

AI-Based Resume Parser with ATS Scoring

Syed Ayaz

Dept. of Computer Science
CMR University
Bengaluru, India
syed.ayaz@cmr.edu.in

Vishwajeet

Dept. of Computer Science
CMR University
Bengaluru, India
vishwajeet@cmr.edu.in

Varun S Manchi

Dept. of Computer Science
CMR University
Bengaluru, India
varun_manchi@cmr.edu.in

Vanchhit Tiwari

Dept. of Computer Science
CMR University
Bengaluru, India
vanchhit.tiwari@cmr.edu.in

Prof. Varsha S

Assistant Professor, Dept. of Computer Science
CMR University
Bengaluru, India
s_varsha@cmr.edu.in

Abstract—Automated resume screening has become an operational necessity for organizations managing large volumes of job applications. This paper presents an AI-Based Resume Parser with Applicant Tracking System (ATS) Scoring, a practical web application that processes candidate resumes in PDF and DOCX formats and evaluates their compatibility against job descriptions. The system extracts resume text using format-specific parsing libraries, applies Natural Language Processing (NLP) techniques including tokenization, stop-word removal, and lemmatization via SpaCy, and computes a numerical ATS compatibility score using TF-IDF vectorization and cosine similarity. A skill-keyword matching phase supplements the similarity score by awarding targeted boosts for explicitly matched recruiter-specified skills. A rule-based resume quality audit module independently evaluates resume structure against eight heuristics covering contact information, section completeness, bullet-point formatting, quantified achievements, and date ranges. A locally hosted language model generates role-specific improvement feedback and resume rewriting suggestions, ensuring that candidate data remains within the institutional environment. The system is deployed as a Flask web application with role-differentiated dashboards for student and recruiter users, backed by a SQLite relational database. The current implementation focuses on functional output delivery rather than quantitative machine learning evaluation, and all scoring logic is deterministic and auditable.

Index Terms—resume parsing, applicant tracking system, natural language processing, TF-IDF, cosine similarity, SpaCy, Flask, candidate screening, text extraction.

I. INTRODUCTION

Recruitment has undergone a fundamental operational shift with the rise of online job portals. A single job posting on a popular platform can attract several hundred applications within a short window, creating a screening bottleneck that human resource teams cannot resolve through manual review alone [1]. Conventional Applicant Tracking Systems address the volume problem through keyword filtering, but this approach is known to exclude qualified candidates whose resumes describe relevant competencies using terminology that differs from the exact words in the job description [2].

At the same time, many candidates, particularly students and early-career professionals, are unaware of how their resumes are evaluated by automated screening tools. They

may submit well-qualified documents that nonetheless score poorly because they lack specific section structures, quantified achievements, or domain-aligned vocabulary. This double-sided information gap, where recruiters receive imprecise rankings and candidates receive no feedback, reduces the overall quality of the hiring process.

A. Motivation

The primary motivation behind this system is to make resume evaluation transparent and actionable for both parties. Rather than treating ATS scoring as an opaque black box, the proposed system exposes its scoring rationale by reporting which keywords are missing, which structural elements are absent, and what specific changes would improve the candidate's score. A secondary motivation is data privacy: the system routes all language model inference through a locally deployed model, so resume content does not leave the institutional network.

B. Scope and Contributions

This paper describes a practical, fully implemented system. The contributions are:

- A format-agnostic resume text extraction module supporting PDF and DOCX inputs.
- A three-phase ATS scoring pipeline combining TF-IDF cosine similarity, skill-keyword boosting, and a rule-based resume quality audit.
- A lemmatization-based missing keyword identification module using SpaCy.
- A locally hosted language model feedback system for privacy-preserving resume coaching and rewriting.
- A Flask web application with student and recruiter dashboards, SQLite persistence, and session-based authentication.

The system does not implement a trained machine learning classifier. All scoring and extraction logic is rule-based and deterministic.

II. LITERATURE SURVEY

A. Resume Parsing Approaches

Automated resume parsing has been studied from rule-based, statistical, and learning-based perspectives. Early systems relied on template matching and regular expressions to identify section boundaries and extract named fields such as name, phone, and education [5]. These approaches work well for structured resumes but fail when candidates deviate from expected layouts.

