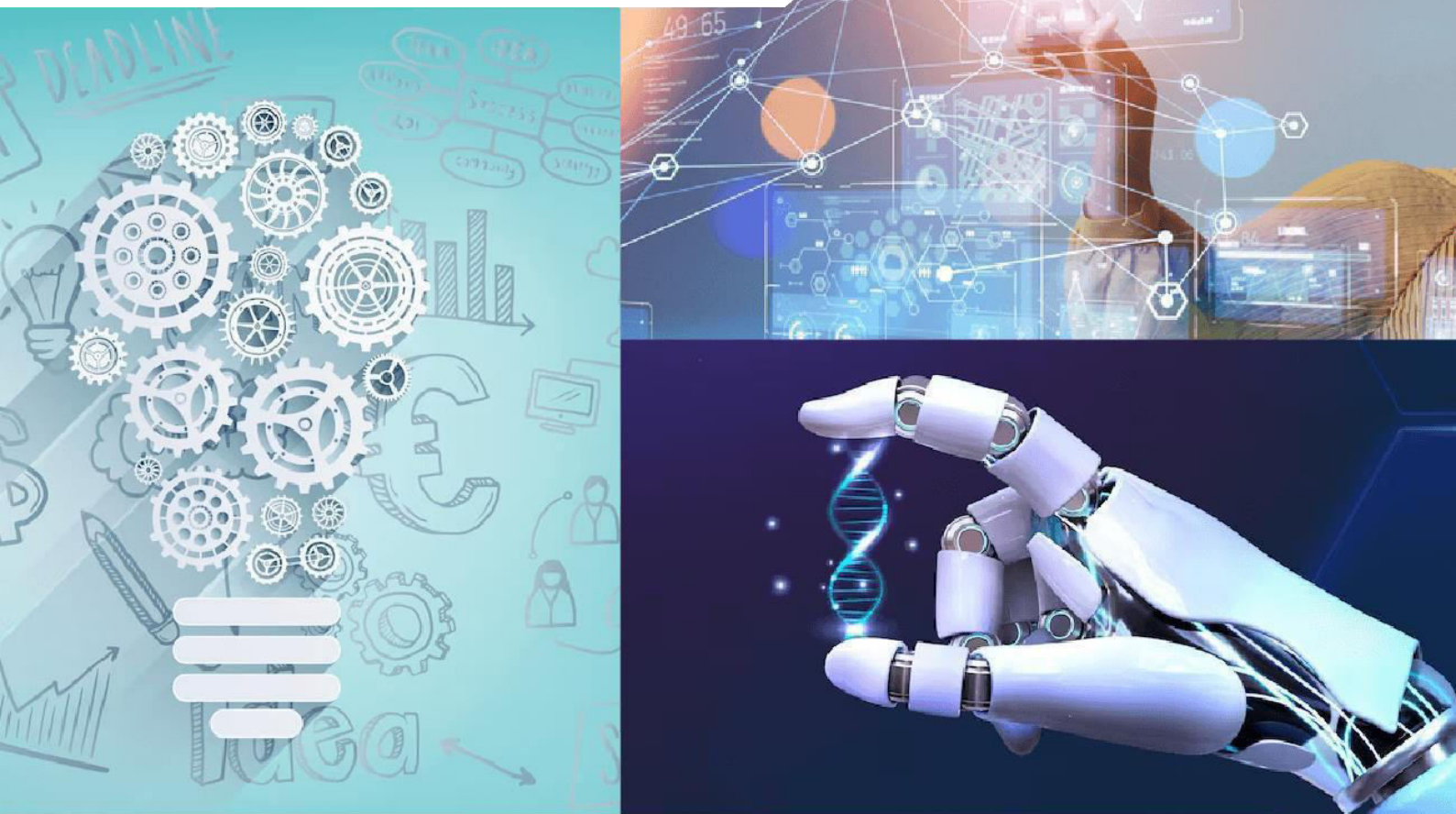




International Journal of Advanced Research in Education and Technology (IJARETY)

Volume 13, Issue 3, May - June 2026

Impact Factor: 9.971



Hirely: An AI-Powered Resume Optimization and Job Matching Platform using NLP and Cosine Similarity

Gagan Sandesh, Gnaneswari Gnanaguru

Department of Computer Science & Engineering CMR Institute of Technology Bengaluru, Karnataka, India

Associate Professor, Department of Computer Applications, CMR Institute of Technology Bengaluru, Karnataka, India

