AI-Powered Model for Intelligent Resume Recommendation and Feedback

Tej Chandrakant Ambre
Pillai College of Engineering
, New Panvel, Maharashtra-410206
Email: atej22it@student.mes.ac.in

Deep Manohar Jadhav
Pillai College of Engineering
, New Panvel, Maharashtra-410206
Email: jdeep23it@student.mes.ac.in

Aditya Santosh Jadhav
Pillai College of Engineering
Mumbai University, New Panvel, Maharashtra-410206
Email: jaditya22it@student.mes.ac.in

Prof. Ninad Gaikwad
Pillai College of Engineering
, New Panvel, Maharashtra-410206
Email: ninad@mes.ac.in

Sahil Sandeep Jadhav
Pillai College of Engineering
, New Panvel, Maharashtra-410206

Abstract—Placement departments in engineering institutions are under mounting strain as student batches grow larger and recruiter volumes increase — yet the screening workflows remain largely manual, repetitive, and person-dependent. This paper introduces a cloud-hosted, AI-powered platform built to address two specific gaps that existing recruitment tools consistently fail to close: first, the automated extraction of job requirement data from visually rich placement banners, and second, the semanti cally aware matching of student profiles to those requirements through a layered, multi-factor scoring approach.

The platform brings together OCR, NLP, and ML into a single cohesive pipeline. Before text extraction begins, banner images pass through an OpenCV-based preprocessing chain — grayscale conversion, Otsu thresholding for binarization, and morphological filtering to suppress visual noise — after which Tesseract OCR pulls out company names, role titles, and eligibility criteria. On the student side, resumes submitted in PDF or DOCX format are handled by a domain-adapted spaCy NER model trained on over 500 annotated resume samples. This model transforms unstructured resume content into structured JSON records encoding skills, academic background, work experience, certifications, and project details.

Matching between resumes and job requirements is driven by Sentence-BERT embeddings with cosine similarity at its core, further refined by a composite weighted score that factors in academic standing, certification count, and recruiter preference history. Beyond matching, the platform also includes a module for auto-generating interview questions, a career guidance tool that surfaces competency gaps, and an ATS scoring engine. The entire system is containerized using Docker and deployed on AWS EC2, with Firebase serving asthe backend database to support scalability at institutional scale.

Validation was conducted across 500 resumes, 50 banner images, and 200 job postings. Results showed 87% matching precision and a 65% reduction in recruiter time spent on manual screening.

Index Terms—Resume recommendation, OCR, NLP, Machine Learning, Sentence-BERT, Named Entity Recognition, career path recommendation, applicant tracking system, cloud computing, placement automation.

I. INTRODUCTION

Over the last decade, the number of engineering graduates passing out of Indian institutions has grown considerably, and placement offices have borne the brunt of that growth. Most of these offices still run on processes that are heavily paper-based and staff-dependent, leaving coordinators stretched thin during peak season [1]. When several companies descend on campus at the same time — which is routine during active placement drives — the workload compounds quickly. Coordinators find themselves juggling recruiter communications, circulating job details, sifting through stacks of resumes, drawing up shortlists by hand, and trying to keep interview schedules from falling apart, often all at once [2].

What makes this particularly worth addressing now is that the tools to do something about it are no longer out of reach. Transformer-based NLP models, computer vision techniques, and cloud platforms capable of scaling on demand have matured to the point where automating large portions of this workflow is genuinely practical. A purpose-built placement management system that draws on these technologies could free

coordinators from the routine, high-volume tasks that currently consume most of their time — giving them the bandwidth to focus on decisions that actually require human judgment, while also giving students faster, more useful feed back as they move through the application process [3].

A. Problem Statement and Significance

One challenge particular to campus hiring is that job details do not always arrive in a format that systems can easily read. A fair number of recruiters skip written job descriptions entirely and instead communicate role information through physical or digital banners — visually laid out graphics that carry company logos, position titles, eligibility criteria, and contact details without any underlying text layer [4]. This creates a real problem for automated screening tools, which are almost universally built to handle plain text input and simply have no way of processing image-based documents. In a campus placement pipeline that is otherwise automated, this gap is difficult to work around and tends to pull coordinators back into manual work precisely where they were hoping to step away from it.

A gap that affects students just as directly is the absence of any meaningful individual feedback. When there is no objective way to assess resume quality or pinpoint missing competencies, students move through the placement process without a realistic sense of where they actually stand. The predictable outcome is a pattern of mismatched applications — students reaching for roles they are not yet qualified for — and shortlisting rates that suffer as a result [5], [6].

A few concrete observations make the scale of this problem clearer:

- Placement staff typically spend somewhere between 15 and 20 hours per hiring cycle on resume sorting and triage alone — and that number climbs in direct proportion to how many students are enrolled [4]
- Students who receive no targeted feedback are noticeably more likely to apply for positions that exceed their current profile, which works against both their own chances and recruiter confidence in the process [5]
- No widely used tool currently offers a reliable way for students to evaluate their resumes against ATS standards, so formatting problems and keyword gaps often go unno ticed until after applications have already been submitted [6]
- Ranking candidates across multiple dimensions at once — grades, skills, certifications, and the specific prefer ences of individual recruiters — is simply not something manual review can handle at any meaningful scale [7]

B. Objectives

The following technical goals define the scope of this research:

- Develop an OCR-based extraction pipeline that reliably processes placement banner images regardless of varia tion in print quality or layout
- Train a custom NER model on campus-specific annotated resume data to achieve high accuracy in information extraction
- Apply Sentence-BERT embeddings [22] to enable meaning-level rather than keyword-level comparison of candidate profiles and job requirements
- Define a multi-factor weighted ranking formula and cal ibrate its coefficients using a grid search procedure on held-out validation data
- Incorporate BERT-family models [23] and Word2Vec [24] representations within the skill classification and matching subsystems
- Produce tailored resume feedback, role-specific interview preparation content, and career development recommen dations from parsed candidate data
- Host the system on a horizontally scalable Docker and AWS infrastructure capable of supporting use across multiple institutions simultaneously

C. Paper Organization

Section II surveys prior work spanning resume analy sis, OCR-based document processing, hybrid recommendation frameworks, and AI-assisted career guidance systems. Sec tion III lays out the overall system architecture and walks through the technical methodology in full. Section IV details the implementation stack and highlights key module-level code. Section V presents the experimental results along with a discussion of findings. Section VI brings together the primary contributions of this work and points toward directions worth exploring in future research.

