

EMPLOYABILITY HUB: Smart System for Skill Analysis and Career Prediction

Mr. Ganesh Prasad , Assistant Professor, Department of Computer Science and Engineering
Shridevi Institute of Engineering and Technology
Tumkur, India

Anusha B.L Student, Department of Computer Science and Engineering
Shridevi Institute of Engineering and Technology
Tumkur, India

Bhavana R Student, Department of Computer Science and Engineering
Shridevi Institute of Engineering and Technology
Tumkur, India

Kavana K.J Student, Department of Computer Science and Engineering
Shridevi Institute of Engineering and Technology
Tumkur, India

Manasa G.R Student, Department of Computer Science and Engineering
Shridevi Institute of Engineering and Technology
Tumkur, India

Abstract - Career guidance that relies on generic, manually administered advice cannot keep pace with a job market whose required skills change quickly. This paper presents Employability Hub, a web pipeline that takes a candidate from resume upload to a personalized career recommendation on a single platform: a Flask backend parses an uploaded PDF or DOCX resume with PyMuPDF and python-docx; the Google Gemini API extracts technical and soft skills and infers career interests; a timed assessment of 20 multiple-choice and 3 coding questions is generated from the candidate's profile, with coding questions executed through the Judge0 API; a skill-gap service compares the assessed profile against a target role; a roadmap service produces a learning plan; and a job-search step queries the Adzuna API for matching openings. We state plainly what is implemented and what is not: resume parsing, skill extraction, interest detection, the assessment engine, skill-gap analysis and roadmap generation are functional; coding-score submission and job-API integration are implemented but still under verification; and full-scale performance testing, UI testing and error-handling remain pending. No accuracy, latency or performance figures are reported because none have been measured; Section VIII fixes an evaluation protocol in advance of data collection so future numbers can be reproduced and checked rather than asserted.

Index Terms - Employability, resume parsing, natural language processing, large language models, skill-gap analysis, career prediction, job recommendation, career roadmap, automated assessment.

I. INTRODUCTION

Career selection has become harder as the number of graduates competing for a given role grows and as the specific skills employers expect shift faster than most curricula are revised. Many students can list their coursework but cannot say, in terms an employer would recognize, which skills they already have, which they are missing for a target role, or what to do next. Counselling that is generic — the same checklist or the same short quiz for every student — does not close that gap, because it does not look at what any individual candidate has actually demonstrated. The result is a widening gap between what curricula certify and what employers are willing to test for at interview, and students are left to close that gap through trial and error rather than through structured feedback on their own profile.

A system that reads a candidate's own resume, tests the skills it finds there, and compares the result against a specific role can, in principle, give more concrete guidance than a generic quiz: which skills are already adequate, which are missing, and what job openings and learning steps follow from that gap. Realizing this in software chains together several components that are each well studied separately — document parsing, large-language-model-based information extraction, automated coding assessment, and job-listing retrieval — but that are less often combined, end to end, on one candidate profile and reported with an explicit account of which stage is finished and which is not. Most existing tools stop at one stage of this journey: a screening tool tells a recruiter whether a resume matches a job description, while a learning platform recommends courses without ever having tested whether the candidate already has the underlying skill.

This paper describes Employability Hub, a pipeline that performs that combination for a single candidate profile: resume parsing, skill and interest extraction, a generated skill assessment, skill-gap analysis relative to a chosen role, job recommendation, and a learning roadmap. We do not claim that any individual technique — resume parsing, LLM-based extraction, automated code execution, or job-API querying — is novel. The contribution is the working combination applied to one candidate journey, together with an explicit statement of which stages are implemented and which are still under verification, so that readers can tell functioning software from a diagram of intent. This distinction matters in a student project setting, where architecture diagrams are often produced before implementation and can silently outlive the code that was actually written.

A. Contributions

- An end-to-end, functional pipeline from resume upload to career recommendation that combines resume parsing (PyMuPDF, python-docx), LLM-based skill/interest extraction (Google Gemini API), a generated and timed skill assessment with coding execution (Judge0 API), skill-gap analysis, job recommendation (Adzuna API), and roadmap generation, with the front end and back-end routes named against the project's own file structure (Section V).
- A status table (Table IV) that states, module by module, whether it is implemented, implemented-but-unverified, or pending, so that every claim in this paper is tied to a stated component rather than to an aggregate impression of completeness.
- An evaluation protocol (Section VII) fixed in advance of any measurement — scenarios, metrics and a measurement procedure — so that when the pending components are verified, the resulting numbers can be produced under a documented method rather than reported without one.
- An explicit list of unresolved discrepancies between the project's own architecture diagram and its technology-stack documentation (Section IV), and of gaps between the project's literature-survey table and its numbered reference list (Section II), so that these are flagged for correction rather than silently carried into a final submission.

