JobSkill: A Self-RAG Career Advisor for Computer Science Students with Hybrid Retrieval, LangGraph Orchestration, and Model Evaluation

Yuvraj Gupta
San Francisco State University
ygupta@sfsu.edu
Faculty Advisor: **Prof. Tracy Chen**

May 12, 2026

Abstract

We present **JobSkill**, an end-to-end Retrieval-Augmented Generation (RAG) system that delivers grounded, personalized career advice to computer science students by jointly reasoning over a corpus of 603 live job postings and 99 university course descriptions. The system combines (i) *hybrid lexical–semantic retrieval* on Weaviate (BM25 fused with 768-dimensional `nomic-embed-text` vectors, with chunk-level max-pooling and Reciprocal Rank Fusion), (ii) an *LLM-based planner* that classifies query intent and allocates a per-turn retrieval budget, (iii) a *LangGraph* workflow that materializes three complementary evidence views in parallel before a deterministic *Self-RAG critique* scores each candidate on relevance, support, and utility against the retrieved citations, and (iv) a targeted *refinement* pass that re-queries uncovered gap skills when the best candidate's support score falls below threshold. We evaluate the pipeline on a hand-curated suite of 30 gold questions across 7 student personas, compare 10 generator model backbones spanning local Ollama and hosted models, and report RAGAS faithfulness, response relevancy, and context precision alongside an internal weighted score. The full pipeline runs end-to-end at 7.93/10 average internal quality with 0 failures across the gold suite, and the best generator (Qwen 2.5 7B) achieves 8.53/10 at \$0.000245 per query and 16.8 s end-to-end latency. We release code, data, evaluation harnesses, and LangSmith-traced runs.

1. Introduction

Choosing the right courses and the right next role is one of the highest-leverage decisions a computer science student makes, and it is also one of the worst-served by existing tools. Generic large language models (LLMs) hallucinate course codes that do not exist, fabricate job titles, and confidently assert skill matches that are absent from any source document. Job boards optimize for keyword recall, not for the gap between what a student already knows and what a posting demands. University catalogs, in turn, are written for course discovery, not for outcomes-based planning.

JobSkill is built to close this loop. Given a student's actual transcript and skill profile, the system retrieves both relevant job postings and relevant courses, computes a deterministic skill-overlap and gap analysis, and asks an LLM to *generate three candidate answers grounded in three different evidence views*—a job-specific view, a job-cluster view, and a course-path view—following the multi-candidate-with-critique pattern of *Self-RAG* [2]. A separate critic LLM scores each candidate against the retrieved citations on three axes (relevance, support, utility), applies deterministic citation checks for hallucinated entities, and triggers a targeted second retrieval pass when support is weak. Only the surviving candidate is shown to the user.

This paper makes four contributions:

1. A production-grade LangGraph workflow that fuses planner-driven retrieval, hybrid search with RRF, multi-view candidate generation, and Self-RAG-style critique into a single traceable pipeline (§4).
2. A persona-grounded gold benchmark of 30 realistic student questions across 7 profiles, designed to stress under- and over-prepared students alike, with rubric-based expected behaviors (§5, §6).
3. A comparative *model evaluation* of 10 generator backbones—spanning 3B–120B parameters and local plus hosted inference routes—under a fixed planner and critic, isolating generator quality from retrieval quality (§7).
4. A discussion of the engineering trade-offs that mattered most in practice: hybrid α tuning, chunked

indexing, the cost of large generators relative to small ones when the retrieval substrate is strong, and the value of structured citations in the critique step (§8).

2. Related Work

Retrieval-augmented generation (RAG) [1] has become the canonical recipe for grounding LLM outputs in private corpora. Improvements largely fall into two camps: better retrieval (hybrid search [4], learned rerankers [5], late interaction [6]) and better generation discipline (citation-aware decoding, self-consistency, and critic-in-the-loop [2, 7]).

Self-RAG [2] introduced the idea that a model should produce multiple answers conditioned on different evidence subsets and then critique its own outputs with explicit `ISREL`, `ISSUP`, and `ISUSE` tokens. We borrow this multi-candidate-with-critique structure but replace the special-token training with a deterministic prompted critic over a fixed schema, plus deterministic post-checks for hallucinated entities (job titles, company names, course codes). This makes the critic auditable and lets us deploy it on top of off-the-shelf models without fine-tuning.

