IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511176

# ResuMatch: AI-Based Résumé–Job Matching and Mock Interview System

**Girithar V.K**

Student, II MCA, Nehru College of Management, Coimbatore, Tamil Nadu, India[2]

**Abstract:** Job seekers often do not know why a résumé is rejected, and recruiters spend considerable time reading résumés one by one. Many companies filter applications by keywords, so a capable candidate can be rejected because the résumé does not reflect the language of the job description. This paper presents ResuMatch, an AI-based web application that compares a résumé with a job description and returns a match score from 0 to 100, a verdict, strengths, weaknesses, matched and missing keywords, a skill-wise breakdown and suggestions for improvement. The user uploads a PDF, DOCX or TXT résumé or pastes its text; the backend extracts the text, checks the user's plan limit, and sends both documents to a large language model with instructions to reply in a fixed JSON schema. The reply is cleaned, bounded and stored so that it can be viewed again or downloaded as a PDF report. ResuMatch also provides a timed AI mock interviewer that asks one question at a time and produces a scored feedback report, a batch mode that allows organizations to screen up to 25 résumés together, subscription plans with usage limits, and an administrator dashboard. The system is built with React, Flask and PostgreSQL (with an automatic SQLite fallback), uses JSON Web Tokens and bcrypt for security, and can switch between Gemini, Groq and Claude if one AI provider fails. The project includes unit tests for the provider-fallback logic, but it does not yet include a labelled evaluation of the match scores against human judgement; therefore, no numerical accuracy result is claimed in this paper. The system is intended as a decision-support prototype that helps candidates improve their résumés and practise interviews, not as a replacement for human hiring decisions.

**Keywords:** Résumé screening, job description matching, large language models, mock interview, applicant tracking system, Flask, React, PostgreSQL, AI-assisted career guidance.

## I. INTRODUCTION

A résumé is usually the first document a recruiter sees about a candidate. Organizations that receive many applications often use Applicant Tracking Systems (ATS) or manual shortlisting that look for specific skills and keywords from the job description. A candidate whose résumé does not use the expected terms may be filtered out even when the underlying skills are present. Students and freshers are especially affected because they rarely have a mentor who can review their résumé against a particular job, and they have few opportunities to practise realistic interviews.

Recent large language models (LLMs) can read long text, compare two documents and produce structured explanations. This makes it practical to build a tool that gives a candidate fast, readable feedback on how well a résumé fits a job, and that conducts a practice interview in natural language. However, an LLM reply is free text; to be useful inside an application it must be constrained, validated and stored in a consistent form. A further challenge is reliability: external AI services can be slow, rate limited or temporarily unavailable.

ResuMatch addresses these needs in one web application. It combines résumé–job matching, an AI mock interviewer, an organization mode for batch screening, subscription-based usage control and an administrator dashboard.

The main contributions of the project are: (1) a résumé analysis workflow that extracts text from common file formats and obtains a structured match report from an LLM; (2) a timed mock-interview module with a scored feedback report; (3) a batch screening mode for organizations; and (4) a multi-provider AI layer with automatic fallback, together with role-based access, plan-based limits and persistent records.

The remainder of this paper is organised as follows. Section II discusses related concepts. Section III describes the data and methodology. Section IV explains the system architecture and implementation. Section V describes the database design, Section VI the technology stack and Section VII the workflow. Section VIII presents the evaluation status and limitations, and Section IX concludes the paper and identifies future work.

 This work is licensed under a Creative Commons Attribution 4.0 International License 

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511176

## II. RELATED WORK

Automatic résumé screening has traditionally relied on keyword matching and term-weighting methods from information retrieval, such as TF-IDF [1], [2], which score how important a word is to a document within a collection. These methods are simple and fast but treat words as independent tokens and cannot recognise that two differently worded phrases describe the same skill.

