IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, November 2026
Impact Factor: 8.2

Survey of Resume Screener with Match Score

Shreeram Shinde, Aditya Wadmare, Abhishek Gamane, Sushil Rahatole, Prof. S. B. Dhadake
Department of Computer Engineering
STES's Smt. Kashibai Navale College of Engineering, Pune, India
Savitribai Phule Pune University Pune, India
shindeshreeram86@gmail.com, adityawadmare472@gmail.com
abhishekgamane14@gmail.com, sushilrahatole@gmail.com, swathidhadake@gmail.com

Abstract: *Judging if a resume fits a job posting isn't just a recruiter's headache—it's tough for applicants too. Reading stacks of resumes by hand takes forever, and results depend a lot on whoever's doing the reading. Most applicants have no clue which skills employers need but don't see on their resume. Previous work in resume-job matching has focused mostly on trained models and ranking scores, but the numbers these systems produce are hard for regular users to understand. Here, we lay out the design for Resume Screener with Match Score, a Flask web app in which someone uploads a resume PDF and a job description. The system will extract text from the resume using pdfplumber, normalize it, and scan both documents for skills using an editable dictionary. It will then output three things: matched skills, missing skills, and extra skills, along with a skill-match percentage $S = 100 \times |M|/|J|$, the ratio between found skills and those listed in the job posting. It will also show a TF-IDF cosine similarity score as a separate number, never mixed into the percentage. The paper covers five related studies, settles how to handle edge cases when scoring, outlines a precision-recall evaluation against hand-annotated data, and confirms that the underlying logic isn't complicated—it sits nicely in class P. Right now, it's just a detailed blueprint: there's no working code or results yet.*

Keywords: resume screening, job description, skill matching, TF-IDF, cosine similarity, explainable scoring, Flask, pdfplumber.

I. INTRODUCTION

One job post can bring in a flood of resumes, and every single one needs to be checked against the requirements. Bhatia and colleagues highlight the sheer volume of applications as a real blocker for employers who sort through them by hand [1]. Khelkhal and Lanasri say manual reading takes a ton of time, leads to mistakes, and opens the door to personal bias [2]. Clearly, there's demand for smarter tools to speed up the first screening step, and plenty of research has already gone into it.

But it's not just recruiters who struggle. Applicants—especially students prepping for campus interviews—have almost no way to tell which required skills show up in their resume and which are missing. If a tool could lay out exactly what's there and what's absent, students could tweak their resumes before sending them off, and recruiters would know exactly how the comparison works.

The actual matching is messy. Everyone writes their own resume, usually as a PDF, and that alone creates headaches. Lavi's group calls parsed resumes "noisy"—formats, tables, bullet points, you name it, everything's different [3]. On top of that, if you just use raw TF-IDF cosine similarity for matching, it barely sets apart good fits from bad ones [3]. A decent screening tool needs to explain what these numbers mean, and what they don't.

This paper introduces Resume Screener with Match Score—a straightforward web app focused on clarity. Contributions include (i) a precise skill-match percentage, with clear rules for tricky cases, (ii) transparency about which skills are matched, missing, or extra, (iii) a separate TF-IDF similarity score, (iv) a way to test the skill detector, and (v) proof the computation stays simple and fast (class P).

Copyright to IJARSCT
www.ijarsct.co.in
DOI: 10.48175/IJARSCT-40826

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 3, November 2026 Impact Factor: 8.2

Current status: We're at the drawing-board stage. There's no code, no tests, no results. Everything here is about the design and what could come next. Section II covers related work, Section III explains the architecture, Section IV dives into scoring and complexity, and Section V wraps up.

II. LITERATURE REVIEW

We picked these studies since they tackle resume-job matching from angles like text similarity, skill extraction, or explainable matching. They're all recent—published between 2019 and 2025. Every number here belongs to the original authors, not us.