Retyk et al. [3] reformulated resume parsing as a hierarchical sequence-labeling task, using machine learning models to segment resume documents into semantic sections and extract entities within each section. This approach achieves better generalization across format variations than rule-based predecessors, at the cost of requiring annotated training data.

B. ATS Scoring and Candidate Ranking

Several systems have explored the use of TF-IDF vectorization and cosine similarity as the basis for resume-to-job-description compatibility scoring [4]. These methods are computationally inexpensive and produce interpretable scores, making them practical for real-time web applications. Their principal limitation is insensitivity to semantic equivalences: a resume that uses “deep learning” will not receive similarity credit from a job description that specifies “neural networks,” despite the conceptual overlap.

Bevara et al. [6] addressed this limitation by embedding both resumes and job descriptions in a shared vector space using transformer-based models, enabling semantic similarity measurement that is robust to vocabulary variation. While this approach yields improved ranking quality, it introduces higher computational requirements.

C. NLP-Based Resume Analysis

The use of NLP libraries such as SpaCy and NLTK for resume analysis has been well documented [5]. Lemmatization, in particular, is a widely applied preprocessing step that reduces vocabulary dimensionality by mapping inflected word forms to their base form, improving the recall of keyword matching at a negligible computational cost. The proposed system applies SpaCy’s `en_core_web_sm` model for lemmatization, following this established practice.

III. METHODOLOGY

A. NLP Preprocessing

Before any scoring or matching, extracted resume and job description text passes through the `_normalize_words_spacy` function in `ats_engine.py`. This function applies the following steps in order:

- 1) **Lowercasing:** All text is converted to lowercase so that “Python” and “python” are treated identically.
- 2) **SpaCy Processing:** The text is processed by the SpaCy `en_core_web_sm` pipeline, which performs tokenization, part-of-speech tagging, and lemmatization.

- 3) **Lemmatization:** Each token is replaced by its base form (e.g., “developing” becomes “develop”), reducing vocabulary size and improving keyword match recall.
- 4) **Filtering:** Stop words, punctuation marks, and whitespace tokens are discarded. Only tokens matching the pattern `[a-zA-Z0-9+#+-.\ ]+` are retained, which preserves technical identifiers such as “c++”, “node.js”, and “asp.net”.
- 5) **Fallback:** If the SpaCy model cannot be loaded, the function falls back to a regular expression that extracts alphanumeric tokens of the form `[a-zA-Z][a-zA-Z0-9+#+-.\ ]*`.

B. ATS Scoring Pipeline

The `ats_score` function computes a candidate’s compatibility with a job description in three sequential phases.

1) *Phase 1 — TF-IDF Cosine Similarity:* The resume text T_r and job description J are vectorized using Scikit-learn’s `TfidfVectorizer` with English stop words removed. The TF-IDF weight of term t in document d is:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log \frac{N}{\text{DF}(t)} \quad (1)$$

where $\text{TF}(t, d)$ is the normalized term frequency, N is the corpus size (two documents: resume and job description), and $\text{DF}(t)$ is the number of documents containing term t . This yields document vectors $\mathbf{v}_r$ and $\mathbf{v}_j$. The base similarity score is:

$$s_{\text{base}} = \frac{\mathbf{v}_r \cdot \mathbf{v}_j}{\|\mathbf{v}_r\|_2 \|\mathbf{v}_j\|_2} \quad (2)$$

$s_{\text{base}} \in [0, 1]$, with higher values indicating greater lexical overlap between the resume and the job description.

2) *Phase 2 — Skill-Keyword Boost:* If the recruiter has explicitly specified core skills K , the system computes a boost term:

$$s_{\text{boost}} = 0.05 \times \{w \in \text{lem}(K) \mid w \in \text{lem}(T_r)\} \quad (3)$$

where $\text{lem}(\cdot)$ denotes the lemmatized token set. Each matched core skill contributes a fixed increment of 0.05 (five percentage points) to the final score, directly rewarding candidates who explicitly list the recruiter’s required skills.