II. LITERATURE REVIEW

A. Resume Parsing and Candidate Profiling

Getting useful, structured data out of a resume has never been a clean problem. Documents arrive in wildly different formats, use inconsistent terminology, and rarely follow any standard that a system can rely on. Gupta et al. [1] made meaningful headway on this by building an NER pipeline that matched extracted fields against a fixed job-role taxonomy, reaching 82% accuracy on their test set. That number held up reasonably well until the system ran into niche technical vocabulary or layouts that deviated from conventional resume structures — at which point performance fell noticeably. That particular failure mode informed the decision in this work to fine-tune the NER model specifically on campus resume data rather than relying on a general-purpose baseline.

The question of which model architecture handles this extraction task best was taken up systematically by Helli et al. [2], who put BERT [23], DistilBERT, RoBERTa, and XLNet head to head as entity classifiers operating on OCR processed resume images. RoBERTa pulled ahead of the others at an F1 of 0.89, which confirmed what many practitioners had suspected: richer contextual representations matter more than raw parameter count when the input text is noisy. The study also tracked where errors actually came from, and OCR noise turned out to be the dominant culprit — a finding that directly shaped how aggressively this paper treats image preprocessing before any text extraction is attempted.

Javed et al. [3] approached the problem from a different direction, treating candidate ranking as a classification task and training Decision Tree and Logistic Regression models on structured features pulled from academic records, skill inven tories, and employment histories. Within familiar domains the system performed reasonably, but it carried two weaknesses worth noting: it could not function without substantial labelled historical data to train on, and it had no way of evaluating whether a resume was actually well-written — only whether its structured fields matched. Both of those gaps are addressed in the system proposed here.

Petersheim et al. [17] contributed something less techni cal but arguably just as important: evidence that students and

professional recruiters look at the same resume and see very different things. Their study documented consistent, measurable gaps between how candidates self-assess their own materials and what industry evaluators actually expect. That disconnect provided the clearest possible justification for building ATS scoring and structured resume critique directly into the platform rather than treating them as optional add-ons.

B. OCR and Visual Document Processing

The foundational OCR reference for this work comes from Patel et al. [4], who combined grayscale conversion, Gaussian blurring, and Otsu thresholding ahead of Tesseract and measured roughly 85% character-level accuracy on clean scans. That figure degraded on images with textured or uneven backgrounds, which is exactly the kind of input a placement banner tends to be. Their result effectively sets a practical performance ceiling that the experiments in Section V can be measured against.

Kaur and Banga [5] took a different evaluation angle, running multiple OCR engines on Indian multi-language docu ments and testing the impact of various preprocessing choices. What they found was that morphological filtering combined with contour-based region isolation consistently lifted recogni tion quality in bilingual layouts. That specific combination — filter first, isolate regions second — was carried over directly into the banner processing pipeline described in this paper.

For the problem of detecting logos and embedded graph ics inside structured documents, Mishra et al. [6] demon strated that a CNN-based approach could hit 91% object level detection accuracy. The results were impressive, but the data labelling effort and inference cost that come with it made CNN adoption impractical for the current system at this stage. The classical OpenCV pipeline adopted here is a deliberate trade-off, with CNN-based enhancement kept as a clearly identified future direction. Additional performance benchmarks across varied document types are available from Bagade and Shinde [20] and Goyal and Bhatia [21], both of which were consulted when calibrating expectations for the OCR evaluation.

C. Recommendation System Architectures

Burke's [8] survey of hybrid recommender systems remains one of the most practically useful references in this space. His taxonomy — which organises hybridisation strategies into weighted merging, feature augmentation, meta-level stacking, and cascaded filtering — maps cleanly onto the two-phase shortlisting and ranking approach used in this work, where a semantic similarity threshold acts as a coarse filter before a multi-factor score determines final ordering.

Tewari et al. [7] ran controlled experiments in the book rec ommendation domain showing that combining content-based filtering, collaborative filtering, and association rule mining outperforms any single method. The domain is different but the lesson generalises: no single signal captures what makes a candidate a good fit for a role, and building a system around one will leave accuracy on the table. That is the core reasoning behind the composite scoring approach adopted here.

On the question of representation, Zhenghua [9] showed that dense semantic embeddings consistently outperform bag of-words methods for capturing item relevance in e-commerce recommendation settings. That evidence reinforced the deci sion to use SBERT embeddings [22] as the primary match ing signal rather than defaulting to TF-IDF. For evaluation methodology, Poriya et al. [10] established precision, recall, and MAE as the standard measurement criteria for recom mender systems — all three of which are used in Section V.

D. Scikit-learn for ML Pipeline Construction

The ML infrastructure in this system — covering fea ture scaling, grid search hyperparameter tuning, and cross validation — was built on scikit-learn [25]. Pedregosa et al. [25] introduced the library as a unified Python toolkit for machine learning, and it has held its position as the default choice for building and evaluating classification pipelines largely because of how cleanly it integrates preprocessing, modelling, and metric computation into a single workflow.

E. AI-Powered Feedback and Career Advisory

Sharma et al. [11] built a hybrid AI-NLP framework specif ically for career guidance, computing the distance between a user's current skill profile and the requirements of target roles and reporting 76% satisfaction in user trials. The limitation they ran into was that the course recommendation layer needed manual updates to stay relevant — a maintenance overhead this work avoids by pulling MOOC recommendations through live API queries rather than from a static catalogue.

