International Journal of Enhanced Research in Science, Technology & Engineering
ISSN: 2319-7463, Vol. 15 Issue 10, October-2026

Skill2Career: A Mobile Application That Matches Student Skills with Jobs and Internships and Shows a Plan to Close Skill Gaps

Bindhushree S M[1], Dhanalakshmi B[2], K. Lavanya[3], Bhaskar[4],
Prof. Shubhra Prakash Puhan[5]

[1,2,3,4] Department of Artificial Intelligence and Machine Learning, AMC Engineering College, Bengaluru, Karnataka, India
[5]Professor in AMC Engineering College, Bengaluru, Karnataka, India

ABSTRACT

Finding a first job or internship is hard for students. Job websites show thousands of postings, but most students do not know which ones fit their skills or what they should learn next. This paper presents Skill2Career, a mobile application that reads a student's skills, finds job and internship postings from several sources, and ranks them by how well they fit. The match score combines skill coverage (70 percent) and text similarity using TF-IDF and cosine similarity (30 percent). The student's skill level—beginner, intermediate or expert—affects the score. The application also provides a skill-gap report, a week-by-week learning plan with course links, and optional alerts for strong matches. The described system uses React Native, Flask, Firebase and scikit-learn. The source paper reports 35 passing automated backend tests and a worked example using sample postings; these are implementation-reported results, not a real-user study. The paper also discusses limitations and future improvements.

Keywords: job recommendation, internship search, skill matching, TF-IDF, cosine similarity, skill gap analysis, mobile application

INTRODUCTION

Every year, many students in India finish their degrees and start looking for work. Online job portals have made it easy to find openings, but they have also created a new problem. A search for a single role can return thousands of results. A student must open many of them, read the long descriptions, and decide whether they are ready to apply. This takes time and can discourage first-time applicants.

A second problem comes after the search. When a student finds that a posting asks for skills they do not have, the portal rarely explains what to learn first. Students may choose courses at random and spend time on topics that few employers ask for.

Skill2Career addresses both problems in one place. The student enters skills or uploads a resume. The application lists matching jobs and internships, shows which required skills are present and missing, and gives a plan to learn missing skills in order of demand.

The main contributions of this work are:

- A clear scoring method that combines skill coverage with text similarity and gives credit according to the student's skill level.
- A skill-gap report that ranks missing skills by how often they appear in the postings found for the student.
- A week-by-week learning plan with course links and an estimate of how the plan may improve the student's match.
- A mobile application design with login, resume reading, fresher filters, job alerts, five languages and an application tracker.

The paper reviews related work, describes the system design and methods, covers implementation and reported testing, discusses limitations, and concludes with future work.

BACKGROUND AND RELATED WORK

Recommender systems

Recommender systems suggest items to a user based on information about the user and the items. Adomavicius and Tuzhilin [1] describe content-based methods, which compare item details with a user's profile; collaborative methods,

which use the choices of similar users; and hybrid methods, which combine both. The Recommender Systems Handbook by Ricci, Rokach and Shapira

[2] surveys these methods and their uses.
Collaborative methods need a history of user actions. A new application may not have this history, and a new student may not yet have clicked or applied to opportunities. Content-based methods can operate using the student's skills and the text of postings. For this reason, Skill2Career uses a content-based approach.

Text similarity with TF-IDF
TF-IDF turns text into numerical representations. A word receives a higher weight when it appears often in one document but less often in other documents. This idea relates to work on term specificity by Sparck Jones [3] and weighting schemes studied by Salton and Buckley [4]. Once documents are represented as TF-IDF vectors, cosine similarity can measure their textual closeness, as in the vector-space model described by Salton, Wong and Yang [5]. Manning, Raghavan and Schütze [6] provide a detailed explanation of these methods.

Why skill coverage is also needed
Text similarity alone can be misleading. A long posting may look like a profile because of shared general words even when the student lacks required skills. A direct count of required skills is easier to interpret. Skill2Career therefore combines both measures and gives more weight to skill coverage.

Existing job platforms
Many job websites rely on keyword search and filters such as location, salary and date. These tools help narrow a list but may not show how well a student fits a role or provide a plan to improve. Skill2Career combines search, matching, gap analysis and learning links in one application intended for students and freshers.

SYSTEM DESIGN

Overall structure
Skill2Career has three main parts. The mobile application is the user-facing interface and is written in React Native with Expo. The backend is a Python Flask server that contains matching logic and communication with other services. Firebase provides login and stores student data. The mobile application routes job-source requests through the backend so service keys can remain on the server rather than in the phone.

Table 1. Overall structure of Skill2Career.