The remainder of the paper is organised as follows. Section II reviews related building blocks and prior work. Section III states the problem and the system's design goals. Section IV presents the architecture. Section V describes the implementation module by module. Section VI states the status of the two components still under verification. Section VII fixes the evaluation methodology. Section VIII reports verification status against that methodology. Section IX discusses trade-offs and risks. Section X states limitations. Section XI gives a prioritised roadmap, and Section XII concludes.

II. BACKGROUND AND RELATED WORK

A. Building Blocks

Resume parsing converts a PDF or DOCX file into plain text that downstream components can process; this project uses PyMuPDF for PDF extraction and python-docx for DOCX extraction. Skill and interest extraction is performed by prompting the Google Gemini API on the parsed resume text, rather than by a hand-written keyword list, so that the extraction step can generalize across resume formats and wording at the cost of being less predictable and harder to audit than a rule-based extractor. Automated coding assessment executes candidate-submitted code against test cases through the Judge0 API, a hosted code-execution service, rather than running arbitrary code inside the project's own infrastructure — a choice that avoids the sandboxing burden of running untrusted code locally but introduces a dependency on a third-party service's availability. Job recommendation is performed by querying the Adzuna job-search API for listings matching a selected role, rather than by maintaining a private job database, which keeps listings current without requiring the project to scrape or license job data itself.

Skill-gap analysis and career-roadmap generation, as implemented here, compare an assessed skill profile against role requirements and derive a learning plan from the result; this project does not use a published skill-gap dataset or benchmark, and no comparison against one is reported. This is consistent with the project's own stated position that no accuracy or benchmark figures are available until the evaluation protocol of Section VII has actually been executed.

B. Related Studies

The project's own literature survey (reproduced as Table I) reviews sixteen works spanning several related areas. Several works apply NLP techniques ranging from keyword and TF-IDF methods to transformer-based semantic models such as BERT and S-BERT to extract information from resumes and match it against job descriptions or roles; these report improved matching accuracy but generally require training data and computational resources, and treat screening or matching as a stand-alone task rather than as one stage of a longer candidate journey. A second group combines resume parsing with domain or role prediction using classical ML classifiers (SVM, Random Forest, Naive Bayes) together with named-entity recognition, achieving accurate role prediction without extending to job recommendation or a learning roadmap.

A third group addresses skill verification and assessment specifically: one work proposes a blockchain-based mechanism for tamper-proof skill verification without any AI-based analysis or prediction, and another combines resume analysis with an adaptive, externally-scraped skill assessment, which raises data-privacy questions that do not arise when the assessment is generated from the candidate's own submitted resume, as in the present system. Job-recommendation work generally relies on recommendation-engine or similarity-based techniques and commonly reports a cold-start problem for new users or listings, with no accompanying skill-gap or roadmap function. One

framework for comprehensive resume analysis, scoring and recommendation is conceptually close to an integrated platform but is reported as requiring further large-scale validation before its findings can be generalized.

Note on citation numbering: the project's literature-survey table (S1–S16 in Table I) and its separately numbered reference list ([1]–[16] in the References section) do not correspond position-for-position in the source material; several entries were matched with confidence by title (S4, S8, S12, S13, S15, S16), while others could not be matched confidently and are discussed above without a specific bracket citation. This should be reconciled by the authors before submission.

C. Positioning

Individually, resume parsing, LLM-based skill extraction, automated coding assessment, skill-gap comparison, and job-listing retrieval are each established techniques, and Table I shows that each has been studied, in isolation or in pairs, in prior work. What the surveyed literature does not show, and what Employability Hub is designed to provide, is all of them applied to one candidate profile in a single pipeline: a candidate uploads one resume and receives an assessed skill profile, a stated skill gap, job recommendations and a learning roadmap from one system, rather than needing separate tools for screening, matching, and guidance. The novelty claimed here is therefore integrative rather than algorithmic — the value is in the chained pipeline and its honest status reporting, not in a new model or a new matching technique.