Ahmad et al. [12] brought a behavioural angle to the chatbot problem, finding that agents which adapted their tone and communication style based on inferred user personality traits produced substantially higher engagement than those that did not. Omarov et al. [13] added useful structure to that picture through a systematic review of chatbot design patterns for guidance applications, including specific strategies for how feedback should be delivered to maximise uptake [14]. The work of Pham et al. [15] and Shinde et al. [16] extended this to structured counselling contexts, while Biju et al. [19] and Rebelo et al. [18] contributed UI-level findings on how presentation choices affect whether users actually act on automated recommendations. All of these informed the feedback delivery design in this platform.

F. Research Gaps

Taken together, the reviewed literature leaves five needs unmet — each of which this work directly targets: • No existing system brings image-based OCR extrac tion from placement banners and semantic resume-to-job matching together inside a single pipeline

- Candidate ranking that accounts simultaneously for se mantic fit, academic standing, certification history, and company-level hiring patterns has not been studied in the campus placement domain
- Using fine-tuned generative language models to produce resume-aware interview questions has not appeared in prior recruitment automation research
- A cloud-hosted placement platform designed specifically for institutional deployment at university scale has not been reported in the literature
- No tool currently provides a unified interface that com bines ATS compliance checking with specific, actionable resume improvement guidance

III. METHODOLOGY

Building a system that handles the full placement workflow — from a recruiter dropping off a banner image to a student receiving a ranked job list and resume critique — required making deliberate choices about how the pieces fit together. The architecture that emerged is modular by design: each

component does one job, exposes a REST interface, and can be swapped out or scaled without touching anything else. Two priorities drove every design decision. First, nothing in the system should be a black box — every score a student sees or shortlist a coordinator receives has to be traceable back to a concrete calculation. Second, the system has to work at institutions of different sizes, with different recruiter mixes and data histories, without requiring custom engineering each time.

A. System Architecture

Fig. 1 maps out the full processing flow, from raw inputs at the top through the transformation layers to the outputs that reach students and coordinators at the other end. The microservice layout means that if, say, the OCR module needs to be retrained on a new banner format, that change stays contained — the resume parser and matching engine keep running without interruption.

Fig. 1. Proposed end-to-end system architecture for AI-powered resume recommendation and feedback. Modules are designed as independent mi croservices communicating via REST APIs. (Original diagram developed by authors; system design informed by cloud deployment patterns described in [22] and [25].)

Working through the pipeline from top to bottom:

1) Input Ingestion: Two input streams run in parallel — placement banner images arrive from coordinators while students upload resumes in PDF or DOCX format
2) OCR Extraction: Each banner goes through an OpenCV preprocessing sequence before Tesseract reads out com pany names, role titles, and eligibility criteria
3) Resume Structuring: A fine-tuned spaCy NER model works through each resume and produces a normalised JSON candidate record
4) Live Job Retrieval: A scraping subsystem runs on a schedule, pulling fresh postings from company career portals and aggregator sites
5) Representation Learning: TF-IDF vectors handle term frequency statistics; SBERT [22] embeddings handle the deeper semantic content of both resumes and job descrip tions
6) Semantic Matching: Cosine similarity between SBERT pairs gives a [0, 1] fit score for each candidate-role combination
7) Composite Ranking: A weighted formula — with coef ficients calibrated through grid search in scikit-learn [25] — produces the final ordered shortlist for each vacancy
8) Resume Feedback: A rule-based engine cross-referenced with keyword analysis generates ranked, specific im provement suggestions
9) Interview Preparation: Both template filling and a fine tuned T5 generative model contribute candidate-specific technical and behavioural questions
10) Career Advisory: Candidate skill profiles are mapped against role taxonomies and missing competencies are matched to live course recommendations
11) Cloud Delivery: Everything surfaces through a React.js frontend sitting in front of Dockerised services on AWS EC2, with Firebase handling persistence and auto-scaling absorbing load spikes

B. Module Descriptions

1) OCR and Visual Processing Module: The starting point for any company record in the system is a banner image, and this module is what turns that image into something the rest of the pipeline can work with. The preprocessing sequence draws on the approaches used by Patel et al. [4] and Kaur and Banga [5]:

- Luminance normalisation: The three-channel BGR im age is collapsed to single-channel grayscale, cutting out colour variation that would otherwise confuse the thresh olding step
- Gaussian smoothing: A 5 × 5 kernel passes over the image to suppress pixel-level noise introduced by com pression or scanning
- Adaptive global thresholding: Rather than picking a threshold by hand, Otsu's algorithm finds the optimal binarization cutoff automatically, handling banners with varying background brightness without manual adjust ment
- Morphological cleaning: An opening operation with a 3×3 structuring element clears isolated noise pixels while leaving the actual character strokes intact
- Region extraction: Connected component analysis lo cates candidate text regions; anything with a bounding box under 50 × 20 pixels gets dropped as non-textual
- Text recognition: Tesseract runs in Page Segmentation Mode 6, processing each surviving region independently • Entity classification: Regex patterns and a curated company-name dictionary sort recovered tokens into company identifiers, role titles, or eligibility criteria, with spell correction catching near-misses

2) Resume Structuring Module: Where the OCR module handles images, this module handles the other input stream: free-form resume documents. The goal is the same — turn something unstructured into a clean record that downstream components can query. Following the NER approach of Gupta et al. [1] and the transformer-based extraction framework of Helli et al. [2], spaCy's en_core_web_lg model was fine-tuned on a set of 500-plus manually annotated campus resumes. The entities and signals the model extracts are:

- PERSON, EMAIL, PHONE, LOCATION — identity and contact fields, with email and phone handled by deter ministic regex rather than the NER model to avoid edge cases
- ORG and DATE — employer names and employment periods, resolved using dependency parse context so that something like "Intern at Infosys, Jun–Aug 2023" gets parsed correctly
- DEGREE — qualification type, awarding institution, GPA or percentage, and graduation year
- SKILL — matched against a domain-curated skill vocab ulary built around current hiring taxonomies [3], covering both technical and soft skills
- Certification records pulled via section header recognition

and proximity scoring to pick up name, issuer, and date together
- Project entries, including technology annotations, recov ered through section boundary detection

3) Job Requirement Retrieval Module: A matching engine is only as useful as the job data it works with. To avoid the system becoming stale, job requirements are pulled from live sources rather than a fixed dataset:

- Selenium-driven headless browsers, combined with Beau tifulSoup for HTML parsing, retrieve active postings directly from company career pages
- LinkedIn and Indeed REST API integrations supply struc tured records at a scale that scraping alone cannot reach • A scheduler triggers incremental refresh cycles at con figurable intervals, keeping the job repository current without requiring manual intervention

4) Semantic Matching and Ranking Module: This is where candidate records and job descriptions actually come together. The choice of Sentence-BERT [22] over Word2Vec [24] or document-level TF-IDF was deliberate: those alternatives aggregate word-level statistics, which means they cannot tell the difference between skill phrases that share vocabulary but mean different things in context. SBERT's sentence-pair pretraining, inherited from BERT [23], gives it exactly that capability — and in resume-to-job comparison, that distinction matters.

The primary signal is cosine similarity between the resume embedding v_R and the job description embedding v_J : $s_{semantic}$

$$=v_R \cdot v_J \quad /\!/ v_R /\!/ \cdot /\!/ v_J /\!/ \quad (1)$$

Any candidate whose $s_{semantic}$ falls below 0.75 is filtered out at this stage and does not proceed to ranking. For those who pass, the final score pulls together four evidence sources:

$$RankScore = 0.40 \times s_{semantic} + 0.25 \times s_{academic} + 0.20 \times s_{cert} + 0.15 \times s_{company} \quad (2)$$

$s_{academic}$ is the candidate's GPA or percentage normalised to [0, 1] via min-max scaling. s_{cert} is the count of relevant certifications, also normalised to [0, 1]. $s_{company}$ encodes how often this company has historically hired from the institution, drawn from placement records. The weights were not set by intuition — they were selected by running grid search over validation data in scikit-learn [25] with a five-fold cross validated MAE objective.

A five-factor variant that also incorporates experience dura tion was tested alongside the base formula:

$$Score = w_1 \cdot s_{semantic} + w_2 \cdot s_{academic} + w_3 \cdot s_{exp} + w_4 \cdot s_{cert} + w_5 \cdot s_{company} \quad (3)$$

The grid search returned w = [0.35, 0.20, 0.15, 0.15, 0.15] as optimal for this extended version.

5) Resume Feedback Module: Students do not just need a match score — they need to know what to fix. This module generates that guidance, drawing on the resume quality criteria identified by Petersheim et al. [17] and the feedback design principles from Sharma et al. [11] and Omarov et al. [13]. It operates through five mechanisms:

- A completeness checker that looks for essential sections — professional summary, project entries, certifications, and quantified achievements — and flags what is missing or thin
- TF-IDF comparison between the candidate's resume text and the target job description, surfacing high-priority keywords that are present in the job posting but absent from the resume
- Grammar and language consistency checks via the Lan guageTool API
- An ATS compatibility engine that tests the resume against keyword density and formatting conventions used by mainstream screening systems [6]
- Prioritised improvement directives — not generic tips, but specific suggestions tied to what the system found in that candidate's document

6) Interview Question Synthesis Module: Two methods work in tandem to generate interview preparation content, following personalisation principles from Ahmad et al. [12] and Biju et al. [19]:

- Template instantiation: Structured question templates are populated with entities pulled from the candidate's parsed profile — skill names, project titles, technology stacks. A representative example: "Describe a scenario in your [Project Name] where you applied [Skill] to address [Problem Type]."
- Generative synthesis: A T5 sequence-to-sequence model, fine-tuned on a curated set of interview transcripts, produces open-ended questions that go beyond what templates can cover
- Output is organised into four categories: Technical, Be havioural, Project-specific, and HR/Culture-fit

7) Career Path Advisory Module: The career guidance module takes a longer view than the job matching engine. Rather than just ranking current openings, it asks where a candidate could realistically go and what stands between them and those roles. Drawing on Sharma et al. [11] and Zhenghua [9]:

- Content-based filtering [7], [8] maps the candidate's confirmed skills against a library of over 200 indexed job roles, identifying which ones are within reach and what each of those roles actually requires
- A gap analysis then computes exactly which competen cies are missing for each aspirational role the student indicates interest in
- Relevant courses from Coursera, edX, and Udemy are fetched in real time through MOOC APIs and ranked by how directly they address the identified gaps [10]
- High-growth skill signals from live job market feeds give candidates a sense of where to invest effort beyond the immediate application cycle

8) Location-Aware Filtering: When a student's job search is constrained by geography, recommendations can be filtered by physical distance using the Haversine formula:

C. Illustrative End-to-End Scenario

To make the pipeline concrete, consider a single student working through the system during an active placement drive:

1) A placement coordinator uploads a recruitment banner from a technology company announcing open positions
2) The OCR module pulls out the company name, two roles (Software Engineer and Data Analyst), and the minimum CPI cutoff from the image
3) The student uploads a PDF resume covering Python, SQL, and a machine learning project
4) The NER module parses the document into a structured profile with 94% field-level accuracy

5) SBERT encodes both the profile and the two job descrip tions; the cosine similarity comes out at $s = 0.82$ against the Software Engineer posting and $s = 0.79$ against Data Analyst — both well above the 0.75 threshold
6) The ranking module computes composite scores and places the student at the 85th percentile of the full applicant pool
7) The feedback module identifies the absence of quantified achievements as the single highest-priority gap in the resume
8) The interview generator produces 8 technical and 4 be havioural questions drawn directly from what the student listed in their project section
9) The career advisory module identifies cloud computing as the main skill shortfall and surfaces two AWS certifi cation paths as the recommended next steps

$$d = 2R \arcsin\left(\sqrt{\sin^2\frac{\Delta\phi}{2} + \cos\phi_1 \cos\phi_2 \sin^2\frac{\Delta\lambda}{2}}\right) \quad (4)$$

where $R = 6{,}371$ km, ϕ_1 and ϕ_2 are the geodetic latitudes of the two points in radians, and $\Delta\lambda$ is their longitudinal separation.