3) *Phase 3 — Final Score:* The two components are combined and capped at 100%:

$$S = \min(1.0, s_{\text{base}} + s_{\text{boost}}) \times 100 \quad (4)$$

C. Missing Keyword Identification

After preprocessing, missing keywords are identified by set subtraction. The target word set T is defined as the lemmatized skill string if one was provided, or the lemmatized job description otherwise:

$$T = \begin{cases} \text{lem}(K) & \text{if } K \neq \emptyset \\ \text{lem}(J) & \text{otherwise} \end{cases} \quad (5)$$

The missing keywords are:

$$M = \text{sort } T \setminus \text{lem}(T_r), \text{ by token length, descending } [: 20] \quad (6)$$

Longer tokens are ranked first on the assumption that specific multi-character technical terms (e.g., “kubernetes”, “tensorflow”) are more informative to a candidate than short generic words.

D. Resume Quality Audit

Independently of any job description, the `resume_quality_score` function evaluates a resume against eight structural heuristics. Starting from a base score of 30 points, it awards incremental points as described in Table I. Each unsatisfied heuristic produces a natural-language feedback item included in the response.

TABLE I
RESUME QUALITY AUDIT HEURISTICS

Heuristic	Points	Detection
Contact information present	+10	Regex: @ or phone
Summary / Objective section	+8	Keyword lookup
Experience section	+12	Keyword lookup
Skills section	+10	Keyword lookup
Education section	+8	Keyword lookup
≥4 bullet points	+10	Count of “-” and “•”
≥2 quantified outcomes	+12	Regex: %, +, \$, users
≥1 date range present	+10	Regex: year–year/present
Base score	30	
Maximum score	100	

IV. PROPOSED SYSTEM

A. System Overview

The system is a web-based resume parsing and ATS scoring platform. A candidate uploads a resume file through a browser interface. The system extracts the text, analyzes it against the target job description using NLP and TF-IDF scoring, identifies vocabulary gaps, audits the resume’s structural quality, and returns a numerical score with accompanying feedback. Recruiters can post job descriptions and view applicants ranked by their ATS scores. All data is stored in a local SQLite database.

The system is not a machine learning research framework. It implements deterministic, rule-based scoring logic that is fully auditable and produces the same output for the same inputs on every run.

B. Processing Pipeline Flowchart

The flowchart below illustrates the file-type detection pathways required for processing applications in the supported formats (PDF and DOCX).

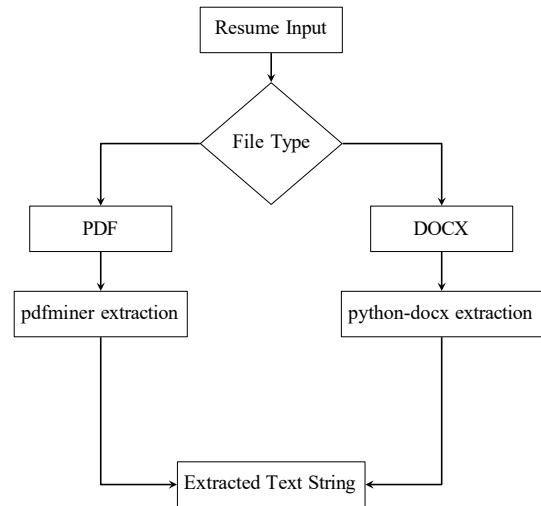


Fig. 1. Flowchart representing the Resume Parsing Pipeline for PDF and DOCX.

C. Resume Text Extraction

The `resume_parser.py` module accepts a file path and returns the plain text content of the resume. Two formats are supported:

- **PDF:** The `pdfminer.high_level.extract_text` function is used. It performs layout analysis and character decoding to recover the textual content from the PDF byte stream.
- **DOCX:** The `python-docx` library’s `Document` class iterates over paragraph objects and joins their `.text` attributes with newline separators.
- **Other formats:** An empty string is returned, and the routing layer presents the user with an error message.

If the extracted text is empty after stripping whitespace, the application rejects the upload and prompts the candidate to try a different file.

