

International Journal Research Publication Analysis

Page: 01-13

SMART CAREER NAVIGATOR WITH ATS RESUME ANALYSER

***Kinthan Thomas, Kruthik, Ravi Surya Reddy V.V.**

Department of CSE, Jyothy Institute of Technology, Bangalore, India.

Article Received: 17 September 2026

*Corresponding Author: Kinthan Thomas

Article Revised: 07 October 2026

Department of CSE, Jyothy Institute of Technology, Bangalore, India.

Published on: 27 October 2026

DOI: <https://doi-doi.org/101555/ijrpa.9534>

ABSTRACT

A companion survey [1] found that career guidance, resume building, Applicant Tracking System (ATS)-style evaluation, and examination support are offered as separate services, and proposed an integrated framework. This paper reports how that framework is implemented as a working web platform for engineering students. We describe the architecture (a FastAPI service layer, SQLite persistence, and a framework-free JavaScript client) and the algorithm behind each module: a three-component ATS compatibility score whose semantic term is computed with TF-IDF cosine similarity, a hybrid career recommender with ranked skill-gap analysis, a sandboxed code judge for programming assessments, an OCR-based document integrity analysis, and a chatbot built on the Anthropic API with a deterministic fallback. We also document the design decisions that make the system explainable, privacy-conscious, and robust to partial failure, together with three explicit scope boundaries: the ATS score is a research-defined measure and not any vendor's score; the integrity analysis flags inconsistencies and does not authenticate documents with their issuers; and the platform does not submit job applications. The paper specifies a verification strategy and an evaluation protocol but reports no experimental results.

KEYWORDS: ATS resume analysis, TF-IDF, career recommendation, skill-gap analysis, sandboxed code execution, document integrity, OCR, explainable scoring, system implementation.

I. INTRODUCTION

Engineering students preparing for placements move between several tools: a counsellor or website for career direction, a template for the resume, a keyword checker for ATS compatibility, a coding platform for practice, and manual checks of their own certificates. A

targeted review of fourteen recent studies [1] found that none of the reviewed systems covers career guidance, resume building, and ATS-style scoring together, and it proposed an integrated framework with formally defined components. A framework is only useful if it can be built, run on modest infrastructure, and explained to the person using it. This paper therefore describes the implementation.

The platform is built to be small enough to deploy on a single machine without a GPU, because the target users are students and the target operators are colleges and student teams. That constraint shapes most of the choices below: a lexical similarity measure instead of a neural encoder as the default, an embedded database, no front-end build toolchain, and an external language-model service that is optional rather than required.

Contributions. (1) A concrete layered architecture in which every function is an independent service behind a thin API layer (Section III). (2) An implementable formulation of the framework's K/S/C ATS compatibility score, including the engineering fixes needed to make TF-IDF meaningful for a single resume-job pair (Section V-B). (3) Safety-oriented designs for the code judge and the document integrity analysis, with explicit statements of what each can and cannot conclude (Sections V-D and V-E). (4) A chatbot gateway with defined failure behaviour (Section V-F). (5) A verification strategy and an evaluation protocol that keep unvalidated claims out of the paper (Section VIII).

II. REQUIREMENTS AND SCOPE

A. Functional requirements

1. *FR1 Profile.* Capture education, skills, interests, and target roles, and use them consistently across modules.
2. *FR2 Resume analysis.* Accept a resume and a target job description (JD); return a compatibility score with the reasons for it and concrete fixes.
3. *FR3 Guidance.* Rank career paths for the profile and list the missing skills for a target role in learning order.
4. *FR4 Assessment.* Deliver objective and coding assessments and feed the outcome back into the skill profile.
5. *FR5 Document checks.* Analyse uploaded academic or certificate documents and report internal inconsistencies for human review.
6. *FR6 Assistance.* Answer questions about the user's own results in natural language, including when the language-model service is unavailable.

B. Non-functional requirements

Explainability: every score or flag is accompanied by its components and rules. *Low footprint*: single-host deployment, no GPU. *Privacy*: minimum data, explicit consent for sensitive fields, and bounded retention. *Graceful degradation*: loss of an optional dependency must reduce capability, not availability. *Reproducibility*: each stored analysis records the parameter and ontology versions that produced it.

C. Scope boundaries

Three boundaries are part of the design, and the user interface repeats them where relevant. First, the ATS score is a compatibility measure defined in [1] and in Section V-B; it is not the score of any employer's ATS and it does not predict shortlisting. Second, the document integrity module reports consistency signals such as mismatched fields, implausible dates, and text-layer disagreement; it does not confirm that a document was issued by the named institution. Third, the platform produces guidance and artefacts for the user; it does not apply to jobs on the user's behalf.