D. Summary of the Gap

Table I's sixteen entries can be read as answering four separate questions — can a resume be parsed and matched to a role (S1, S3, S5, S8, S10–S16), can a candidate's skill be verified rather than merely claimed (S4, S7), can a job be recommended given a profile (S6, S8), and can keyword or context be extracted efficiently (S9) — but no single entry answers all four for the same candidate in the same run. Employability Hub is scoped narrowly to answer all four together for one profile, which is why its contribution is stated in Section I-A as an integration claim rather than an accuracy or novelty claim: the paper does not assert that its Gemini-based extraction step out-performs the BERT- or S-BERT-based methods in S5 and S16, only that it is chained to the assessment, gap-analysis, recommendation and roadmap stages those methods are not.

TABLE I. SUMMARY OF THE LITERATURE SURVEY (S1–S16; DISTINCT FROM REFERENCES [1]–[16])

No.	Title (short)	Yr	Method / Limitation
S1	Automated Resume Evaluation using NLP	2023	NLP, scoring/ranking; weak on complex formats
S2	Software Defect Detection (ML/Static Analysis)*	2024	SVM, RF, LR, NB, MLP; off-domain
S3	Intelligent Resume Analysis & Skill Matching	2024	NLP extraction; limited experience-quality sense

S4	Smart Contract Skill Verification	2023	Blockchain; no AI-based analysis
S5	Job–Resume Matching using NLP	2024	BERT, semantic similarity; needs training data/compute
S6	Job Recommendation (ML + Rec. Engine)	2024	Rec. engine, ML, NLP; cold-start problem
S7	NLP Resume Analysis + Adaptive Assessment	2024	Scraping, Word2Vec; privacy, data quality
S8	Job Recommendation Web Platform	2023	ML, Jaccard similarity; keyword-based, no gap
S9	Keyword Extraction (RAKE)	2023	RAKE, word frequency; limited context
S10	Resume Matching (SVM/NER)	2024	SVM, RF, NER, embeddings; needs tuning, data
S11	Resume Screening using ML	2024	KNN, SVM, TF-IDF; screening only
S12	Resume Parsing & Domain Prediction	2023	NLP, NER, SVM/RF/NB; domain only
S13	ResuMate AI Framework	2025	NLP, scoring engine; needs large-scale validation
S14	Resume Shortlisting using NLP	2024	NLP, NER, ranking; weak semantics
S15	Smart Assistant: Best Resume (NLP/ML)	2024	ML classification; accuracy depends on dataset
S16	AI Resume Screening & Job Matching	2025	S-BERT, cosine sim., ML; no career guidance

* Concerns software-defect prediction, outside the resume/employability domain; relevance to be reconfirmed by authors.

III. PROBLEM STATEMENT AND DESIGN GOALS

A. Problem Statement

Students and job seekers struggle to identify their strengths, skill gaps, and suitable career paths because personalized, real-time career-guidance systems are not widely available; most existing tools address at most one of resume screening, skill assessment, job matching, or learning guidance, leaving the candidate to reconcile the results of several disconnected tools. A student may, for instance, run a resume through one screening tool, take an unrelated online quiz for a certification, and separately browse job boards, with no single system connecting what the quiz showed to what the job listings actually require.

B. Design Goals

- Coverage across the candidate journey. Resume analysis, skill assessment, skill-gap identification, job recommendation, and learning guidance should all be available from one platform rather than requiring separate tools.
- Modularity. Resume parsing, skill extraction, assessment, skill-gap analysis, job recommendation, and roadmap generation are implemented as separate services (Section V) so that any one can be modified or replaced without redesigning the others.
- Repeatable assessment. The generated assessment (20 MCQs, 3 coding questions) is produced from the candidate's own extracted profile, so evaluation does not depend on externally scraped or purchased test content.
- Bounded automation. Automated actions during the assessment — the tab-switch warning and auto-submit —

are time-boxed and limited to the assessment session rather than applied elsewhere in the system.

- Honest reporting. Components that are implemented but unverified, or not yet built, are labelled as such throughout this paper (Sections VI, VII) rather than presented as complete.

C. Non-Functional Considerations

Beyond the functional goals above, three non-functional concerns shape how the system should be read. Latency is compounded across stages: because skill extraction, question generation, gap analysis and roadmap generation each call the same external LLM API, a slow or rate-limited Gemini response delays every downstream stage rather than just one, which is why Section VII treats per-service latency as a metric in its own right rather than measuring only end-to-end time. Data sensitivity follows from the inputs the system collects — an uploaded resume and, during the assessment, camera access — both of which are personal data that the source material does not yet describe a retention or consent policy for (Section VII-D). Availability depends on three external services (Gemini, Judge0, Adzuna) that the project does not control, so the system's own uptime is bounded by the combined uptime of all three, not by the Flask application alone.