D. Resume Improvement Feedback

The `llm_engine.py` module communicates with a locally hosted Ollama server via HTTP POST requests to its `/api/generate` endpoint. The default model is `gemma3:1b`, configurable through the `OLLAMA_MODEL` environment variable. A temperature of 0.2 is used to produce consistent, factual outputs. Three feedback functions are implemented:

- **Job-specific coaching:** Sends the job title, job description, current ATS score, and missing keywords to the model and requests five actionable improvement bullet points.
- **General quality feedback:** Sends the resume text, quality score, and list of failed heuristics, and requests five general structural improvement tips.
- **Resume rewriting:** Sends the full resume text, job description, and missing keywords with an instruction to produce a complete rewritten resume integrating the missing keywords naturally.

Because Ollama runs locally, no resume content is transmitted to any external server or API.

V. SYSTEM ARCHITECTURE

The system follows a modular client-server architecture designed to cleanly separate web presentation concerns from the intensive NLP parsing operations.

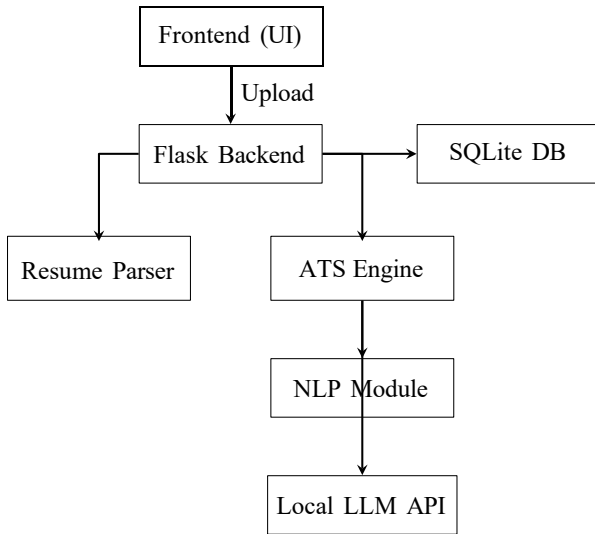


Fig. 2. System Architecture Block Diagram mapping components.

A. Component Description

The application is organized into the following functional layers:

- **Presentation Layer:** Jinja2 HTML templates and static CSS/JavaScript assets, providing separate dashboards for student and recruiter roles.
- **Routing Layer:** Flask route handlers managing HTTP requests, file uploads, session state, and flash messaging.
- **Text Extraction Layer:** Dispatches to format-specific parsers based on file extension.
- **Scoring and NLP Layer:** Implements all text preprocessing, similarity scoring, skill matching, and quality auditing.
- **Feedback Layer:** Sends structured prompts to a locally hosted language model and returns coaching feedback.
- **Data Layer:** SQLite database with four tables managing users, jobs, applications, and student profiles.

B. Implementation Details

Table II summarizes the complete technology stack of the implemented system.

The database schema includes four tables. The `users` table stores name, email (unique), hashed password, and role (student or recruiter). The `jobs` table stores job title, description, required skills, recruiter email, and application deadline. The `applications` table records the student email, job identifier, resume filename, computed ATS score, and JSON-serialized missing keyword list for each submission. The

TABLE II
SYSTEM IMPLEMENTATION STACK

Component	Technology
Web framework	Flask 2.x (Python 3)
NLP processing	SpaCy (en_core_web_sm)
TF-IDF and similarity	Scikit-learn
PDF extraction	pdfminer.six
DOCX extraction	python-docx
Local LLM inference	Ollama (gemma3:1b)
Database	SQLite (database.db)
Password security	Werkzeug generate_password_hash
File upload security	Werkzeug secure_filename
Frontend templates	Jinja2, HTML/CSS, JavaScript

`student_profiles` table stores extended candidate profile details including phone, location, LinkedIn URL, portfolio link, and a freeform skills description.

VI. RESULTS AND ANALYSIS

The system currently focuses on functional output delivery rather than quantitative machine learning evaluation. No trained classifier is used, and therefore no accuracy, precision, recall, or F1-score metrics are applicable to the scoring pipeline. The following section describes the functional behavior observed during system operation.

