International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 09 | Sep 2026 www.irjet.net p-ISSN: 2395-0072

# AI Resume ATS Analyzer

**B. Pravalika[1], Vasa Rakshitha[2], Sri Madhavi[3], T. Jashvathi[4], Tony Eric[5]**

*[1]Assistant Professor, [2345]Student, & Hyderabad Institute of Technology and Management*
*Department of Computer Science and Engineering, [HITAM], [Telangana], India*

---

**ABSTRACT-***Applicant Tracking Systems now stand between almost every job seeker and the recruiter who might otherwise read their resume. A candidate can be genuinely well suited to a role and still never get a callback, simply because the resume was not written in a way the screening software recognises. This paper walks through the design of an AI Resume ATS Analyzer built to close that gap - a small web tool that takes a candidate's resume and a target job description and tells them, in plain terms, how an ATS is likely to see it. Under the hood, the system pulls text out of PDF or DOCX resumes, cleans it up with standard NLP steps such as tokenisation, stop-word removal and lemmatisation, and pulls out the skills and named entities that matter for the role. It then measures how closely the resume lines up with the job description using TF-IDF vectors and cosine similarity, backed up by a straightforward keyword-gap check. What comes out the other end is a single compatibility score, a list of keywords the resume is missing, and a few concrete pointers on what to fix. When we tried this hybrid scoring approach against plain keyword counting on a small set of sample resumes, it came out noticeably ahead - suggesting that combining the two signals gets closer to how a real ATS (and a real recruiter) would judge a resume.*

**Key Words: Applicant Tracking System (ATS), Resume Parsing, Natural Language Processing (NLP), TF-IDF, Cosine Similarity, Named Entity Recognition (NER), Keyword Extraction, Resume Screening**

## 1. INTRODUCTION

Recruitment has quietly become an automated process long before most candidates realise it. A single job opening at a large company can pull in hundreds, sometimes thousands, of applications, and no HR team could realistically read through all of them by hand. That is where Applicant Tracking Systems come in - platforms such as Workday, Taleo, Greenhouse and iCIMS that scan, parse and rank resumes before a human being ever opens one. These systems pull structured details out of a resume - skills, past roles, education, certifications - and check that against what the job description is actually asking for. The trouble is, a resume that does not use the expected keywords, or that leans on formatting an ATS parser cannot read cleanly (multi-column layouts, tables, text boxes, headers and footers, embedded images, unusual fonts), often gets pushed down the list or filtered out entirely, regardless of how qualified the person behind it actually is.

That creates a barrier most candidates cannot even see. Someone might have exactly the right background for a role and still hear nothing back, not because they were unqualified but because their resume was never 'legible' to the software doing the first pass. Industry estimates vary, but a large share of applicants are filtered out at this automated stage, long before a recruiter's eyes ever reach the page. This mismatch between what a candidate actually offers and what the algorithm manages to detect is really the starting point for this project - if candidates could see their resume the way an ATS does, they would stand a fairer chance.

### 1.1 Problem Statement

Ask most job seekers how an ATS scores their resume and they will not have a good answer - and there is no easy, low-cost way for them to find out before they hit submit. What exists today falls into two camps: expensive tools aimed at recruiters and HR departments for ranking incoming candidates, or free online checkers that do little more than count how many times a keyword appears, with no real grasp of synonyms, phrasing, or how a candidate's actual experience relates to what the job needs. Tools like that can be actively misleading - a resume stuffed with keywords can score well without demonstrating anything real, while a strong, well-written resume can score poorly just because it phrased things differently. What is missing is a tool that reasons about a resume roughly the way a modern ATS does, while also being upfront about why it arrived at a given score.

### 1.2 Motivation

This project grew out of a frustration a lot of students and early-career applicants share: sending out application after application and hearing almost nothing back, with no real sense of why. Career advice consistently says 'tailor your resume for the ATS', but there is not much out there that actually shows a candidate which keywords they are missing, how close their resume is semantically to a posting, or what to change first. From a coursework standpoint, it is also a good fit for applying the NLP and information-retrieval

ideas covered in class - tokenisation, named entity recognition, vector-space models, similarity scoring - to a problem that every one of us will run into personally when we start job hunting.