D. Relation to the proposed framework

Table I records where the implementation follows the framework in [1] and where it deliberately differs.

TABLE I: FRAMEWORK COMPONENT AND IMPLEMENTATION CHOICE.

Component	Framework [1]	Implementation
Semantic term S	Pre-trained Sentence-BERT cosine	TF-IDF cosine over a reference corpus; encoder behind a replaceable interface
Keyword term K	Ontology-normalised coverage m/n	As specified
Structure term C	p/q parsing rules passed	As specified; versioned rule set
Recommender	Hybrid CF + content, α tuned	As specified; CF gated on data volume
Weights	(0.4, 0.4, 0.2), then calibrated	Same defaults; configurable
Eligibility checker	Three-state rule engine	Specified in Section V-G; design-level in this version
Added modules	—	Code judge, OCR integrity analysis, chatbot gateway

III. SYSTEM ARCHITECTURE

Fig. 1 shows the architecture. A browser client written in plain JavaScript, HTML, and CSS calls a FastAPI application over JSON. The application layer is deliberately thin: it authenticates the session, validates request bodies against typed schemas, and delegates to a

service module. Services hold all domain logic and communicate through function calls and the shared stores, not through each other's internals, so a module can be replaced or tested alone.

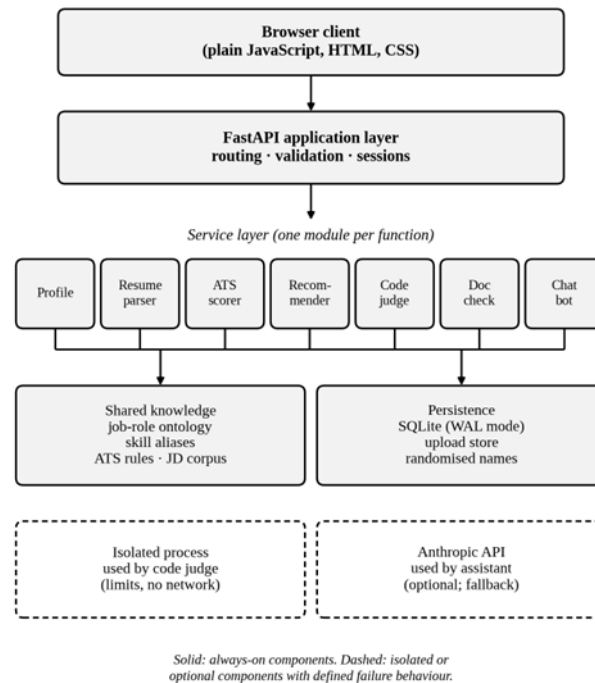


Fig. 1. Layered architecture with isolated and optional components.

A. Technology choices and their rationale

- **FastAPI** provides typed request validation and machine-readable API documentation, which reduces input-handling code and gives a testable contract between client and server.
- **SQLite** removes database administration, stores everything in one file that can be backed up by copying, and is adequate for a pilot at college scale. Write-ahead logging is enabled so readers do not block the writer. All data access goes through one repository module, so a move to a client-server database changes one layer.
- **Framework-free JavaScript** avoids a build step and keeps the front end small enough to audit. State is held in the server session and re-fetched, so the client has little logic that can diverge from the server.

B. Request lifecycle

A typical request, such as scoring a resume, proceeds as follows: the client uploads the file and JD; the application layer checks size, type, and schema; the parser service extracts text and layout flags; the scorer returns component scores and an explanation; the repository

stores the analysis with its parameter versions; and the response carries the explanation payload that the client renders. Long-running work, namely code execution and OCR, runs outside the request thread so that other users are not blocked.

IV. DATA DESIGN

Table II lists the logical entities. Two decisions matter beyond the schema. First, every analysis row stores the weights, rule-set version, and ontology version used, so that a score can be reproduced or compared after the parameters change. Second, uploaded files are stored under randomised names outside the web root, and the database holds the reference and a content hash, never the file bytes.

TABLE II: LOGICAL ENTITIES.

