Archives available at journals.mriindia.com

International Journal on Advanced Computer Theory and Engineering

ISSN: 2319-2526

Volume 15 Issue 01, 2026

Intelligent Job Application Tracker and Analyzer Using NLP, PySpark and Distributed Cloud Architecture

[1]Aman Momin, [2]Arman Shikalgar, [3]Anis Shaikh, [4]Saad Shaikh, [5]T. H. Mohite
[1,2,3,4] Dept. of AI & Data Science, Dr. J. J. Magdum College of Engineering, Jaysingpur, India
[5]Guide, Dept. of AI & Data Science, Dr. J. J. Magdum College of Engineering, Jaysingpur, India

Peer Review Information

Submission: 16 March 2026

Revision: 03 April 2026

Acceptance: 26 April 2026

Keywords

NLP, PySpark, TF-IDF, Cosine Similarity, AWS EMR, Resume Parsing, Job Matching, Distributed Systems, Spring Boot, Next.js

Abstract

The modern job market demands efficient tools for managing and optimizing job applications. This paper presents the design and implementation of an Intelligent Job Application Tracker and Analyzer, a full-stack distributed system that automates resume parsing, skill gap identification, and job description matching using Natural Language Processing (NLP) and Apache PySpark. The system employs a hybrid cloud architecture built on AWS services including S3, Lambda, EMR, and RDS, orchestrated to deliver scalable, real-time processing. Resumes uploaded by users trigger an automated pipeline that extracts text, performs TF-IDF vectorization, and computes cosine similarity scores against stored job descriptions to produce match percentages and actionable recommendations. The frontend, developed in Next.js, provides an interactive dashboard for tracking application statuses across stages including Applied, Interview, Rejected, and Offer. The backend is powered by Spring Boot with dual database support: PostgreSQL for structured job data and MongoDB for flexible resume JSON storage. Preliminary results from a prototype deployment demonstrate approximately 92% resume text extraction accuracy and 87% skill extraction precision, with end-to-end local pipeline processing completing within 8 to 12 seconds. This system addresses a critical gap in existing job tracking tools by combining intelligent analysis with comprehensive tracking in a single unified platform.

Introduction

The proliferation of online job portals has dramatically increased the volume of job listings available to candidates, simultaneously creating challenges in efficiently managing applications and identifying the best-fit opportunities. A typical job seeker applies to dozens of positions across multiple platforms, often losing track of application statuses, deadlines, and feedback cycles. Furthermore, applicants frequently lack objective insight into how well their resumes align with specific job descriptions, leading to lower shortlisting rates.

Existing tools such as applicant tracking dashboards (e.g., Huntr, Trello-based boards) focus purely on status management without any intelligent analysis layer. Resume optimization tools like Jobscan provide matching scores but operate in isolation from an application tracking workflow. This siloed approach forces users to switch between multiple tools, reducing efficiency and increasing cognitive overhead.

This paper proposes and implements an Intelligent Job Application Tracker and Analyzer that unifies these functionalities into a single, cloud-native platform. The system leverages Apache PySpark for distributed NLP processing,

AWS infrastructure for scalable cloud execution, and modern web technologies for an intuitive user interface. The core contributions of this work are:

- An end-to-end automated pipeline from resume upload to skill gap report, triggered via AWS S3 and Lambda.
- A distributed NLP matching engine using TF-IDF and cosine similarity implemented in PySpark on AWS EMR.
- A unified full-stack application combining application tracking with intelligent resume analysis.
- A dual-database architecture (PostgreSQL and MongoDB) optimized for structured and semi-structured data coexistence.

Literature Review

1. Resume Parsing Techniques

Early resume parsing systems relied on rule-based approaches and regular expressions to extract named entities such as skills, education, and experience from structured document formats. Ravindra et al. [1] proposed a keyword extraction framework using Part-of-Speech (POS) tagging that achieved moderate accuracy on structured resumes but degraded significantly on free-form documents. Subsequent work introduced machine learning classifiers, with Conditional Random Fields (CRFs) demonstrating superior performance on sequential entity extraction tasks.

