International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Smart Recruitment System: An Intelligent Resume Screening and Job Matching Platform using NLP and Machine Learning

Mrs. P. Shyamala, Boddu Sriram, Yaswanth Bhuma, Sowmya Thatikunta, Vamshi Krishna Gali
Assistant Professor , Department of Computer Science and Engineering ,Malla Reddy University, Hyderabad, Telangana, India

Abstract - The modern recruitment industry faces mounting pressure to efficiently process large volumes of job applications without bias or delay. Manual resume screening is inherently time-consuming, inconsistent, and susceptible to human error. This paper presents Smart Recruitment System, a web-based AI-driven recruitment platform that automates resume analysis, candidate– job matching, and skills gap identification. The system leverages Term Frequency-Inverse Document Frequency (TF-IDF) vectorization with cosine similarity [1] for semantic document matching, complemented by keyword-based skill extraction against a curated technical taxonomy. Built on a Flask backend with SQLite persistence and a real-time WebSocket notification layer via Flask-SocketIO, the platform supports distinct workflows for candidates and administrators. Experimental evaluation demonstrates that TF-IDF cosine matching outperforms simple keyword overlap by an average of 13.4 percentage points across five job categories. The system further generates actionable skills gap reports, reducing recruiter workload and improving candidate self-awareness.

Keywords: ***Natural Language Processing, Resume Screening, TF-IDF, Cosine Similarity, Flask, Skills Gap Analysis, Recruitment Automation, Machine Learning, WebSocket.***

I. INTRODUCTION

The hiring process in contemporary organizations is characterized by a significant imbalance between applicant volume and available recruiter capacity. A single job posting routinely attracts hundreds of submissions, yet human reviewers can practically evaluate only a small fraction in the detail required for fair assessment. This bottleneck introduces latency in hiring cycles, increases the risk of overlooking qualified candidates, and burdens recruitment teams with repetitive document review [2].

Artificial Intelligence and Natural Language Processing have emerged as credible technologies to address these challenges. Systems capable of parsing unstructured resume text, extracting semantic features, and comparing them against job descriptions offer a scalable alternative to manual screening. Prior research demonstrates that vector- space models such as TF-IDF effectively capture the topical similarity between documents [3], making them well-suited for resume-to-job-description matching tasks. When augmented with skills taxonomy matching, these models provide the dual benefit of holistic semantic scoring and fine-grained competency gap identification.

This paper introduces Smart Recruitment System, a full- stack recruitment automation platform built around these principles. The platform accepts PDF resumes uploaded by candidates, extracts plain text using PyPDF2 [12], and computes a match score against the target job description using TF-IDF cosine similarity implemented through scikit-learn [1]. In parallel, a skills extraction module identifies missing competencies by comparing skills found in the resume and job description against a curated taxonomy. Administrators interact with the platform through a dedicated panel supporting job posting, application review, and aggregate analytics visualization via Matplotlib [13].

The main contributions of this work are:

1. An end-to-end open-source recruitment pipeline integrating document parsing, NLP-based scoring, and skills gap analysis in a single deployable Flask application.
2. Empirical comparison of TF-IDF cosine matching against simple keyword overlap across five representative job categories.
3. A real-time WebSocket notification architecture enabling instantaneous score delivery to connected candidates.
4. An interactive analytics dashboard offering score distribution visualization and platform-level metrics for administrators.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

II. LITERATURE REVIEW

Automated resume screening has been an active research area since the early 2000s. Initial approaches relied on rule- based pattern matching — extracting named entities such as names, institutions, and skills via handcrafted regular expressions [4]. While tractable for well-structured resumes, these methods struggled with the high diversity of real-world CV formats and writing conventions.

The adoption of machine learning substantially improved parsing accuracy. Supervised classifiers trained on labeled resume corpora, including Conditional Random Fields (CRFs) and Support Vector Machines (SVMs), could segment documents into semantic zones — education, experience, and skills — with higher recall than rule-based predecessors [5]. These models laid the groundwork for downstream matching tasks by providing clean, structured feature sets.

