WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

AI Job Application Agent: Resume-Based Job Discovery, Recommendation and Application Tracking

Tripti Sharma, Khushi Tiwari, Rashi Jain, Dr. Sonu Agarwal

Project Team, Department of Computer Science, Shri Shankaracharya Professional University

ABSTRACT

Looking for a job usually means visiting several portals, reading long descriptions and then checking each one against your own resume. Students searching for entry-level roles find this especially tiring, and they also lose track of where they have already applied. This paper describes the AI Job Application Agent, a web application that tries to put these tasks in one place.

A candidate signs in with Firebase Authentication and uploads a PDF or DOCX resume. The text is passed to the Gemini API, which pulls out skills, education and experience, and the candidate can correct the resulting profile. Job postings are collected from the public Greenhouse, Lever and Ashby interfaces, stored in MongoDB Atlas and filtered by location and eligibility. A Python service built with scikit-learn then turns the profile and the postings into TF-IDF vectors and ranks the postings by cosine similarity. Candidates can save jobs, open the employer's own application page, and record statuses such as Applied, Interview, Rejected or Offer themselves. Nothing is submitted automatically.

The current dataset holds 624 postings synced from four public job boards. The match score only measures how similar two pieces of text are; it is not a prediction of being hired. We describe the design and implementation, and we discuss the limitations we found: matching on words misses synonyms, few suitable India and remote student roles appear in the feed, and the recommendations have not yet been tested with a larger group of real users.

Keywords: job recommendation, resume parsing, TF-IDF, cosine similarity, content-based filtering, application tracking

1. Introduction

Anyone who has looked for a job recently knows how scattered the process is. Openings are posted on company career pages and on several job boards, each with its own search box and its own filters. After finding a posting, the candidate still has to read the description and decide whether the skills it asks for match what is written on the resume. Students and fresh graduates have it harder, because the entry-level roles they are eligible for are mixed in with hundreds of senior ones and are easy to overlook. Once applications start going out, remembering which company replied, which one called for an interview and which one said no usually ends up in a notebook or a spreadsheet.

Much of this effort is repetitive, and part of it can be handled by software. Comparing a resume with a job description is a text-matching problem, and text matching has well-known tools. In the vector space model every document is written as a list of weighted terms, and the angle between two such lists tells us how alike the documents are [1], [3]. TF-IDF weighting combined with cosine similarity is simple, quick to compute and needs no training data. It also gives a score that an ordinary user can understand, which is why it is still used in resume and job matching work, often as the baseline against which newer methods are measured.

In this project we built the AI Job Application Agent, a web application that brings job discovery, resume-based recommendation and application tracking into one workspace. The candidate uploads a resume, the Gemini API extracts the main fields, and the candidate reviews them. Jobs fetched from public applicant tracking system (ATS) boards are then ranked against this profile with TF-IDF and cosine similarity. The candidate can save jobs and keep a record of the status of each application. We made one design choice on purpose. The Apply Now button only opens the employer's own page; the system never submits an application on the candidate's behalf, so every decision stays with the person applying.

The rest of the paper is organised as follows. Section 2 reviews related work. Section 3 explains the methodology and the mathematics behind TF-IDF and cosine similarity. Section 4 describes the system architecture and the technologies used, and Section 5 walks through the working of the algorithm. Sections 6 and 7 cover the implementation, the results and the limitations, and Sections 8 and 9 give the future scope and the conclusion.

2. Related Work

Content-based filtering is the usual starting point for job recommendation when there is little or no history of user behaviour, because it only needs the text of the profile and the text of the postings. Collaborative approaches learn from what similar users did, and they need a user base that a new system does not yet have. Within content-based work, TF-IDF with cosine similarity is a common choice for comparing resumes with job descriptions, mainly because the resulting score is easy to explain to the user [3].

The weakness of this approach is that it compares words and not meanings. A resume that says “ML” and a posting that says “machine learning” share no term, so the score does not show that they are about the same thing. Embedding-based models try to fix this. In a recent master’s study, Power compared Sentence-BERT with a TF-IDF baseline for matching resumes to job descriptions and reported that the transformer model ranked jobs more accurately [5]. We keep TF-IDF in the present version because it is transparent and cheap to run, and we list embeddings under future work.

