INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 3, MARCH 2026 | ISSN: 2456-4184 | IJNRD.ORG

AI-POWERED JOB SCREENING AND RESUME INTELLIGENCE SYSTEM

A Full-Stack Platform Using Generative AI, NLP, and REST APIs to Automate Recruitment and Career Guidance

Deepak Bajaj, Sahil Yadav, Ronit Kumar, AdityaRaj Singh Chouhan, Atharv Chaurasiya
Students, Department of Data Science
Oriental Institute of Science and Technology, Bhopal, India

Abstract: Recruiters have always had a difficult time finding the best candidate, and it is rarely simple for job seekers to get relevant feedback on their applications. During our final year of the Data Science program, we set out to create an AI-powered resume intelligence and employment screening platform in order to solve all sides of this issue. The end product is a full-stack system that was developed over the course of more than six months. It includes basic NLP approaches for parsing and extracting information from resumes, TF-IDF cosine similarity for matching resumes to job descriptions, and Large Language Models for understanding and creating text. The frontend is developed in React, the backend is powered by Flask, data is saved in MongoDB and cached via Redis, and the entire system is containerised using Docker. The platform has 56 successful automated tests, 33 API endpoints, and 17 AI features. This paper describes the system's operation, the design choices that went into it, the difficulties we encountered during its construction, and the findings of the evaluation.

I. INTRODUCTION

Hiring has always been a slow process. A recruiter at a mid-sized company might receive three hundred applications for a single role and realistically has time to spend only a few seconds on each resume before deciding whether to continue reading or move on. On the other side, a candidate submits an application and often hears nothing back — no feedback, no explanation, just silence. Both sides of this process are frustrating, and we think a lot of that frustration is unnecessary.

As Data Science students, we spent time working with text data, NLP libraries, and machine learning models as part of our coursework. At some point it occurred to us that the same techniques — TF-IDF, cosine similarity, named entity recognition — could actually be applied to the resume screening problem in a meaningful way. And with LLMs becoming more accessible through APIs, there was an opportunity to go further than just matching keywords.

So for our final year major project we built an AI-powered platform that handles both sides of recruitment. Job seekers can upload their resume, get detailed feedback, generate a cover letter tailored to a specific job, practise for interviews through a conversational AI simulator, and track how their resume improves over multiple versions. Recruiters can use the same platform to score candidates against a job description, generate professional emails, write job descriptions, and produce Boolean search strings for sourcing. The project took more than six months to complete.

This paper covers the full system. Section II looks at related work. Section III describes the architecture. Section IV explains the AI and NLP pipeline. Section V covers the coaching system. Section VI documents the API. Section VII presents evaluation results. Section VIII discusses challenges and limitations, and Section IX concludes.

II. BACKGROUND AND RELATED WORK

The earliest automated resume screening tools worked through keyword matching. If a resume contained specific words that appeared in the job description, it was considered a match. The obvious problem with this is that language does not work that way. A developer who writes 'built predictive models' and a job description that asks for 'machine learning experience' are clearly talking about the same thing, but a keyword matcher would miss the connection entirely.

Research in Named Entity Recognition (NER) helped improve things by identifying structured information from resume text — pulling out job titles, company names, and skills as distinct entities [1]. Word embedding models like Word2Vec [2] then brought semantic understanding. BERT [3] and later transformer models took this further by understanding words in context rather than in isolation.

More recent work has looked at using LLMs directly for candidate evaluation. The advantage is that a well-prompted LLM can not only assess fit but explain its reasoning in natural language — something no previous approach could do well [4]. The limitation, which we ran into personally during this project, is that LLM outputs can be inconsistent, and getting reliable structured responses requires careful prompt design.

What we found lacking across all of this existing work was a complete deployable system. Most research produces a model or a pipeline evaluated on a dataset, but nothing a real recruiter or job seeker could actually open in a browser and use. That gap is what motivated the design of this platform.

III. SYSTEM ARCHITECTURE

The system is built across three layers: a React frontend, a Flask backend, and a data layer using MongoDB and Redis. Table 1 gives a breakdown of the full technology stack and what each component is responsible for.