A. ATS Scoring Behavior

The scoring pipeline produces consistent results for a given resume and job description pair. Resumes that contain domain-relevant vocabulary from the job description receive higher TF-IDF cosine similarity scores. The skill-keyword boost correctly increases scores by five percentage points for each recruiter-specified skill found in the resume. Candidates who receive their missing keyword list and revise their resume accordingly observe a measurable improvement in their ATS score on resubmission, which is the intended design outcome.

B. Resume Quality Audit Behavior

The eight-heuristic quality audit reliably identifies structural deficiencies in student resumes. Commonly detected issues include the absence of a professional summary section, insufficient use of bullet-point formatting, and a lack of quantified achievement statements. The audit produces a score between 30 (base) and 100 (all heuristics satisfied) with corresponding natural-language feedback for each unsatisfied check. This behavior is deterministic: the same resume always produces the same quality score.

C. Missing Keyword Output

The lemmatized set-difference operation produces a ranked list of up to twenty missing keywords ordered by token length. Technical terms with many characters (e.g., “postgresql”, “kubernetes”, “authentication”) appear first, followed by shorter terms. This ordering provides candidates with the most specific and actionable vocabulary gaps at the top of the list.

D. System Output Fields

Table III summarizes the data fields produced by the system for each resume submission.

TABLE III
SYSTEM OUTPUT FOR A RESUME SUBMISSION

Output Field	Source
ATS compatibility score (0–100%)	TF-IDF cosine + skill boost
Missing keywords (up to 20)	Lemmatized set subtraction
Resume quality score (0–100%)	Eight structural heuristics
Quality feedback items (up to 8)	Failed heuristic messages
LLM coaching tips (5 bullets)	Locally hosted gemma3:1b
Rewritten resume text	LLM rewrite with keyword injection

E. Recruiter-Side Behavior

Recruiters can post job descriptions with an explicit skills field. After candidates submit applications, the recruiter dashboard displays a ranked list ordered by descending ATS score, allowing the recruiter to identify the highest-matching candidates without reviewing individual resumes manually.

VII. DISCUSSION

A. Advantages

The primary practical advantage of the system is the transparency of its scoring logic. Because every component of the ATS score is deterministic and derives from auditable operations (TF-IDF weighting, lemmatized set membership, and structural regex matching), both candidates and recruiters can understand exactly why a particular score was produced. This contrasts with black-box ML classifiers whose decision boundaries are opaque.

The privacy-preserving architecture is a meaningful benefit for institutional deployments. By running language model inference through a locally hosted Ollama instance, the system avoids transmitting sensitive personal documents to third-party cloud APIs.

The dual-dashboard design cleanly separates workflows: candidates receive structured feedback on how to improve their resume, while recruiters obtain a ranked applicant list without manual review effort.

B. Limitations

No semantic similarity: TF-IDF cosine similarity measures lexical overlap, not semantic equivalence. A resume that lists “PyTorch” will not receive similarity credit from a job description that specifies “deep learning framework,” even though the two are closely related. Sentence-transformer embeddings would address this limitation.

Rule-based extraction only: The system does not parse and extract individual resume fields such as candidate name, specific educational qualifications, or job titles from experience entries. Text extraction is document-level only. A trained Named Entity Recognition model would be required to achieve field-level extraction.

Fixed skill boost increment: Every matched skill contributes an identical 5-percentage-point boost regardless of its relative importance to the role. A core requirement and a peripheral tool receive equal weight.

Two-document TF-IDF corpus: The vectorizer is fitted on a two-document corpus (resume and job description) at inference time. This means IDF values have limited statistical meaning compared to a vectorizer fitted on a large background corpus.

LLM output format variability: Despite the low temperature setting, the locally hosted model occasionally deviates from the requested five-bullet output format. The frontend renders whatever text the model returns, without structural validation.