Entity	Contents and purpose
User, Profile	Credentials (salted hash), education, skills with proficiency, interests, target roles; sensitive fields optional and consent-flagged
Resume, Analysis	File reference and hash; per-analysis K, S, C, total, matched and missing skills, failed rules, parameter versions, timestamp
Assessment, Submission	Question sets and coding problems with test cases; per-attempt verdicts, scores, resource usage
Document, Finding	Uploaded document reference; extracted fields; list of flags with evidence and severity
Recommendation	Ranked paths with component scores and contributing skills; skill-gap roadmap
Chat session	Turns, mode used (assistant or offline guidance), and the structured context supplied

V. MODULE IMPLEMENTATION

A. Resume ingestion and normalisation

PDF resumes are read through their text layer and DOCX resumes through their paragraph and table structure. The parser also records layout flags used later by the structure term: whether tables are present, whether text flows in more than one column, whether text-bearing images are present, and whether a text layer exists at all. A resume without a text layer is reported as unparsable instead of being scored, because a score on empty text would mislead.

Text is Unicode-normalised, lower-cased, and tokenised with a tokeniser that preserves technical tokens such as C++, C#, .NET, and Node.js, which a default word tokeniser would damage. Sections are located by matching heading lines against a lexicon with fuzzy tolerance. Contact fields are extracted by pattern. Skills are recognised by longest-match lookup in a trie built from the ontology's canonical names and aliases (for example “js” and “javascript” map to one skill identifier). Normalisation is linear in text length.

B. ATS compatibility score

The score implements the definition in [1] for a resume against a target JD:

$$ATS = 100 \times (w_1 K + w_2 S + w_3 C), w_i \geq 0, \sum w_i = 1 \quad (1)$$

K (keyword coverage) is m/n , where n is the number of required skills identified in the JD and m is the number of those found in the normalised resume. C (structural compliance) is p/q over a versioned set of q parsing rules: standard section headings, single-column text flow, no text carried in images, parsable email and phone, no layout tables, and a present text layer. S (semantic similarity) is

$$S = \max(0, \cos(\phi(r), \phi(j))) \quad (2)$$

Where ϕ maps a normalised document to its TF-IDF vector. Three implementation details make this behave sensibly.

- *Reference corpus for IDF.* With only two documents, inverse document frequency is nearly constant and TF-IDF collapses toward raw term overlap. The vectoriser is therefore fitted on a reference corpus of role descriptions from the ontology plus the current JD, so that common words are down-weighted and distinctive technical terms dominate.
- *Term configuration.* Unigrams and bigrams, sub-linear term frequency, L2 normalisation, a stop-word list, and a minimum document frequency to suppress noise. Ontology-mapped skill identifiers are added as tokens so that aliases contribute to the same dimension.
- *Degenerate cases.* If no required skill can be detected in the JD, K is undefined; it is excluded and the remaining weights are renormalised, and the interface says so. If the resume fails parsing, no score is produced.

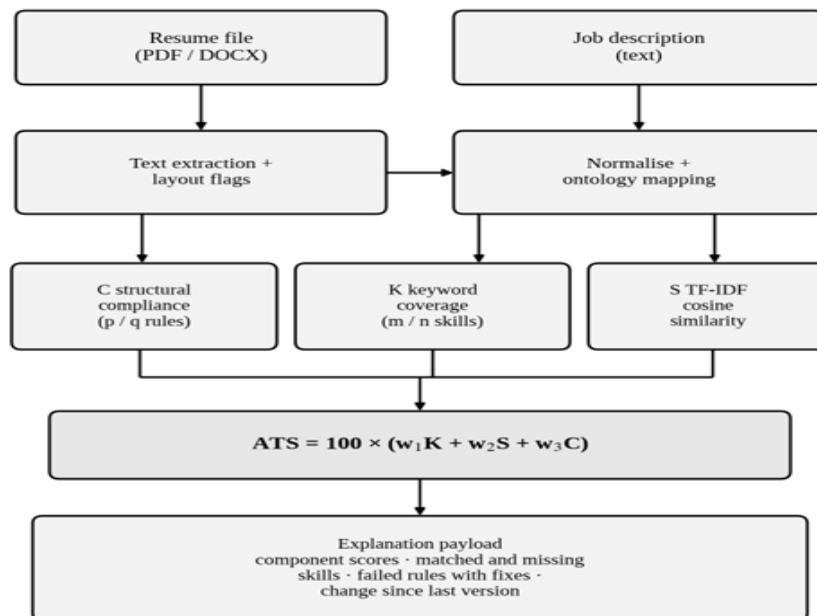


Fig. 2. ATS scoring pipeline.