3. Methodology

The system follows the five stages shown in Fig. 1. In the input stage the candidate uploads a PDF or DOCX resume and fills in profile details. In preprocessing, the text is extracted from the file and then cleaned and normalised so that skills, education and experience can be handled in a uniform way. In the AI processing stage the Gemini API parses the qualifications from the resume, and the TF-IDF and cosine similarity model ranks the jobs. The database stage saves profiles, jobs and application data in MongoDB Atlas so that they can be retrieved quickly later, and the output stage shows the personalised job matches and lets the candidate save jobs and follow the status of applications.

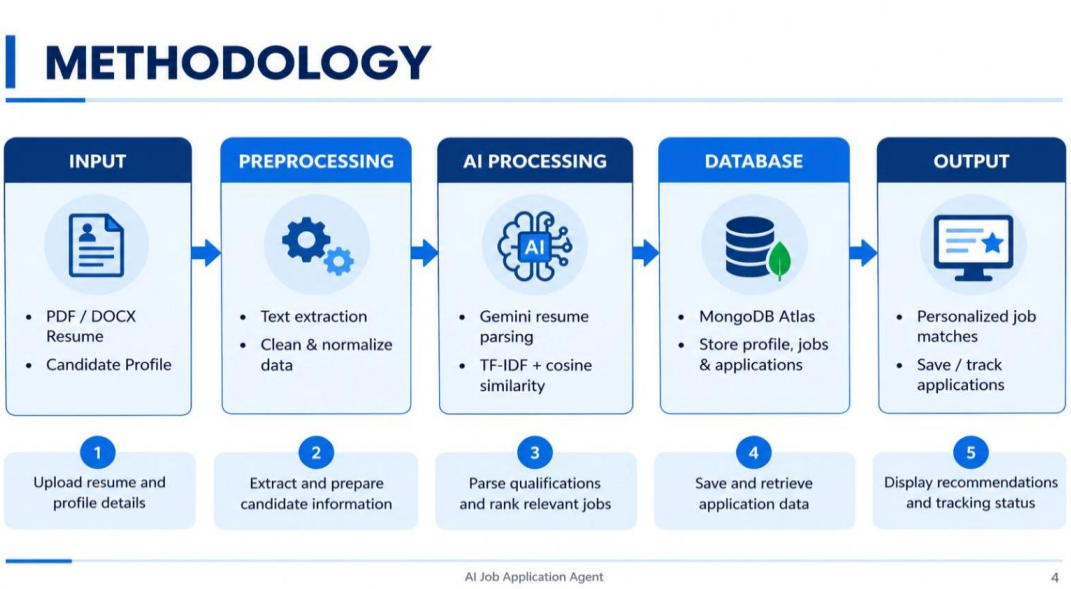


Fig. 1. Methodology of the AI Job Application Agent.

3.1 Term frequency–inverse document frequency

TF-IDF gives every word in a document a number that says how important that word is for that document. A word that appears many times in one document gets a high score. If the same word also appears in almost every document of the collection, it tells us little about any one of them, so its score is pushed down. In our setting a *document* is either the candidate’s profile text or the text of one job posting, and the *corpus* is the set of all such texts that are compared together. The symbols used below are:

- t: a term, that is, one word of the vocabulary;
- d: a document (the profile or one job posting);
- n: the number of documents in the corpus;
- $\mathrm{tf}(t, d)$: term frequency, the number of times t occurs in d;
- $\mathrm{df}(t)$: document frequency, the number of documents that contain t;
- $\mathrm{idf}(t)$: inverse document frequency, which is small for common terms and large for rare ones;
- m: the number of distinct terms (the vocabulary size), so every document becomes a vector with m entries.

The weight of term *t* in document *d* is the product of the two parts [1]:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t) \qquad (1)$$

The idea of idf comes from Spärck Jones [2]. In textbooks it is often written as $\log(n / \text{df}(t))$. We use the version that scikit-learn applies by default [4], which adds one to the numerator and the denominator so that no division by zero can occur, and adds one to the result so that a term found in every document is not given a weight of exactly zero:

$$\text{idf}(t) = \ln[(1 + n) / (1 + \text{df}(t))] + 1 \qquad (2)$$

Long documents have larger raw counts than short ones, so each vector is divided by its Euclidean length. After this step every document vector has length 1:

$$u(d) = v(d) / \|v(d)\|, \quad \text{where } \|v\| = \sqrt{(w_1^2 + w_2^2 + \ldots + w_m^2)} \qquad (3)$$

Here $v(d)$ is the vector $(w_1, w_2, \ldots, w_m)$ of tf-idf weights of document *d*, and $u(d)$ is the normalised vector.