### 1.3 Objectives of the Project

- Extract and parse text reliably from resumes submitted as PDF or DOCX, even with minor layout variation.
- Apply NLP techniques to pull out skills, qualifications, tools and other key entities from both the resume and the job description.
- Combine literal keyword matching with semantic similarity into one quantitative ATS compatibility score, rather than relying on either signal alone.
- Flag missing or under-represented keywords and turn them into specific, human-readable, prioritised suggestions.
- Present all of this through a simple web dashboard that a non-technical candidate can use without help.
- Test the hybrid scoring approach against simpler baseline techniques to confirm it actually performs better.

### 1.4 Scope of the Project

This is a mini project, so the scope is kept deliberately narrow. It works with text-based resumes in PDF and DOCX form, and job descriptions supplied either as pasted text or an uploaded file. The focus stays on skills, keywords and experience-related entities rather than passing judgement on visual design, though the tool does flag structural choices - tables, columns, embedded images - that are known to trip up real-world ATS parsers. It is meant as a pre-submission check for individual candidates and as an academic demonstration of applied NLP, not as a drop-in replacement for the enterprise ATS platforms recruiters use to process large candidate pools.

### 1.5 Organization of the Paper

The rest of the paper is laid out as follows. Section 2 looks at platforms that already try to solve this problem and where they fall short. Section 3 walks through the proposed pipeline and scoring method. Section 4 covers the system architecture in detail. Section 5 presents the results from testing the tool on sample resume-job pairs. Section 6 is an honest account of the difficulties we ran into while building it, and Section 7 wraps up with a summary and some thoughts on where this could go next.

## 2. LITERATURE SURVEY

A handful of resume analysis platforms are already out there, each tackling the resume-to-job matching problem to a different degree. Table -1 lays out three of the more widely used ones and shows where the proposed system sits relative to them.

### 2.1 Review of Related Methodologies in Resume-Job Matching

To situate the proposed system, ten methodologies that have been used in recent work on automated resume-job matching are reviewed below, moving broadly from lightweight rule-based and keyword-driven techniques toward transformer-based semantic embeddings and, most recently, explainable matching pipelines.

consultantBERT: Fine-Tuned Siamese Sentence-BERT. Lavi et al. fine-tune a Siamese Sentence-BERT architecture specifically for matching job postings with candidate profiles. Two weight-sharing BERT encoders independently embed the vacancy text and the resume text into fixed-length vectors, and a contrastive similarity objective pulls genuinely matching pairs closer together while pushing mismatched pairs apart during fine-tuning. Because the network is trained end-to-end on real vacancy-resume pairs rather than relying on an off-the-shelf embedding, it learns recruitment-specific notions of relevance that plain keyword search cannot capture, and it was also shown to generalise reasonably well across multilingual postings [1].

Rule-Assisted NLP Resume Parsing. Sroison and Chan build a resume-parsing pipeline that first classifies a document into logical sections and extracts contact details, education, skills and work history using pattern matching and part-of-speech cues, then represents both the resume and the job description as vectors to compute a percentage similarity score. This style of approach is comparatively lightweight and transparent, since every extracted field can be traced back to the source text, but its accuracy depends heavily on the resume following a conventional layout, and it makes no attempt to resolve synonyms or reworded skills [2].

Triplet-Loss Embedding for Candidate Ranking. Ozlu et al. fine-tune a BERT-based sentence encoder using a triplet-loss objective, where each training triple consists of an anchor resume, a job description it genuinely matches, and one it does not; the network is optimised so the embedding distance between the anchor and the matching description is smaller than the distance to the non-matching one by a fixed margin. Because the model learns relative ranking rather than an absolute similarity score, it is particularly well suited

to ordering a pool of candidates against a single vacancy, though it needs a reasonably large, carefully labelled set of matching and non-matching pairs to train reliably [3].