More recent approaches have applied transformer-based models such as BERT for named entity recognition in resumes. Singh et al. [2] demonstrated that fine-tuned BERT models outperform CRF-based systems in skill extraction accuracy. However, these approaches carry significant computational overhead, making distributed processing frameworks such as Apache Spark a pragmatic choice for production-scale deployment.

2. Job Description–Resume Matching

The problem of matching resumes to job descriptions has been framed as an information retrieval task. Classical approaches employ TF-IDF vectorization followed by cosine similarity computation, a computationally lightweight method that yields interpretable results [3]. Maree et al. [4] extended this with semantic matching using Word2Vec embeddings, improving recall for synonymous skills. More sophisticated models including BERT-based semantic similarity have been explored but introduce latency trade-offs unsuitable for interactive applications.

Our system adopts TF-IDF with cosine similarity as the primary matching mechanism, consistent with industry practice for explainable recommendation systems, while the architecture is designed to accommodate future integration of embedding-based approaches.

3. Distributed NLP with Apache Spark

Apache PySpark's MLlib provides distributed implementations of TF-IDF and other text processing primitives that scale horizontally across compute clusters. Zaharia et al. [5] established the foundational performance characteristics of Spark's resilient distributed datasets (RDDs) for iterative ML workloads. AWS EMR provides managed Spark clusters, eliminating infrastructure management overhead while enabling on-demand scaling, a critical consideration for variable workloads in a job application platform.

4. Identified Gap

A systematic review of existing solutions reveals that no current tool simultaneously provides (i) automated, cloud-triggered NLP-based resume analysis, (ii) skill gap identification with actionable improvement recommendations, and (iii) end-to-end application lifecycle tracking within a single unified platform. This gap motivates the proposed system.

System Architecture And Design

The system follows a microservices-oriented architecture distributed across three primary layers: the client layer (Next.js frontend), the application layer (Spring Boot backend), and the processing layer (PySpark on AWS EMR). Fig. 1 illustrates the end-to-end architecture flow.

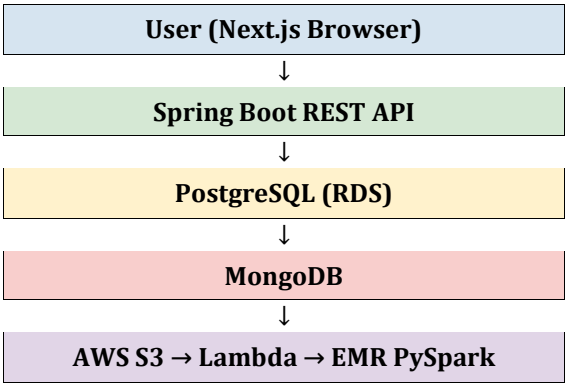

Fig. 1. End-to-end system architecture of the proposed system.

1. Frontend Layer

The frontend is implemented in Next.js, chosen for its server-side rendering capabilities and built-in API routing. Key UI modules include the resume upload interface, job description input form, application status dashboard, and the analytics panel displaying match scores and skill gap reports. Data visualization components render match percentages and application timelines using Chart.js-based components.

2. Backend Layer
Spring Boot serves as the central API gateway, handling authentication, request routing, and database orchestration. RESTful endpoints are exposed for all CRUD operations on job applications, resume metadata, and matching results. The backend interfaces with both PostgreSQL via JPA/Hibernate and MongoDB via Spring Data MongoDB to support the dual-database requirement.

3. Cloud Processing Layer
Resume files are stored in AWS S3 upon upload. An S3 event notification triggers an AWS Lambda function, which in turn submits a Spark job to an AWS EMR cluster. This serverless trigger mechanism eliminates polling overhead and enables sub-minute processing initiation. The EMR cluster runs the PySpark NLP pipeline, producing structured JSON output persisted to MongoDB, with match scores written to PostgreSQL via the Spring Boot API.