Pseudocode for the scorer follows. The similarity function is accessed through an interface, so the TF-IDF implementation can be replaced by a Sentence-BERT encoder [3], as proposed in [1], without changing the rest of the pipeline.

```

ATS(resume_file, jd, W, ontology, corpus):
  R, flags <- extract_text(resume_file)
  Rn, Jn <- normalise(R), normalise(jd)
  need <- required_skills(Jn, ontology)
  have <- skills(Rn, ontology)
  K <- |need & have| / |need| (if |need|>0)
  S <- max(0, sim(Rn, Jn, corpus))
  C <- passed(RULES, R, flags) / |RULES|
  total <- 100*(w1*K + w2*S + w3*C)
  return total, explain(K,S,C,need,have)

```

Explanation payload. Each result lists the three component scores and their weighted contributions, the matched and missing skills ordered by their weight in the JD, each failed structural rule with a concrete fix, and the change since the user's previous analysis of the same resume. The last item makes revision iterative: the user edits, re-scores, and sees which change moved which component, which is the candidate-facing feedback loop identified as a gap in [1].

Properties relied on in testing. S , K , and C each lie in $[0,1]$, so the total lies in $[0,100]$; the total is non-decreasing in each component for fixed weights; and adding to the resume a required skill that was missing cannot lower K .

Known limitation. TF-IDF is lexical. It cannot match a resume phrase and a JD phrase that share meaning but no terms, which is the main reason [1] proposes a pre-trained encoder. The ontology aliases recover part of this gap for skills but not for free-text descriptions.

C. Career recommendation and skill gap

For user u and career path c the hybrid score is

$$R(u,c) = \alpha \cdot CF(u,c) + (1-\alpha) \cdot CB(u,c) \quad (3)$$

Where CB is the cosine between the user's weighted skill vector and the path's skill vector, with skills weighted by inverse frequency across paths so that ubiquitous skills do not dominate. CF is a neighbourhood score from profiles similar to u . A new platform has little interaction data, so CF is gated: while the number of profiles with outcome labels is below a configured threshold, α is zero and ranking is content-based, which is the cold-start behaviour specified in [1]. A rule-based sector filter removes paths incompatible with the profile (for example, qualification mismatches) before ranking.

The skill gap for a target role is $G = S_{\text{role}} \setminus S_{\text{user}}$, computed over canonical skill identifiers. Gap skills are ordered by their frequency across the role's postings and then adjusted by

prerequisite edges in the ontology, so that foundations precede dependent skills in the learning roadmap. Each recommendation lists the profile skills that contributed most to its score.

D. Assessments and the sandboxed code judge

Objective assessments are scored deterministically, and results update the proficiency of the skills they test. Coding problems are evaluated by the judge, whose central design problem is running untrusted code safely on the same host as the application.

Each submission is written to an ephemeral directory and, where the language requires, compiled first. Each test case then runs in a separate child process started without inherited secrets, as an unprivileged user, with limits on CPU time, wall-clock time, address space, process count, and output size, and with network access disabled. Standard input is the test input; standard output is captured, truncated at a cap, and compared with the expected output after whitespace normalisation. Verdicts are accepted, wrong answer, time limit exceeded, memory limit exceeded, runtime error, and compile error. The score is the weighted fraction of passed tests, and the per-test resource usage is stored.

Security position. Process-level limits contain accidental and casual abuse such as infinite loops, memory exhaustion, and fork bursts, but they are weaker than container or virtual-machine isolation. For a deployment that accepts code from the public internet, the judge process should additionally run inside a container with a restricted system-call profile; the interface between the judge and the application is kept narrow so that this wrapper can be added without changing callers.

E. OCR-based document integrity analysis

For an uploaded document the pipeline renders pages to images, runs OCR, extracts labelled fields (name, identifier, institution, dates, marks or grades) by pattern and layout rules, and applies a set of independent checks. Examples are: the extracted name against the profile name; internal date ordering; arithmetic consistency between component marks and totals; agreement between a PDF's embedded text layer and the OCR of the rendered page, since a mismatch can indicate overlaid or edited text; and plausibility of identifier formats. Each check yields a finding with its evidence and a severity.

What the result means. The output is one of consistent, needs review, or inconsistent, each with the findings behind it. It is never reported as verified or authentic, because that would require confirmation from the issuing body, which the platform does not have. Low-quality scans, handwriting, and non-Latin scripts reduce OCR accuracy, so extraction confidence is

reported and low-confidence fields are excluded from checks instead of being guessed. Findings are intended to direct a human reviewer, not to decide a case.