Neural language models address this limitation. Transformer-based models [3] learn contextual representations of text, and encoders such as BERT [4] and Sentence-BERT [5] allow two documents to be compared by semantic similarity. Large generative models [6], [7] go further: given a résumé and a job description, they can reason about both and write an explanation, not just a similarity number. The use of an LLM to judge or rate text has been studied, and strong models have been reported to agree with human preferences in many tasks, although the approach has known biases and inconsistencies [8].

Algorithmic hiring tools also raise fairness concerns. Raghavan et al. [9] review claims made by vendors of automated hiring systems and discuss the difficulty of validating them and of detecting bias. For this reason, a résumé-matching system should be positioned as a support tool whose output is reviewed by a person.

ResuMatch uses a hosted LLM for the matching and interview tasks and concentrates on the engineering needed to make such a tool dependable: structured output, validation, usage limits, persistence, security and provider fallback.

## III. PROPOSED METHODOLOGY

### *A. Input Data Description*

ResuMatch does not train a model on a fixed dataset; instead it processes user-supplied documents at run time. The inputs are summarised in Table I. The exact number and type of documents used during testing should be added from the final test set.

TABLE I INPUTS USED BY THE SYSTEM

| Input | Format | Purpose |
|---|---|---|
| Résumé | PDF, DOCX, TXT file or pasted text | Candidate profile to be analysed |
| Job description (JD) | Pasted text | Requirements against which the résumé is compared |
| Job title | Short text | Context for the analysis and for history |
| Interview role | Admin-defined role or custom role text | Topic of the mock interview |
| Account details | Name, email, account type | Access control and plan limits |

### *B. Text Extraction and Validation*

When a file is uploaded, the backend extracts plain text using pypdf for PDF files and python-docx for Word files; TXT files are read directly. Pasted text is accepted as it is. Before analysis, the backend checks that the résumé and the job description contain a minimum amount of text (at least 20 characters) and that the user has not exceeded the limit of the current subscription plan. Scanned, image-only PDFs cannot be read because the system does not perform optical character recognition.

### *C. Prompt-Based Match Scoring*

The résumé text and job description are sent to the LLM together with instructions to act as a recruiter, to base each statement only on the supplied text, and to reply only in a fixed JSON schema. The schema contains the score, verdict, summary, strengths, weaknesses, matched keywords, missing keywords, improvement suggestions and a skill-wise breakdown (Table II). Requesting a fixed schema makes the reply directly usable by the user interface.

TABLE II FIELDS RETURNED BY THE ANALYSIS

| Output field | Description |
|---|---|
| score | Overall match, integer 0–100 |
| verdict | Short label for the match level |

 This work is licensed under a Creative Commons Attribution 4.0 International License 



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.1511176

| Output field | Description |
|---|---|
| summary | Plain-language overview |
| strengths / weaknesses | Lists of points found in the text |
| matched / missing keywords | Keywords present or absent in the résumé |
| improvements | Practical suggestions |
| skill breakdown | Match level for individual skills |

*D. Output Normalisation*

LLM replies may contain extra text or code fences around the JSON. The backend therefore extracts the JSON object, limits the score to the range 0–100, trims over-long lists and fills missing fields with safe defaults before saving the record. This step prevents a malformed reply from breaking the interface.

*E. Mock Interview Methodology*

For a mock interview, the user selects an administrator-defined role or types any role. If no job description exists, the system asks the model to infer typical duties for that role. A session is created with a time limit. A separate prompt makes the AI take the part of an interviewer named Ava who asks one question at a time, follows up on weak answers and does not reveal that it is an AI model. The backend limits the interview to 14 questions and informs the model when time is almost over so that it asks a closing question. When the session ends, a second prompt generates a feedback report containing an overall score, verdict, communication and technical scores, strengths, weaknesses, suggestions and a note for each question. The transcript and feedback are stored.

*F. Provider Fallback*

All AI calls pass through one common function that supports Gemini (default), Groq and Claude. If the selected provider returns a temporary error such as overload or rate limiting, the function retries once and then tries the next configured provider. The administrator can change the provider, model and key from the dashboard without restarting the server.