Layer	Technology	Purpose
Frontend	React 18, TypeScript, Vite	User interface, PDF upload, speech recognition
Backend API	Python 3.12, Flask, Gunicorn	REST endpoints, business logic, auth middleware
AI / NLP	Cohere, OpenAI, scikit-learn, spaCy	LLM calls, TF-IDF matching, NER, parsing
Task Queue	Redis + Celery	Async processing, LLM response caching
Database	MongoDB Atlas	Analysis history, user records, coaching data
Auth	Firebase Auth	Google Sign-In, token validation, RBAC
DevOps	Docker, Docker Compose, GitHub Actions	Containerisation, automated CI/CD

Table 1: Technology stack overview

3.1 Backend Design

We built the entire backend in Flask. One design decision we kept coming back to was how to handle the LLM integration without locking the codebase into a single provider. The solution was a single call_llm() function that reads the provider and model name from an environment variable. Switching from Cohere to OpenAI or back is a one-line config change — the rest of the application does not need to know or care which provider is active.

The backend exposes 33 endpoints across six groups: public health and metrics routes that require no authentication, core analysis routes, job seeker tools, recruiter tools, a coaching system, and admin-only endpoints. Every response follows the same JSON structure with status, data, and error fields so the frontend always knows what to expect.

3.2 Asynchronous Processing and Caching

LLM API calls are slow — typically two to four seconds per request. To handle this, we set up Celery with Redis as the broker. When a resume analysis request comes in, the task is queued and a job ID is returned immediately. The frontend polls a status endpoint until the result is ready. If Redis is unavailable, the system falls back to synchronous processing rather than throwing an error.

We also cached LLM responses in Redis with a 24-hour TTL. Prompts are fingerprinted with a SHA-256 hash and cached results are returned without hitting the LLM API again. In production this brought response times for repeat queries from around three seconds down to under two hundred milliseconds.

3.3 Security

Integrating Firebase authentication was one of the harder parts of the project. We ran into several issues around token validation between the React frontend and Flask backend before it worked consistently. Once stable, we added role-based access control, rate limiting using a sliding window per user IP, and HTTP security headers including CSP, HSTS, and X-Content-Type-Options on every response.

IV. AI AND NLP PIPELINE

4.1 Resume Parsing

Before the AI can do anything useful, the resume needs to be read and structured. We wrote a regex-based section parser that identifies standard headings — Summary, Experience, Education, Skills, Projects — and extracts the text under each. The content goes through spaCy for tokenisation and Named Entity Recognition. A curated list of over 30 technical skills is cross-checked against extracted tokens to build a skills inventory for each candidate.

4.2 Hybrid Matching Model

Getting the matching score to feel meaningful was probably the most iterative part of the whole project. Our first attempt used simple keyword overlap and the results were poor. We then tried pure TF-IDF semantic matching, which was better at handling synonyms but sometimes missed specific technical requirements that recruiters care about exactly.

The approach we landed on combines both. Lexical matching measures exact keyword overlap between the resume and job description (Equation 1). Semantic matching uses TF-IDF vectors and cosine similarity to capture meaning beyond exact phrasing (Equation 2). The final combined score averages both with equal weight (Equation 3).

Lexical Score	= |Resume Skills ∩ JD Keywords| / |JD Keywords| × 100 (Eq. 1)
Semantic Score	= cosine(TF-IDF(Resume), TF-IDF(JD)) × 100 (Eq. 2)
Combined Score	= 0.5 × Lexical Score + 0.5 × Semantic Score (Eq. 3)

4.3 Prompt Engineering

Getting consistent output from LLMs was harder than we expected. Early prompts produced responses that varied in format, which made parsing unreliable. For features that need structured data — like skill gap analysis returning a list of missing skills with learning resources — we moved to JSON-enforced prompts where the model is explicitly told to return only a JSON object with defined keys. We also added retry logic so that if the response cannot be parsed, the system retries up to three times before falling back to a default response.

4.4 AI Feature Modules

The platform includes 17 AI-powered features in total, split between job seekers and recruiters. Table 2 lists all features and the method behind each one.