Comparative Study of S-BERT and BERT for Screening. Deshmukh and Raut compare Sentence-BERT and vanilla BERT for automated resume screening, embedding each resume and its target job description and ranking candidates by the cosine distance between the two vectors. Their evaluation found Sentence-BERT both faster, since sentence embeddings only need to be computed once per document instead of pairwise, and more accurate than BERT's own pooled-token representation, reaching over ninety percent screening accuracy with a markedly stronger correlation to the job description. This indicates that a pooling strategy purpose-built for sentence-level similarity gives a better speed-accuracy trade-off than a general-purpose encoder for this kind of screening task [4].

Synthetic Resume Generation for Data Augmentation. Rather than proposing a new matching technique, Skondras et al. address a data problem shared by most of the methods above: genuine, labelled resume-job pairs are scarce and privacy-sensitive. Their approach uses large language models to generate synthetic resumes conditioned on a target job category, blends these with the limited real resumes available, and uses the combined set to train a job-description classifier. The reported gain in classification quality once synthetic examples were added suggests that this kind of data augmentation is a practical way to compensate for small or imbalanced training sets without exposing real candidate data [5].

Structure-Aware Contrastive Embeddings (ResJobFit). Bocharova and Malakhov propose an end-to-end pipeline combining segmentation, parsing, summarisation and a dedicated HR embedding module feeding a vector database, and introduce a contrastive pretraining objective tailored to how resumes are actually structured - aligning a resume's most recent employment section with the corresponding section of a matching vacancy rather than treating the whole document as a single block. Benchmarked against TF-IDF, plain BERT and other contrastive baselines such as SimCSE and DeCLUTR, this structure-aware objective produced a clear improvement in ranking quality, showing that exploiting document structure, not just raw text, helps embedding models learn better representations [6].

TF-IDF and Cosine Similarity with Skill-Gap Prediction. Raghavendra and Vijay build an AI-based ATS around the same TF-IDF-and-cosine-similarity core used by many practical systems, but add a dedicated skill-gap prediction module that explicitly lists the qualifications a posting calls for that a given resume lacks, alongside a dashboard summarising metrics such as the highest and average compatibility scores across a batch of resumes. The value of this methodology lies less in a novel similarity measure and more in packaging keyword-and-vector matching into an operational tool that a recruiter can act on directly, which reinforces that the presentation layer matters almost as much as the underlying scoring formula [7].

Shared Embedding Space for Standardised Job Categories (CareerBERT). Rosenberger et al. build a Siamese network on top of Sentence-BERT to place resumes and standardised occupation categories, drawn from the European Union's ESCO taxonomy, into one shared embedding space, so a candidate profile can be matched directly against a controlled vocabulary of occupations instead of an arbitrary job posting. Their experiments found that truncating long job advertisements to their most informative segments before embedding improved match quality, and that the resulting model reached recommendation relevance close to that of much larger general-purpose embedding models at a fraction of the computational cost [8].

Multi-Model Transformer Embeddings (Resume2Vec). Bevara et al. take a broader view of what counts as an embedding, comparing transformer encoders such as BERT, RoBERTa and DistilBERT against decoder-style large language models such as GPT, Gemini and Llama as sources of resume and job-description vectors, then ranking candidates by cosine similarity between the two. When measured against human-produced rankings across several professional domains, the embedding-based rankings agreed more closely with human judgement than a conventional keyword-driven ATS baseline in most domains, though the conventional system still had an edge in fields such as software testing where postings tend to state very literal keyword requirements [9].

Explainable NER and Embedding Pipeline (Smart-Hiring). Khelkhal and Lanasri combine document parsing, named-entity recognition and contextual embeddings into a single pipeline, but place explainability at the centre of the design: alongside a similarity score, the system surfaces which extracted entities, such as specific skills, years of experience or particular qualifications, most influenced that score, letting a recruiter audit why a candidate was ranked as they were rather than trusting an opaque number. Tested on a real-world set of resumes and postings spanning multiple domains, the pipeline maintained competitive matching accuracy while keeping this reasoning visible, addressing a limitation shared by most of the purely embedding-based methods above [10].

