Resume Parser & Job Matcher

This report explains and shows our process of using machine learning
and natural language processing techniques to build our resume parser and job matcher

Nicole Chen
Brandon Cheung
Devarsh Shah
I 320D: Text Mining & Natural Language Processing
Dr. Abhijit Mishra
May 1, 2025

Introduction

As we're approaching graduation, everyone is starting to look for jobs and it's honestly overwhelming. It's extremely time-consuming to go on different job websites, scroll through listings, and check each one to see if it matches your skillset. A lot of the time, people end up missing out on good opportunities just because they don't have time to go through everything one by one. That's the problem we wanted to solve with this project. We wanted to build a Resume Parser and Job Matcher system that can take someone's resume, understand what skills and experience they have, and automatically find job descriptions that are a good fit for them. Our goal is to make the job search process faster, easier, and more accurate for people looking for a job. Moreover, we wanted to build something smarter than just catching keywords, we want the model to understand the meaning of what the information means in a resume or job post. Hence, we used NLP techniques and machine learning models to achieve that.

Data Description

We used two datasets from Kaggle for this project, one for resumes and the other for job descriptions. The resume dataset contains over 2,400 resumes in PDF format, and were a bit messy to deal with at first. It's loosely categorized into broad fields like IT, Marketing, or Teaching, and many of them were still inconsistently labeled. Hence, to extract and process the resume data, we used pdfminer.six to convert the PDFs into raw text. We then focused on extracting relevant skills from the resumes. We used regular expressions to find common skill section headers, like "Skills," "Technical Skills," or "Programming Languages." Once located, we cleaned and standardized the extracted phrases by removing unnecessary characters like brackets, quotes, and trailing punctuation. We also removed leading conjunctions, for instance "and," "for," and limited each skill phrase to a maximum of five words, because we want it to be

consistent and minimize noise. Moreover, in order to validate the extracted skills, we used a zero-shot classification model to filter out irrelevant phrases, and keep only those that were confidently categorized as actual skills. For the job description, we did a similar extraction method. We parsed text from job listings and used regular expressions to extract skill-related segments, then we cleaned these tokens by lowercasing the text, removing non-alphanumeric characters, filtering out stopwords using NLTK, and keeping only meaningful tokens. We applied this across all rows to extract a comparable set of job-related skills.

Methodology

Our methodology begins by normalizing each resume and job description by converting all text to lowercase, stripping punctuation and extra whitespace using regular expressions, and removing stopwords. To cut residual noise, we then drop tokens of two characters or fewer, trading occasional loss of abbreviations for overall clarity. Next, we split each cleaned document into sentences using NLTK's sentence tokenizer, isolating self-contained ideas like "managed cross-functional team." Each sentence is re-cleaned (lowercased, punctuation and stopwords removed, short tokens filtered) and any fragment under three tokens is dropped, yielding a focused list of meaningful sentences. We then convert those sentences into vectors with Hugging Face's "paraphrase-MiniLM-L6-v2" model (six transformer layers, 384-dimensional hidden size). For every token, the model sums its learned token and position embeddings into a 384-dim vector, stacks these into a (sequence_length × 384) matrix, and processes them through six self-attention and feed-forward layers, which prompts every token to "look at" every other token, integrating contextual cues and then expands and then shrinks the vector's dimension, letting the model learn more complex transformations. Finally, we apply mean pooling across tokens to get

one fixed 384-dimensional embedding per sentence. This three-step pipeline, cleaning, sentence splitting, and transformer encoding, yields reliable, fixed-length vectors that accurately capture meaningful skills mentioned in both the resumes and job descriptions. These embeddings serve as the basis for our similarity matching and analyses.