A. Parsing and Supervised Ranking

Bhatia's team built a web tool that parses resumes and uses a BERT-based classifier to rank candidates [1]. If a past job doesn't have a description, they borrow one from previous experience. Their parser nailed 100% accuracy for LinkedIn format, and the candidate ranking hit a baseline 73% accuracy. Lavi and colleagues fine-tuned a multilingual Sentence-BERT model on over 274,000 resume-job pairs, using recruiters' labeling [3]. Their TF-IDF baseline reached a ROC-AUC of 0.5730, while their advanced model jumped to 0.8459. Vocabulary mismatch is a big culprit: one resume talks about "warehousing," the job asks for a "logistics worker," but the words don't match. Both studies lean on labeled datasets and trained models, but neither really tries to explain skill matches to users.

B. Explainable Pipelines

Smart-Hiring, from Khelkhal and Lanasri, runs resumes through a pipeline—starting with pdfplumber for PDFs, plus layout-aware tools and OCR for tricky or scanned resumes [2]. They search for skills using a fuzzy dictionary lookup, compare skill phrases through a sentence-embedding model, and blend weighted similarities into a single score. Their explainability layer highlights shared keywords and overlapping skills. They tested the approach on about a thousand CVs with input from HR experts. They also note that tricky formats and graphics can make skill extraction lose structure. It's the closest thing we've found to our own design, though their pipeline is bigger and depends on domain-specific weights.

C. TF-IDF and Cosine Similarity for Retrieval

Grech and Suda came up with a résumé search method for a recruitment agency that blends skip-gram word embeddings, averages them with TF-IDF weights, and then ranks results using cosine similarity [4]. They tested this on a set of 500 job descriptions and 2,000 résumés. To check how well it worked, they used the F1 score to evaluate the top ten résumés for three different job descriptions. One thing they noticed: the model ignores word order, which means if a job description mentions another job title, the search can pull up unrelated résumés.

Ajjam and Al-Raweshidy took a different approach. They built a real-time recruitment tool in Python that mixes TF-IDF, cosine similarity, and domain-specific keyword weights [5]. In a simulated test for software engineers, their weighted model reached a similarity score of 0.74, much higher than the 0.35 from just plain keyword matching. But, to be clear, these scores only measure similarity on their own data—they're not measures of accuracy.

TABLE I: Summary of Reviewed Papers

Ref.	Method	Limitation for this work
[1]	Resume parser; BERT sentence-pair ranking	Needs labeled data and a trained model; no explanation of skill level
[2]	pdfplumber, fuzzy lexicon, embeddings, explainability layer	Large pipeline; weights set by expertise; some info gets lost in layout
[3]	Siamese SBERT on 274,407 pairs; TF-IDF baseline	Proprietary labeled data; can't reproduce results at small scale

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 3, November 2026 Impact Factor: 8.2

[4]	TF-IDF-weighted skip-gram embeddings; cosine retrieval	Only three test queries; embeddings must be trained beforehand
[5]	TF-IDF, cosine, and keyword weighting in a web app	Only reports similarity (not accuracy); weighting hasn't been replicated

D. Research Gap

Most of these studies focus on ranking quality, trained models, or big, complicated pipelines. A lot of the results depend on datasets and models that a small student team simply can't build or use. That doesn't mean there's nothing else like them out there—but it does show there's space for a small prototype with a different priority: something transparent and reproducible. This prototype's main feature is a well-defined set overlap score, where users always see which skills matched and which are missing, and TF-IDF similarity gets reported on the side, with its limits clearly stated. Unlike most of the reviewed work, this approach doesn't need any training data. On the flip side, there's no quality evidence for it yet—evaluation will happen only after it's actually built.