IV. SYSTEM ARCHITECTURE

The architecture, as depicted in the project's own diagram, is organised into four layers. A User Layer represents students and job seekers who register, complete a profile, upload a resume, take the skill assessment, view results, and request recommendations.

An Application Layer performs the core processing in five stages: a Data Collection stage that gathers profile, education, skills, interest and resume/test data; a Preprocessing Module that performs data cleaning, normalization and feature extraction; a Skill Analysis (NLP) Module responsible for resume parsing, skill extraction and keyword matching; an ML Prediction Module that performs career prediction; and a Recommendation Engine that suggests career paths, courses and certifications.

A Data Layer maintains a database of user profiles, a skills dataset, a career dataset, a course-and-certification dataset, and an industry/job-market dataset, populated from External Data Sources (education data, job-market/industry data, course-and-certification data, and online resources accessed through APIs and websites). Results are presented through an Output Layer — a dashboard showing predicted career paths, skill-gap analysis, recommended courses, certification suggestions, and job-opportunity insights.

Read end to end, a single request therefore crosses all four layers at least once: the User Layer collects a resume upload, the Application Layer's Skill Analysis and Recommendation stages call the Gemini, Judge0 and Adzuna services described in Section V, the Data Layer holds the resulting profile and dataset lookups, and the Output Layer renders the dashboard the candidate sees. A failure at any Application-Layer stage

is therefore visible to the candidate only through the Output Layer, which is why Table VI recommends surfacing explicit success/failure status rather than letting a stage fail silently.

Table II summarizes how responsibility divides among the components actually named in the project's implementation (Section V), as distinct from the generic layer names in the architecture diagram described above.

TABLE II. DIVISION OF RESPONSIBILITY AMONG NAMED COMPONENTS

Component	Stage	Role
Resume parser (PyMuPDF, python-docx)	Upload	Extracts raw text from uploaded resume
Gemini API integration	Analysis	Skills/interests, MCQ+coding generation, skill gap, roadmap content
Assessment engine (quiz.html, question_generator.py)	Assessment	20-min timed test; camera permission; tab-switch auto-submit
Judge0 API	Assessment	Executes submitted code for coding questions
skill_gap_service.py	Post-assessment	Compares assessed profile to target role
jobs_service.py + Adzuna API	Recommendation	Retrieves job listings for selected role
roadmap_service.py	Recommendation	Generates learning roadmap
Dashboard (result/final_result/skill_gap/roadmap.html)	Output	Displays skills, scores, gaps, jobs, roadmap

It should be noted that the architecture diagram labels its component technologies as Python, Flask, MySQL, HTML, CSS, JavaScript and Scikit-learn, whereas the technologies actually described for the implemented pipeline (Section V, Table III) are the Google Gemini API, PyMuPDF, python-docx, the Judge0 API and the Adzuna API, without MySQL or Scikit-learn. This discrepancy between the architecture diagram and the implementation description is unresolved and should be checked by the authors before submission, since a reviewer comparing Fig. 1 against Section V would otherwise reasonably ask where the database and the classical ML model are implemented.

V. IMPLEMENTATION

A. Resume Ingestion and Parsing

The candidate journey begins at index.html, where a resume is uploaded in PDF or DOCX format. On the back end, resume_service.py extracts the raw text using PyMuPDF for PDF files and python-docx for DOCX files. The extracted text is passed to the Gemini-based analysis step described next; no OCR or scanned-image handling is described in the

source material, so scanned (image-only) resumes are assumed to be out of scope for the present implementation.

B. Skill Extraction and Interest Detection

Technical and soft skills are extracted from the parsed resume text through a Gemini API call rather than a fixed keyword list, and the resulting skill set is shown to the candidate together with role-selection options on result.html. interest_service.py derives broad career interests from the extracted technical skills, again using the Gemini API rather than a rule table, so that a candidate whose resume lists an unusual combination of skills can still be routed toward a plausible interest area rather than falling outside a fixed category scheme.

C. Assessment Engine

Once a role is selected, question_generator.py uses the Gemini API to generate 20 multiple-choice questions and 3 coding questions from the candidate's profile. The assessment runs on quiz.html under a 20-minute timer, requests camera permission for basic monitoring, and issues a tab-switch warning that can auto-submit the assessment if the candidate leaves the browser tab. Coding questions are executed through the Judge0 API; MCQ, coding and overall scores are computed and displayed on final_result.html. AI-generated questions and results are exchanged in JSON format between the front end and back end, which keeps the assessment engine loosely coupled from the specific question-generation logic.