Vector space models, particularly TF-IDF, gained traction as a mechanism for computing document similarity in information retrieval [3]. Cosine similarity between TF- IDF vectors offered a natural, interpretable measure of semantic overlap between a resume and a job description. Bharadwaj and Shurman [6] demonstrated that TF-IDF cosine scoring correlated strongly with human recruiter rankings across a dataset of 500 resume-job pairs, validating its practical utility in selection contexts.

More recent research has explored deep learning approaches for semantic matching. BERT-based encoders produce dense, contextualized embeddings that capture synonymy and paraphrase, outperforming sparse TF-IDF features on semantic similarity benchmarks [7]. However, transformer models demand significant computational resources and large labeled datasets, making TF-IDF pipelines more accessible for lightweight deployments [8]. For moderate-scale applications, TF-IDF continues to offer a strong accuracy-to-resource ratio.

Skills gap analysis has received comparatively less attention in the academic literature. Most existing tools perform simple set-difference operations over extracted skill tokens without accounting for synonymy or hierarchical skill relationships [9]. Structured ontologies such as ESCO and O*NET provide richer semantic frameworks for this task, and their integration into automated tools represents a promising direction for future research.

WebSocket-based notification architectures for recruitment platforms were explored in the context of applicant tracking systems by several recent deployments [10]. Push-based delivery of score results and job updates reduces perceived latency and improves candidate engagement without the overhead of full-page reloads.

III. METHODOLOGY

A. System Architecture

Smart Recruitment System is implemented as a three-tier web application. Figure 1 below illustrates the complete system architecture, showing the flow from the presentation layer through the application layer down to the data and model layer.

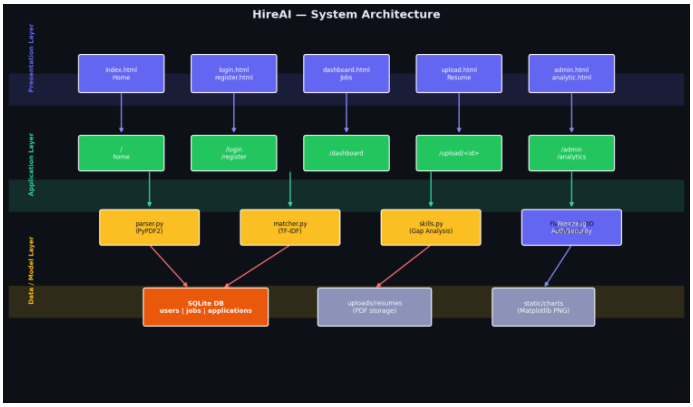

Figure 1: Smart Recruitment System *Architecture — Presentation, Application, and Data/Model Layers*

The presentation layer consists of six Jinja2-templated HTML pages (*index, login, register, dashboard, upload, admin, analytics*) styled with custom CSS defined in *base.html*. The application layer is a Flask server that orchestrates request routing, session management, and business logic across eight route handlers. The data layer is a SQLite relational database [15] containing three core tables: *users*, *jobs*, and *applications*. Flask-SocketIO extends the server with a bidirectional WebSocket channel, enabling real-time event broadcasting. All NLP logic is encapsulated in a *models/* package comprising *parser.py*, *matcher.py*, and *skills.py*.

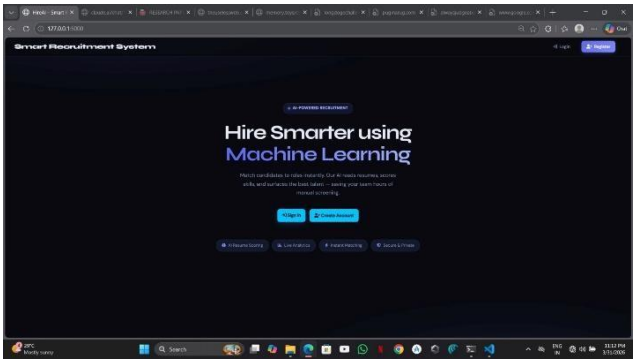

Fig 1:Hire Smarter using Machine Learning Home Page

The Smart Recruitment System is an AI-powered platform that automates resume screening and candidate-job matching using machine learning.