Mobile App React Native + Expo	**Backend API** Python + Flask	**Data & Auth** Firebase
Profile, resume, jobs, skill gaps and roadmap	Matching, providers, courses and alerts	Login, profile, saved/applied jobs

Main features
- Login and profile: registration, email verification, password reset, and a profile containing education, experience, links and job preferences.
- Skills: skills can be read from a PDF or Word resume, pasted text, or manual entry. Each skill can be marked beginner, intermediate or expert.
- Jobs and internships: postings from enabled providers are ranked and shown with match scores, location information and filters.
- Skill gap and roadmap: missing skills are ranked by demand and linked to courses from learning platforms, with a phased plan and progress tracking.
- Student and fresher mode: senior roles are hidden and entry-level roles are prioritized.
- Job alerts: users can set a minimum match score, role types, locations and notification frequency.
- Application tracker: saved, viewed and applied jobs can be stored with status, notes and a timeline.
- Languages and comfort: the source paper describes English, Hindi, Kannada, Tamil and Telugu support, along with light/dark modes and an offline reading cache.

METHODOLOGY

Reading skills from text
The source paper describes a list of 103 skills, each with a standard name and alternative names. For example, “JS” is counted as JavaScript. A skill is detected when its name appears as a whole word, helping avoid errors such as detecting “Java” inside “JavaScript” and supporting terms such as C++, C# and .NET.

Match score
For one student and one posting, the final score is calculated as follows:

$$Score = 0.70 \times C + 0.30 \times T$$

Here, C is the skill coverage score and T is the text similarity score. Both are on a 0–100 scale, and the final score is limited to this range.

Skill coverage (C). Let N be the number of skills required by the posting. Each required skill the student has is counted with a weight based on proficiency: 0.6 for beginner, 0.85 for intermediate and 1.0 for expert. The sum is divided by N and multiplied by 100. When a posting has no explicit required-skill list, the system counts how many of the student's skills appear in the text and limits C to 90.

Text similarity (T). The student's profile is represented as a document. Each skill is repeated once for beginner, twice for intermediate and three times for expert. The headline and desired role are added. Each posting is represented using its title, required skills and description, with the skills repeated to emphasize them. TF-IDF vectors are built, and cosine similarity is calculated. The source paper states that the raw cosine value is multiplied by 3 and limited to 100 before combination with C.

Tie-breaking. When postings receive the same score, the posting with text closer to the student's profile is ranked first.
Skill gap report

For every posting, the system lists required skills not detected in the student's profile and counts how many postings ask for each missing skill. A skill is marked High priority if it appears in at least 40 percent of the postings, Medium if it appears in at least 20 percent, and Low otherwise. High-priority gaps can be shown first with course links.

Learning roadmap
The roadmap takes up to six missing skills in priority order. The source paper describes approximate study times of five weeks for larger topics such as Python, SQL or machine learning, two weeks for lighter tools such as Excel, Git or Power BI, and three weeks for other skills. Skills are arranged in phases of three. When a student marks a skill as learned, it is added to the profile as a beginner skill and the match scores are updated.

The system can compare the average match of the top ten postings before and after the first three skills are added at beginner level. This is an estimate based on the postings available at that time.

Fresher filter
The system uses simple rules to identify fresher-friendly and senior postings. Internships and titles containing terms such as trainee, graduate, junior or entry level are treated as fresher friendly. Titles containing senior, lead or principal, or descriptions asking for three or more years of experience, are treated as senior. In fresher mode, senior postings are hidden and entry-level postings are shown first among similar scores. Unspecified postings remain visible.

Job alerts
A background task checks students who have enabled alerts, finds new postings above the chosen minimum score, and sends notifications according to the selected frequency (instant, daily or weekly). The source paper states that alerts are sent only to verified email accounts by default.

Collecting postings from many sources
Postings are requested from enabled providers concurrently. Failed providers or providers without keys are skipped. Results are cleaned, combined and deduplicated using pandas. If all providers fail, the application displays a built-in sample set, so the student does not see a blank screen.

IMPLEMENTATION

Technology

Table 2 summarizes the technology stack described in the uploaded paper.

Layer	Technology
Mobile application	React Native with Expo SDK 54 (Android, iOS and web)
Layer	Technology
Backend	Python 3.11 and Flask 3.0.3, served with Unicorn
Login	Firebase Authentication (email/password and password reset)
Database	Firebase Fire store (profile, skills, saved, applied and viewed jobs)
Matching	scikit-learn 1.5.1 (TF-IDF and cosine similarity)

Data handling	pandas 2.2.2 (merging and removing duplicate postings)
Resume reading	PyPDF2 (PDF), mammoth (Word), pasted text
Hosting	Render for backend; Expo Application Services for Android build

Security and fair use
The uploaded paper describes backend requests authenticated with Firebase login tokens, database rules that prevent direct phone access, and job/course service keys kept on the server. Uploaded resumes are limited to 15 MB. The server is described as limiting each client to 120 requests per minute and external-service search endpoints to 40 requests per minute.