3.2 Cosine similarity

Let **p** be the normalised vector of the candidate profile and **j** the normalised vector of a job posting. Cosine similarity is the cosine of the angle between them:

$$\text{sim}(p, j) = (p \cdot j) / (\|p\| \, \|j\|) \qquad (4)$$

where the dot product is the sum of products of matching entries:

$$p \cdot j = p_1 j_1 + p_2 j_2 + \ldots + p_m j_m \qquad (5)$$

All tf-idf weights are zero or positive, so the score lies between 0 and 1. A score of 0 means the two texts share no term at all, and a score close to 1 means they use nearly the same terms in similar proportions. Since both vectors were already normalised in (3), the denominator in (4) equals 1 and the similarity is simply the dot product. For each candidate, the jobs are sorted by this score from highest to lowest.

3.3 A small worked example

To check the formulas, take three very short documents. D1 is a profile, “python machine learning python”; D2 is a job posting, “python developer machine learning”; and D3 is another posting, “sales manager marketing”. With $n = 3$, the word “python” occurs in two documents, so idf = ln(4/3) + 1 = 1.288, while “developer” occurs in one, so idf = ln(4/2) + 1 = 1.693. In D1 the word “python” occurs twice, so its raw weight is 2 × 1.288 = 2.575; after the raw weights of D1 are divided by their length (3.154) it becomes 0.816. Table I lists all the values; they agree with the output of scikit-learn’s TfidfVectorizer.

TABLE I. TF-IDF WEIGHTS FOR THE WORKED EXAMPLE (NORMALISED)

Term	df	idf	D1 (profile)	D2 (job A)	D3 (job B)
developer	1	1.693	0	0.605	0
learning	2	1.288	0.408	0.460	0
machine	2	1.288	0.408	0.460	0
manager	1	1.693	0	0	0.577
marketing	1	1.693	0	0	0.577
python	2	1.288	0.816	0.460	0
sales	1	1.693	0	0	0.577

The similarity of the profile to job A is 0.408 × 0.460 + 0.408 × 0.460 + 0.816 × 0.460 = 0.751, and to job B it is 0, because they have no word in common. Job A is therefore ranked first, which is what we would expect.

3.4 What the score means

The match score estimates how similar the text of a job description is to the text of the resume and profile. It is not a hiring probability and it does not guarantee anything about the outcome of an application, and the interface presents it only as an aid for deciding which postings to read first.

4. System Architecture

The architecture is shown in Fig. 2. The frontend is built with React, Vite and Tailwind CSS and has screens for sign-up and login, resume upload, viewing and editing the profile, job search with filters, recommendations, saved jobs and application tracking. It talks to the backend over HTTPS using a REST API with JSON requests and responses.