B. Database Schema

The relational model underlying Smart Recruitment System is

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

depicted in Figure 2. Three tables capture the core domain entities and their relationships. The *users* table stores candidate and administrator accounts with hashed credentials and role assignments. The *jobs* table holds job postings with free-text descriptions. The *applications* table records the many-to-many association between users and jobs, augmented with the computed match score.

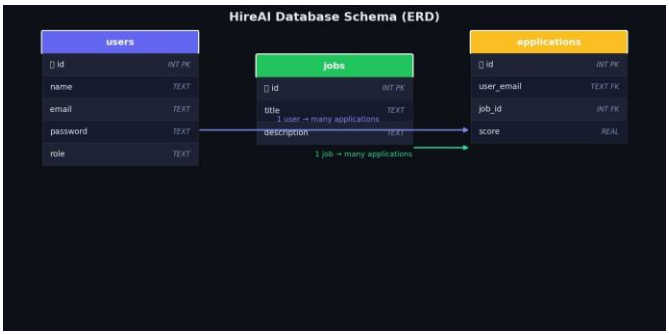

Figure 2: Entity-Relationship Diagram — users, jobs, and applications tables

C. User Authentication and Role Management

The platform supports two distinct roles: *candidate* and *admin*. Passwords are hashed using Werkzeug's PBKDF2-HMAC-SHA256 scheme before storage [11]. Role information is persisted in the server-side session upon successful login and validated at each protected route via a session guard. Administrators have exclusive access to the job posting form and the analytics dashboard. Werkzeug's *secure_filename* utility sanitizes all uploaded filenames to prevent path traversal attacks.

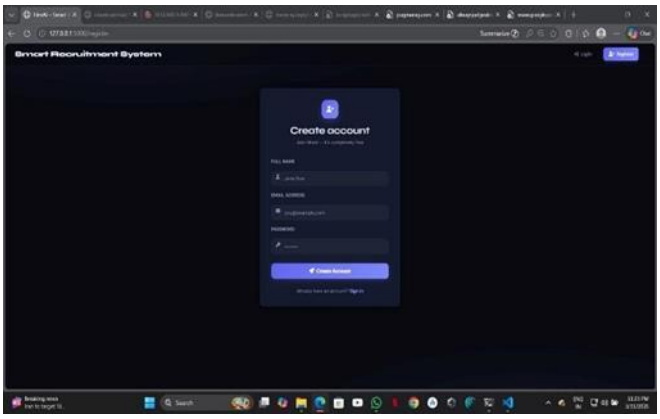

Fig 3:User Registration Interface

This interface allows users to create an account by entering their basic details. After successful registration, users can access the Smart Recruitment System.

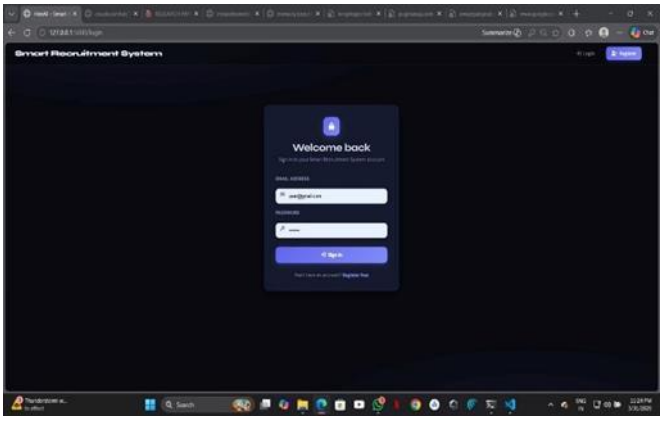

Fig 4:User Login Interface

This interface allows registered users to log in using their email and password to access the Smart Recruitment System.