Code organisation
The backend is divided into modules for matching, skills, roadmap, experience rules, resume reading, job sources, courses, alerts and storage. The source paper states that each part has tests. When Firebase credentials are unavailable, the server uses in-memory storage for demonstrations and testing.

TESTING AND RESULTS

Automated tests
The uploaded paper reports 35 automated backend tests covering skill reading, match scoring, proficiency levels, tie-breaking, skill-gap priority, resume reading, fresher filters, alert settings, learning roadmap, rate limiting and API routes. It reports that all 35 tests pass, including a test in which all job providers fail, and the sample set is displayed.

Worked example
The paper states that a real-user study has not yet been carried out. To demonstrate ranking behaviour, the project's matching code was run on the sample set included with the project: eight postings (five jobs and three internships) and three sample profiles.

Table 3. Top result for each sample profile, as reported in the uploaded paper.

Profile	Top posting	Score	Skills still missing
A	Data Analyst	78	Statistics
B	ML Intern	72	AWS
C	Frontend Developer (React)	55	React, Redux

Profile A contains Python, SQL, Power BI, Tableau and Excel at intermediate level and seeks a Data Analyst role. Profile B contains Python, Machine Learning, NLP and Statistics at intermediate level and seeks an ML Engineer role. Profile C contains HTML, CSS and JavaScript at beginner level and seeks a Frontend Developer role.

The paper reports that unrelated postings received scores between 0 and 4. For Profile A, Statistics and Machine Learning were the most frequently requested missing skills, each appearing in three of the eight postings. A six-skill roadmap was estimated at 28 weeks. Adding the first three skills at beginner level increased the average match of the top ten postings from 24 to 38 out of 100 in the worked example.

What the example shows
The example illustrates the intended behaviour of the scoring rules: matching roles appear near the top, missing skills are listed, and the roadmap can change the calculated match. It does not prove that the system outperforms other methods. That would require more postings and feedback from real students.

Limitations
- The worked example uses only eight sample postings, so the resulting skill roadmap may include topics that are not representative of a wider job market.
- TF-IDF compares words rather than meaning. Different words describing the same skill may not be recognized as similar, and the 103-skill list may not cover every field.
- The fresher filter relies on keyword rules. Postings that do not state experience requirements may remain visible to freshers.
- The salary filter checks whether salary information is listed, because provider salary fields may be free text.
- Scanned or photo-based resumes need OCR. If the resume has no readable text, the student may paste the text or add skills manually.
- The source paper notes that one course-catalogue address may stop working, offline mode is read-only, and push notifications require a suitable phone build.
- Translations into Hindi, Kannada, Tamil and Telugu require review by native speakers, and some screens may remain in English.
- A match score is guidance only and does not predict whether an employer will shortlist a student.

Future Work

- Conduct a study with real students and measure the usefulness of top-ranked postings and time saved.
- Compare the current score with other approaches, such as word embeddings, using a larger set of postings.
- Add resume feedback to show missing skills and help students describe their projects more clearly.
- Use saved, viewed and applied interactions to improve ranking over time.
- Expand skill and language coverage and add interview preparation resources for each role.

CONCLUSION

This paper presents Skill2Career, a mobile application intended to help students and freshers find jobs and internships that fit their skills and understand what to learn next. The match score combines skill coverage with text similarity and accounts for the student's skill level. The skill-gap report and learning roadmap turn missing requirements into a structured learning plan. The uploaded paper reports that the implementation passes its automated tests and demonstrates ranking behaviour on sample postings. Further evaluation with real students and a larger set of live postings is needed to measure its practical value and improve the system.

REFERENCES

1. G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art andpossible extensions,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.
2. F. Ricci, L. Rokach and B. Shapira (Eds.), *Recommender Systems Handbook*. New York: Springer, 2011.
3. K. Sparck Jones, “A statistical interpretation of term specificity and its application in retrieval,” *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, 1972.
4. G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.
5. G. Salton, A. Wong and C. S. Yang, “A vector space model for automatic indexing,” *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.
6. C. D. Manning, P. Raghavan and H. Schütze, *Introduction to Information Retrieval*. Cambridge: Cambridge University Press, 2008.
7. F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
8. W. McKinney, “Data structures for statistical computing in Python,” in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 56–61.