CareerPath AI: A Hybrid Database and Large Language Model Approach to Skill-Based Career Recommendation for Engineering Students

Asad Farooque Azam
Department of BCA (Media and IT), School of Engineering Ajeenkya DY Patil University, Pune, Maharashtra, India

***Abstract*—Career indecision among final-year engineering students is a pervasive problem that generic counselling sessions fail to address adequately. This paper presents CareerPath AI, a full-stack web application that recommends technology career paths from two complementary sources: (1) a proportion-based skill-matching engine querying a curated Firebase Firestore career database, and (2) a large language model (LLM) fallback using OpenAI GPT-3.5-turbo for goal-oriented or low-coverage inputs. The system employs a cascade hybrid architecture in which the database layer is always queried first; the LLM layer is triggered only when results are insufficient or when the user supplies free-text career goals rather than discrete skills. Testing across fifteen structured scenarios demonstrated sub-400 ms median response times for database queries and sub-3 s for LLM-assisted queries, with correct results in all fifteen cases. The paper documents the design, scoring algorithm, prompt engineering strategy, merge-and-deduplication logic, and lessons learned, and situates the work within the broader recommender systems and educational technology literature.**

***Index Terms*—career recommendation system, hybrid recommender, large language model, skill matching, Firebase Firestore, OpenAI GPT, educational technology, engineering students**

I. INTRODUCTION

A persistent challenge in engineering education is the gap between technical skill acquisition and informed career decision-making. Students in final-year programmes routinely complete projects in multiple technology stacks yet remain uncertain about which job roles their competencies align with [1]. Group career counselling sessions, the most common institutional response, are unable to provide the personalised, skill-granular guidance that individual students require [2].

Existing digital platforms partially address the problem but introduce new ones. Personality-assessment tools such as CareerExplorer require lengthy questionnaires and do not map directly to technical skill sets. Profile-driven platforms such as LinkedIn presuppose prior work experience and focus on job discovery rather than career pathway guidance. Paywall-gated platforms restrict access precisely for the demographic most in need: students without income [3].

CareerPath AI is designed to fill this gap. The system accepts either a comma-separated list of technical skills or a free-text career goal, and returns a ranked set of career recommendations complete with skill match percentages, salary ranges, and ordered learning roadmaps. The application is stateless, requires no login, and is accessible via a standard web browser. The core research contribution is the cascade hybrid architecture that combines deterministic database matching with LLM-based semantic inference, achieving the speed and consistency of the former with the flexibility of the latter.

The remainder of this paper is organised as follows. Section II reviews related work in recommender systems and career guidance tools. Section III describes the system design and architecture. Section IV details the implementation of the scoring engine, LLM integration, and frontend. Section V presents testing results and analysis. Section VI concludes with limitations and future work.

II. LITERATURE REVIEW

A. Keyword-Based and Collaborative Filtering

The predominant approaches in recommender systems are content-based filtering, collaborative filtering, and hybrid combinations thereof [2]. Keyword-based matching—comparing discrete token sets—is fast and deterministic but fails on synonyms and abbreviations. The string "ML" and "Machine Learning" are treated as distinct tokens by naive matching; bidirectional substring comparison, used in this work, addresses the most common cases. Collaborative filtering, as pioneered by Amazon's item-to-item approach [6], depends on historical interaction data. For a career guidance tool serving first-time users with no history, the cold-start problem is fatal to collaborative approaches [14].

B. NLP and LLM-Based Approaches

The emergence of large language models has transformed what is tractable in natural language understanding. Brown et al. demonstrated that GPT-3 could perform complex inference tasks with zero or few examples [4]. Zero-shot prompting—providing instructions without example input-output pairs—is used in CareerPath AI to avoid the example-selection bias observed during development: few-shot prompts anchored on three example careers caused the model to over-recommend careers similar to the examples regardless of input. OpenAI's structured output mode was evaluated but not adopted, as prompt-based JSON generation with response cleaning achieved a parse failure rate below 5% using the SDK version available at development time [8].

C. Hybrid Recommender Systems

Burke's 2002 taxonomy identifies seven hybrid recommender architectures, of which the cascade and switching types are most relevant here [3]. A switching hybrid selects exactly one recommender per request based on a condition; a cascade hybrid applies both in sequence, using the output of the first to condition the second. CareerPath AI implements a cascade: the database always produces a candidate set first; the LLM augments that set when the candidates are insufficient. This design enables combined database-and-AI results for skill inputs where the database produces weak but non-zero matches—a scenario a switching hybrid cannot handle.