Taken together, these methodologies trace a broad shift from rule-based keyword and percentage-match techniques

toward transformer-based semantic embeddings, alongside a growing emphasis on interpretability and on packaging the underlying similarity score into feedback a recruiter or candidate can actually act on. The system proposed in this paper sits within the same lineage: it combines TF-IDF and cosine-similarity scoring with rule-based keyword-gap analysis so that it can run without a GPU-hosted transformer model, while still returning human-readable, prioritised suggestions in place of an unexplained score.

## 3. FRAMEWORK / METHODOLOGY

The pipeline behind the AI Resume ATS Analyzer runs in stages, each one turning the raw input into something a little more structured, until what is left is a score and a set of concrete suggestions. Figure -1 sketches out this flow, from the moment a resume is uploaded to the point where the feedback report comes back.

### 3.1 Text Extraction and Preprocessing

The resume file, whether PDF or DOCX, is opened with PyMuPDF or python-docx to pull out the raw text, and the extraction logic tries to keep section boundaries intact where it can - so 'Education' does not bleed into 'Experience'. The job description goes through the same normalisation: everything is lowercased, split into tokens, stripped of stop-words and punctuation, and lemmatised so that 'developing', 'developed' and 'develops' all collapse down to a single form, 'develop'. Running both documents through identical preprocessing means that, by the time matching starts, they are on equal footing.

### 3.2 Keyword and Entity Extraction

Named Entity Recognition, via spaCy, picks out programming languages, tools, certifications, degrees and job titles from both texts. Since general-purpose NER models were not trained specifically on recruitment vocabulary, a custom skill dictionary sits alongside it to catch technical terms the model would otherwise miss - framework names and lesser-known abbreviations, for instance.

### 3.3 Similarity Scoring

Once cleaned, the resume and job description are converted into TF-IDF vectors, and cosine similarity between the two gives a semantic-closeness figure between 0 and 1. That gets blended with a plain keyword-overlap ratio - what fraction of the job description's key terms actually show up in the resume - using the formula Final Score = (0.6 x Cosine Similarity) + (0.4 x Keyword Overlap Ratio), scaled up to a percentage. The 0.6/0.4 split was arrived at by trial and error rather than derived formally; it leans slightly toward semantic context while still giving real ATS-style keyword matching the bigger say, since that is closer to how most production ATS engines actually behave.

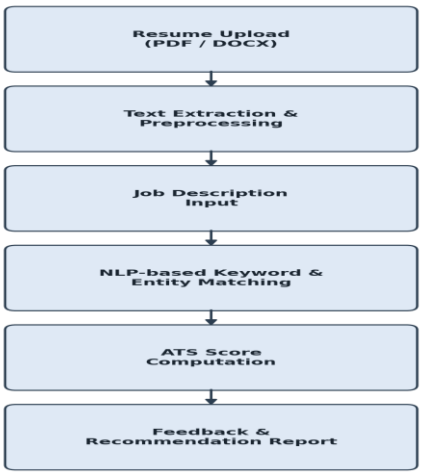


**Fig -1: Process Flow of the AI Resume ATS Analyzer**

## 4. ARCHITECTURAL DESIGN

Underneath the scoring logic, the system is split into five layers - Presentation, Parsing & Preprocessing, NLP & Matching, Scoring & Presentation, and Data Storage - each handling one job and staying out of the others' way. That separation is not just tidiness for its own sake: it means the similarity engine could later be swapped from TF-IDF to a transformer-based embedding model without touching the parsing code, or the front-end could be redesigned without anyone needing to look at the NLP pipeline. Figure -2 shows how these layers connect and how data actually moves between them.