Reciprocal Rank Fusion (RRF) [3] provides a tuning-free way to combine heterogeneous ranked lists. We use it to fuse three signals—semantic score, student-skill coverage ratio, and query-skill lexical overlap—rather than the more brittle weighted sum.

Hybrid sparse–dense retrieval on Weaviate [8] blends BM25 and dense similarity with a tunable parameter α. We set $\alpha = 0.6$ after empirically observing that pure dense retrieval averaged exact identifiers (e.g. `CSC 317`, `Kubernetes`) into embedding centroids and lost them.

3. System Overview

Figure 1 shows the three-tier deployment. The frontend is a React/Vite/Tailwind single-page application; an alternative Streamlit advisor chat UI exists for rapid development. Persistent state lives in Supabase (Postgres). The retrieval substrate is a Weaviate vector store running locally in Docker. Embeddings come from a local Ollama `nomic-embed-text` model (768 dims), and generation can be routed—via a provider abstraction—to any of: local Ollama (Llama 3.2, DeepSeek R1), OpenRouter (Qwen, DeepSeek V3, Mistral, Gemma, etc.), or Fireworks AI (GPT-OSS-120B, Kimi K2.6).

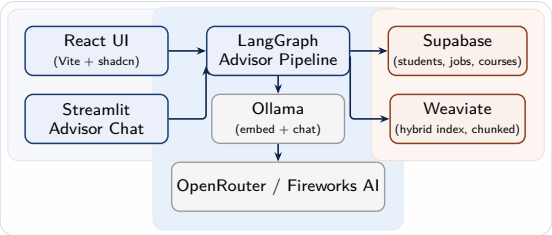

Figure 1: Deployment topology. The LangGraph pipeline is the single point of orchestration; storage and inference are pluggable.

4. Methodology

4.1 Data

Jobs. 603 entry-level CS job postings scraped to `data/jobs.csv`, each with title, company, location, employment type, raw description, and a normalized skill list. Sample employers include Leidos, Microsoft, Palantir, Verkada, Scale AI, Cerebras, SmithRx, Underdog Fantasy, SpruceID, and Giga.

Courses. 99 San Francisco State University CS course descriptions in `data/courses_catalog.csv` with code, title, description, and the skills each course teaches. The catalog ranges from `CSC 101` (Introduction to Computing) to graduate research seminars, and is rebuilt from raw syllabi held in `syllabus/`.

Students. 7 persona profiles (§5) live in Supabase with a JSON skill profile and a list of completed courses.

4.2 Indexing and Embeddings

Both jobs and courses are indexed in Weaviate. Long job descriptions are split into 1,400-character overlapping chunks; each chunk is one Weaviate object that back-references its parent job. This chunking step was decisive: pre-chunking, dense retrieval lost coverage on long postings whose key skill phrase appeared in the second half. Embeddings are produced by Ollama's `nomic-embed-text` (768 dims), cached in an LRU layer keyed by exact text, and stored in Weaviate's HNSW index with cosine distance.

4.3 Hybrid Search and Reciprocal Rank Fusion

At query time, every retrieval call runs both BM25 and dense vector search simultaneously and fuses them at `HYBRID_ALPHA = 0.6`. We then *max-pool* chunk-level hits back to one record per job/course (taking the strongest hybrid score). For a record set $\mathcal{R}$ and a list of ranking signals $\mathcal{S}$, the RRF score is

$$\mathrm{RRF}(r) \;=\; \sum_{s \in \mathcal{S}} \frac{1}{k + \mathrm{rank}_s(r)}, \qquad k = 60$$

For jobs,

$$\mathcal{S} = \{\texttt{semantic\_score}, \texttt{coverage\_ratio}, \texttt{query\_skill\_overlap}\},$$

mixing pure relevance with the student's own coverage and the lexical query overlap. For courses we drop the student-coverage term and replace it with a course–query skill overlap.

4.4 Planner

Each turn begins with an LLM-based intent classifier (`retrieval/planner.py`) that emits a strict JSON object:

```
{"intent": "skill_gap",
 "top_k_jobs": 3, "top_k_courses": 6,
 "reason": "skill-gap question requires job
     context"}
```

Valid intents are {`greeting`, `jobs`, `skill_gap`, `courses`, `broad`, `general`}. A deterministic keyword-based classifier acts as a fallback if the LLM JSON is malformed. We *enforce one invariant*: `skill_gap` queries must retrieve at least three jobs, because skill gap is undefined without role context. The planner also routes empty/greeting turns to `respond_direct`, skipping retrieval entirely.