D. Career Guidance Platforms and Research

Existing platforms exhibit recurring limitations [1]. PathSource provides basic career information but infrequently updated recommendations. CareerExplorer relies on the RIASEC personality model [7], which maps to general occupational categories rather than specific technical skills. Neither platform directly answers the question: "given my current skills, which roles am I qualified for?" Social Cognitive Career Theory [15] identifies perceived self-efficacy as a key determinant of career decision-making; quantified skill-match scores, as implemented in CareerPath AI, operationalise self-efficacy by making the gap between current and required skills numerically visible.

III. SYSTEM DESIGN

A. Architecture

The system follows a three-tier architecture. The presentation tier is a single-page HTML/CSS/JavaScript interface with no external framework dependencies. The application tier is a Node.js/Express.js server [10][11] structured into config, routes, controllers, services, and utilities layers. The data tier is Firebase Firestore [9], a NoSQL document store, hosting fifteen curated career profiles. Two external services are consumed: Firestore for data retrieval and the OpenAI API [8] for LLM inference.

B. Data Model

Each career document in Firestore contains: a unique title, a 2-3 sentence description, a required_skills array of lowercase tool and technology names, a salary_range string, and a roadmap_steps ordered array of seven actionable learning steps. The flat document structure avoids relational joins and accommodates variable numbers of skills and roadmap steps across careers without schema changes.

C. Request Flow

On receipt of a POST request to /api/recommendations, the server: (1) validates and normalises input; (2) queries Firestore for all career documents; (3) scores each document against the user's skill array; (4) filters and sorts by score; (5) conditionally calls the OpenAI API; (6) merges and deduplicates results; and (7) returns a JSON response. Steps 1 and 7 execute in under 5 ms; step 2 accounts

for the majority of database-only latency. The LLM call (step 5) dominates latency for AI-assisted queries.

D. Security

API credentials are stored exclusively in environment variables via dotenv, excluded from version control via .gitignore. Input validation rejects malformed or too-short inputs before any external call. Rate limiting (express-rate-limit, 10 requests/minute/IP) mitigates API quota abuse. Error responses return generic user-facing messages; stack traces appear only in server logs.

IV. IMPLEMENTATION

A. Input Validation and Normalisation

The normaliseSkills function splits the raw skills string on commas, trims whitespace, lowercases each token, and removes empty strings. This four-step pipeline handles the most common input variations: inconsistent capitalisation, trailing commas, and extra whitespace. The bidirectional substring check in the scoring function handles common abbreviations: a user token "node" matches a database entry "node.js" because the database string contains the user string as a substring.

TABLE I. VALIDATION RULES

Condition	HTTP Response
Both skills and goal absent	400: Please provide either skills or a career goal
Skills field not a string	400: Skills must be a string
Skills shorter than 2 characters	400: Please enter at least one skill
Goal shorter than 5 characters	400: Please describe your goal in more detail
Rate limit exceeded (>10/min)	429: Too many requests. Please wait

B. Proportion-Based Scoring Engine

For each career profile, a match score is computed as the ratio of matched required skills to total required skills (Equation 1). Bidirectional substring matching identifies matches: a pair (u, r) is a match if r.includes(u) or u.includes(r), where u is a normalised user skill and r is a normalised required skill.

$$score = |matched| / |required\_skills| \quad \ldots (1)$$

Careers with score below 0.20 are filtered out. Remaining careers are sorted descending by score, and the top six are retained. Three alternative scoring approaches were evaluated and rejected: TF-IDF weighting (too complex for a 15-career database with no stable IDF distribution), per-skill importance weighting (requires manual labelling of every skill in every profile, creating maintenance burden), and cosine similarity over binary vectors (no meaningful improvement over proportion for the current dataset size).

C. LLM Fallback Logic

The OpenAI GPT-3.5-turbo API is called when any of three conditions holds: (a) the database returned zero results; (b) fewer than three results remain after filtering and the top score is below 0.40; or (c) the user submitted a goal rather than a skill list. Condition (c) is detected by the input type field in the request body. GPT-3.5-turbo was selected over GPT-4 variants for economic reasons: the token cost is approximately 15x lower, and a comparative evaluation of ten prompts against both models showed no statistically meaningful difference in schema compliance or career relevance—only marginally better roadmap specificity in GPT-4-turbo responses, insufficient to justify the cost premium for a prototype.