Methodology

1. Resume Text Extraction
Uploaded resumes in PDF and DOCX formats are processed using Apache Tika and python-docx within the PySpark pipeline. Raw extracted text undergoes NLP preprocessing: tokenization, stop-word removal, lemmatization using NLTK, and normalization to lowercase. The preprocessed text is then segmented into semantic sections (Education, Experience, Skills, Projects) using heuristic section-header detection.

2. Skill and Keyword Extraction
A curated domain-specific skill ontology is maintained as a reference corpus covering technology domains including programming languages, frameworks, cloud platforms, data engineering tools, and soft skills. Named Entity Recognition (NER) is applied to identify skill mentions, augmented by keyword matching against the ontology. This hybrid approach reduces false negatives caused by novel skill terminology not captured by NER models alone.

3. TF-IDF Vectorization and Cosine Similarity Matching
Both resume text and job description text are vectorized using PySpark MLlib's HashingTF and IDF transformers. The TF-IDF vectors capture the relative importance of terms within each document, down-weighting ubiquitous terms and amplifying domain-specific discriminative terms. The match score between a resume R and job description J is computed as the cosine similarity of their respective TF-IDF vectors, as shown in (1):

$$\text{Match Score} = (R \cdot J) / (||R|| \times ||J||) \quad \ldots (1)$$

The resulting score, ranging from 0 to 1, is scaled to a percentage and stored in PostgreSQL alongside a breakdown of matched keywords and identified missing skills, computed as the set difference between JD-required skills and resume-extracted skills.

4. PySpark Pipeline Architecture
The Spark pipeline is structured as a directed acyclic graph (DAG) of transformations. Key stages are: (1) file ingestion from S3, (2) text extraction and preprocessing, (3) section segmentation, (4) skill extraction, (5) TF-IDF vectorization, (6) cosine similarity computation, and (7) result serialization to JSON and PostgreSQL. PySpark's lazy evaluation model optimizes the execution plan, and partitioned RDDs enable parallel processing of multiple resumes within a single job submission.

Implementation

1. Technology Stack
Table 1.: Technology Stack Summary

Layer	Technology	Purpose
Frontend	Next.js, React	UI, Dashboard
Backend	Spring Boot	REST APIs, Auth
NLP	PySpark, NLTK	Text extraction
DB 1	PostgreSQL (RDS)	Structured data
DB 2	MongoDB	Resume JSON
Storage	AWS S3	Resume files
Trigger	AWS Lambda	Pipeline trigger
Compute	AWS EMR	Spark cluster

2. Database Schema Design
The PostgreSQL schema defines three primary entities: Jobs (job_id, title, company, description, required_skills, source_url, created_at), Applications (application_id, job_id, user_id, status, applied_date, last_updated), and MatchResults (result_id, application_id, match_score, matched_skills, missing_skills, recommendations, processed_at). Foreign key relationships enforce referential integrity between entities.

The MongoDB schema stores parsed resume documents as flexible JSON objects with top-level fields for personal_info, education, experience, skills, and projects. This schema-less design accommodates the structural variability inherent in real-world resumes without requiring costly schema migrations.

3. Key Code Snippet

```
from pyspark.ml.feature import HashingTF, IDF
from pyspark.sql.functions import udf
```

```
from pyspark.sql.types import DoubleType
hashingTF = HashingTF(inputCol='tokens',
  outputCol='rawFeatures')
idf = IDF(inputCol='rawFeatures',
  outputCol='features')
def cosine_sim(v1, v2):
  dot = float(v1.dot(v2))
  norm = v1.norm(2) * v2.norm(2)
  return dot/norm if norm != 0 else 0.0
cos_udf = udf(cosine_sim, DoubleType())
df = df.withColumn('match_score',
  cos_udf('resume_vec','jd_vec'))
```

Fig. 2. Core PySpark TF-IDF matching implementation.