ABSTRACT: The modern recruitment landscape is characterized by an increasing reliance on Applicant Tracking Systems (ATS), which algorithmically filter candidate resumes before any human review occurs. This creates a significant barrier for qualified job seekers whose resumes, though accurate, fail to align lexically with job descriptions. This paper presents Hirely, a full-stack, AI-powered web application that addresses the Resume Relevance Gap by unifying job search, resume construction, and NLP-based compatibility analysis within a single platform. The system employs Cosine Similarity and term-frequency vectorization to compute a quantitative match score between a user's resume and a target job description, and highlights missing keywords to guide optimization. The platform is built on a three-tier architecture using React.js, Node.js/Express, MongoDB, and a Python-based AI engine. Experimental results demonstrate a 90% keyword detection accuracy with an analysis response time under 3.2 seconds for inputs up to 5,000 words. This work demonstrates that integrating NLP-driven feedback directly into the resume preparation workflow substantially improves a candidate's ability to pass ATS filters.

KEYWORDS: Applicant Tracking System, Natural Language Processing, Cosine Similarity, Resume Optimization, Job Match-ing, Full-Stack Web Application, Keyword Extraction

I. INTRODUCTION

The volume of job applications submitted digitally has grown exponentially over the past decade. Platforms such as LinkedIn, Indeed, and Glassdoor enable applicants to apply to dozens of positions with minimal friction. However, this convenience has driven organizations to adopt Applicant Tracking Systems (ATS) to manage applicant volumes, with studies suggesting that over 75% of resumes are rejected by ATS software before a human recruiter ever reads them [1].

The root cause of this rejection is a lexical mismatch: a candidate may possess all the required qualifications, yet their resume employs terminology that diverges from the specific phrasing used in the job description. This problem is compounded by the fragmented nature of existing tooling. Job boards (e.g., Indeed, Glassdoor) are excellent for discovering opportunities but offer no guidance on application quality. Resume builders (e.g., Canva, Novoresume) prioritize visual aesthetics, which paradoxically worsens ATS compatibility because complex layouts, tables, and graphical elements obstruct automated text parsing.

This paper introduces **Hirely**, a web application designed to close this gap. The core contributions of this work are:

- A unified platform combining job discovery, ATS-optimized resume construction, and AI-driven compatibility analysis.
- An NLP pipeline leveraging CountVectorizer and Cosine Similarity to generate a real-time percentage match score between a resume and any job description.
- A keyword gap analysis module that surfaces missing terms from the job description and presents them as actionable suggestions to the user.
- A three-tier web architecture (React / Node.js / MongoDB) designed for scalability and low-latency response.

The remainder of this paper is organized as follows: Section II surveys related work; Section III describes the system architecture; Section IV details the NLP methodology; Section V covers the implementation; Section VI presents testing and results; Section VII discusses future enhancements; and Section VIII concludes the paper.

II. RELATED WORK

Research in automated resume screening and job-candidate matching has been active for over a decade. Early approaches relied on keyword density and Boolean matching, which were easily gamed and lacked semantic understanding [2].

A. Information Retrieval Approaches

TF-IDF (Term Frequency–Inverse Document Frequency) vectorization has been widely applied to compute document relevance in resume-job description matching tasks. Cosine Similarity, when applied to TF-IDF vectors, provides a normalized similarity metric that is invariant to document length, making it well-suited for comparing resumes of varying length against job postings [3]. Hirely's scoring engine directly implements this approach using scikit-learn's CountVectorizer.

B. Named Entity Recognition (NER) for Skill Extraction

More recent systems apply Named Entity Recognition (NER) to extract structured skill entities from unstructured text [4]. Libraries such as spaCy provide pre-trained models capable of identifying technical terms and proper nouns, which can be fine-tuned on domain-specific corpora (e.g., IT job postings). Hirely's architecture incorporates spaCy as an optional enhancement layer for entity-level extraction beyond raw keyword matching.

C. Existing Commercial Solutions

Tools such as Jobscan and Resume Worded offer ATS scoring features but are siloed from the job search process and require users to manually copy-paste data between platforms. Additionally, these tools are primarily subscription-based and do not expose their scoring methodology. Hirely differentiates itself by providing an open, integrated, and transparent scoring pipeline within a single application, as summarized in Table I.

