where text extraction produces fragmented content. This limitation is addressed in Section VI and identified as a key area for future improvement.

VI. LIMITATIONS

Despite the strong performance demonstrated in experimental testing, the current system has several limitations that should be acknowledged.

First, the system's skill extraction accuracy is constrained by the completeness of the skill dictionary. Emerging technologies, domain-specific jargon, and newly coined frameworks that are not present in the dictionary will not be identified regardless of how clearly they are stated in the resume. Continuous dictionary maintenance is required to keep pace with evolving technology landscapes.

Second, PDF text extraction quality is highly dependent on resume formatting. Resumes that use complex multi-column layouts, decorative fonts, or heavily graphic designs may produce low-quality extracted text that degrades NLP performance. Resumes submitted as scanned images embedded within PDF files cannot be processed by the current system, as PyPDF2 does not support Optical Character Recognition (OCR).

Third, the TF-IDF vectorization approach does not capture semantic equivalences between terms. For example, "ML" and "Machine Learning", or "JS" and "JavaScript", will not be recognized as equivalent by the similarity scoring model unless both forms are explicitly included in the preprocessing normalization rules. This limits matching recall for candidates who use abbreviated or alternative terminology.

Fourth, the system does not currently support batch processing of multiple resumes against a single job description, which would be a critical feature for real-world recruiter use. Each resume must currently be uploaded and evaluated individually.

Fifth, no persistent data storage is implemented in the current version. Match results are not saved, making it impossible to compare candidates across sessions or build a historical screening database.

VII. FUTURE WORK

Several directions for future enhancement have been identified based on both the experimental findings and the limitations described in Section VI.

A. Semantic Matching with Transformer Models

Replacing TF-IDF vectorization with sentence embeddings generated by pre-trained transformer models such as BERT or Sentence-BERT would significantly improve the system's ability to recognize semantic equivalences between skills and handle terminology variations. Fine-tuning such models on a domainspecific resume and job description corpus would further improve matching accuracy for specialized technical roles.

B. OCR Integration for Scanned Resumes

Integrating an OCR engine such as Tesseract would enable the system to process scanned or image-based resume PDFs, substantially broadening the range of acceptable resume formats. This is particularly relevant for screening legacy resume archives where digital originals may not be available.

C. Multi-Resume Batch Processing and Candidate Ranking

Implementing batch upload functionality would allow recruiters to submit multiple resumes simultaneously against a single job description and receive a ranked shortlist of candidates sorted by match percentage. This would transform the system from a single-candidate evaluation tool into a practical mass screening platform.

D. Database Integration and Persistent Storage

Integrating a relational or document database such as PostgreSQL or MongoDB would enable persistent storage of match results, candidate profiles, and job descriptions. This would support historical analysis, trend identification across hiring cycles, and longitudinal recruiter workflow improvements.

E. Advanced Candidate Analytics Dashboard

A recruiter-facing analytics dashboard displaying aggregate statistics across multiple candidates — including skill coverage distributions, common skill gaps, and candidate comparison visualizations — would significantly enhance the system's value as a decision support tool.

F. Cloud Deployment and API Exposure

Deploying the system on a cloud platform such as AWS, Azure, or Google Cloud and exposing core matching functionality as a REST API would enable integration with existing HR Information Systems (HRIS), Applicant Tracking Systems (ATS), and job portal platforms, extending the system's impact beyond standalone use.

G. Bias Detection and Fairness Auditing

Future versions of the system should incorporate bias detection mechanisms to identify and mitigate potential discriminatory patterns in matching outcomes. Auditing match scores across demographic groups using anonymized data would ensure that the system supports equitable hiring practices in line with employment regulations.

VIII. CONCLUSION

This paper presented the complete design, development, and evaluation of an AI Resume Screening and Job Description Matching System. The system addresses the critical challenge of manual resume screening in modern recruitment by automating skill extraction and candidate-job alignment scoring through a hybrid NLP and text similarity approach.

The proposed system successfully extracts technical skills and qualifications from candidate resumes in PDF format using an NLP pipeline combining NLTK tokenization and lemmatization with spaCy named entity recognition and a comprehensive skill dictionary. The extracted candidate profile is compared against recruiter-provided job descriptions using a weighted combination of keyword overlap scoring and TF-IDF cosine similarity, producing an interpretable match percentage alongside detailed matched and missing skill analyses.

Experimental evaluation across twenty resumes and three job profiles demonstrated average skill extraction F1 scores of 0.84 and match percentage consistency with expert human evaluation. Average processing time of 2.1 seconds per resume confirms the system's suitability for real-time recruiter use without specialized hardware.

The modular architecture, open-source technology stack, and lightweight NLP approach make the system accessible for deployment in academic institutions, startup environments, and small enterprises that lack the infrastructure for deep learning-based recruitment platforms. At the same time, the identified future work directions — including transformer-based semantic matching, OCR integration, batch processing, and cloud deployment — provide a clear roadmap for evolving the system into an enterprise-grade recruitment automation platform.

This work demonstrates the practical viability and significant potential of combining AI, NLP, and web technologies to automate recruitment workflows, reduce hiring inefficiencies, and support more objective and data-driven talent acquisition decisions.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to Ravi Kumar Chaudhary, Assistant Professor, Department of Computer Science and Engineering, Raffles University, for his valuable guidance, continuous support, and constructive suggestions throughout the project.

I am also deeply thankful to Rajendra Singh, Dean, Department of Computer Science and Engineering, Raffles University, for his encouragement, academic support, and motivation during the course of this research work.

REFERENCES

[1] A. Raza, C. Munir, M. Asif, and N. Akhtar, "Automated Resume Screening Using Natural Language Processing and Machine Learning," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 8, pp. 212–218, 2019.

[2] D. Celik and A. Balaban, "A Resume Classification System Using Support Vector Machines," *Procedia Computer Science*, vol. 3, pp. 747–753, 2011.

[3] S. Maheshwari, A. Sainani, and P. Rekh, "An Approach to Improve Recommendation System Using Text Mining," *International Journal of Computer Applications*, vol. 73, no. 8, pp. 30–35, 2013.

[4] M. Zhao, T. Liu, X. Li, and H. Zhang, "Resume Information Extraction with Cascaded Hybrid Model," *Proceedings of the 45th Annual Meeting of the ACL*, pp. 917–924, 2007.

[5] Y. Zhang, X. Li, and H. Wang, "Named Entity Recognition for Resume Information Extraction Using Bidirectional LSTM," *Proceedings of the IEEE International Conference on Natural Language Processing and Knowledge Engineering*, pp. 1–6, 2018.

[6] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.

[7] C. Qin, L. Zhu, T. Xu, C. Zhu, and E. Chen, "Enhancing Person-Job Fit for Talent Recruitment: An AbilityAware Neural Network Approach," *Proceedings of the 41st International ACM SIGIR Conference*, pp. 25– 34, 2018.

[8] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, O'Reilly Media, 2009. Available: https://www.nltk.org/

[10] Explosion AI, "spaCy: Industrial-Strength Natural Language Processing in Python," Available: https://spacy.io/

[11] Pallets Projects, "Flask Web Development Framework Documentation," Available: https://flask.palletsprojects.com/

[12] Python Software Foundation, "Python Language Reference, version 3.x," Available: https://docs.python.org/3/

[13] MDN Web Docs, "HTML, CSS and JavaScript Reference," Mozilla Foundation. Available: https://developer.mozilla.org/

[14] GitHub Inc., "GitHub Documentation," Available: https://docs.github.com/

[15] A. Vaswani et al., "Attention Is All You Need," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5998–6008, 2017.