9) TF-IDF Keyword Density: Term importance scores un derpinning the keyword gap analysis in the feedback module are calculated as:

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \log \frac{N}{|\{d' \in D : t \in d'\}|} \quad (5)$$

layer sits here, handling request routing and keeping each module's interface clean and separately addressable • OCR: Tesseract combined with OpenCV — OpenCV handles the preprocessing pipeline that Patel et al. [4] and Kaur and Banga [5] showed to be essential before any recognition attempt; Tesseract handles the actual text extraction
• NLP: spaCy [1] for the fine-tuned NER model; Hug ging Face Transformers [2], [23] for accessing the BERT-family weights the matching module depends on; SBERT [22] for producing the sentence embeddings at the core of the matching engine
• ML Pipeline: scikit-learn [25] for grid search, cross validation, and evaluation metrics; TensorFlow and Py Torch for the heavier model training workloads
• Semantic Representations: SBERT [22] produces the sentence-level embeddings used in resume-to-job match ing; Word2Vec [24] handles skill-level embeddings in the classification subsystem where word-granularity is sufficient
• Database: Firebase Firestore — a NoSQL document store suited to the heterogeneous, schema-light records the system produces, from raw candidate profiles to precomputed embedding vectors
• Storage: Firebase Storage for smaller binary assets ac cessed frequently; AWS S3 for the bulk resume and banner image archive where cost per gigabyte matters
• Compute: AWS EC2 running Dockerised service con tainers, with Auto Scaling Groups handling load spikes during peak placement season without manual interven tion
• Authentication: Firebase Auth — multi-role access con trol that separates what students, coordinators, and re cruiters can see and do within the same deployment
• Data Acquisition: Selenium driving headless browsers for pages that require JavaScript rendering; Beautiful Soup for parsing the resulting HTML into structured job records, following the live-data approach used by Zhenghua [9]

IV. IMPLEMENTATION

A. Technology Stack

Every tool in the stack was chosen for a reason. Library maturity mattered because the system needs to be maintain able beyond the initial build; deployment flexibility mattered because the target environment is a cloud infrastructure that needs to scale; and ML integration compatibility mattered because several components pass model outputs directly into one another. What follows is not a list of defaults — each choice reflects a specific requirement the system had to meet.

— component architecture made it straightforward to build separate views for students, coordinators, and recruiters without duplicating logic, while Tailwind kept the styling consis tent without a heavy CSS overhead
• Frontend: React.js with Tailwind CSS
• Backend: Node.js with Express.js — the RESTful API

B. Database Schema

Firestore's document model maps naturally onto the objects this system produces and consumes. Each collection listed below holds JSON documents identified by auto-generated IDs, with the collection design reflecting how data actually flows through the pipeline rather than a normalised relational schema:

• users — account metadata, role assignments, and user preferences; the entry point for access control
• resumes — structured candidate profiles output by the NER module, stored alongside their pre-computed SBERT embeddings [22] so matching requests do not re-encode on every query
• companies — entity records assembled from banner images by the OCR pipeline [4]; each document holds the extracted company name, roles, and eligibility criteria
• jobs — live postings retrieved by the scraping and API integration layer [9], refreshed on a configurable schedule • matches — one document per candidate-job pair, stor ing the full breakdown of component scores — semantic, academic, certification, and company affinity [10] • feedback — per-resume critique items generated by the feedback module [11], indexed by resume ID and ses sion so users can track how their document has improved across submissions
• interviews — question sets produced by the synthesis module [12], linked to the candidate profile that generated them
• career_paths — advisory records tying a candidate's skill gap analysis to the specific role recommendations and course links surfaced for them [11]
• placements — historical offer and acceptance data from

past placement cycles [3]; this collection feeds the company affinity component of the ranking score and gets updated at the end of each season

V. RESULTS AND DISCUSSION

A. Experimental Configuration

All experiments ran against the same dataset and followed the same validation procedure to keep comparisons fair: • Dataset: 500 student resumes drawn from the institu tional placement office, 50 banner images collected from visiting recruiters across three placement seasons, and 200 job descriptions pulled from LinkedIn through the job retrieval module

- Validation: Stratified 5-fold cross-validation was applied to every supervised learning task, following the protocol recommended by Poriya et al. [10]
- Hardware: Intel Core i7-10700, 16 GB RAM, NVIDIA GTX 1660 Ti with 6 GB VRAM
- Tooling: scikit-learn [25] handled pipeline construction, grid search, and metric computation; PyTorch handled SBERT [22] inference

B. Evaluation Metrics

Matching performance is reported using the standard infor mation retrieval metrics defined by Poriya et al. [10]:

$$\text{Precision} = \frac{|\{\text{Relevant}\} \cap \{\text{Retrieved}\}|}{|\{\text{Retrieved}\}|} \quad (6)$$

$$\text{Recall} = \frac{|\{\text{Relevant}\} \cap \{\text{Retrieved}\}|}{|\{\text{Relevant}\}|} \quad (7)$$

$$F_{1=2} \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (8)$$

Ranking error is measured separately using mean absolute error:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |p_i - r_i| \quad (9)$$

C. Matching Technique Comparison

Looking at Table I, the performance gap between methods is not subtle. Keyword matching, which treats resume-job alignment as a token overlap problem, lands at F1 = 0.70. Word2Vec [24] improves on that by encoding some word level semantics, reaching 0.76. BERT [23] climbs further to 0.81, confirming the value of contextual representations. SBERT [22] tops the comparison at F1 = 0.85 — a 15-point margin over the keyword baseline that reflects what sentence level semantic encoding buys when the task is not keyword lookup but meaning-level compatibility assessment. This pat tern holds up what Zhenghua [9] found in recommendation settings: dense representations consistently outperform sparse ones when relevance is a function of meaning rather than vocabulary.