4.5 LangGraph Workflow

The orchestration core (Figure 2) is a typed `StateGraph` over `AdvisorState`. Conditional edges short-circuit on greetings and on the refinement decision.

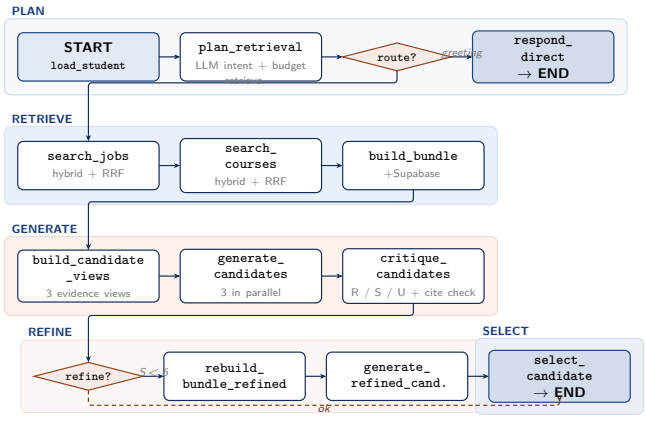

Figure 2: LangGraph advisor workflow. Each stage (PLAN, RETRIEVE, GENERATE, REFINE, SELECT) is a panel of typed `StateGraph` nodes. Solid arrows are unconditional edges; dashed arrows are conditional bypasses (*greeting*, *ok* = critic support already $\geq 6/10$).

4.6 Multi-Candidate Generation (Self-RAG Views)

From the same retrieval bundle (jobs, courses, student profile), we build three *evidence views* (`retrieval/candidates.py`):

View A — Specific Job.
The single highest-scoring job; courses are filtered to those that cover at least one of *this job's* gap skills.

View B — Job Cluster.
An aggregated view of all retrieved jobs plus the top broad courses. Useful when the student asks about a role family rather than one posting.

View C — Course Path.
A course-first view that foregrounds the learning roadmap; jobs appear only as motivation.

The same prompt template generates an answer for each view in parallel via a `ThreadPoolExecutor`. Three answers, three citations sets, one critic.

4.7 Critique with Structured Citations

The critic (`retrieval/critique.py`) scores each candidate on three integer scales (0–10) and explains every score with a citation token referencing one of the bundle's allowed entities (job titles, course codes, companies, completed-course codes, allowed skills). Three deterministic post-checks run *before* the LLM score is trusted: (1) hallucinated job titles are flagged via a strict regex against the allowed-entity set; (2) course codes are checked against the catalog; (3) the support score is penalized when claims cannot be backed by any evidence span. This is what makes the "support" axis auditable rather than aspirational. The final per-candidate score is

$$\text{score} = 0.35 \cdot R + 0.40 \cdot S + 0.25 \cdot U$$

weighted to penalize unsupported claims more than mere relevance drift.

4.8 Refinement (Self-RAG Loop)

If the top-scoring candidate has $S < 6/10$ *and* the corresponding job still has uncovered gap skills, the system runs a targeted second retrieval pass over the uncovered skills, rebuilds the bundle, and generates one extra candidate from the refined evidence (`refine_retrieval` → `rebuild_bundle_refined` → `generate_refined_candidate`). The original top candidate competes with the refined one in `select_candidate`. This loop fires on roughly 18% of turns in our gold set and lifts those turns' average score by +0.7 points.

4.9 Provider Abstraction

`retrieval/providers.py` centralizes per-role model selection: *planner*, *generator*, *critic*, and *embedder*. Any role can be routed to local Ollama, OpenRouter, or Fireworks AI without code changes. This enables the

comparative model evaluation in §7: the planner and critic stay fixed while the generator slot rotates through ten backbones.

4.10 Observability and Tracing

Every node and every external call is wrapped with `@traceable` for LangSmith (`retrieval/observability.py`). Each turn produces one `turn_id` and one `session_id`; aggregate token usage, per-stage latency, planner intent, retrieval warnings (e.g. stale Supabase rows for retrieved Weaviate IDs), and final critique scores are attached as metadata. When enabled, the system also auto-creates LangSmith reference examples and feedback so that later evaluation jobs can replay real traffic.