F. Conversational assistant with fallback

The assistant gateway calls the Anthropic API from the server only, so the API key never reaches the browser. Each request supplies the user's question together with a compact structured context, such as the latest ATS explanation payload, rather than raw documents, which limits both cost and disclosure. The system instructions direct the model to explain results using the supplied components, to avoid promising outcomes, and not to claim capabilities the platform lacks, in particular job application and certificate authentication.

The gateway enforces timeouts, a bounded retry, and per-user rate limits. When the call fails or the service is not configured, a deterministic responder takes over: it matches the question to an intent by keyword and TF-IDF similarity against a curated set of answers and fills templates from the same structured context, for example by restating which ATS component is lowest and what to change. Every reply is labelled with the mode that produced it, so the user can tell a model-generated answer from rule-based guidance.

G. Eligibility rule engine (design)

The framework in [1] adds government-examination eligibility. Its implementation is specified here and is not part of the validated modules. Each notification is normalised into a record of required qualification, age bounds at a cut-off date, category-based relaxation, and other conditions. Evaluation is a pure function of the profile and the record:

```
eligibility(profile, rule):  
  if not meets(profile.qual, rule.qual): return NOT  
  age <- age.on(profile.dob, rule.cutoff)  
  hi <- rule.max_age + relax(rule, profile.cat)  
  if not (rule.min_age <= age <= hi): return NOT  
  unmet <- failed(rule.other, profile)  
  if unmet == {}: return ELIGIBLE  
  if all(attainable(u) for u in unmet):  
    return CONDITIONAL  
  return NOT
```

Each state is returned with the rule that decided it. Because notifications are amended often, every record carries its source and retrieval date, and records extracted automatically are held for manual verification before use. Category fields used here are sensitive and follow the consent rules of Section VI.

VI. EXPLAINABILITY, PRIVACY, AND HONEST SCOPE

Explainability. The scoring and rule-based components were chosen so that explanations are exact rather than approximated: the ATS result shows K, S, and C and the facts behind them, the recommender shows contributing skills, the integrity analysis shows the evidence for each

finding, and the eligibility engine shows the deciding rule. This follows the argument in [5], [6] that post-hoc attributions have limited fidelity and that human-centred transparency matters in hiring-related decisions.

Fairness. Embedding-based screening can prefer particular names and groups [4]. The default semantic term is lexical, which reduces dependence on pre-trained representations, but this is not evidence of fairness. The evaluation protocol therefore includes a counterfactual name-substitution audit for whichever similarity function is deployed, and any replacement encoder must pass it before use.

Privacy. Sensitive attributes such as category and disability status are optional, requested only where a feature needs them, and stored with a consent flag. Retention is bounded and users can delete their data. The design aims to respect the Digital Personal Data Protection Act, 2023 [8]; legal compliance review is outside the scope of this paper. Only the minimum structured context is sent to the external language-model service.

Honest scope. The scope statements of Section II-C are surfaced in the interface at the point of use: next to the ATS score, next to integrity findings, and in the assistant's instructions. We treat accurate statements of capability as a functional requirement, because overstated authentication or prediction would harm the users the system is meant to help.

VII. ROBUSTNESS AND SECURITY CONTROLS

- *Uploads:* size caps, content-type verification by file signature rather than extension, sanitised and randomised names, and no execution or serving of uploaded content.
 - *Data access:* parameterised queries only; schema validation of every request body; passwords stored as salted hashes; session tokens with expiry.
 - *Isolation:* untrusted code runs only in the judge's child processes (Section V-D); OCR and parsing run with time limits so a malformed file cannot stall a worker.
 - *Secrets:* the language-model key and other secrets are read from the server environment, never stored in the repository or sent to the client.
- Degradation:* each optional dependency has a defined failure mode: the assistant falls back to rule-based guidance, an unparsable resume yields an explicit error instead of a score, and low-confidence OCR fields are skipped and reported.

VIII. VERIFICATION STRATEGY AND EVALUATION STATUS

A. Verification of the implementation

The verification plan for deterministic components has four levels. Unit tests cover normalisation (including technical tokens), alias mapping, K, C, the eligibility function, and

the score arithmetic. Property tests check the invariants listed in Section V-B. Integration tests exercise each API route with valid, malformed, and oversized input. Adversarial tests submit programs that loop forever, allocate until exhaustion, spawn processes, write outside the working directory, attempt network access, and print unbounded output, and assert the expected verdict and that the host is unaffected. A further group disables the language-model service and asserts that the fallback answers and labels itself correctly. Regression fixtures with stored expected component scores guard against accidental changes to normalisation or rules.