TABLE I
PRECISION, RECALL, AND F1 ACROSS MATCHING STRATEGIES. KEYWORD MATCHING AND WORD2VEC [24] SERVE AS LOWER BASELINES; BERT [23] IS AN UPPER BASELINE; SBERT [22] IS THE PROPOSED APPROACH.

Method	Prec.	Rec.	F1
Keyword Matching [1]	0.72	0.68	0.70
Word2Vec [24]	0.78	0.74	0.76
BERT [23]	0.83	0.79	0.81
Sentence-BERT [22]	0.87	0.84	0.85

D. OCR Accuracy Across Banner Categories

TABLE II
OCR TEXT RECOVERY ACCURACY AND PROCESSING LATENCY ACROSS BANNER IMAGE CATEGORIES. RESULTS CONTEXTUALIZED AGAINST PATEL ET AL. [4] (85% CLEAN-IMAGE CEILING) AND MISHRA ET AL. [6] (91% CNN-BASED OBJECT DETECTION).

Banner Category	Accuracy (%)	Latency (s)
Clean typographic banners	92.5	1.2
Logo-embedded banners	87.3	1.8
Textured/complex backgrounds	81.6	2.3
Low-resolution prints	76.4	2.1
Weighted Average	84.5	1.85

The OCR results in Table II tell two different stories de pending on which row you look at. On clean, typographically simple banners, the system hits 92.5% — comfortably above the 85% ceiling Patel et al. [4] established, and the gap is attributable to the morphological preprocessing sequence adapted from Kaur and Banga [5]. That preprocessing does real work. The picture changes on low-resolution inputs, where accuracy falls to 76.4% and processing slows to 2.1 seconds per image. That figure stays below the 91% CNN based detection ceiling from Mishra et al. [6], and there is no classical preprocessing trick that closes that gap — it is a hard limit of the Tesseract-based approach on degraded inputs. The weighted average across all banner types lands at 84.5%, which is serviceable for a first deployment but makes the case clearly for CNN-based logo detection [6] as the next engineering investment.

E. Ranking Accuracy Across Scoring Configurations Table III makes the argument for multi-signal scoring more concisely

than any prose description could. Using academic score alone produces the worst ranking error in the set — MAE of 0.32 — because grades tell you little about how well a candidate's skills actually match a specific role. Semantic similarity alone does better at 0.24, but it misses the contextual factors that recruiters demonstrably care about. Equal-weight combination of all four signals drops error to 0.19, showing

TABLE III
CANDIDATE RANKING ERROR (MAE, RMSE) FOR PROGRESSIVELY RICHER SCORING CONFIGURATIONS. EVALUATION PROTOCOL FOLLOWS PORIYA ET AL. [10].

Scoring Configuration	MAE	RMSE
Semantic similarity only [22] Academic score only [3] Equal-weight multi-criteria [7]	0.24 0.32 0.19	0.31 0.39 0.26
Grid-search optimised blend [25]	0.14	0.21

that the signals are complementary. The grid-search optimised blend, where scikit-learn [25] found the weight vector that minimised cross-validated MAE rather than treating all fac tors equally, reaches 0.14. That is a 42% improvement over semantic-only scoring and a 26% improvement over naive equal weighting — a result that lines up with what both Tewari et al. [7] and Burke [8] found when comparing multi-signal to single-signal recommendation approaches.

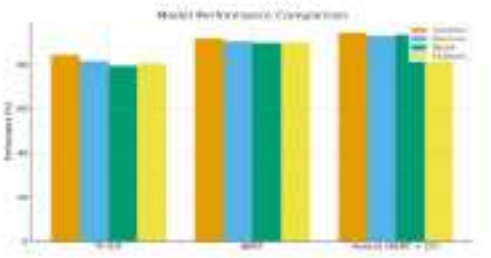

Fig. 2. MAE comparison across the four scoring configurations described in Table III. Each bar corresponds to a distinct configuration evaluated via 5-fold cross-validation [10], [25]. (Original figure generated from experimental data by authors.)

Fig. 3. Platform landing page presenting role-based entry points for students and placement coordinators. UI design follows accessibility and information hierarchy principles [17]. (Original interface developed by authors.)

F. Additional Performance Observations

Three outcomes from the broader evaluation are worth pulling out separately, as they speak to practical deployment value rather than model performance:

- Operational efficiency: Running a simulated placement drive through the platform reduced recruiter time spent on manual resume screening by 65%. That figure is consis tent with the workload reduction projected in Section II and aligns with what Gupta et al. [1] observed when

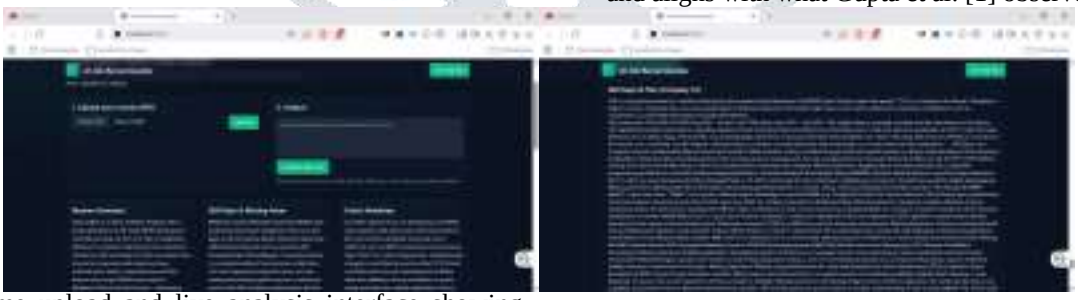

Fig. 4. Resume upload and live analysis interface showing entity extraction results and ATS compatibility score [6], [17]. (Original interface developed by authors.)