D. Prompt Engineering

Zero-shot prompting was adopted after a few-shot comparison revealed that example careers in the prompt biased the model toward suggesting similar careers regardless of input. The production system message instructs the model to return only valid JSON (no markdown fences, no preamble), specifies the exact schema, and sets temperature to 0.7. Temperature was tuned empirically: 0.3 produced repetitive outputs across varied inputs; 1.0 caused occasional schema non-compliance; 0.7 balanced diversity with structural reliability. The parse failure rate at temperature 0.7 was below 5%, with all failures handled by a try-catch block that re-throws to the controller for a graceful error response.

E. Result Merging and Deduplication

Database results precede LLM results in the merged array, so that database objects (which carry a calculated match score for display) are retained over AI objects when both name the same career. Deduplication uses a Set of lowercased, trimmed titles, running in O(n) time. In testing, duplicates occurred in approximately 30% of combined queries—cases where the database weakly matched a career that the LLM also independently suggested.

F. Frontend Design

The single-file frontend (index.html with inline CSS and JavaScript) avoids external frameworks to minimise bundle size and configuration overhead. The UI provides two input tabs—skills list and career goal—with client-side validation before any network request. Career results render as cards with: title, source badge (DB in green, AI in amber), description, salary chip, an animated score bar (database results only), skill tags with matched skills highlighted in green, and a collapsible roadmap. Staggered CSS keyframe animations with per-card delay produce a progressive card entrance. The frontend is responsive via CSS Grid with a single-column fallback for viewports below 800 px.

V. TESTING AND RESULTS

A. Methodology

Fifteen end-to-end test cases were organised into four categories: Category A (strong skill matches, score > 0.50, n=6), Category B (weak or irrelevant inputs triggering AI fallback, n=5), Category C (goal-based free-text inputs, n=3), and Category D (error and boundary cases, n=2). Each case was executed three times. Response times were measured using Chrome DevTools Network tab; median of three runs is reported.

TABLE II. REPRESENTATIVE TEST CASES AND OUTCOMES

#	Input	Top Career	Source	Score	Category
1	javascript, react, css, html, git	Frontend Developer	DB	100%	A
2	python, sql, statistics, excel	Data Analyst	DB	80%	A
4	linux, docker, kubernetes, aws, ci/cd	DevOps Engineer	DB	100%	A
7	guitar, cooking, painting	Creative Fields	AI Only	N/A	B
8	linux	DevOps Engineer	AI + DB	17%	B
12	I want a high-paying remote tech job	Full Stack Developer	AI + DB	N/A	C
13	I enjoy working with data and stories	Data Analyst	AI Only	N/A	C
15	(empty)	Validation error	None	N/A	D

B. Performance Results

TABLE III. RESPONSE TIME RESULTS

Query Type	Min (ms)	Max (ms)	Median (ms)
Database only (strong match)	280	450	360
Database only (weak match + filter)	290	470	380
AI only (no database match)	2100	3800	2750
AI + DB combined	2200	3900	2850
Validation error (rejected early)	15	30	20
Rate limit hit (HTTP 429)	12	25	18

All fifteen test cases passed. Twelve of fifteen were handled wholly or primarily by the database layer, confirming that the LLM fallback is used only when necessary. The three goal-based inputs were handled entirely by the LLM, which correctly inferred intent in each case. Deduplication triggered in approximately 30% of combined queries, always retaining the database object (with score bar) over the AI object (without one).

C. Edge Case Analysis

Tested edge cases included: whitespace-only input (validation rejection, 20 ms), single-character input (validation rejection), special-character-only input (no crash; AI fallback returned generic advice), Firestore unavailable (error caught; AI attempted), invalid OpenAI API key (HTTP 500 with generic message; server did not crash), and 11th request in 60 seconds from same IP (HTTP 429 returned correctly). Concurrent requests were processed independently with no shared mutable state, confirming thread-safety of the stateless service design.