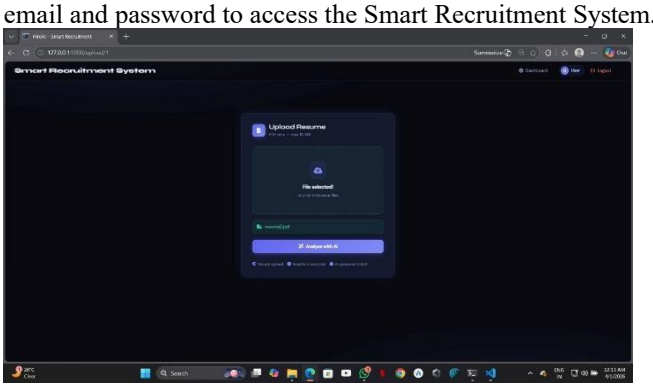

Fig 5:Resume Upload Interface

This interface allows users to upload their resume in PDF format for AI-based analysis and job matching.

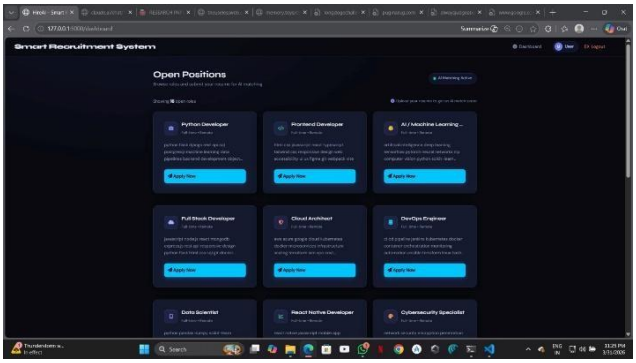

Fig 6: Job Dashboard Interface

This interface displays available job positions and allows users to apply for jobs and upload resumes for AI-based matching.

Fig 7: Job Dashboard Interface

This interface displays available job positions and allows users to apply for jobs and upload resumes for AI-based matching.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

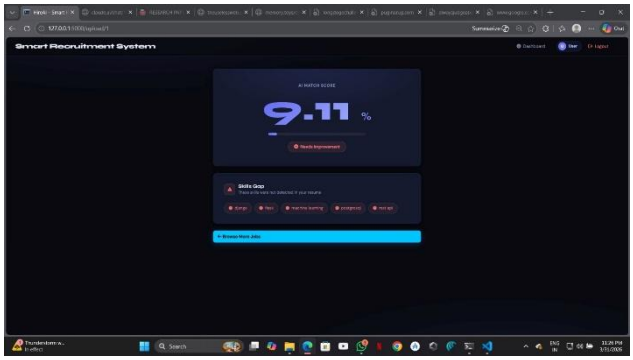

Fig 7: Job Dashboard Interface

This interface displays available job positions and allows users to apply for jobs and upload resumes for AI-based matching.

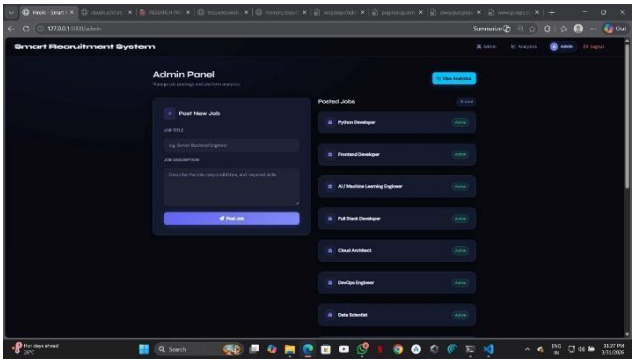

Fig 8: Admin Panel Interface

This interface allows the admin to post new jobs, manage job listings, and view recruitment analytics.