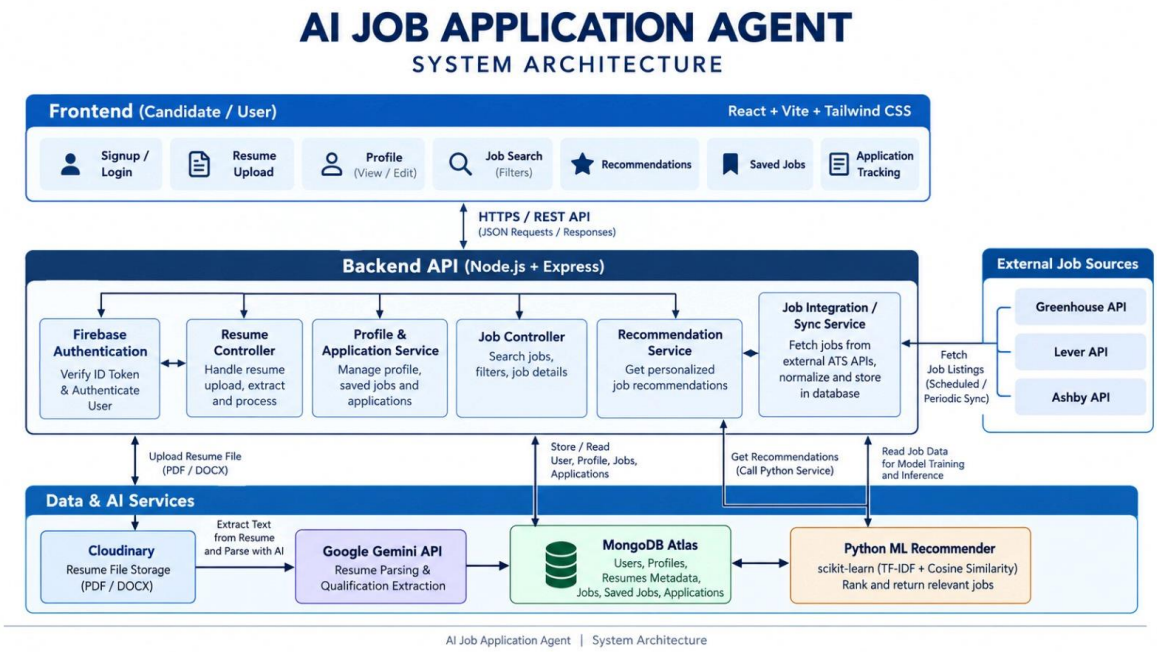


Fig. 2. System architecture of the AI Job Application Agent.

The backend is a Node.js and Express application that is divided into services. Firebase Authentication verifies the ID token that comes with each request. A resume controller handles the upload, extraction and processing of resumes. A profile and application service manages profiles, saved jobs and applications, and a job controller takes care of searching, filtering and showing job details. The recommendation service asks the Python recommender for personalised results. A job integration and sync service fetches listings from the Greenhouse, Lever and Ashby APIs on a schedule, normalises them and stores them in the database.

Behind the backend are the data and AI services. Resume files (PDF or DOCX) are kept in Cloudinary, and the text extracted from them is sent to the Google Gemini API for parsing and qualification extraction. MongoDB Atlas stores users, profiles, resume metadata, jobs, saved jobs and applications. The Python recommender, built on scikit-learn, reads the job data, ranks the jobs with TF-IDF and cosine similarity and returns the relevant ones. The technologies are summarised in Table II.

TABLE II. TECHNOLOGIES USED

Layer	Technologies	Responsibility
Frontend	React, Vite, Tailwind CSS, React Router, Framer Motion	User interface and navigation
Backend	Node.js, Express.js	REST APIs and validation
Database	MongoDB Atlas, Mongoose	Profiles, jobs, applications
Auth and storage	Firebase Authentication, Cloudinary	Sign-in and resume files
Resume parsing	Google Gemini API	Skills, education, experience
Recommendation	Python, scikit-learn	TF-IDF and cosine similarity

5. Algorithm and Working

Fig. 3 summarises how the system works from the first upload to the final output. The user uploads a resume and provides profile details. The resume text is extracted and cleaned to prepare structured data. Gemini then extracts the qualifications, and the TF-IDF and cosine similarity step ranks the jobs. The results are stored in MongoDB Atlas for fast retrieval, and finally the personalised recommendations are shown and the applications can be saved and tracked.