*G. Evaluation Protocol*

A proper evaluation of the matching feature should use a set of résumé–job pairs scored independently by human reviewers, and report the agreement between the system and the reviewers (for example, rank correlation of scores), the consistency of repeated runs on the same input, and the usefulness of the suggestions as rated by users. Response time and failure rate with and without provider fallback should also be measured. Such an evaluation has not yet been carried out for this project; it is listed as a required next step.

## IV. SYSTEM ARCHITECTURE AND IMPLEMENTATION

*A. Overall Architecture*

ResuMatch follows a three-tier architecture, shown in Fig. 1. The presentation tier is a React single-page application that runs in the browser. It communicates with the Flask application tier through REST [10] endpoints that exchange JSON over HTTPS; protected requests carry a JSON Web Token (JWT) [11]. The application tier contains the authentication, analysis, interview and administration modules and is the only component that talks to the database and to the AI providers, so that secret API keys never reach the browser. The data tier consists of a PostgreSQL database (SQLite is used automatically when no PostgreSQL address is configured) and the external AI services. Table III lists the responsibilities of each layer.

 This work is licensed under a Creative Commons Attribution 4.0 International License 

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511176

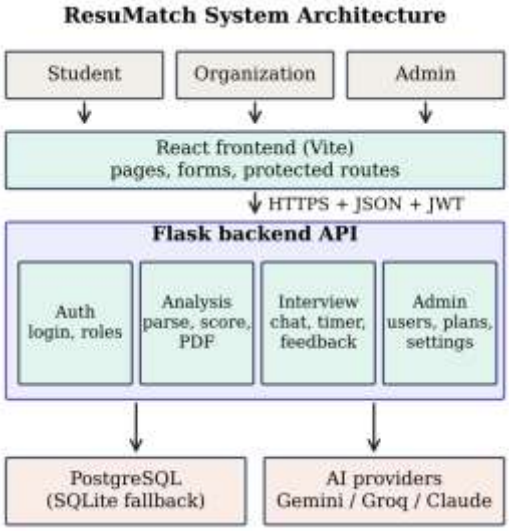



Fig. 1 Overall architecture of the ResuMatch application.

TABLE III LAYERS AND RESPONSIBILITIES

| Layer | Technology | Responsibility |
|---|---|---|
| Presentation | React, Vite, React Router | Pages, forms, route guards, login state |
| Application | Flask, Flask-CORS | API, validation, limits, business logic |
| File processing | pypdf, python-docx | Extract résumé text |
| AI layer | Gemini, Groq, Claude | Analysis and interview replies |
| Reporting | ReportLab | PDF reports |
| Security | JWT, bcrypt, Google Sign-In | Authentication and authorisation |
| Persistence | PostgreSQL / SQLite | Users, results, sessions, settings |

*B. Application Modules*

- User module: registration and login by email–password or Google Sign-In, profile and password change, résumé analyzer, result page, history with deletion and PDF download.
- Mock interview module: selection of a ready-made or custom role, a chat interface with a timer, and a final feedback report with PDF download.
- Organization module: a separate account type that analyses up to 25 résumés against one job description and labels each candidate so that results can be compared.
- Subscription module: plans that limit the number of analyses and interviews. Student plans range from a free plan (one analysis and one interview) to an unlimited yearly plan; organization plans are counted per month. Expired paid plans fall back to the free limits.
- Admin module: statistics, management of users, analyses, interview roles and plans, and site settings such as name, logo, sign-up permission, maintenance message and AI provider, model and key.

*C. Security Measures*

Passwords are stored as bcrypt [12] hashes. Login tokens expire after one day. Protected pages and API routes require a valid token, and administrator routes additionally require the admin role. Each user can access only their own analyses and interviews. Cross-origin requests are accepted only from the configured website address, Google Sign-In tokens are verified on the server, and API keys are kept in environment variables or in the database and are displayed masked in the dashboard.

## V. DATABASE DESIGN

The application stores its data in PostgreSQL; tables are created automatically at the first start, and a default administrator account is seeded. The main tables are shown in Table IV. A user has many analyses and many interview sessions (one-to-many relationships).