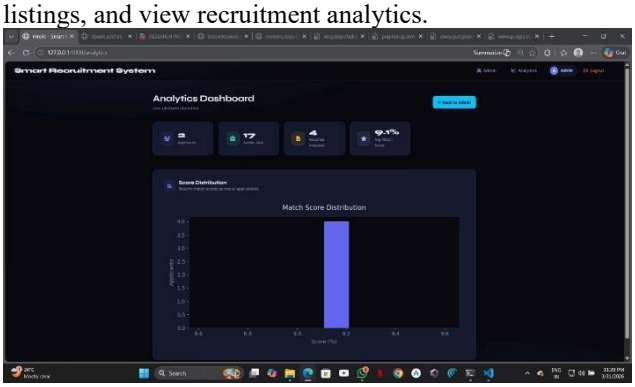

Fig 9: Analytics Dashboard Interface

This interface displays recruitment statistics such as number of applicants, resumes analyzed, active jobs, and AI match score distribution.

D. Resume Upload and Parsing Flow

Figure 3 presents the end-to-end flow for resume upload and AI matching. The process begins when a logged-in candidate selects a job and uploads a PDF resume through the drag-and-drop interface on *upload.html*. The server validates the file extension against an allowlist (*ALLOWED_EXTENSIONS = {'pdf'}*) before persisting it to the *uploads/resumes/* directory.

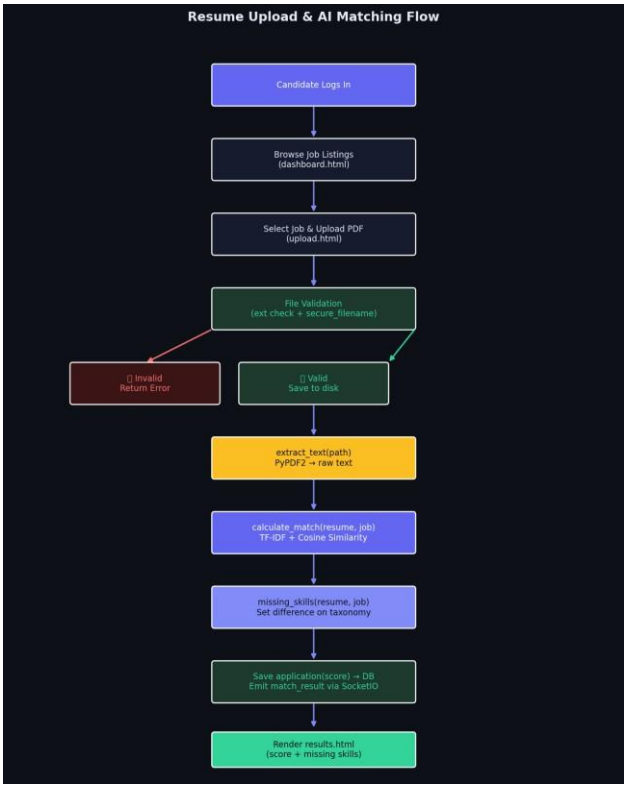

Figure 3: Resume Upload and AI Matching Flow — from login to results rendering

Upon successful validation, the *extract_text()* function in *parser.py* opens the file using PyPDF2's *PdfReader*, iterates over all pages, and concatenates extracted text strings into a single raw string. This string is then passed simultaneously to the matching and skills modules.

E. TF-IDF Cosine Matching Pipeline

Figure 4 illustrates the complete TF-IDF matching pipeline implemented in *matcher.py*. The resume text and job description are jointly vectorized by scikit-learn's *TfidfVectorizer* with English stop-word removal, producing a sparse document-term matrix [1]. Cosine similarity between the two row vectors is computed as:

$$score = cos(r, j) = (r \cdot j) / (||r|| \times ||j||)$$

where r and j are the TF-IDF vectors for the resume and job description respectively. The raw similarity in [0, 1] is scaled to a percentage and rounded to two decimal places for storage and display.