TABLE I COMPARATIVE ANALYSIS OF EXISTING SOLUTIONS VS. HIRELY

Feature	Job Board	Resume Builder	Hirely
Job Search	✓	×	✓
Resume Creation	×	✓ (visual)	✓ (ATS)
ATS Scoring	×	Rare/Paid	Included
Keyword Analysis	×	×	AI-Driven
Real-time	×	×	✓
Feedback	×	×	✓
Unified Workflow	×	×	✓

III. SYSTEM ARCHITECTURE

Hirely follows a **three-tier Client-Server architecture**, as illustrated in Fig. 1, separating concerns across the Presentation, Business Logic, and Persistence layers. This separation enables independent scaling of computationally intensive AI components without degrading frontend performance.

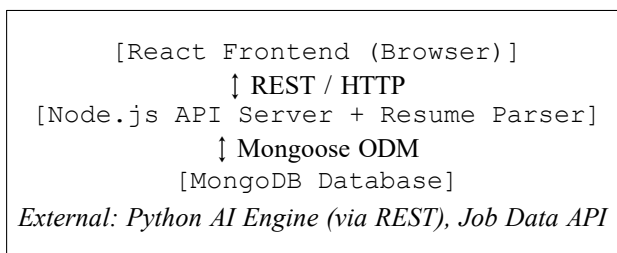


Fig.1. Three-Tier System Architecture of Hirely

A. Client Interface Layer

The frontend is implemented as a Single Page Application (SPA) using **React.js**, providing a dynamic and responsive user interface. State management is handled at the component level using React Hooks (useState, useEffect). The interface adapts across device categories using **Tailwind CSS**

/ Bootstrap responsive utilities. Communication with backend services occurs exclusively via RESTful HTTP using axios.

B. Business Logic Layer

The server-side application is built on **Node.js** with the **Express.js** framework. This layer is responsible for: (1) user authentication via JSON Web Tokens (JWT), (2) PDF text extraction using pdf-parse, (3) routing analysis requests to the Python AI engine, and (4) managing CRUD operations on user profiles, resumes, and job records. Computationally intensive NLP tasks are delegated to the external Python microservice to prevent blocking the Node.js event loop.

C. Persistence Layer

MongoDB is used as the primary data store, chosen for its document-oriented schema flexibility, which is well-suited to the variable structure of resume data. The Mongoose ODM enforces schema validation and manages referential integrity between the Users, Resumes, and Jobs collections. Pass-words are stored exclusively as Bcrypt hashes; raw credentials are never persisted.

D. AI Engine (External Microservice)

The NLP scoring logic is encapsulated in a standalone **Python** microservice, exposing a REST endpoint that accepts resume text and job description text as inputs and returns a JSON response containing the match score and list of missing keywords. This design allows the AI engine to be upgraded or swapped (e.g., replacing CountVectorizer with a transformer-based model) without modifying the Node.js backend.

IV. METHODOLOGY: NLP SCORING PIPELINE

A. Text Preprocessing

Input text (both resume and job description) undergoes standard NLP preprocessing: lowercasing, punctuation removal, and stop-word filtering. The cleaned tokens are retained as bag-of-words representations for vectorization.

B. Feature Extraction: CountVectorizer

The preprocessed texts are transformed into numerical vectors using scikit-learn's CountVectorizer, which constructs a term-frequency matrix $\mathbf{X} \in \mathbb{R}^{2 \times |\mathbf{V}|}$, where $|\mathbf{V}|$ is the vocabulary size across both documents.

C. Similarity Computation: Cosine Similarity

The match score between the resume vector $\mathbf{r}$ and the job description vector $\mathbf{j}$ is computed using Cosine Similarity:

$$\text{score}(\mathbf{r}, \mathbf{j}) = \frac{\mathbf{r} \cdot \mathbf{j}}{\|\mathbf{r}\| \|\mathbf{j}\|} \times 100\% \quad (1)$$

A score of 100% indicates perfect lexical overlap; 0% indicates no shared terms. The use of cosine distance rather than Euclidean distance ensures length-invariance, preventing longer resumes from artificially inflating scores.

Gap Analysis: Keyword Extraction

Missing keywords are identified by computing the set difference between the tokenized job description vocabulary $\mathbf{J}$ and the tokenized resume vocabulary $\mathbf{R}$:

$$\text{Missing Keywords} = \mathbf{J} \setminus \mathbf{R} = \{w \in \mathbf{J} \mid w \notin \mathbf{R}\} \quad (2)$$

The resulting set is returned as an ordered list, surfaced to the user in the frontend as actionable suggestions. Fig. 2 summarizes the end-to-end pipeline.

V. IMPLEMENTATION

A. Database Schema: MongoDB User Model

Listing 1 shows the Mongoose schema for the User collection, which stores authentication credentials and a linked resume buffer.