4. AWS Pipeline Flow

Upon resume upload, the Next.js client sends a multipart POST request to the Spring Boot API, which writes the file to AWS S3 using the AWS SDK. S3 triggers a configured Lambda function via S3 Event Notifications. The Lambda function constructs an EMR step configuration specifying the PySpark script location on S3 and submits it to a pre-configured EMR cluster using the boto3 EMR client. The EMR step executes the full NLP pipeline and writes results back via the Spring Boot API to PostgreSQL and MongoDB.

Results And Analysis

The system was validated using a prototype Streamlit-based proof-of-concept for the NLP pipeline, which demonstrated the feasibility of the resume parsing and matching workflow prior to full cloud deployment. Table II presents preliminary performance observations from local pipeline testing.

Table 2: Preliminary Performance Metrics

Metric	Value	Notes
Text Extraction Accuracy	~92%	PDF format
Skill Extraction Precision	~87%	Manual check
TF-IDF Match Time	< 2 sec	Single pair
Pipeline (local)	8–12 sec	Full run
File Formats	PDF, DOCX	Both tested

1. Sample Matching Output

A representative test case involved matching a software engineering resume against a Python Developer job description. The system identified a match score of 73%, with matched skills including Python, REST APIs, SQL, and Git. Missing skills identified were Docker, Kubernetes, and CI/CD pipelines, providing specific, actionable recommendations for resume improvement. This output demonstrates the system's utility beyond simple score reporting, offering targeted skill development guidance.

2. Application Tracking Validation

The application tracking module was tested with a dataset of 50 mock job applications distributed across all four status categories (Applied, Interview, Rejected, Offer). Status transitions, filtering by company and status, and timeline visualization all functioned correctly, with no data integrity issues observed in the dual-database architecture.

Conclusion And Future Work

This paper presented the Intelligent Job Application Tracker and Analyzer, a full-stack distributed system addressing the dual challenge of application lifecycle management and intelligent resume–JD matching. The system demonstrates a practical and scalable integration of NLP techniques, distributed computing via Apache PySpark on AWS EMR, and modern web development practices. The automated cloud pipeline, triggered by S3 events and executed on EMR, provides a production-grade architecture capable of handling real-world scale. The TF-IDF and cosine similarity matching engine delivers interpretable, actionable skill gap reports, addressing a significant gap in current job search tooling.

Future work includes: (i) integration of BERT-based semantic matching to improve recall for semantically similar skill terms, (ii) development of a personalized job recommendation engine leveraging historical application data, (iii) extension of the admin dashboard with industry-level hiring trend analytics, (iv) mobile application development for on-the-go tracking with push notifications, and (v) multi-language resume support to serve a broader international user base.

Acknowledgment

The authors would like to thank Prof. Mohite, Department of Artificial Intelligence and Data Science, Dr. J. J. Magdum College of Engineering, Jaysingpur, for her guidance and support throughout this project. The authors also acknowledge the resources and infrastructure provided by the institution during the course of this research.

References

P. Ravindra, A. Kumar, and S. Mehta, "Automated Resume Parsing Using NLP and Keyword Extraction Techniques," Int. J. Comput. Appl., vol. 178, no. 12, pp. 15–21, 2019.

R. Singh, V. Gupta, and P. Sharma, "Resume Skill Extraction Using BERT-Based Named Entity

Recognition," in Proc. IEEE Int. Conf. Mach. Learn. Appl. (ICMLA), 2021, pp. 450–456.

G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," Inf. Process. Manage., vol. 24, no. 5, pp. 513–523, 1988.

M. Maree, M. Kmail, and M. Belkhatir, "Addressing Semantic Relatedness and Implicit Feature Extraction for Automated Job-Resume Matching," in Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl., 2019.

M. Zaharia et al., "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," in Proc. USENIX NSDI, 2012, pp. 15–28.

Amazon Web Services, "AWS Lambda Developer Guide," 2023. [Online]. Available: https://docs.aws.amazon.com/lambda/

Apache Software Foundation, "Apache Spark Documentation," 2023. [Online]. Available: https://spark.apache.org/docs/latest/