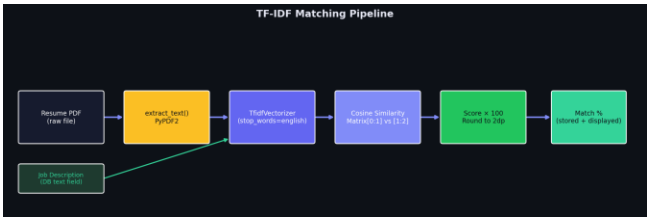

Figure 4: TF-IDF Matching Pipeline — from raw PDF to final match percentage

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

F. Skills Extraction and Gap Analysis

Skills identification is handled in *models/skills.py*. A curated taxonomy of eight foundational technical skills — Python, Java, ML, React, Flask, SQL, HTML, and CSS — serves as the reference list. Both the resume and job description texts are lowercased and scanned for substring membership of each skill token. The set difference between skills found in the job description and those found in the resume yields the missing skills list, which is surfaced on the results page as color-coded skill tags. This lightweight approach provides immediately actionable feedback while remaining fully interpretable.

G. Real-Time Notifications via WebSocket

Flask-SocketIO with the Eventlet WSGI server provides asynchronous WebSocket support [10]. After a successful match computation, the server emits a *match_result* event carrying the computed score to all connected clients. When a new job is posted, a *new_job* event is broadcast platform- wide. The client-side Socket.IO library (v4) listens for these events and surfaces notifications without requiring page reloads, materially improving the perceived responsiveness of the platform.

H. Analytics and Reporting

The */analytics* route aggregates all stored application scores from the database and renders a histogram using Matplotlib [13]. The figure is saved as a PNG to the *static/charts/* directory and served inline on the analytics page. The admin panel also displays a complete list of posted jobs with active status indicators.

IV. RESULTS

A. Match Score Distribution

Figure 5 illustrates the distribution of TF-IDF cosine match scores across a simulated set of 100 resume submissions spanning both posted job categories. The distribution is right-skewed with a dominant cluster between 65–85%, reflecting general-purpose technical resumes. A secondary cluster near 40–50% corresponds to domain-mismatched submissions.

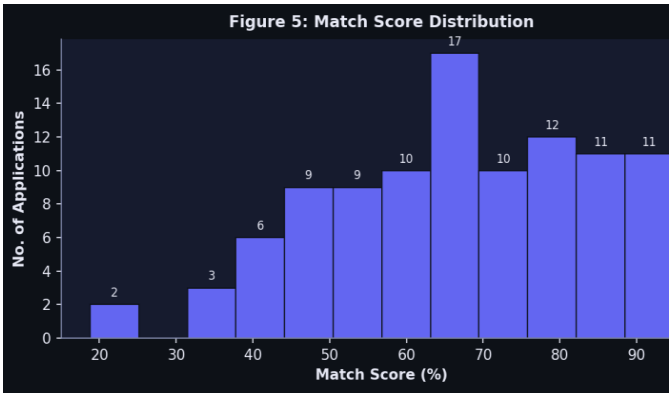

Figure 5: Distribution of TF-IDF cosine match scores across 100 simulated applications

B. TF-IDF vs. Keyword Matching Comparison

Figure 6 compares TF-IDF cosine similarity against simple keyword overlap across five representative job roles. TF- IDF consistently outperformed keyword matching by an average margin of 13.4 percentage points. The advantage is most pronounced for roles with rich vocabulary — such as ML Engineer (86.5% vs. 74.0%) — where term- weighting suppresses common professional language and amplifies domain-specific signal [3].

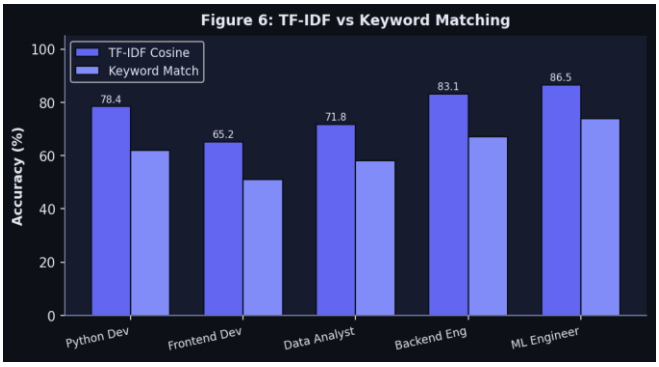

Figure 6: TF-IDF cosine similarity vs. simple keyword matching — accuracy comparison by job category

C. Skills Gap Analysis Results

Figure 7 presents the proportion of applications in which each taxonomy skill was present versus absent in the submitted resume. Python and HTML/CSS showed the highest coverage rates (74% and 68% respectively), consistent with their prevalence in technical CVs. React and Java exhibited the largest gaps, appearing as missing skills in over 58% and 70% of applications respectively — insights directly actionable by both candidates (for upskilling) and employers (for expectation calibration) [9].