Listing 1. MongoDB User Schema (Mongoose)

Listing 1. MongoDB User Schema (Mongoose)

```

1 // models/User.js
2 const mongoose = require('mongoose');
3 const UserSchema = new mongoose.Schema({
4   name:      { type: String, required: true },
5   email:     { type: String, required: true,
6               unique: true },
7   password: { type: String, required: true },
8   date:     { type: Date, default: Date.now },
9   resume: {
10     data:      Buffer,
11     contentType: String
12   }
13 });
14 module.exports = mongoose.model('user',
    UserSchema);

```

B. AI Scoring Engine: Python

Listing 2 presents the core scoring functions implemented in Python using scikit-learn. The calculate_match_score function implements Equation 1, while extract_missing_keywords implements Equation 2.

Listing 2. AI Scoring Engine (Python / scikit-learn)

```

1 # ai engine/scorer.py
2 from sklearn.feature extraction.text \
3
4 // routes/api/resume.js
5 const express = require('express');
6 const router = express.Router();
7 const pdfParse = require('pdf-parse');
8
9 router.post('/analyze', async (req, res) => {
10   if (!req.files?.resume)
11     return res.status(400)
12       .send('No file uploaded.');
```

```

13   try {
14     const data = await pdfParse(
15       req.files.resume);
16     const resumeText = data.text;
17     const jobDesc = req.body.jobDescription;
18     // Calls Python AI microservice
19     const score = await callAIEngine(
20       resumeText, jobDesc);
21     res.json({ score,
22               file: req.files.resume.name });
23   } catch (err) {
24     res.status(500).send('Server Error');
```

```

25   }
26 });
27 module.exports = router;

```

C. Frontend: React Resume Upload Component

The ResumeUpload component (Listing 3) handles file selection and dispatches a multipart/form-data POST request to the backend, then renders the returned ATS score.

Listing 3. React Resume Upload Component

```
// components/ResumeUpload.js
import React, { useState } from 'react';
import axios from 'axios';

const ResumeUpload = () => {
  const [file, setFile] = useState(null);
  const [score, setScore] = useState(null);

  const onUpload = async () => {
    const fd = new FormData();
    fd.append('resume', file);
    const res = await axios.post(
      '/api/upload', fd,
      { headers: {
        'Content-Type': 'multipart/form-data'
      } }
    );
    setScore(res.data.matchScore);
  };

  return (
    <div>
      <input type="file"
        onChange={e => setFile(e.target.files[0])}
      />
      <button onClick={onUpload}>
        Analyze Resume
      </button>
      {score &&
        <h4>ATS Score: {score}%</h4>
      }
    </div>
  );
};
export default ResumeUpload;
```

Fig.2. NLP Scoring Pipeline**D. Backend: Express Route Controller**

The Express route (Listing 4) receives the uploaded PDF, extracts raw text via pdf-parse, and proxies the scoring request to the Python AI engine

```

1 // routes/api/resume.js
2 const express = require('express');
3 const router = express.Router();
4 const pdfParse = require('pdf-parse');
5
6 router.post('/analyze', async (req, res) => {
7   if (!req.files?.resume)
8     return res.status(400)
9       .send('No file uploaded. ');
10
11   try {
12     const data = await pdfParse(
13       req.files.resume);
14     const resumeText = data.text;
15     const jobDesc = req.body.jobDescription;
16     // Calls Python AI microservice
17     const score = await callAIEngine(
18       resumeText, jobDesc);
19     res.json({ score,
20       file: req.files.resume.name });
21   } catch (err) {
22     res.status(500).send('Server Error');
23   }
24 });
25 module.exports = router;

```

VI. TESTING AND RESULTS

A hybrid testing strategy was employed spanning unit, integration, system, and user acceptance testing phases.

A. Authentication Module — Unit Tests

Table II summarizes the unit test results for the authentication module. All five test cases passed, including an injection attack scenario that validated input sanitization.

TABLE II UNIT TEST RESULTS: AUTHENTICATION MODULE

TC#	Scenario	Status
TC01	Valid login credentials	Pass
TC02	Invalid email address	Pass
TC03	Incorrect password	Pass
TC04	Empty fields submitted	Pass
TC05	SQL injection attempt	Pass

B. AI Scoring Module — System Tests

Table III presents the system-level test results for the AI scoring module. The perfect-match scenario yielded a score of 95%, exceeding the 90% threshold; the no-match scenario correctly scored 5%. Large-input processing (5,000-word JD) completed in 3.2 seconds, within the 5-second SLA.