D. Skill Gap Analysis

skill_gap_service.py compares the candidate's assessed skill set against the requirements associated with the selected role and returns matched skills, missing skills, and improvement suggestions, presented on skill_gap.html. Because the comparison is driven by the same Gemini-derived skill list used earlier in the pipeline, an extraction error at the resume-parsing stage would propagate into the gap analysis, which is one reason Table VI recommends a candidate-facing confirmation step.

E. Job Recommendation

jobs_service.py queries the Adzuna job-search API for listings that match the candidate's selected role, presented through job_search.html/jobs.html. This step depends on network access to a third-party API and is therefore one of the two integrations still under verification (Section VI-B). Because the query is built from the candidate's selected role rather than from the full extracted skill list, the relevance of returned listings depends on how well that single role label captures the candidate's actual profile, which is one reason Section VII-B measures listing relevance as a distinct metric rather than assuming a role-based query is sufficient on its own.

F. Learning Roadmap

roadmap_service.py generates a roadmap specifying duration, difficulty level, a learning plan, resources, and a practice project, presented on roadmap.html. All of the above back-end services are exposed through Flask blueprint routes

with Flask-CORS enabled for cross-origin requests from the HTML/CSS/JavaScript front end; Visual Studio Code was used as the development environment.

VI. STATUS OF COMPONENTS UNDER VERIFICATION

Two integrations are implemented in code but are explicitly reported, in the project's own testing-status material, as still under verification and fixing rather than confirmed working end to end. A third element, named only in the architecture diagram, has no confirmed implementation at all.

A. Coding-Score Submission

The coding-assessment flow calls the Judge0 API to execute submitted code, but the project's source material states that coding-score submission is still under verification/fixing. This paper does not claim that a coding score is reliably computed and persisted for every submission; that claim would require the verification step the source material says has not yet been completed.

B. Job-API Integration

jobs_service.py calls the Adzuna API, but the same source material states that job-API integration is still under verification/fixing. Reliability under API rate limits, malformed responses, or network failure is therefore unconfirmed, and no fallback behaviour for these failure modes has been described.

C. ML Prediction Module

The architecture diagram names an "ML Prediction Module" using Scikit-learn for "career prediction using Machine Learning algorithms," distinct from the Gemini-API-based skill extraction and roadmap generation described in Section V. No file, route, or technology reference for a Scikit-learn-based prediction step appears elsewhere in the source material. This paper does not claim such a module is implemented; it is listed here as an architecture-diagram element whose implementation status is unconfirmed and should be checked by the authors.

TABLE III. TECHNOLOGIES USED AND THEIR PURPOSE

Category	Technology	Purpose
Front end	HTML, CSS, JavaScript	UI and assessment pages
Back end	Python, Flask, Flask-CORS	Routing, processing, app logic
NLP / AI	Google Gemini API	Extraction, interest, questions, gap, roadmap
Resume parsing	PyMuPDF, python-docx	PDF/DOCX text extraction
Data handling	pandas, CSV datasets	Role/skill/quiz/roadmap data
Coding evaluation	Judge0 API	Online code execution
Job search	Adzuna API	Job-listing retrieval
Dev environment	Visual Studio Code	Development and testing

TABLE IV. IMPLEMENTATION STATUS

Component	Artifact	Status
Resume upload & parsing	index.html, resume_service.py	Implemented

Skill extraction	Gemini in resume_service.py	Implemented
Interest detection	interest_service.py	Implemented
AI skill assessment (generation)	question_generator.py, quiz.html	Implemented
Performance evaluation (scoring)	quiz.html, final_result.html	Implemented
Skill-gap analysis	skill_gap_service.py, skill_gap.html	Implemented
Roadmap generation	roadmap_service.py, roadmap.html	Implemented
Coding-score submission	Judge0 API integration	Under verification/fixing
Job-API integration	jobs_service.py, Adzuna API	Under verification/fixing
ML Prediction Module	Scikit-learn (per diagram)	Not confirmed
Performance/UI testing, error handling	—	Pending

VII. EVALUATION METHODOLOGY

No part of this section has been executed on the system; it is written down in advance of any measurement, as a fixed protocol, so that once the pending components (Section VI) are verified, results can be produced under a documented method rather than asserted after the fact. Fixing the protocol before data collection also makes it possible for a reader to check, later, whether a reported number actually followed the stated procedure.