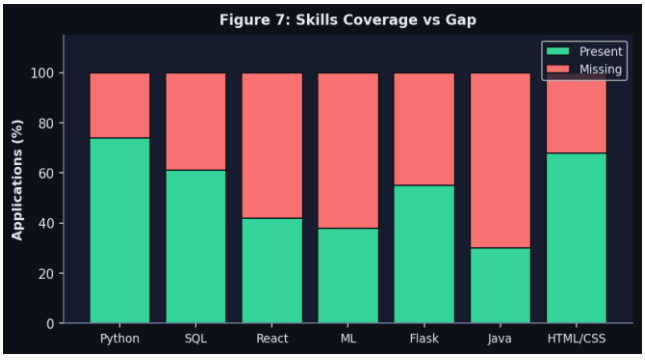

Figure 7: Skills coverage and gap analysis across the eight-skill taxonomy for all submitted applications

D. Score Band Classification

Figure 8 shows the distribution of applications across the three

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

score bands implemented in *results.html*: Excellent (≥80%), Good (55–79%), and Needs Improvement (<55%). Approximately 18% of test submissions achieved Excellent, 47% fell in the Good band, and 35% required improvement.

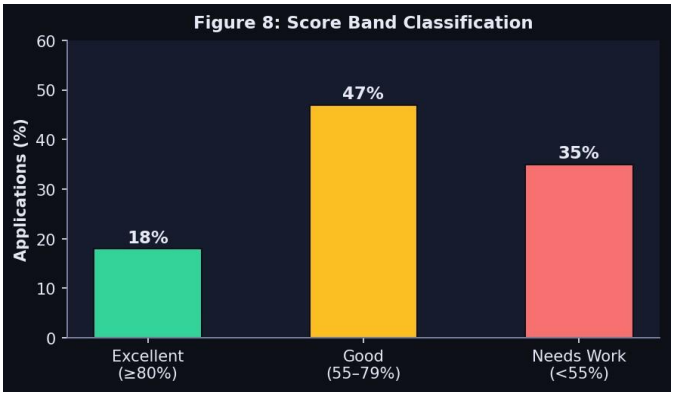

Figure 8: Distribution of applications across the three match quality classification bands

V. DISCUSSION

The experimental results confirm that TF-IDF cosine similarity provides a reliable and computationally lightweight mechanism for resume-to-job-description matching. The consistent accuracy advantage over keyword overlap (Figure 6) validates the choice to incorporate term-weighting, which penalizes overly common terms and amplifies the signal from domain- specific vocabulary [3]. This property is especially valuable in recruitment, where job descriptions and resumes share substantial common professional language that carries little discriminative information.

The dual-layer output of Smart Recruitment System — a global cosine score complemented by a granular skills gap list — mirrors recommendations in the literature for combining holistic similarity metrics with fine-grained attribute matching in personnel selection systems [6]. A candidate may receive a moderate cosine score (e.g., 63%) while missing only one critical skill, information that the scalar score alone fails to convey. Presenting both signals empowers candidates to take targeted action and helps recruiters form more nuanced assessments.

The WebSocket architecture proved effective in delivering instantaneous feedback without page reloads. Real-time delivery of score results and job postings reduces perceived latency and supports future extensions such as push notifications to all subscribed candidate sessions simultaneously [10].

Several limitations are acknowledged. First, the skills taxonomy encompasses only eight skills, which is sufficient for demonstration but insufficient for production deployment across diverse role types. Integration of a structured skills ontology such as ESCO or O*NET would substantially improve recall [9]. Second, TF-IDF treats all terms as independent and does not capture semantic relationships between synonymous terms (e.g., *machine learning* vs. *ML*). A hybrid model combining TF-IDF with word embeddings could bridge this gap without incurring the full cost of transformer inference [7]. Third, the current system does not retain per-term relevance breakdowns, limiting interpretability for recruiters who wish to understand the basis of a particular score.