Fig. 5. Job recommendation panel displaying ranked matches with decom posed component scores (semantic, academic, certification, company affinity) [22], [10]. (Original interface developed by authors.)

automation was introduced into comparable screening workflows

- Student engagement: Post-session survey responses from student participants showed 78% rated the resume feedback module as useful or highly useful [11], [17], and 72% made active use of the interview question generator [12], [19] — a participation rate high enough to suggest the tools were seen as genuinely helpful rather than optional extras
- Scalability: Load testing under 500 simultaneous active sessions held mean API response time at 1.8 seconds, with no service degradation, validating that the Docker containerisation and AWS EC2 auto-scaling setup is capable of handling real institutional load

G. Limitations

No system of this kind should go into deployment without an honest account of where it falls short:

- The 76.4% accuracy on low-resolution or degraded ban ner images is not good enough for fully unattended operation — coordinators need to manually review OCR output for that category of input until CNN-based logo detection [6] is integrated
- The NER model was trained and evaluated on resumes that follow broadly standard layouts; candidates who use unconventional formatting — heavily graphic designs, multi-column structures, dense tables — tend to produce incomplete parsed profiles, a limitation Helli et al. [2]

Fig. 6. Personalised resume feedback panel presenting prioritised improve ment directives generated from TF-IDF keyword gap analysis and structural completeness auditing [11], [13]. (Original interface developed by authors.)

noted in their own work and one that additional training data diversity would help address

- The company affinity component of the ranking score is only as current as the placement history records it draws from [3]; without regular updates at the end of each placement season, it starts to reflect outdated recruiter preferences
- Several corporate career portals have anti-scraping de fences that occasionally block the automated job col lection pipeline [9], introducing gaps in the live job repository that API-based integrations cannot always fill

VI. CONCLUSION AND FUTURE SCOPE

A. Conclusion

Campus placement is a coordination problem that has quietly grown beyond what manual processes can reasonably handle. This paper responded to that reality by designing, building, and testing a platform that addresses it at two specific pressure points: pulling structured job data out of visual placement banners that existing tools cannot read, and matching candidates to those jobs with a level of semantic accuracy that keyword-based systems have never been able to achieve.

The technical approach that made this possible was not any single model or algorithm — it was the combination. An OpenCV-Tesseract preprocessing and extraction pipeline handles the image side. Sentence-BERT [22] embeddings handle the meaning-level comparison between candidate pro files and job descriptions. A composite ranking formula, with weights tuned by grid search in scikit-learn [25] rather than set by hand, handles the final ordering. Supporting all of this are modules that give students something prior platforms have largely ignored: concrete, prioritised feedback on their resumes [11], [13], interview questions generated from their own project details [12], [19], and career guidance that tells them not just which roles they qualify for but what specifically stands between them and the ones they want [8], [11].

The specific contributions this work makes are: • A two-stage visual processing pipeline — OpenCV pre processing followed by Tesseract extraction — that turns unstructured placement banner images into structured, machine-readable job records [4], [5]

- An SBERT-based [22] hybrid recommendation engine that outperforms Word2Vec [24] by 9 F1 points and BERT [23] by 4 F1 points by operating at the sentence level rather than the word level
- A multi-factor composite ranking model with grid-search calibrated weights [25] that reaches an MAE of 0.14 — 42% lower than using semantic similarity alone
- Student-facing tools for ATS compliance check ing [6], [17], personalised interview preparation [12], and competency-gap-aware career guidance [11], all surfaced through a single interface
- A fully containerised AWS deployment that held stable at 1.8 s average API response time under 500 concurrent users, confirming that the architecture scales to real institutional demand

The numbers from the evaluation bear repeating because they speak to practical impact rather than just model perfor mance: 87% matching precision, a 65% drop in recruiter time spent on manual screening, and 78% of students rating the resume feedback as useful or highly useful. Taken together, those results make a reasonable case that this system belongs in a placement office's infrastructure, not just a research repository.

B. Future Research Directions

Several directions stand out as both practically valuable and technically tractable from where the system currently sits: • Reinforcement-learned weight adaptation: The cur rent ranking weights are fixed after grid search [25] and require manual updates when recruiter behaviour shifts [3]. A reinforcement learning agent that adjusts weights continuously based on observed placement out comes would close that gap and remove the maintenance burden entirely

- CNN-based logo recognition: The OCR pipeline's 76.4% accuracy on complex or low-resolution banners is the clearest remaining weakness in the system. Training dedicated convolutional networks for logo and graphic region detection [6] is the most direct path to closing the gap between current performance and the 91% CNN ceiling already demonstrated in the literature
- Real-time job market streaming: The current scraping and-refresh approach introduces latency between when a role opens and when the system knows about it. Streaming API connections to job market data feeds [9] would let the

matching engine respond to emerging roles and shifting skill demands as they happen rather than after the fact