Feature	User	Method
Resume Analysis & Scoring	Both	LLM + TF-IDF Hybrid Matching
Cover Letter Generator	Job Seeker	LLM with resume and JD context
Mock Interview Simulator	Job Seeker	Conversational LLM + Speech API
Interview Performance Scorer	Job Seeker	LLM session analysis
Skill Gap Analysis	Job Seeker	LLM + curated skill vocabulary
LinkedIn Profile Builder	Job Seeker	LLM generation
Salary Estimator	Job Seeker	LLM with role and location context
Resume Tailoring Tool	Job Seeker	LLM keyword-aligned rewriting
Career Path Generator	Job Seeker	LLM roadmap generation
Resume Health / ATS Check	Job Seeker	LLM formatting and ATS analysis
Networking Message Generator	Job Seeker	LLM
Recruiter Email Generator	Recruiter	LLM (invite / reject / offer)
Job Description Generator	Recruiter	LLM
Boolean Search Generator	Recruiter	LLM for sourcing strings
Semantic Candidate Matching	Recruiter	TF-IDF + Cosine Similarity
Coaching Progress Tracker	Job Seeker	Versioned metrics + Chart.js
Interview Question Generator	Both	LLM with role context

Table 2: All 17 AI features and their methods

4.5 Mock Interview with Speech Recognition

The mock interview feature was something we were particularly eager to build. The LLM is given a system prompt containing the candidate's resume and target job role, and it plays the role of an interviewer. The frontend integrates the browser Web Speech API so candidates can answer by speaking rather than typing, and interview questions are read aloud through text-to-speech. After the session, a separate endpoint analyses the full conversation and gives a performance score across technical depth, communication, and problem-solving.

V. COACHING AND PROGRESS TRACKING

Most AI resume tools give you a one-time analysis and nothing else. But improving a resume is something you do gradually — you make changes, submit again, and want to know whether you are actually getting better over time. The coaching module was built to support exactly that.

After each resume submission, users can save a snapshot of their results. Each snapshot records the skills count, ATS compatibility score, section completeness, and match percentage. These are displayed as a time-series chart in the React frontend using Chart.js. A diff endpoint compares any two saved versions and highlights which skills were added, removed, or changed between them. A study pack endpoint maps identified skill gaps to learning resources generated by the LLM based on the specific skills that are missing.

This turns the platform from a one-time scanner into something closer to a continuous development tool, which we think is more useful for candidates who are actively preparing for job applications.

VI. API DESIGN

The backend exposes 33 REST endpoints. Table 3 lists the core endpoints that power the main features. All endpoints return a consistent JSON envelope with status, data, and error fields.

Endpoint	Method	Description
/analyze	POST	Resume and JD analysis with AI scoring
/status/<job_id>	GET	Poll async job status
/generate-cover-letter	POST	AI cover letter tailored to job description
/mock-interview	POST	Start a conversational mock interview
/analyze-mock-interview	POST	Score a completed interview session
/analyze-skills	POST	Skill gap detection with learning resources
/estimate-salary	POST	Market salary range and negotiation tips
/generate-linkedin-profile	POST	AI-generated LinkedIn headline and about
/tailor-resume	POST	Rewrite resume bullets to match JD
/generate-career-path	POST	Career roadmap with milestones
/resume-health-check	POST	ATS compatibility and formatting score
/generate-job-description	POST	Job description generation for recruiters
/generate-boolean-search	POST	Boolean search string for sourcing
/coaching/save-version	POST	Save versioned resume snapshot
/coaching/progress	GET	Retrieve all saved resume versions
/coaching/diff	GET	Compare two resume versions
/history	GET	Past analysis results from MongoDB

Table 3: Key API endpoints (authentication required)

VII. EVALUATION AND RESULTS

7.1 Matching Accuracy

To test how well the hybrid matching pipeline actually works compared to human judgment, we collected 50 resume and job description pairs and asked three people with HR and technical backgrounds to independently rate each pair on a 0 to 100 fit scale. We then compared their average scores against the three matching methods the system supports. The results are in Table 4.

Method	Correlation (r)	Mean Abs. Error	RMSE
Keyword Matching Only	0.61	14.3 pts	17.8 pts
TF-IDF Semantic Only	0.74	10.1 pts	13.2 pts
Hybrid Combined (Eq. 3)	**0.82**	**7.6 pts**	**9.9 pts**

Table 4: Matching accuracy vs human expert ratings (n=50 pairs)

The hybrid approach had the strongest correlation with human scores at 0.82, and the lowest mean absolute error at 7.6 points. The jump from keyword-only to the combined method — Pearson r from 0.61 to 0.82 — confirms that adding semantic understanding on top of exact matching genuinely improves how well the score reflects human judgment.