III. PROPOSED SYSTEM ARCHITECTURE

A. System Overview

The proposed architecture is layered. At the top is the presentation layer—the HTML pages the user sees. Next, the application layer is a Flask app [6] that accepts requests, validates inputs, and sends them to a screening service. The processing layer contains separate Python modules for pulling text from PDFs, normalizing it, checking for skill matches, and running similarity calculations. The data layer is a configuration file with a list of relevant skills. Features like storing a history in SQLite, displaying results on a Chart.js dashboard, or exporting with ReportLab are extras for later on. Each time a user uploads something, it will be read into memory, processed, and then discarded—only the derived results go back to the user. Development will proceed in increments; the first increment covers only the path from upload to result page. See Figure 1 for the workflow.

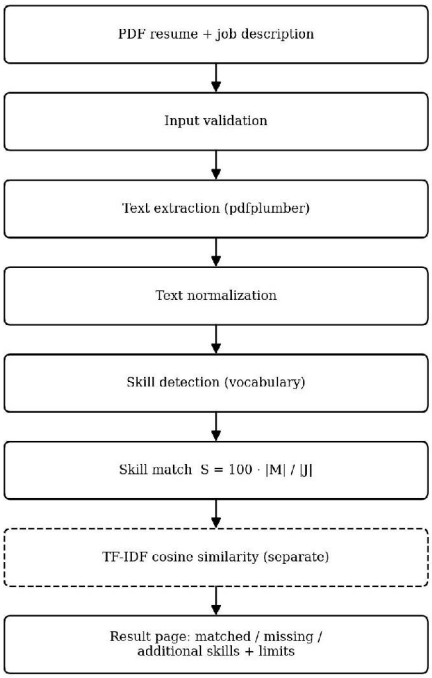


Fig. 1. Proposed workflow. The dashed box shows a separate indicator.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 3, November 2026 Impact Factor: 8.2

B. Input Validation and Text Extraction

Before anything gets processed, the system will check that a file was uploaded, that it is a PDF, that it carries the proper file signature, and that its size stays under a set limit. The filename supplied by the client will never be used to save or load files. The job description also has to be filled in and kept under a set length. The text will be read with pdfplumber [7], page by page. If the PDF is corrupted, or is just a scanned image with no real text, the system will show a clear error message and skip scoring. OCR isn't covered here.

C. Text Normalization

After extraction, the text will be normalized: Unicode normalization, conversion to lowercase, replacement of bullets and dashes with a standard form, and collapsing of repeated spaces. Specific characters like "+", "#", and "." stay untouched, so names like C++, C#, and Node.js aren't mangled.

D. Skill Detection

The skill detection method starts with a list of canonical skills—the vocabulary—which can be configured or edited without touching the code. Each skill has several aliases. A starter vocabulary of about 150 to 300 common software and IT skills is proposed, and the list stays easy to swap out or update.

For any skill, call its set of aliases A(s). The detection function D runs on a normalized text T like this:

$$D(T) = \{ s \in V : \exists\, a \in A(s), \text{ a occurs in T as a delimited token } \} \quad (1)$$

Matching respects token boundaries: an alias counts only when the characters directly before and after it are not letters, digits, "+" or "#", so "java" is not picked up inside "javascript" or "c" inside "c++". Several aliases may map to one skill, but each skill lands in the result just once.

Some names like C, R, or Go are just too short or ambiguous for a simple word search. The first version will either look for context phrases like "C programming" or just skip these cases and note it as a limitation. Also, the detection can't handle negation, so "no experience with Docker" still counts "Docker" as present.

When the vocabulary gets large, swapping to a faster pattern-matching algorithm like Aho–Corasick [11] makes alias searches efficient.

Table II walks through a hand calculation with invented sets. It illustrates the formula and is not a system result.

TABLE II: Hypothetical Hand Calculation of the Skill Match

Quantity	Value
J (job description)	{Python, Flask, SQL, Docker, Git}; $\|J\| = 5$
R (resume)	{Python, SQL, Git, HTML}; $\|R\| = 4$
$M = J \cap R$	{Python, SQL, Git}; $\|M\| = 3$
Missing $J \setminus R$	{Flask, Docker}
Additional $R \setminus J$	{HTML}
Skill match S	$100 \times 3/5 = 60.0\%$
Case $\|J\| = 0$	Unavailable (no division attempted)

And if the job description has no matching skills, the system will report "unavailable" instead of showing 0% or 100%, since either would be misleading.

