# Project Discussion

We built a resume-to-job recommender with a React/Vite front end and FastAPI back end. The pipeline ingests a PDF/DOCX resume, parses sections and skills, fetches postings from RapidAPI's JSearch feed, and ranks them with a hybrid scorer (backend/nlp_model_stub.py).

## Dataset & EDA

Jobs are fetched on demand via JSearch (LinkedIn/Indeed aggregate) using the API contract in resume-recommender-docs/03_backend_api_skeleton_v4.md. Each query hits multiple pages (job_fetcher.py, MAX_PAGES=3) and dedupes on (title, company). The API natively filters by role and location, so the dataset spans diverse geos (remote + user-specified cities) and roles. We cache one run in backend/cache.json for offline analysis, for example: a Washington, D.C. "Data Scientist" slice with 10 postings, average description length 3,834 characters (361–7,732), mean score 0.63 (0.35–0.80). Skills co-occur around Data Analysis, Machine Learning, Python, and SQL (4/10 each). Future EDA will aggregate multiple live pulls (different roles/geos) to quantify skill drift and balance remote/on-site splits.

## Engineering Decisions

Resume parsing uses pdfplumber/python-docx plus heuristic sectioning with keyword fallback (resume_parser.py). A curated skill dictionary with aliases normalizes variants. API results are deduped, retain API-side location filtering (fixing "VA" vs. "Virginia" misses), and receive skill extraction via regex over the shared dictionary (extract_job_skills_from_list.py). Uploads land in temp files that are deleted post-parse; results cache to backend/cache.json for "load more" reuse.

## Algorithms

Baseline: TF–IDF cosine between resume and job descriptions (tfidf_matcher.py). The final hybrid model (nlp_model_stub.py, summarized in the Group 8 deck) layers:

- **Resume parsing & intent**: ResumeParser extracts sections, skills, and infers target roles from summary/experience (with keyword fallbacks) to seed intent matching.
- **Job normalization**: Each JD is skill-tagged via regex over the shared dictionary; results are deduped by title+company and carry API-side geo filtering.
- **Five weighted factors**: (1) skill overlap (40%, capped denominator to avoid long JD penalty), (2) boosted TF–IDF semantic similarity (25%), (3) role intent match (15%), (4) experience fit with "no preference" guard (10%), (5) location match with state-name abbreviation bridging and remote override (10%). Scores are clipped to [0,1].
- **Explainability**: For each recommendation we surface a brief summary and top overlapping skills; caching supports "load more" without recomputing.

Weights were tuned after class feedback to privilege explicit skills and reduce noisy semantic dominance.

## Evaluation

No labeled judgments yet. Current evaluation is pragmatic and mirrors the demo in the final presentation: (1) inspect score spread and factor contributions in logs during live queries, (2) eyeball top-k results in the deployed UI for overlap/intent plausibility, and (3) sanity-check caching/pagination via /match/more. In the cached slice, skill-dense cybersecurity roles surface near 0.8 while generic posts fall below 0.4, matching qualitative expectations. Next steps from the deck/docs: gather a small set of human preference labels to compute MAP@n/precision@k, add regression tests with synthetic resumes to catch parsing/sectioning regressions, and log per-factor weights to detect drift as dictionaries or prompts evolve.

## Adaptation to Feedback

Changes driven by class reviews: true multipart upload handling, reliance on API filtering instead of brittle substring checks, state-abbreviation matching for locations, higher skill weight with a cap on TF–IDF boost, and deduping keys to avoid repeated jobs.

## Results & Analysis

The deployed site (React/Vite + FastAPI) demonstrates end-to-end matching and deployment (per the final deck): resume upload, live API fetch, hybrid scoring, and paginated display with keyword-based explanations. In the cached D.C. slice, top results cluster around cybersecurity/data-heavy roles with rich skill overlap, while sparse or generic postings sink to lower scores—evidence that the high skill weight and TF–IDF cap are working as intended. Location uniformity in that slice highlights the need for geo-aware reweighting when queries span multiple metros or remote roles. User-facing explanations (top overlapping skills and concise summaries) were well received in demos and helped build trust despite the lack of ground-truth labels.

## Challenges & Conclusions

Key challenges from code and deployment: (a) heterogeneous PDF/DOCX formats strain section detection; (b) long JD boilerplate (e.g., clearance language) can dominate TF–IDF; (c) lack of labeled relevance data forces manual/qualitative evaluation; (d) API rate limits and geo variance slow broad EDA; (e) end-to-end deployment (Docker/HF Spaces) requires tight coupling of CORS, static build, and multipart handling. The hybrid scorer currently delivers explainable, skill-forward rankings and handles location/experience edge cases. Next reliability boosts: introduce lightweight human labels, add regression tests for parsing/intent, and, budget permitting, explore embedding-based semantic signals while keeping the transparent skill overlay.

## References

[1] G. Salton and C. Buckley. “Term-Weighting Approaches in Automatic Text Retrieval.” Information Processing & Management, 24(5):513–523, 1988.

[2] C. D. Manning, P. Raghavan, and H. Schütze. Introduction to Information Retrieval. Cambridge University Press, 2008.

[3] T. Joachims. “Text Categorization with Support Vector Machines: Learning with Many Relevant Features.” In ECML, 1998.

[4] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python.” Journal of Machine Learning Research, 12:2825–2830, 2011.

[5] J. Leskovec, A. Rajaraman, and J. Ullman. Mining of Massive Datasets. Cambridge University Press, 2nd ed., 2014.

[6] R. Mihalcea and P. Tarau. “TextRank: Bringing Order into Texts.” In EMNLP, 2004.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” In NAACL-HLT, 2019.

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean. “Efficient Estimation of Word Representations in Vector Space.” In ICLR (Workshop), 2013.