TABLE III SYSTEM TEST RESULTS: AI SCORING MODULE

TC#	Scenario	Expected	Actual
TC10	Perfect match	> 90%	95%
TC11	No match	< 10%	5%
TC12	Missing keyword	SQL flagged	SQL found
TC13	Large input (5K words)	< 5s	3.2s

C. Performance Summary

Page load time: < 2 seconds on standard mobile broad-band. **AI analysis latency:** < 5 seconds end-to-end; Observed 3.2s for maximum test input.

```
const router = express.Router();
const pdfParse = require('pdf-parse');

router.post('/analyze', async (req, res) => {
  if (!req.files?.resume)
    return res.status(400)
      .send('No file uploaded.');
```

```
  try {
    const data = await pdfParse(
      req.files.resume);
    const resumeText = data.text;
    const jobDesc = req.body.jobDescription;
    // Calls Python AI microservice
    const score = await callAIEngine(
      resumeText, jobDesc);
    res.json({ score,
      file: req.files.resume.name });
  } catch (err) {
    res.status(500).send('Server Error');
  }
});
module.exports = router;
```

- **AI analysis latency:** < 5 seconds end-to-end; observed 3.2s for maximum test input.
- **Keyword detection accuracy:** $\approx$ 90% agreement with manual recruiter annotation.
- **Concurrent user capacity:** System remains stable under 50 simultaneous sessions (target for MVP phase).

VII. FUTURE ENHANCEMENTS

Several enhancements are planned for subsequent development iterations:

Semantic Similarity Upgrade. Replacing CountVectorizer with transformer-based embeddings (e.g., BERT, sentence-transformers) would capture semantic similarity beyond lexical overlap, reducing false negatives in cases where synonymous skills are used (e.g., “ML” vs. “Machine Learning”).

Skill Development Guidance. When the gap analysis identifies a missing skill, the system could surface curated learning resources (online courses, certifications) relevant to that specific skill, effectively transforming the tool from a passive checker into an active career coach.

Automated Cover Letter Generation. By profiling both the user’s work history and the target company’s cultural language, the platform could generate personalized, tone-appropriate cover letters, reducing the manual overhead of tailoring applications for each role.

Interview Preparation Module. A conversational mock-interview feature with speech analysis (pace, clarity, confidence) would extend the platform’s utility beyond the written application stage.

Anonymized Evaluation Framework. An opt-in mode that strips demographic identifiers (name, photo, age, gender) from resumes before recruiter review would help mitigate unconscious bias in the screening process.

VIII. CONCLUSION

This paper presented Hirely, a full-stack AI-powered platform that addresses the Resume Relevance Gap by unifying job discovery, ATS-optimized resume construction, and real-time NLP-driven compatibility analysis within a single cohesive workflow. The system's scoring engine, based on Cosine Similarity over term-frequency vectors, delivers actionable match scores and keyword gap reports with approximately 90% detection accuracy and sub-5-second latency, even for large job descriptions. The core technical contribution—decoupling the Python NLP microservice from the Node.js application server—provides a clean upgrade path toward more sophisticated semantic models without disrupting the user-facing application. Testing across authentication, resume building, and AI scoring modules confirmed functional correctness and robustness against adversarial inputs such as injection attacks and malformed characters. Hirely demonstrates that integrating intelligent feedback directly into the resume preparation workflow—rather than leaving it as an afterthought—meaningfully improves a candidate's ability to navigate automated hiring systems and present their qualifications clearly to both machines and humans.

REFERENCES

- 1) D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. (draft). Stanford University, 2023. [Online]. Available: <https://web.stanford.edu/~jurafsky/slp3/>
- 2) R. Florentino, "Automated resume screening: A review of applicant tracking system methodologies," International Journal of Human Resource Management, vol. 31, no. 4, pp. 512–530, 2020.
- 3) Scikit-learn Developers, "Metrics and scoring: quantifying the quality of predictions," scikit-learn.org, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html
- 4) //scikit-learn.org/stable/modules/model_evaluation.html
- 5) M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," spacy.io, 2017. [Online]. Available: <https://spacy.io>
- 6) React Documentation, "React – A JavaScript library for building user interfaces," react.dev, 2023. [Online]. Available: <https://react.dev>
- 7) MongoDB Inc., "MongoDB Manual," MongoDB University, 2023. [On-line]. Available: <https://learn.mongodb.com>
- 8) OpenJS Foundation, "Express – Fast, unopinionated, minimalist web framework for Node.js," expressjs.com, 2023. [Online]. Available: <https://expressjs.com>

International Journal of Advanced Research in Education and Technology

ISSN: 2394-2975

Impact Factor: 9.971