E. Skill-Match Score

Running the detection on the job description and the resume gives J and R. The set M holds the skills they both have. Missing skills are those in J not found in R. Extras are in R but not J. The skill-match percentage is:

$$S = 100 \times |M| / |J|, \; |J| > 0 \quad (2)$$

The ratio is evaluated on unrounded values and rounded only for display. The system will show the percentage together with the counts |M|, |J| and |R|, because a small set makes the score coarse (with only two job skills, only 0%, 50%, or

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 3, November 2026 Impact Factor: 8.2

100% can occur).
This score only measures which skills overlap—it says nothing about proficiency, hiring chance, or model accuracy. It fully depends on the vocabulary and how the matching works.

F. TF-IDF Cosine Similarity

TF-IDF gives each word a weight: words that show up often in a document but not in every document stand out more, a principle first stated for term specificity by Spärck Jones [8]. Scikit-learn's TfidfVectorizer [9], [10] computes these weights, and by default it scales each document's vector to unit length.
For the resume and job description, we get TF-IDF vectors x and y, and their cosine similarity is:

$$\cos(x, y) = (x \cdot y) / (\| x \| \| y \|) \quad (3)$$

The value is always between 0 and 1, since the weights can't go below zero. This number isn't combined with the skill match—it's just another data point. It compares weighted word occurrences without capturing meaning, so two texts that express the same idea in different words get a low similarity, the vocabulary gap reported in [3].
But there's a catch: with only two documents, the default smoothed inverse document frequency is 1 for a word present in both and about 1.41 for a word present in one, so common and rare words are barely separated and the value behaves like a normalized word-count similarity. Two options remain open: fitting the vectorizer on a small reference collection of job descriptions, or limiting the comparison to skill phrases. The method, plus stopwords and n-gram lengths, will get finalized in the first version.

G. Planned Evaluation

Once we have the basic version ready, the team will pull together a set of about twenty resume and job description pairs—some from their own resumes, others anonymized with consent, plus public job postings. Two people will go through each text separately, listing out the skills vocabulary, and they'll settle any disagreements by discussion. We'll compare their lists to what the skill detector picks up and count up true positives, false positives, and false negatives. That gives us:

$$\text{Precision} = TP / (TP + FP) \quad (4)$$

$$\text{Recall} = TP / (TP + FN) \quad (5)$$

$$F1 = 2 \cdot \text{Precision} \cdot \text{Recall} / (\text{Precision} + \text{Recall}) \quad (6)$$

We'll report these numbers for skill detection, along with the difference between the screener's match percentage and the percentage calculated from the reference lists. Processing time gets reported too—on whatever machine we're using. Just to be clear, this only checks skill detection, not whether someone would actually get hired, since the system doesn't try to make that prediction. The sample size is small, so results will be a first look, and we'll dig into any failures.

IV. THEORETICAL ANALYSIS AND COMPLEXITY

A. Score Properties

Since M is a subset of J, the score S always falls between 0 and 100 when J isn't empty. You only hit S = 100 if every job skill is found in the resume, and S = 0 when there's no overlap at all. For a given job, the score jumps in increments of 100 divided by how many skills the job lists. This is the granularity limit mentioned in Section III-E.