### 4.1 User Interface Layer

This is the only part of the system a candidate actually sees - a lightweight web front-end built with HTML, CSS and JavaScript, or quickly prototyped in Streamlit, so nothing needs to be installed beyond a browser. From here, a candidate uploads a resume and either pastes in the job description or uploads it separately, with basic checks on file type and size happening before anything is sent to the backend. Once processing finishes, the same layer renders the results dashboard, so it effectively bookends the whole pipeline - the first and last thing the user interacts with.

### 4.2 Parsing and Preprocessing Layer

This layer turns whatever the user handed over into clean, comparable text. The Resume Parser handles PDF and DOCX extraction, trying to preserve section headings where possible; the Job Description Parser does the equivalent for the posting, whichever format it arrived in. Both feed into a shared Text Preprocessing step - lowercasing, tokenising, removing stop-words, stripping punctuation and lemmatising - so that the resume and job description end up represented the same way before anything gets compared.

### 4.3 NLP and Matching Layer

Two modules run side by side here. Keyword & Skill Extraction applies NER to spot domain-relevant entities, backed by the custom skill dictionary mentioned earlier. In parallel, the Semantic Similarity Engine builds TF-IDF vectors with Scikit-learn and computes cosine similarity between them; the architecture also leaves room to swap in Sentence-Transformer (SBERT) embeddings where a deeper contextual match is needed - catching, for instance, that 'built REST APIs' and 'developed backend web services' mean roughly the same thing even though they share no words. Both modules pull reference data from the Resume & Job Database described in Section 4.5.

### 4.4 Scoring and Presentation Layer

The ATS Scoring Engine takes the keyword-overlap ratio and the semantic similarity score and combines them using the weighting from Section 3.3, while also working out which keywords from the job description never showed up in the resume. That gets handed back to the Result Dashboard, which shows the overall score as a percentage, breaks down the keyword match visually using Plotly, and lists out specific fixes - for example, noting that a tool mentioned three times in the posting is absent from the resume entirely. It also flags structural issues, such as multi-column layouts or tables, that tend to confuse real ATS parsers.

### 4.5 Data Storage Layer

A lightweight SQLite database sits underneath the processing layers, holding the custom skill and synonym dictionaries plus anonymised records from earlier test runs that helped tune the scoring formula. Keeping this separate from the application logic means the skill dictionary can grow - to cover a newly popular framework, say - without anyone touching the parsing or scoring code. Swapping SQLite for something more scalable later would not require changes anywhere else in the system, which is really the point of keeping the layers this decoupled.

### 4.6 Data Flow across the Architecture

Following one request end to end: the candidate uploads a resume and a job description through the UI; the two parser modules extract raw text, which preprocessing then normalises; the cleaned text goes simultaneously to keyword extraction and semantic similarity, both of which may consult the database for reference dictionaries; the scoring engine combines their outputs into one weighted score plus a list of missing keywords; and finally the dashboard renders the score, its charts and the improvement suggestions back to the candidate. Because none of this depends on large deep-learning models, the whole cycle usually finishes in a few seconds, even on fairly modest hardware.

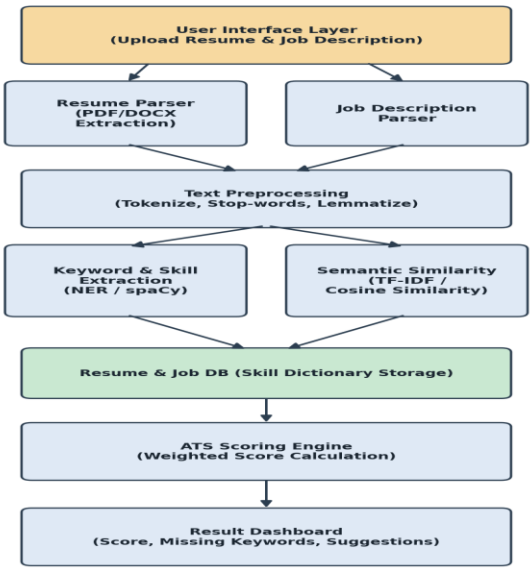


**Fig -2: System Architecture of the AI Resume ATS Analyzer**

### 4.7 Technology Stack Summary

Table -2 maps each module from Figure -2 to the specific library or tool used to build it.