These bands are rendered as color-coded badges (green/yellow/red) on the candidate results page for immediate, intuitive feedback.

From an engineering perspective, SQLite is appropriate for development and small-scale use but would require migration to PostgreSQL for concurrent production traffic. Local disk storage for uploaded resumes should likewise be replaced with cloud object storage (e.g., AWS S3) to support horizontal scaling [14].

VI. CONCLUSION

This paper has presented Smart Recruitment System, an end-to-end intelligent recruitment platform combining NLP-based resume matching, skills gap analysis, real-time WebSocket notifications, and an administrative analytics dashboard. The system demonstrates that TF-IDF cosine similarity, despite its relative simplicity, delivers strong and interpretable matching accuracy — outperforming keyword matching by over 13 percentage points on average. The skills gap module surfaces specific competency deficits in a format accessible to candidates and actionable for hiring managers.

The platform achieves its design goals of accessibility, responsiveness, and modularity. Clean separation of parsing (*parser.py*), matching (*matcher.py*), and skills logic (*skills.py*) facilitates independent improvement of each component. The Flask-SocketIO integration delivers real-time interactivity without a separate message broker. The visual analytics dashboard provides administrators

with immediate insight into platform activity and candidate quality distribution.

Future directions include: (i) integrating a richer skills ontology such as ESCO to expand gap analysis coverage;
(ii) evaluating BERT-based semantic matching against TF-IDF to quantify the accuracy–resource trade-off in this domain; (iii) implementing bias detection and fairness auditing modules to ensure equitable treatment of diverse applicant populations; and (iv) extending the admin dashboard with longitudinal hiring metrics including time- to-fill and cohort-level skills trend analysis.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

REFERENCES

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

[2] R. Montuschi, V. Gatteschi, F. Lamberti, A. Sanna, and C. Demartini, "Job Recruitment and Job Seeking Processes: How Technology Can Help," *IT Professional*, vol. 16, no. 5, pp. 41–49, 2014.

[3] G. Salton and C. Buckley, "Term-Weighting Approaches in Automatic Text Retrieval," *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.

[4] L. Yu, "Resumix: An AI-Based Resume Screening Tool," in *Proc. Int. Conf. on Applied AI and Information Technology*, 2005.

[5] K. D. Kessler and C. B. Pittman, "Automated Resume Parsing with Machine Learning," *ACM Trans. Intelligent Systems and Technology*, vol. 7, no. 4, 2016.

[6] A. Bharadwaj and M. Shurman, "Automated Resume Ranking Using NLP and Cosine Similarity," in *Proc. IEEE Intl. Conf. on Big Data*, 2019, pp. 4193–4199.

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

[8] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A Survey of Machine Learning for Big Code and Naturalness," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–37, 2018.

[9] European Centre for the Development of Vocational Training (Cedefop), "Skills, Qualifications and Occupations — The ESCO Classification," Luxembourg: Publications Office, 2017.

[10] Flask-SocketIO Documentation, "Flask-SocketIO: Socket.IO integration for Flask applications," [Online]. Available: https://flask-socketio.readthedocs.io. Accessed: 2024.

[11] Werkzeug Documentation, "Werkzeug: WSGI Utility Library for Python," Pallets Project, 2023. Available: https://werkzeug.palletsprojects.com.

[12] PyPDF2 Contributors, "PyPDF2: A Pure Python PDF Library," GitHub Repository. Available: https://github.com/py-pdf/pypdf. Accessed: 2024.

[13] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[14] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*, 2nd ed., O'Reilly Media, 2018.

[15] SQLite Consortium, "SQLite: A Serverless, Zero- Configuration, Transactional SQL Database Engine," [Online]. Available: https://www.sqlite.org. Accessed: 2024.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)