7.2 System Performance

Metric	Result
Average LLM response (no cache)	~3.1 seconds
Average LLM response (cached)	< 200 milliseconds
Redis cache hit rate (production)	~34%
Automated tests	56 (all passing)
Test domains	Auth, NLP, API, database, coaching
Production uptime over 30 days	99.7%
Total development time	More than 6 months

Table 5: System performance metrics

7.3 Test Coverage

We wrote 56 pytest test cases covering health and system endpoints, authentication and security header validation, resume parsing, semantic matching correctness, all AI feature endpoints, the coaching module, MongoDB integration, and utility functions. All 56 pass on every commit through the GitHub Actions CI pipeline. Having that safety net made it much easier to refactor parts of the code without worrying about breaking something silently.

VIII. CHALLENGES AND LIMITATIONS

Three problems took up a disproportionate amount of time during development.

The first was LLM output variability. Even with the same prompt the model would sometimes return data in a slightly different structure, which broke the parsing code downstream. We fixed this by moving to JSON-enforced prompts, adding retry logic, and validating the response schema before using it. It took several iterations to get stable.

The second was tuning the matching accuracy. Our initial keyword-only implementation scored candidates poorly when they used different but equivalent terminology. Switching to the hybrid approach required a fair amount of experimentation to find the right balance between the lexical and semantic components across different types of job descriptions and resume styles.

The third was Firebase authentication. Token validation between the React frontend and Flask backend had edge cases around token refresh, guest mode, and session expiry that were not well documented. Debugging those took significant time.

In terms of current limitations, the TF-IDF semantic matching is effective but has a known ceiling — it cannot capture deep contextual relationships the way dense embeddings from models like sentence-transformers can. The skill vocabulary is also general and would need domain-specific extension for specialised fields like healthcare, legal, or finance. These are the two main areas we plan to improve in future versions.

IX. CONCLUSION

This project started as an observation — recruitment takes too long and gives too little feedback — and became a full-stack AI platform built from scratch over more than six months. Along the way we applied and expanded concepts from our Data Science programme including TF-IDF, cosine similarity, NLP text processing, and LLM prompt engineering, into a production environment with real authentication, async processing, containerisation, and a comprehensive test suite.

The hybrid matching pipeline outperforms keyword-only and semantic-only approaches, reaching a Pearson correlation of 0.82 with human expert ratings and a mean absolute error of 7.6 points. The platform delivers 17 AI features across resume analysis, career coaching, mock interviews, and recruiter tools. All 56 automated tests pass and the system has maintained 99.7% uptime in production.

Future work will focus on replacing TF-IDF with dense vector embeddings using FAISS for better semantic recall, adding multi-tenant organisational support, and connecting with external ATS platforms. We believe AI, when applied carefully, can meaningfully improve processes that millions of people go through every day.

X. ACKNOWLEDGEMENT

We would like to thank the Department of Data Science, Oriental Institute of Science and Technology, Bhopal, for the academic support and guidance throughout this final year project. We are grateful to the faculty members whose teaching of machine learning, NLP, and software systems directly shaped the decisions made in this work. We also acknowledge the open-source communities behind Flask, React, scikit-learn, spaCy, and the developer programmes offered by Cohere and OpenAI that made the AI integration possible.

REFERENCES

[1] Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. Proceedings of ICML 2001, pp. 282-289.

[2] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[3] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of NAACL-HLT 2019, pp. 4171-4186.

[4] Bang, Y., Cahyawijaya, S., Lee, N., Dai, W., Su, D., Wilie, B., and Fung, P. (2023). A multitask, multilingual, multimodal evaluation of ChatGPT on reasoning, hallucination, and interactivity. arXiv preprint arXiv:2302.04023.

[5] Qin, C., Zhu, H., Xu, T., Zhu, C., Ma, J., Chen, W., and Xiong, H. (2018). Enhancing person-job fit for talent recruitment: An ability-aware neural network approach. Proceedings of SIGIR 2018, pp. 25-34.

[6] Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. Foundations and Trends in Information Retrieval, 3(4), pp. 333-389.

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems (NeurIPS), 30.

[8] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. NeurIPS 2020.

[9] Bajaj, D. (2025). AI Job Screening and Talent Intelligence Platform. GitHub. https://github.com/deepakbajaj12/AI-Job-Screening-Analyzer

[10] Cohere Inc. (2024). Command R: A scalable LLM for enterprise use. https://cohere.com/command