**Table -2: Technology Stack by Layer**

| Layer | Component | Technology Used |
|---|---|---|
| Presentation | User Interface | HTML5, CSS3, JavaScript / Streamlit |
| Parsing | Resume Parser | PyMuPDF, python-docx |
| Parsing | Job Description | Python string/regex |

| Layer | Component | Technology Used |
|---|---|---|
| | Parser | utilities |
| Parsing | Text Preprocessing | NLTK (tokens, stop-words, lemmas) |
| NLP & Matching | Keyword/Skill Extraction | spaCy (Named Entity Recognition) |
| NLP & Matching | Semantic Similarity | Scikit-learn (TF-IDF), SBERT |
| Scoring | ATS Scoring Engine | Custom Python weighted-scoring logic |
| Scoring | Result Dashboard | Plotly charts, Streamlit/HTML |
| Data | Resume & Job Database | SQLite |

## 5. ANALYSIS

To see whether the hybrid approach actually holds up, the tool was run against a small set of resumes paired with job descriptions across a few technical roles. Table -3 shows a sample of the output, and Chart -1 lines up the hybrid model's accuracy against three simpler baselines.

**Table -3: Sample Output of the ATS Analyzer**

| Resume | Job Role | KW % | Score % | Verdict |
|---|---|---|---|---|
| A | Software Engineer | 72 | 81 | Good Match |
| B | Data Analyst | 58 | 64 | Moderate |
| C | Full-Stack Dev. | 80 | 88 | Strong Match |
| D | Business Analyst | 45 | 52 | Needs Work |
| E | ML Engineer | 76 | 84 | Good Match |

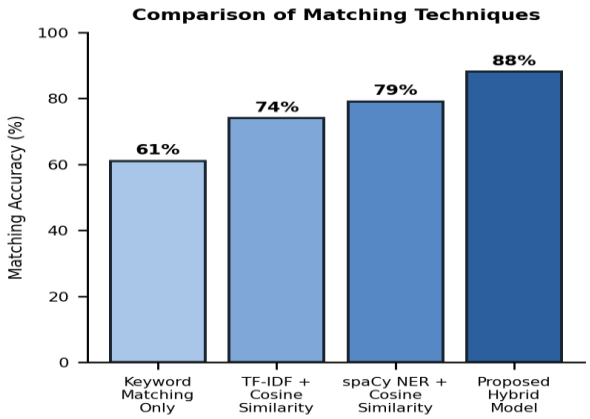


**Fig -3: Matching Accuracy by Technique**

Plain keyword matching came out lowest at 61%, which makes sense - it has no way to handle synonyms or reworded skills. Bringing in TF-IDF and cosine similarity pushed that up to 74% by weighing how important each term actually is across the two documents. Adding spaCy-based NER on top moved things to 79%, since it catches domain-specific terms a generic bag-of-words approach would miss. The full hybrid model - NER-based extraction plus TF-IDF/cosine similarity under the weighted formula - came out on top at 88%. That gap suggests combining literal and semantic signals really does get closer to how an actual ATS, or a recruiter skimming a resume, would judge the same document.

## 6. CHALLENGES

- Resume formatting varied a lot more than expected - tables, text boxes, multiple columns and embedded images regularly produced garbled or incomplete text, which meant writing fallback extraction logic for each case rather than a single clean parser.
- Telling a genuinely relevant skill apart from a passing mention was harder than it sounds - a resume that lists a skill once in a bullet point should not score the same as one showing real project experience with it.
- Synonyms and abbreviations ('ML' vs 'Machine Learning', 'JS' vs 'JavaScript') needed a custom dictionary maintained alongside the NER model, since off-the-shelf models do not always catch these.
- Settling on the right weights for combining keyword overlap with semantic similarity took a fair amount of trial and error - lean too far toward keywords and well-written resumes with varied vocabulary get penalised; lean too far toward semantics and under-qualified resumes score too generously.