A. Scenarios

Table V lists controlled scenarios covering resume ingestion, the assessment engine, skill-gap analysis, job recommendation and roadmap generation. Each scenario is defined by a trigger a tester would perform and the outcome that would count as a pass, so that pass/fail is not left to interpretation once the pending components are ready to be exercised.

TABLE V. CONTROLLED EVALUATION SCENARIOS

ID	Trigger	Expected Outcome
R1/R2	Upload PDF / DOCX resume	Text extracted; skills and interests returned
A1	Complete 20 MCQ + 3 coding items	MCQ, coding, overall score computed
A2	Switch tab during assessment	Warning issued; auto-submit if repeated
G1	Compare profile against role	Matched/missing skills returned
J1	Request job listings	Listings returned, or clear failure message
P1	Request learning roadmap	Duration, difficulty, plan, resources, project

B. Metrics

For resume parsing and skill extraction: extraction precision and recall against a manually labelled set of resumes (not yet constructed). For the assessment engine: completion rate, and agreement between the computed overall score and an independently graded reference score for the same submissions. For skill-gap analysis: precision of the reported missing-skill set against a manually defined role requirement.

For job recommendation: relevance of returned listings to the selected role, and the API failure rate. For roadmap generation: whether the generated plan references the specific missing skills identified for that candidate. For every stage: end-to-end response latency, measured separately per service, since resume parsing, Gemini calls, Judge0 execution, and the Adzuna call each involve a separate network round trip and can fail or slow down independently of the others.

C. Measurement Procedure

Each scenario would be run against a fixed set of candidate resumes (a mix of synthetic and de-identified sample resumes, not yet assembled), repeated a fixed number of times, with all runs — successful or failed — counted and reported rather than discarded, so that a component with an occasional silent failure is not made to look more reliable than it is. Latency would be measured from a single host clock at the point each request is issued to the point its response is received, and reported as mean and standard deviation over the repeated runs.

D. Data and Privacy Considerations

Any evaluation involving real candidate resumes or camera data raises privacy considerations that this protocol does not yet resolve; the source material does not describe a data-retention or consent policy for uploaded resumes or for camera access during the assessment, and none is assumed here. Evaluation should therefore proceed on synthetic or explicitly consented data until such a policy is documented, and any future deployment involving real candidates should have that policy in place before data collection begins.

E. Success Criteria

A run of the protocol would be considered a pass for a given stage only if the stage returns a well-formed result for every scenario in Table V and does so within a latency bound the authors would need to set once real timings are available, since no bound is asserted here in the absence of measurement. A failed or malformed response — including a timeout from Gemini, Judge0, or Adzuna — counts against the stage's completion rate rather than being excluded as an outlier, consistent with the counting rule in Section VII-C. This keeps the pass/fail criterion tied to what a candidate would actually experience rather than to a best-case run of the system.

VIII. RESULTS AND VERIFICATION STATUS

The verified status reported here is functional, not quantitative. Per the project's own testing-status material, resume parsing, skill extraction, the assessment engine, skill-gap analysis, and roadmap generation are implemented; coding-score submission and job-API integration are implemented but still under verification and fixing; and final performance testing, UI testing and error handling are pending. No detection, extraction, scoring, or latency figures have been measured on this system under the protocol of Section VII, and none is reported or estimated here. Any number that might appear elsewhere in the project's own

materials that was not produced by that protocol should be treated as a placeholder rather than a result, and should not be cited as an evaluation outcome until it has been reproduced under Section VII.

TABLE VI. RISKS IN PENDING/AUTOMATED COMPONENTS AND MITIGATIONS

Risk	Mitigation
Coding score lost if Judge0 call fails silently	Explicit success/failure status; timeout and retry with visible error state
Job-API downtime/rate limiting blocks recommendations	Cache last successful response; graceful degradation message
Tab-switch false trigger auto-submits prematurely	Short grace period, explicit warning, event logged
Camera permission requested without stated policy	Explicit consent notice on storage and duration before prompt
Gemini extraction reports an unverified skill	Candidate confirms/corrects extracted skills before assessment

IX. DISCUSSION

A. What Is Demonstrated

The main finding at this stage is architectural rather than empirical: a resume-to-roadmap candidate journey can be assembled from a document parser, one general-purpose LLM API used for several distinct extraction and generation tasks, a hosted code-execution API, and a job-listing API, with most of the integration work expressed as Flask routes and service modules rather than as new machine-learning models. This suggests that a working end-to-end candidate journey does not, by itself, require the team to train or host any model of its own — the harder engineering problem turns out to be reliably chaining several third-party services, not any single algorithmic step.