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511176

TABLE IV MAIN DATABASE TABLES

| Table | Main fields |
|---|---|
| users | id, name, email (unique), password hash, role, account type, auth provider, plan id, plan start and expiry |
| analyses | id, user id, job title, score, verdict, summary, strengths, weaknesses, keywords, improvements, skill breakdown, candidate label |
| plans | id, name, price label, duration, résumé limit, interview limit, features, audience |
| interview_configs | id, title, job description, duration, active flag |
| interview_sessions | id, user id, title, status, transcript, question count, feedback, start, end and completion times |
| settings | site name, tagline, sign-up flag, maintenance message, AI provider, model and key, logo, favicon |

## VI. TECHNOLOGY STACK

The technologies used to build ResuMatch and the role of each are listed in Table V.

TABLE V TECHNOLOGY STACK

| Technology | Role in ResuMatch |
|---|---|
| Python, Flask | Backend language and web framework [13] |
| React, Vite | Frontend interface and build tool [14] |
| PostgreSQL (Neon), SQLite | Persistent storage with local fallback [15] |
| Gemini, Groq, Claude | LLM providers for analysis and interviews |
| pypdf, python-docx | Text extraction from PDF and Word files |
| ReportLab | PDF report generation [16] |
| JWT, bcrypt, Google Sign-In | Authentication and authorisation |
| Gunicorn, Vercel | Deployment of backend and frontend |

## VII. ANALYSIS WORKFLOW

The résumé analysis workflow is shown in Fig. 2 and proceeds as follows.

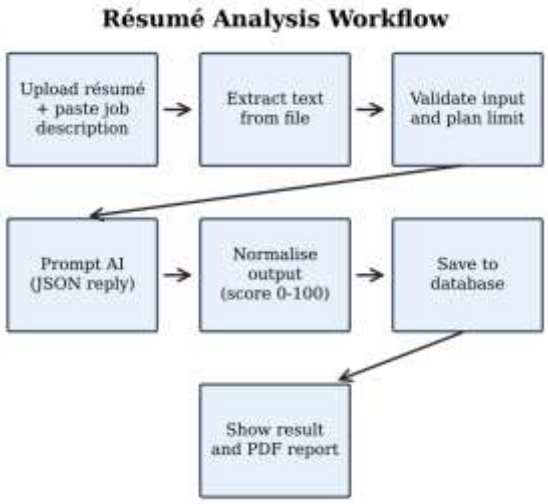



Fig. 2 Résumé analysis workflow.

1. The user logs in and opens the analyzer page.
2. The user uploads a résumé or pastes its text; the backend extracts the text.
3. The user pastes the job description and job title and submits the form.
4. The backend validates the input and checks the plan limit.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

DOI: 10.17148/IJARCCE.2026.1511176

5. The résumé and job description are sent to the AI provider with the JSON-schema instructions; on failure the fallback logic is used.
6. The reply is cleaned and bounded, and the result is saved in the database.
7. The result page shows the score, keywords, skill breakdown and suggestions, and the user may download the PDF report.

## VIII. RESULTS, DISCUSSION AND LIMITATIONS

*A. Implemented Functionality*

The project provides an end-to-end prototype that connects file upload, text extraction, LLM-based analysis, structured storage, PDF reporting, a mock interviewer, organization batch screening, subscription limits and an administrator dashboard. These functions demonstrate how an LLM can be integrated into a practical career-guidance and screening workflow.

*B. Experimental Results Status*

Automated unit tests exist for the AI provider-fallback logic: they check that a temporary error such as a service-unavailable response causes a retry and that persistent failure causes a switch to another provider. Manual functional testing covered registration, login, file upload, plan-limit enforcement, admin access control, interview completion and PDF download. However, the project does not include a labelled evaluation of the quality of the match score or of the interview feedback. Therefore, no numerical accuracy figure is reported (Table VI).