B. Evaluation protocol

Whether the modules perform well is an empirical question that the verification above does not answer. The protocol specified in [1] applies unchanged: hypotheses H1 to H4 on ranking quality, component contribution, recommender quality, and group-wise score gaps; annotated consented resumes and job descriptions, with the public SkillSpan dataset [7] for skill extraction, two annotators, and Cohen’s κ ; training, validation, and test splits with weights and α tuned on validation only; and means with 95% bootstrap confidence intervals and paired tests. Table III extends the protocol to the modules added in this implementation.

TABLE III: MEASUREMENT PLAN BY MODULE.

Module	Measures	Status
ATS scorer	NDCG@k and Spearman ρ vs. human ranking; K/S/C ablation; weight sensitivity; name-swap score gaps	Protocol defined; not run
Recommender	Precision@k, Recall@k, NDCG@k vs. popularity, CF-only, content-only	Protocol defined; not run
Skill extraction	Entity-level P, R, F1 vs. dictionary and default NER	Protocol defined; not run
Code judge	Verdict agreement with reference runs; adversarial pass rate; per-run overhead	Tests specified; no figures reported
Integrity	Per-check precision and recall on constructed edits of sample documents; OCR field accuracy	Protocol defined; not run
Assistant	Fallback activation correctness; answer faithfulness to supplied context (human rating)	Protocol defined; not run
System	Latency per route; peak memory; concurrent users on single host	Not measured

C. RESULTS

No experimental results are reported. The evaluation has not been completed, and we make no claims about the accuracy, fairness, or user benefit of the implemented system. Figures

reported by related systems, such as the extraction scores in [2], were obtained on small samples with different data and are not comparable to the planned protocol.

IX. LIMITATIONS

The default semantic term is lexical and misses paraphrase. The ontology is curated and bounded, so skills outside it are invisible to K. The recommender's collaborative component has no practical effect until enough labelled profiles exist. Process-level code isolation is not sufficient for open public exposure. OCR-based checks are sensitive to scan quality and script, and a consistent document may still be forged. SQLite limits write concurrency, so a deployment beyond pilot scale needs a client-server database. The eligibility engine is specified but not validated, and it depends on how current its notification records are. Finally, the system has not been validated with users or against recruiter judgments, so its practical value remains to be shown.

X. CONCLUSION AND FUTURE WORK

We described the implementation of an integrated career-support platform that realises the framework proposed in [1]: a layered FastAPI and SQLite system whose ATS scoring, recommendation, assessment, document analysis, and assistant modules are independently replaceable, explain their outputs, degrade gracefully, and state their own limits. The immediate next steps are running the evaluation protocol, replacing the TF-IDF similarity with a pre-trained encoder subject to the fairness audit, containerising the code judge for public deployment, implementing and validating the eligibility engine, and migrating persistence when usage requires it.

REFERENCES

1. Anonymous, "Smart Career Navigator with ATS resume analyser: A targeted review and proposed integrated framework," manuscript under review.
2. A. A. A. Hussin, S. A. Suparman, and H. S. Rushdan, "An intelligent NLP-based system for resume screening and Applicant Tracking Systems (ATS)-optimised resume generation," *Int. J. Perceptive Cogn. Comput.*, vol. 12, no. 2, pp. 129–134, 2026, doi: 10.31436/ijpcc.v12i2.719.
3. N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. EMNLP-IJCNLP*, 2019, pp. 3982–3992.

4. K. Wilson and A. Caliskan, “Gender, race, and intersectional bias in resume screening via language model retrieval,” in *Proc. AAAI/ACM Conf. AI, Ethics, and Society (AIES)*, vol. 7, 2024, pp. 1578–1590, doi: 10.1609/aies.v7i1.31748.
5. A. Beretta *et al.*, “Requirements of eXplainable AI in algorithmic hiring,” in *Proc. AIMMES Workshop*, 2024.
6. S. Fabeyo, “Explainable AI in employment decision-making: A systematic review of transparency methods in hiring algorithms,” *Issues Inf. Syst.*, vol. 26, no. 3, pp. 127–135, 2025, doi: 10.48009/3_iis_2025_2025_110.
7. M. Zhang, K. N. Jensen, S. D. Sonniks, and B. Plank, “SkillSpan: Hard and soft skill extraction from English job postings,” in *Proc. NAACL-HLT*, 2022, pp. 4962–4984, doi: 10.18653/v1/2022.naacl-main.366.
8. Government of India, “The Digital Personal Data Protection Act, 2023,” Act No. 22 of 2023.