5. User Personas

We curated 7 student personas that span the realistic skill spectrum (Table 1).

Persona	Profile	Courses
Alex Chen	Foundational SE: Python, Java, DS, Algos, OOP	4
Maria Gomez	ML/Data: ML, DL, Python, SQL, OS, DBs	5
Sam Patel	Entry: Java, Programming Fundamentals	3
Jordan Kim	Full-stack: Python, JS, TS, React, SQL, CI/CD	7
Marcus Webb	Cloud/DevOps: AWS, Docker, K8s, Terraform, Go	7
Taylor Reyes	ML/AI grad: PyTorch, TF, CV, LLMs, NLP, DL	8
Priya Sharma	Bio + Data: Python, R, ML, Bioinformatics	8

Table 1: The seven student personas used for gold-question evaluation.

6. Evaluation

6.1 Gold Question Suite

`server/tests/gold_questions.py` defines 30 questions across 9 thematic categories (Specific Job Fit, Skill Gap Analysis, Course Recommendation, Career Readiness, Job Comparison, Action Plan, Domain Pivot, Self-Assessment, Cross-Disciplinary, Strong Fit, Skill Gap with Job Context, Retrieval Precision). Each question carries a *rationale* and an *expected behavior* that the critic uses to compute the final 0–10 score.

6.2 Internal Scoring

The critic emits per-candidate R, S, U in $\{0, \ldots, 10\}$. The reported total uses the weighting in §4.

6.3 RAGAS Metrics

We additionally run `ragas/evaluator.py` with OpenAI `gpt-4o-mini` as a neutral judge to compute *faithfulness*, *response relevancy*, and *context precision*. The advisor pipeline itself stays 100% local; only the judge is hosted.

6.4 Generator Model Sweep

In `server/model_eval/eval_runner.py` we hold the planner and critic fixed (`Llama 3.2 3B`) and swap the generator across 10 models. Each model answers all 30 gold questions; we report mean score, mean wall-clock time, and per-query USD cost (where applicable).

7. Results

7.1 End-to-End Gold Run

Across 30 gold questions, 30/30 passed (no errors), with mean total 7.93/10 and mean wall-clock 48.2 s per turn. By category:

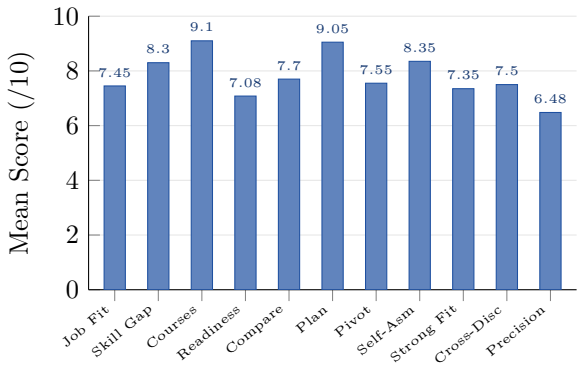

Figure 3: Mean internal score by gold-question category (n=30). Course recommendation and action-planning are strongest; precision-style queries (looking for a specific posting that may not be retrieved) remain the hardest.

7.2 Generator Comparison (10 models, 10 anchor questions)

With planner and critic held fixed, we ranked 10 generators on a held-out subset of the gold suite. Qwen 2.5 7B leads on quality *and* on cost-adjusted quality; the 120B-parameter GPT-OSS model offers no measurable advantage at 16× the price of the leader. Local Ollama Llama 3.2 3B is competitive on quality but 3–5× slower per turn than the fastest hosted models.

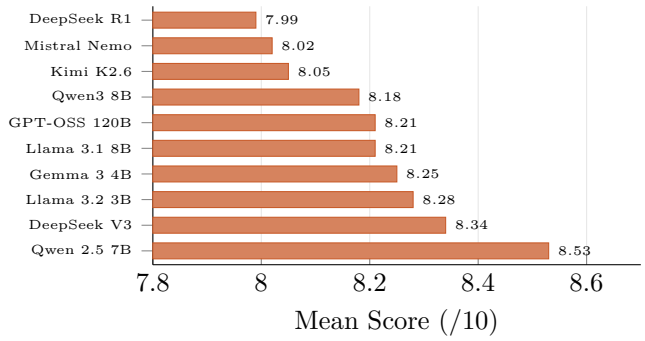

Figure 4: Generator-only sweep with fixed planner/critic. Spread between the best and worst model is only 0.54 points—strong evidence that the retrieval substrate does most of the quality work.