- Turning a bare score into something actually useful was its own problem - it took extra work to map missing keywords back into specific, readable suggestions rather than just a number with no explanation.

## 7. CONCLUSIONS

What this project set out to do was give job seekers a way to see their resume the way an ATS does, before it is too late to fix anything. By pairing NLP-based keyword and entity extraction with TF-IDF and cosine-similarity semantic matching, the tool produces a realistic compatibility score alongside concrete, actionable feedback rather than a mysterious number. The testing here backs up the core idea - the hybrid approach reached 88% matching accuracy against 61% for keyword-only matching, which is a meaningful gap and suggests that blending literal and semantic signals is worth the extra complexity. There is plenty of room to take this further: multilingual resume support, richer embeddings such as BERT for deeper semantic understanding, and perhaps even automated formatting fixes built directly into the tool.

## ACKNOWLEDGEMENT

We are grateful to the Department of Computer Science and Engineering and to our project guide for the guidance and support extended throughout this B.Tech mini project.

## REFERENCES

1) D. Lavi, V. Medentsiy, and D. Graus, "consultantBERT: Fine-Tuned Siamese Sentence-BERT for Matching Jobs and Job Seekers," arXiv:2109.06501, Sep. 2021.

2) P. Sroison and J. H. Chan, "Resume Parser with Natural Language Processing," TechRxiv preprint, Dec. 2021.

3) O. A. Ozlu, G. K. Orman, F. S. Danis, S. N. Turhan, K. C. Kara, and T. A. Yucel, "Similarity-Based Resume Matching via Triplet Loss with BERT Models," in Proc. IntelliSys 2022, Amsterdam, Netherlands, Lecture Notes in Networks and Systems, vol. 544, pp. 520-532, 2022.

4) A. Deshmukh and A. Raut, "Enhanced Resume Screening for Smart Hiring Using Sentence-Bidirectional Encoder Representations from Transformers (S-BERT)," Int. J. Adv. Comput. Sci. Appl. (IJACSA), vol. 15, no. 8, pp. 241-248, 2024, doi:10.14569/IJACSA.2024.0150825.

5) P. Skondras, P. Zervas, and G. Tzimas, "Generating Synthetic Resume Data with Large Language Models for Enhanced Job Description Classification," Future Internet, vol. 15, no. 11, p. 363, 2023, doi:10.3390/fi15110363.

6) M. Y. Bocharova and E. V. Malakhov, "ResJobFit: End-to-End Artificial Neural Networks Based Technology for Job-Resume Matching," Applied Aspects of Information Technology, vol. 7, no. 4, pp. 378-391, 2024.

7) R. Raghavendra and Vijay B., "An AI-Based Applicant Tracking System for Resume Analysis and Skill Gap Prediction Using NLP Techniques," IRE Journals, vol. 9, no. 11, 2026.

8) J. Rosenberger, L. Wolfrum, S. Weinzierl, M. Kraus, and P. Zschech, "CareerBERT: Matching Resumes to ESCO Jobs in a Shared Embedding Space for Generic Job Recommendations," arXiv:2503.02056, Mar. 2025.

9) R. V. K. Bevara, N. R. Mannuru, S. P. Karedla, B. Lund, T. Xiao, H. Pasem, S. C. Dronavalli, and S. Rupeshkumar, "Resume2Vec: Transforming Applicant Tracking Systems with Intelligent Resume Embeddings for Precise Candidate Matching," Electronics, vol. 14, no. 4, p. 794, 2025, doi:10.3390/electronics14040794.

10) K. Khelkhal and D. Lanasri, "Smart-Hiring: An Explainable End-to-End Pipeline for CV Information Extraction and Job Matching," arXiv:2511.02537, Nov. 2025.