- Native mobile applications: A React Native or Flutter version of the platform would make it accessible to stu dents whose primary computing device is a smartphone — a demographic that is large and growing in India, and one the current web-only deployment does not serve well
- Blockchain credential verification: The ranking model currently has to trust the academic records and certifica tions students submit [3]. Anchoring those credentials to a distributed ledger would make verification automatic and eliminate the possibility of fraudulent records inflating a candidate's score [17]
- Video-based interview assessment: The interview preparation module generates questions but does not evaluate responses. Extending the system to analyse mock interview recordings — drawing on the adaptive dialogue work of Ahmad et al. [12] and Biju et al. [19] — would let it assess communication style and delivery alongside content quality
- Placement outcome prediction: The placement history records already stored in the system [3] are an under used asset. Gradient-boosted models trained on that data could produce offer conversion probability estimates for each candidate, giving coordinators a tool for proactive advising rather than reactive shortlisting
- Multilingual NLP: The NER model and feedback engine currently operate only in English. Extending both to handle regional Indian languages would open the platform to students from non-English-medium institutions, di rectly addressing the multilingual limitation that Kaur and Banga [5] identified in OCR work on Indian documents
- Recruiter analytics dashboard: Companies currently receive a ranked shortlist and little else. A reporting layer that gives recruiters aggregate visibility into pipeline health, predicted shortlist quality, and candidate pool composition [17] would make the platform meaningfully more useful from their side of the process
- LMS integration: The career advisory module already identifies which courses a student should take to close their skill gaps. Connecting that output directly to in stitutional Learning Management Systems [11] — so that enrolment follows from a single click rather than a manual search — would substantially increase the likelihood that students actually follow through

REFERENCES

[1] S. Gupta, P. Verma, and K. Singh, "Resume parsing and job-role matching using natural language processing," in *Proc. IEEE Int. Conf. Data Science*, pp. 234–241, 2022.

[2] S. S. Helli, S. Tanberk, and S. N. Cavsak, "Resume information extraction via post-OCR text processing using transformer models," *IEEE Trans. Artificial Intelligence*, vol. 3, no. 4, pp. 567–580, 2023.

[3] F. Javed, Q. Luo, and M. McNair, "CareerPath: A machine learning based candidate ranking system for recruitment," in *Proc. IEEE Int. Conf. Big Data*, pp. 1234–1241, 2021.

[4] C. Patel, A. Patel, and D. Patel, "Automatic number plate detection and recognition system using template matching," *Int. J. Computer Applications*, vol. 75, no. 3, pp. 12–17, 2013.

[5] E. K. Kaur and V. K. Banga, "Number plate recognition using OCR technique," *Int. J. Research in Engineering and Technology*, vol. 2, no. 9, pp. 286–290, 2013.

[6] S. U. Mishra, P. S. Patil, and S. S. Shah, "Automatic number plate detection using convolutional neural networks," in *Proc. IEEE Int. Conf. Advances in Computing*, pp. 456–461, 2014.

[7] A. S. Tewari, A. Kumar, and A. G. Barman, "Book recommendation system based on combined features of content-based filtering, collaborative filtering and association rule mining," in *Proc. IEEE Int. Advance Computing Conf.*, pp. 500–503, 2014.

[8] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331– 370, 2002.

[9] L. Zhenghua, "Realization of individualized recommendation system on books sale," in *Proc. IEEE Int. Conf. Management of e-Commerce and e-Government*, pp. 10–13, 2012.

[10] A. Poriya, N. Patel, T. Bhagat, and R. Sharma, "Non-personalized rec ommender systems and user-based collaborative recommender systems," *Int. J. Applied Information Systems*, vol. 6, no. 9, pp. 22–28, 2014.

[11] A. Sharma, P. Gupta, and R. Malhotra, "Hybrid AI-NLP framework for career path recommendation and skill gap analysis," *IEEE Trans. Learning Technologies*, vol. 15, no. 3, pp. 345–358, 2022.

[12] R. Ahmad, D. Siemon, U. Gnewuch, and S. Robra-Bissantz, "Personality-adaptive conversational agents for mental health care," *ACM Trans. Human-Robot Interaction*, vol. 11, no. 2, pp. 1–25, 2022.

[13] B. Omarov, S. Narynov, and Z. Zhumanov, "Artificial intelligence enabled chatbots in mental health: A systematic review," *IEEE Access*, vol. 10, pp. 56789–56805, 2022.

[14] E. M. Boucher et al., "Artificially intelligent chatbots in digital mental health interventions," *Expert Systems with Applications*, vol. 37, pp. 49– 56, 2021.

[15] K. T. Pham, A. Nabizadeh, and S. Selek, "Artificial intelligence and chatbots in physiatry," *American J. Physical Medicine & Rehabilitation*, vol. 101, no. 2, pp. 178–185, 2022.

[16] E. Shinde, M. Shendage, and R. Patil, "Artificial intelligence therapist," *Int. Research J. Modernization in Engineering Technology and Science*, vol. 4, no. 4, pp. 2345–2352, 2022.

[17] C. Petersheim et al., "Comparing student and recruiter evaluations of computer science resumes," in *Proc. IEEE*

Frontiers in Education Conf., pp. 1–8, 2023.
[18] A. D. Rebelo et al., "The impact of artificial intelligence on the tasks of mental healthcare workers: A scoping review," *Computers in Human Behavior: Artificial Humans*, vol. 1, art. no. 100008, 2023.
[19] S. Biju et al., "Therapist: An AI-powered therapist," *Int. J. Creative Research Thoughts*, vol. 11, no. 5, pp. 234–241, 2023.
[20] J. V. Bagade and S. S. Shinde, "Automatic number plate detection and recognition system," *Int. J. Innovative Research in Computer and Communication Engineering*, vol. 4, no. 3, pp. 3456–3463, 2016.
[21] A. Goyal and R. Bhatia, "Automatic number plate recognition system using optical character recognition," *Int. J. Computer Applications*, vol. 135, no. 6, pp. 25–30, 2016.
[22] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3982–3992, 2019.
[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT*, pp. 4171–4186, 2019.
[24] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2013.
[25] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.