B. LLM-Based Extraction versus Rule-Based Matching

Using the Gemini API for skill extraction, interest detection, question generation, skill-gap comparison and roadmap generation makes the system flexible across resume formats and wording, but it is correspondingly less predictable and harder to audit than a fixed keyword or rule-based extractor: two resumes describing the same skill differently could plausibly be scored differently, and no accuracy figure for this extraction step has been measured (Section VIII). A rule-based fallback or a candidate-facing confirmation step, as noted in Table VI, would reduce this risk without requiring a different extraction technology, and would also give the candidate a chance to correct an extraction error before it propagates into the skill-gap and roadmap stages that depend on it.

C. Automated Proctoring versus Candidate Experience

The 20-minute timer, camera permission and tab-switch auto-submit are intended to preserve the integrity of a self-administered assessment, but each introduces a trade-off: a strict auto-submit policy protects against deliberate circumvention but can also penalize an accidental tab switch, and a camera-permission request without a stated data-handling policy may deter legitimate candidates or raise privacy concerns unrelated to assessment integrity. Section X

and Table VI treat these as open risks rather than settled design choices, and a production deployment would need a documented policy before this behaviour could reasonably be enabled by default.

D. Reliability of Pending Integrations

Coding-score submission and job-API integration are implemented but explicitly unverified (Section VI). Table VI lists the principal risks this creates — a silently lost coding score, or a job-recommendation step that fails without a clear message to the candidate — together with mitigations that do not depend on completing the underlying verification work first, so that some of the risk can be reduced immediately while verification continues in parallel.

E. Relation to Prior Work

Section II-B grouped prior work into NLP-based matching, classifier-based domain prediction, skill-verification systems, and job-recommendation engines, none of which combine assessment, gap analysis, recommendation and roadmap generation in one candidate-facing system. Employability Hub's contribution, per Section II-C, is this integration rather than an improvement to any one stage; the trade-off is that end-to-end reliability now depends on every stage succeeding, which is why Sections VI and VIII are explicit about which stages are not yet confirmed.

F. Deployment and Scalability Considerations

The current implementation is described as a single Flask application with blueprint routes and no stated deployment target; the source material does not describe process management, horizontal scaling, or a production WSGI server in front of Flask's own development server. Because three of the pipeline's stages (skill extraction, question generation, roadmap generation) call the same Gemini endpoint, request volume from concurrent candidates would compound on that single upstream dependency before it compounded on the Flask application itself, so any future load testing should measure Gemini-call concurrency specifically rather than only overall request throughput. This is listed as a scalability question for the roadmap (Section XI) rather than as a defect, since no deployment or load-testing claim is made in the source material to begin with.

G. Threats to Validity

Three threats bear on any future reading of this paper's claims. First, every implementation-status label (Table IV) is drawn from the project's own testing-status material rather than from independent code review, so a component labelled "implemented" has been checked for presence, not for correctness under adversarial input. Second, the literature comparison in Section II rests on the project's own survey table, which Section II-B already flags as inconsistently linked to its reference list; conclusions drawn from Table I should be checked against the original sources once the bibliographic gaps in Section XVIII-equivalent references are closed. Third, because no evaluation data has been collected (Section VIII), every claim about relative strengths and weaknesses in Section IX is a design-level argument, not

an empirical finding, and should be read as motivation for Section VII's protocol rather than as its result.

X. LIMITATIONS AND SCOPE BOUNDARIES

- Evaluation. No dataset, accuracy figure, or performance measurement exists for any component; Section VII defines a protocol but it has not been executed.
- Verification. Coding-score submission and job-API integration are implemented but not confirmed reliable; the project's own materials describe both as still under verification/fixing.
- Scope of input. Only PDF and DOCX resumes are described as supported; scanned or image-only resumes are not addressed in the source material.
- Architecture-documentation mismatch. The architecture diagram's MySQL and Scikit-learn references are not reflected in the implementation described in Section V, and this has not been resolved (Section IV).
- Bibliographic uncertainty. The literature-survey table and the numbered reference list do not fully correspond (Section II-B), and several references are missing standard bibliographic fields.
- Testing coverage. Final performance testing, UI testing and error-handling are explicitly reported as pending in the project's own materials.
- Single-model dependency. Every AI-driven stage relies on one external LLM provider (Gemini); an outage, quota change, or pricing change at that provider affects extraction, question generation, gap analysis and roadmap generation simultaneously.