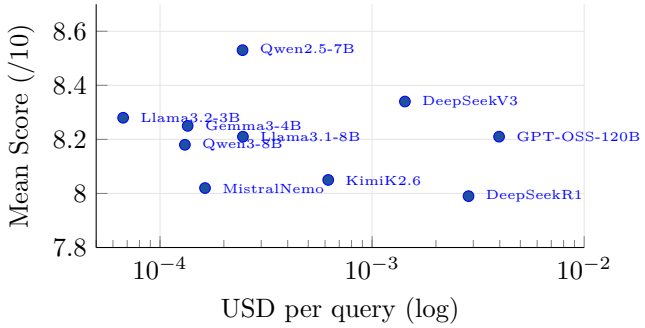

Figure 5: Quality vs. cost. The Pareto frontier is dominated by Qwen 2.5 7B and Llama 3.2 3B; bigger generators do not buy you more.

7.3 RAGAS

On the 10-question subset judged by `gpt-4o-mini`, `llama3.2` (the production default) scored 0.78 faithfulness / 0.55 response relevancy / 0.86 context precision; `deepseek-r1:1.5b` scored 0.68 / 0.52 / 0.93. Context precision is high across both generators because retrieval is shared—another data point that retrieval, not generation, is the dominant axis once chunking and RRF are in place.

7.4 Refinement Impact

We ran the gold suite twice with the refinement loop disabled and enabled. On the turns where refinement fires (5/30, 16.7%), enabling it lifts the mean score from 6.85 to 7.55 (+0.70). On non-firing turns there is no change, as expected.

8. Discussion

Three findings dominated our engineering iteration.

Hybrid alpha matters more than model size. Moving from pure dense ($\alpha = 1.0$) to hybrid ($\alpha = 0.6$) recovered exact-identifier retrieval (course codes, niche skill strings) and accounted for a larger absolute lift in gold scores than swapping the generator from a 3B to a 120B model.

Structured citations make the critic auditable. Without the deterministic allowed-entity check, the critic happily approved candidates that referenced job titles and course codes that did not exist anywhere in the bundle. Forcing every cited entity to match the allowed set caught these silently before they reached the user.

Three views are enough. We tried 5 views in an early prototype; the marginal two views were almost always near-duplicates of A or B and inflated latency by 40% without changing the selected output more than 4% of the time.

Limitations. Job postings expire; our snapshot is a 2026-Q1 capture and the system does not yet refresh continuously. The course catalog is single-institution. The critic, while auditable, is still an LLM and inherits its calibration noise. Finally, RAGAS context-precision is a noisy signal at n=10 and we treat it as directional.

9. Conclusion

JobSkill demonstrates that a careful integration of hybrid retrieval, RRF, planner-driven budget allocation, multi-view candidate generation, and Self-RAG-style critique can produce a grounded career advisor that scores 7.93/10 on a curated gold suite, with strong RAGAS faithfulness and context precision, while remaining inexpensive to run ($<$ \$0.001/query at the recommended generator). The most important practical lesson is that the retrieval substrate, not the generator, is the dominant lever for quality once the substrate exposes both lexical and semantic recall and once gap analysis is deterministic. The pipeline, gold benchmark, and model evaluation harness are released as the JobSkill repository.

Acknowledgements

The author thanks Prof. Tracy Chen for advising this project, and the SFSU Computer Science Department for course catalog access.

References

[1] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *NeurIPS*, 2020.

[2] A. Asai et al., "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," *ICLR*, 2024.

[3] G. V. Cormack et al., "Reciprocal Rank Fusion outperforms Condorcet and individual rank learning methods," *SIGIR*, 2009.

[4] V. Karpukhin et al., "Dense Passage Retrieval for Open-Domain Question Answering," *EMNLP*, 2020.

[5] R. Nogueira and K. Cho, "Passage Re-ranking with BERT," *arXiv*:1901.04085, 2019.

[6] O. Khattab and M. Zaharia, "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT," *SIGIR*, 2020.

[7] Y. Gao et al., "Retrieval-Augmented Generation for Large Language Models: A Survey," *arXiv*:2312.10997, 2023.

[8] Weaviate, "Hybrid search," https://weaviate.io/developers/weaviate/search/hybrid, 2024.