VIII. CONCLUSION

This paper presented an AI-Based Resume Parser with ATS Scoring, a practical web application that automates the evaluation of candidate resumes against job descriptions using rule-based NLP and TF-IDF similarity scoring. The system extracts text from PDF and DOCX files, applies SpaCy-based lemmatization and stop-word removal, computes a three-phase ATS compatibility score combining cosine similarity with skill-keyword boosting, and audits resume structure against eight quality heuristics. Missing keywords are identified through lemmatized set subtraction and returned to the candidate in ranked order. A locally hosted language model generates role-specific coaching tips and resume rewrites while keeping candidate data within the institutional environment.

The system is implemented as a Flask web application with role-differentiated student and recruiter dashboards, SQLite persistence, and session-based authentication. All scoring logic is deterministic and auditable, with no trained machine learning classifier in the current implementation.

The system fulfills its core design objective: providing candidates with a transparent, actionable breakdown of their resume’s alignment with a specific job description, and providing recruiters with a ranked applicant list derived from an interpretable scoring process.

IX. FUTURE WORK

The following enhancements are identified for future development:

- 1) **Semantic similarity:** Replace TF-IDF with Sentence-BERT [7] embeddings to capture semantic equivalences between synonymous skill expressions.
- 2) **Named Entity Recognition:** Integrate a SpaCy NER model trained on resume data to extract individual fields such as candidate name, phone number, educational degrees, and job titles from experience entries.
- 3) **Weighted skill scoring:** Replace the flat 5-point skill boost with a recruiter-configurable importance weight per skill.
- 4) **Corpus-fitted TF-IDF:** Pre-fit the TF-IDF vectorizer on a large corpus of job descriptions to obtain statistically meaningful IDF weights.

- 5) **ML-based ranking:** Train a supervised ranking model on recruiter shortlisting outcomes to learn an empirically calibrated scoring function.
- 6) **Bias auditing:** Implement a fairness check to detect demographic disparities in ATS score distributions, in line with recommendations from the algorithmic hiring fairness literature [10].
- 7) **Mobile interface:** Develop a responsive mobile frontend to improve accessibility for candidates across different devices.

REFERENCES

- [1] K. Khelkhal and D. Lanasri, "Smart-Hiring: An Explainable End-to-End Pipeline for CV Information Extraction and Job Matching," *arXiv preprint arXiv:2511.02537*, 2025.
- [2] A. Akkasi, "Job Description Parsing with Explainable Transformer-Based Skill Extraction," *Expert Systems with Applications*, vol. 240, 2024.
- [3] F. Retyk, H. Fabregat, J. Aizpuru, M. Taglio, and R. Zbib, "Re'sume' Parsing as Hierarchical Sequence Labeling," in *Proc. EMNLP Workshop on Document Intelligence*, 2022, pp. 45–54.
- [4] S. Patil and A. Kulkarni, "Resume Screening Automation Using Natural Language Processing Techniques," *International Journal of Innovative Science and Research Technology*, vol. 8, no. 3, pp. 1023–1031, 2023.
- [5] S. Karhale, P. Pawade, P. Sukala, S. Swami, and A. Khan, "Resume Analysis System Using Natural Language Processing," *International Journal of Latest Technology in Engineering, Management & Applied Science*, vol. 14, no. 2, 2025.
- [6] R. V. K. Bevara, N. R. Mannuru, S. P. Karedla, B. D. Lund, T. Xiao, and H. Pasem, "Resume2Vec: Transformer-Based Embeddings for Candidate-Job Compatibility," in *Proc. IEEE ICMLA*, 2024, pp. 312–319.
- [7] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *Proc. EMNLP-IJCNLP*, Hong Kong, 2019, pp. 3982–3992.
- [8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proc. NAACL-HLT*, Minneapolis, MN, 2019, pp. 4171–4186.
- [9] Z. Chen, "Ethics and Discrimination in Artificial Intelligence-Enabled Recruitment Practices," *Humanities and Social Sciences Communications*, vol. 10, no. 1, p. 567, 2023.
- [10] A. Fabris, "Fairness and Bias in Algorithmic Hiring: A Multidisciplinary Survey," *ACM Computing Surveys*, vol. 57, no. 4, 2025.
- [11] M. Langenkamp, A. Costa, and C. Cheung, "Hiring Fairly in the Age of Algorithms," *arXiv preprint arXiv:2004.07132*, 2020.