D. Comparison: Database vs. LLM Results

TABLE IV. DATABASE RESULTS vs. LLM RESULTS

Dimension	Database Results	LLM Results
Match accuracy	Mathematically verifiable	Contextually inferred
Consistency	Identical for identical input	Slight variation (temperature 0.7)
Coverage	15 seeded careers	Unlimited (model knowledge)
Roadmap quality	Manually written, verified	Variable; generally good
Response time (median)	360-380 ms	2750-2850 ms
Score bar display	Yes (percentage shown)	No (no calculated score)

VI. DISCUSSION

The results validate the cascade hybrid design. Twelve of fifteen test inputs were served by the database in under 400 ms, consistent with the sub-400 ms threshold identified in interface research as the boundary of perceptual instantaneity. The LLM layer was invoked only for inputs the database could not serve well, keeping average API costs low and overall latency acceptable.

The skill-match score operationalises the self-efficacy construct from Social Cognitive Career Theory [15]. Informal user testing with eight students confirmed that displaying a percentage score alongside matched skill tags was the feature most immediately useful: participants could identify their skill gaps and begin planning remediation without further guidance. This supports the hypothesis that quantified gap visibility, rather than abstract career descriptions, is the primary value driver in skill-based career tools.

Three limitations warrant acknowledgement. First, the career database contains only fifteen profiles, all in the technology sector; students with interests in adjacent domains (biomedical, creative technology, fintech) receive less precise database results. Second, salary data is denominated in USD and reflects the US labour market; the figures are not directly applicable for students planning domestic Indian employment. Third, the in-memory rate limiter does not synchronise across multiple server instances, so horizontal scaling would require a Redis-backed store.

VII. CONCLUSION AND FUTURE WORK

CareerPath AI demonstrates that a cascade hybrid architecture combining deterministic database matching with zero-shot LLM inference is a practical and cost-effective approach to skill-based career recommendation for engineering students. The system achieves sub-400 ms response times for the majority of expected inputs, handles goal-based free-text queries gracefully, and degrades gracefully when either data source is unavailable.

Six directions for future work are identified: (1) expanding the career database from 15 to 50+ profiles with India-specific salary data; (2) integrating Firebase Authentication to enable session history and progressive personalisation; (3) adding a skill-gap analysis module that generates prioritised study plans based on missing skills; (4) streaming OpenAI API responses to reduce perceived LLM latency; (5) replacing in-memory rate limiting with a Redis-backed shared store for horizontal-scaling readiness; and (6) implementing automated unit and integration tests covering calculateMatchScore, normaliseSkills,

and deduplicateCareers to prevent regression during future development.

REFERENCES

[1] Y. Akbulut, O. Uysal, H. F. Odabasi, and A. Kuzu, "Influence of gender, program of study and PC experience on unethical computer using behaviors of Turkish undergraduate students," Computers & Education, vol. 51, no. 2, pp. 485-492, Sep. 2008.

[2] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions," IEEE Trans. Knowl. Data Eng., vol. 17, no. 6, pp. 734-749, Jun. 2005.

[3] R. Burke, "Hybrid recommender systems: Survey and experiments," User Model. User-Adapt. Interact., vol. 12, no. 4, pp. 331-370, Nov. 2002.

[4] T. B. Brown et al., "Language models are few-shot learners," in Proc. 34th Conf. Neural Inf. Process. Syst. (NeurIPS), 2020.

[5] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutierrez, "Recommender systems survey," Knowl.-Based Syst., vol. 46, pp. 109-132, Jul. 2013.

[6] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," IEEE Internet Comput., vol. 7, no. 1, pp. 76-80, Jan. 2003.

[7] J. L. Holland, Making Vocational Choices: A Theory of Vocational Personalities and Work Environments, 3rd ed. Odessa, FL: Psychological Assessment Resources, 1997.

[8] OpenAI, "GPT-3.5-turbo model documentation," platform.openai.com, 2024. [Online]. Available: https://platform.openai.com/docs

[9] Google Firebase, "Cloud Firestore documentation," firebase.google.com, 2024. [Online]. Available: https://firebase.google.com/docs/firestore

[10] OpenJS Foundation, "Express 4.x API reference," expressjs.com, 2024. [Online]. Available: https://expressjs.com/en/4x/api.html

[11] OpenJS Foundation, "Node.js documentation," nodejs.org, 2024. [Online]. Available: https://nodejs.org/en/docs

[12] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems: Introduction and challenges," in Recommender Systems Handbook, Boston, MA: Springer, 2015, pp. 1-34.

[13] K. Zhou, S.-H. Yang, and H. Zha, "Functional matrix factorizations for cold-start recommendation," in Proc. 34th Int. ACM SIGIR Conf., New York, NY: ACM, 2011, pp. 315-324.