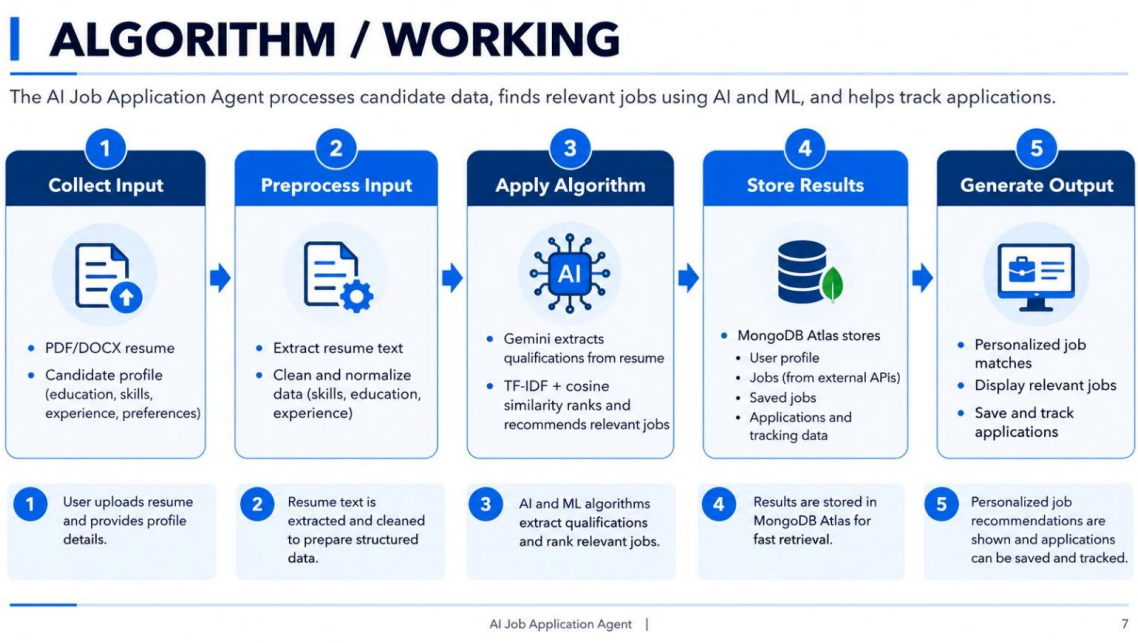


Fig. 3. Algorithm and working of the system.

The ranking part of this flow can be written as the following procedure.

1. Extract the text of the uploaded resume, parse it with Gemini and let the candidate confirm or correct the profile; this gives the profile text *P*.
2. Read the job postings from the database and remove those that fail the location and eligibility filters.
3. Fit a TF-IDF vectoriser on *P* and the text of the remaining postings, using Eqs. (1) to (3).
4. For every posting *j*, compute sim(*p*, *j*) with Eq. (4).
5. Sort the postings by score in descending order and return the top ones with the score and the matched skills.

The matched skills are shown next to every result so that the candidate can see why a job was ranked highly. When the candidate decides to apply, Apply Now opens the employer's application page and the application itself is made there, not inside our system.

6. Implementation

The following modules have been implemented and are working: sign-up and login with Firebase, resume upload and parsing, an editable profile, job search with filters, a recommended-jobs endpoint with its user interface, saving and unsaving jobs, application status tracking, and a dashboard with counts and an overview. The dashboard numbers are calculated from the records stored in the database. Statuses (Applied, Interview, Rejected and Offer) are set by the candidate, since the system has no access to the employers' own systems.

7. Results and Limitations

In the current dataset snapshot, 624 postings were synced from four public ATS boards. At this stage the work demonstrates a functioning system. We have not yet measured the quality of the recommendations with a proper evaluation, so we do not report accuracy figures in this paper.

Several limitations should be kept in mind. First, suitable India and remote roles for students are limited in the current feed, and no ranking method can recommend jobs that are not in the data. Second, TF-IDF is lexical matching, so synonyms and related terms can be missed. Third, the relevance of the recommendations needs to be tested with more real users and resumes. Finally, application statuses are typed in by the candidate and are not checked against any employer.

8. Future Scope

The first step is to test the recommendations with real resumes and to collect feedback from users. We also plan to add verified entry-level sources for India and remote work, and to evaluate the ranking with a measure such as precision@k. Semantic embeddings could then be explored to deal with synonyms, in the way reported in [5]. An optional automation feature, in which the candidate confirms each step before anything is sent, may also be considered.

9. Conclusion

The AI Job Application Agent gives a candidate one workspace for resume parsing, job discovery, text-similarity recommendations, saved jobs and application tracking. By using TF-IDF and cosine similarity, the ranking stays easy to explain, and by leaving the actual application to the candidate, the system keeps the person in control. The most important next steps are broader job sources and a proper evaluation with real users, followed by semantic matching.

References

1. G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.
2. K. Spärck Jones, “A statistical interpretation of term specificity and its application in retrieval,” *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, 1972.
3. C. D. Manning, P. Raghavan and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge University Press, 2008.
4. F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
5. A. Power, “Comparative evaluation of resume–job matching using transformer-based semantic similarity and TF-IDF,” M.Sc. thesis, National College of Ireland, Dublin, 2025.
6. React documentation. [Online]. Available: https://react.dev
7. Vite documentation. [Online]. Available: https://vite.dev
8. Tailwind CSS documentation. [Online]. Available: https://tailwindcss.com
9. Express documentation. [Online]. Available: https://expressjs.com
10. MongoDB documentation. [Online]. Available: https://mongodb.com/docs
11. Firebase documentation. [Online]. Available: https://firebase.google.com/docs
12. Google Gemini API documentation. [Online]. Available: https://ai.google.dev
13. scikit-learn documentation. [Online]. Available: https://scikit-learn.org
14. Greenhouse Job Board API. [Online]. Available: https://developers.greenhouse.io/job-board.html
15. Lever Postings API. [Online]. Available: https://github.com/lever/postings-api