XI. ROADMAP

The items below are ordered by dependency rather than by importance: verification (item 1) gates the evaluation protocol (item 2), since running Section VII's scenarios against an unverified coding-score or job-API path would conflate a measurement problem with an implementation bug.

- Complete and verify coding-score submission and job-API integration end to end, including failure-mode handling, and update Table IV accordingly.
- Execute the evaluation protocol of Section VII on a documented set of resumes and report the resulting metrics, replacing the placeholders in Table V.
- Resolve the architecture-diagram discrepancy of Section IV by confirming whether MySQL and a Scikit-learn-based ML Prediction Module are part of the implemented system or should be removed from the diagram.
- Reconcile the literature-survey table with the numbered reference list and complete the missing bibliographic fields identified for the sixteen references.
- Add the future enhancements identified in the source presentation: more advanced AI/ML models for career prediction and job matching, a mock-interview feature, an

AI-assisted resume builder, a mobile application, and cloud deployment for scalability.

- Document a data-retention and consent policy for uploaded resumes and camera access before any evaluation involving real candidate data is carried out.

XII. CONCLUSION

Employability Hub combines resume parsing, LLM-based skill and interest extraction, a generated and timed skill assessment with automated code execution, skill-gap analysis, job recommendation, and roadmap generation into one candidate-facing pipeline. The core analytical modules are implemented; coding-score submission and job-API integration are implemented but not yet verified; and no performance or accuracy figures are reported because none have been measured. The most valuable next step is the same one this paper argues for throughout: run the fixed evaluation protocol of Section VII, complete the verification work of Section VI, and report the resulting numbers under that documented method rather than asserting them in advance.

Availability

Source code and configuration: repository link to be supplied by the authors; none was present in the source presentation.

Acknowledgment

Portions of this manuscript were drafted with an AI language model (Claude, Anthropic) based on the authors' project presentation, and should be reviewed against the project's own repository and codebase and edited by the authors before submission. No experimental results were generated or estimated by the AI system; every implementation-status and testing claim restates the source presentation, with no accuracy, latency or performance figures reported because none were present in that source. The authors should re-verify every status label in Table IV against the current codebase at the time of submission, since implementation status can change between the drafting of this manuscript and its final review.

REFERENCES

- [1] S. Tayal, T. Sharma, S. Singhal, and A. K. Thakur, "Resume screening using machine learning," *Int. J. Eng. Research*, 2023.
- [2] V. Khatri, V. Kumar, K. Thakur, et al., "AI-powered automated resume screening and job matching system using NLP and machine learning," *Int. J. Research in Eng., Sci. and Mgmt.*, 2025.
- [3] A. K. Sinha, M. A. K. Akhtar, and M. Kumar, "Automated resume parsing and job domain prediction using machine learning," 2023.
- [4] P. Bedi, T. Roy, V. Arora, and P. Gole, "Smart contract based skill verification system for recruitment," 2023.
- [5] R. Sharma, A. Rajput, and V. K. Srivastav, "Job description and resume matching system using NLP," 2025.
- [6] I. Marin and H. Amel, "Web platform for job recommendation based on machine learning," 2023.
- [7] S. Khandare et al., "ResuMate AI: A comprehensive framework for intelligent resume analysis, scoring, and recommendation," 2025.
- [8] H. E. Sharif et al., "Smart assistant to choose the best resume using NLP and machine learning," 2024.
- [9] S. Ambareesh, N. K. Thakur, U. Bhattarai, et al., "Resume shortlisting using NLP," in *Proc. ICDECS*, 2023.

- [10] X. Li et al., "Resume screening using deep learning and NLP," Knowledge-Based Systems, 2021.
- [11] J. Oh and S. Lee, "Extraction of competencies from job postings using NLP," Expert Systems with Applications, 2019.
- [12] C. Xu et al., "Resume screening using deep learning and NLP," Knowledge-Based Systems, 2021.
- [13] R. Bhowmik et al., "Resume screening using semantic similarity and clustering," 2021.
- [14] S. Huang et al., "Resume screening and ranking with NLP techniques," Applied Sciences, 2021.
- [15] Y. Kang and J. Lee, "Resume analysis for job matchmaking using word embedding," 2020.
- [16] X. Wang et al., "An intelligent resume screening system based on NLP and ML," Future Generation Computer Systems, 2020.

Note on citation numbering: the survey table (S1–S16) and this reference list ([1]–[16]) do not correspond position-for-position; several entries were matched with confidence by title (S4, S8, S12, S13, S15, S16) while others could not be matched confidently. This should be reconciled by the authors before submission.