TABLE VI EVALUATION STATUS

| **Evaluation item** | **Status** |
|---|---|
| Unit tests: provider fallback | Implemented |
| Manual functional tests | Performed on main features |
| Agreement with human reviewers | Not yet evaluated |
| Score consistency on repeated runs | Not yet evaluated |
| Response time and failure rate | Not yet measured |
| User satisfaction study | Not yet conducted |

*C. Limitations*

- The match score is produced by a language model and may vary between runs or differ from human judgement; its accuracy has not been measured.
- Scanned or image-only résumés cannot be read because there is no OCR step.
- Résumé text is sent to a third-party AI provider; a real deployment needs user consent, a data-protection review and a clear retention policy.
- The model may reflect biases present in its training data; scores could be unequal across groups of candidates.
- Payment is not integrated, so paid plans are activated without an online payment gateway.
- The mock interview is text-based and does not assess voice, tone or body language.
- A default administrator account is created at first start-up; in production it must be removed or forced to change its credentials.
- Free-tier AI services have usage limits that can restrict throughput.

*D. Responsible Use*

ResuMatch should be used as a decision-support tool. Candidates should treat the score and suggestions as guidance for improving a document, and should not change their résumé to include skills they do not have. Recruiters should not use the score as the only basis for rejecting a candidate; a person should review the résumé and make the final decision. The system should be tested for unequal performance across candidate groups before it is used in consequential hiring settings [9].

## IX. CONCLUSION AND FUTURE WORK

This paper presented ResuMatch, an AI-based résumé–job matching and mock interview system. It extracts text from résumé files, obtains a structured match report from an LLM, stores and presents the result with a downloadable PDF, and adds a timed mock interviewer, organization batch screening, subscription limits, an administrator dashboard and automatic AI-provider fallback. The system is implemented with React, Flask and PostgreSQL and secured with JWT and bcrypt.



International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026
DOI: 10.17148/IJARCCE.2026.1511176

The main gap is quantitative evaluation. Future work therefore includes measuring agreement between the scores and human reviewers, testing consistency and latency, and studying user benefit. Further improvements include an online payment gateway, OCR for scanned résumés, voice-based interviews, automatic résumé rewriting and templates, job recommendations, multilingual support, bias testing, and a mobile application.

## REFERENCES

[1]. K. Spärck Jones, "A statistical interpretation of term specificity and its application in retrieval," *J. Documentation*, vol. 28, no. 1, pp. 11–21, 1972.
[2]. G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manage.*, vol. 24, no. 5, pp. 513–523, 1988.
[3]. A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.
[4]. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT*, 2019, pp. 4171–4186.
[5]. N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. EMNLP-IJCNLP*, 2019, pp. 3982–3992.
[6]. T. Brown et al., "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–1901.
[7]. Gemini Team, Google, "Gemini: A family of highly capable multimodal models," arXiv:2312.11805, 2023.
[8]. L. Zheng et al., "Judging LLM-as-a-judge with MT-Bench and Chatbot Arena," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.
[9]. M. Raghavan, S. Barocas, J. Kleinberg, and K. Levy, "Mitigating bias in algorithmic hiring: Evaluating claims and practices," in *Proc. ACM FAT**, 2020, pp. 469–481.
[10]. R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, 2000.
[11]. M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, IETF, May 2015.
[12]. N. Provos and D. Mazières, "A future-adaptable password scheme," in *Proc. USENIX Annual Technical Conf. (FREENIX Track)*, 1999, pp. 81–91.
[13]. Pallets Projects, "Flask Documentation." Available: https://flask.palletsprojects.com. Accessed: Nov. 21, 2026.
[14]. Meta Open Source, "React Documentation." Available: https://react.dev. Accessed: Nov. 21, 2026.
[15]. PostgreSQL Global Development Group, "PostgreSQL Documentation." Available: https://www.postgresql.org/docs. Accessed: Nov. 21, 2026.
[16]. ReportLab, "ReportLab User Guide." Available: https://docs.reportlab.com. Accessed: Nov. 21, 2026.

 This work is licensed under a Creative Commons Attribution 4.0 International License 