B. Computational Feasibility

Let's call n the text length and L the total length of all aliases. Scanning the text once for each of a aliases costs O(na), but Aho–Corasick does it in O(n + L + z) for z matches [11]. With hash sets, comparing job and resume skills takes expected time O(|J| + |R|), just a single division for the percentage, and building sparse TF-IDF vectors is linear in text length and distinct terms.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 3, November 2026 Impact Factor: 8.2

The main question, "does S $\geq \theta$ with |J| > 0?" boils down to a Boolean expression of skill variables, x(s), which are true when a skill shows up in a resume. That can look like a satisfiability problem—which is NP-complete [12]—but here, detection locks down each x(s), so no search is needed, and the answer comes in linear time. So, it lands in class P. A related challenge would be finding the smallest set of courses to cover all missing skills for multiple jobs, a Set Cover problem, which is NP-hard—but that's beyond what we're tackling right now.

V. CONCLUSION

This paper laid out the design of Resume Screener with Match Score, an app for comparing a resume in PDF against a job description. The main thing: keep the match score simple and fully defined—S = 100 × |M|/|J| for distinct detected skills, reported as unavailable if the job description yields no skill. Alongside the score itself, the app will show which skills matched and which are missing. TF-IDF cosine similarity will also be provided, with its limits and weak standalone performance [3] documented. Everything runs in polynomial time, so the boundaries here are about how good the skill vocabulary and PDF extraction are, not computational restrictions.

We're still at the design stage—no app exists yet, so there are no results on accuracy, speed, or user feedback. The takeaways are about solid planning, not tested performance. Known gaps include: heavy reliance on the vocabulary, no handling for negation, problems with short ambiguous skill names, and no support for scanned PDFs, non-English text, or Word docs. Next steps are to get sign-off on score definitions, build and review the first vocabulary, build out the modules (with unit tests), fix the TF-IDF setup, and carry out Section III-G's evaluation plan. Possible add-ons: OCR for scanned resumes, DOCX support, section-aware extraction to split out skills from experience and from lists, separate "required" vs. "preferred" skills, and testing with sentence-embedding methods [2], [3]. Anything that ranks candidates needs a fairness review before going live.

REFERENCES

[1] V. Bhatia, P. Rawat, A. Kumar and R. R. Shah, "End-to-end resume parsing and finding candidates for a job description using BERT," arXiv:1910.03089 [cs.IR], 2019.

[2] K. Khelkhal and D. Lanasri, "Smart-Hiring: An explainable end-to-end pipeline for CV information extraction and job matching," arXiv:2511.02537 [cs.CL], Nov. 2025.

[3] D. Lavi, V. Medentsiy and D. Graus, "conSultantBERT: Fine-tuned Siamese Sentence-BERT for matching jobs and job seekers," in Proc. RecSys in HR 2021 Workshop, Amsterdam, The Netherlands, Oct. 2021, arXiv:2109.06501.

[4] B. Grech and D. Suda, "A neural information retrieval approach for résumé searching in a recruitment agency," in Proc. 9th Int. Conf. Pattern Recognit. Appl. Methods (ICPRAM), 2020, pp. 645–651, doi: 10.5220/0009355006450651.

[5] M.-H. Ajjam and H. S. Al-Raweshidy, "AI-driven semantic similarity-based job matching framework for recruitment systems," Inf. Sci., vol. 724, Art. no. 122728, 2025, doi: 10.1016/j.ins.2025.122728.

[6] Pallets Projects, "Flask documentation." [Online]. Available: https://flask.palletsprojects.com/

[7] J. Singer-Vine and contributors, "pdfplumber: Plumb a PDF for detailed information about each text character, rectangle, and line." [Online]. Available: https://github.com/jsvine/pdfplumber

[8] K. Spärck Jones, "A statistical interpretation of term specificity and its application in retrieval," J. Documentation, vol. 28, no. 1, pp. 11–21, 1972.

[9] scikit-learn developers, "sklearn.feature_extraction.text.TfidfVectorizer," scikit-learn documentation. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.Tfidf Vectorizer

[10] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[11] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," Commun. ACM, vol. 18, no. 6, pp. 333–340, 1975.

[12] S. A. Cook, "The complexity of theorem-proving procedures," in Proc. 3rd Annu. ACM Symp. Theory Comput. (STOC), 1971, pp. 151–158.