To rank resumes against job descriptions, we first compute cosine similarity between their 384-dimensional embeddings from Sentence-BERT. By measuring the angle θ between a resume vector A and job vector B, we obtain

$$\text{cosine\_sim}(A, B) = \frac{A \cdot B}{\|A\| \, \|B\|}$$

which ranges from 1 (same direction, high match) down toward 0 (little shared meaning). We calculate this score for every resume-job pair, then sort each resume's job list in descending order to surface the top matches. We test another form of matching called FAISS (Facebook AI Similarity Score). FAISS matching begins by loading our precomputed 384-dim resume and job embeddings into NumPy arrays, one of shape (N_resumes, D) for resumes and another of shape (N_jobs, D) for jobs. We then initialize an IndexFlatL2, FAISS's simplest "exact" index that stores each job vector in memory, precomputes and caches its squared norm

$$\|x_i\|^2 = \sum_{j=1}^{D} x_{i,j}^2$$

to turn every distance computation into a fast dot-product plus two norm lookups. That lets FAISS scan millions of vectors per second using optimized BLAS or SIMD calls without recomputing norms every time. At every query, each resume embedding q (reshaped to shape (1, D)) is passed to the index, which returns two arrays:

- distances[0][i]: the squared L_2 distance (Euclidean Distance) to the i^{th} nearest job,
- indices[0][i]: the 0-based position of that job in our original embeddings.

We then convert each raw distance into a bounded similarity via $(1 / (1 + \text{dist}^2))$ and sort by this score to present the top-k job matches for every resume. This pipeline leverages FAISS's precomputed norms and accelerated dot products to deliver exact nearest-neighbor searches in real time.

Results

Evaluation Methodology

To evaluate our Resume_Job Matcher system, we created a test dataset of 20 resumes with manually assigned "correct" job titles which we agreed upon as a group. Each resume was processed through our pipeline to generate embeddings, which were then matched against a database of approximately 20,000 job postings using two different similarity matching methods, FAISS with Euclidean distance, and cosine similarity. Our evaluation focused on the following three metrics:

Top-1 Accuracy: The percentage of resumes for which the correct job title was the top match.

Top -5 Accuracy: The percentage of resumes for which the correct job title appeared within the top 5 matches.

Top-10 Accuracy: The percentage of resumes for which the correct job title appeared within the top 10 matches.

Comparison of Matching Methods

Metric	FAISS (L2)	Cosine Similarity
Top-1 Accuracy	20%	10%
Top-5 Accuracy	30%	50%
Top-10 Accuracy	60%	60%

FAISS showed superior performance for exact matches (Top-1) while cosine similarity outperformed when considering a broader range (Top-5). Both methods, however, achieved identical Top-10 accuracy making it difficult to decide which one was the better performer.

Analysis

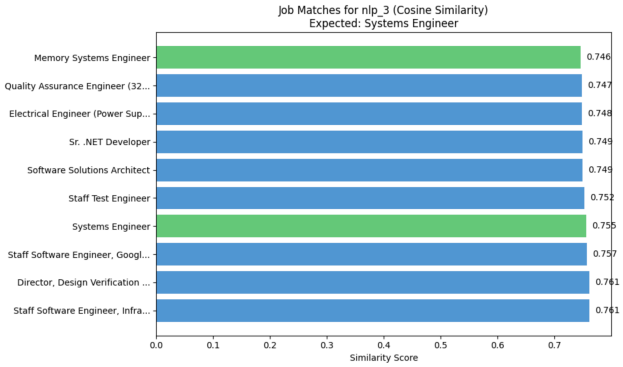


Figure 1: cosine similarity charts for Systems Engineer

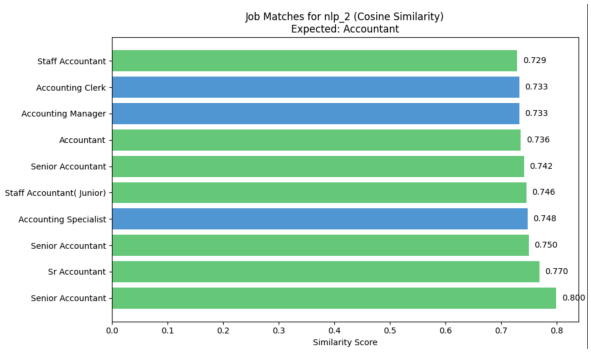


Figure 2: cosine similarity charts for Accountant

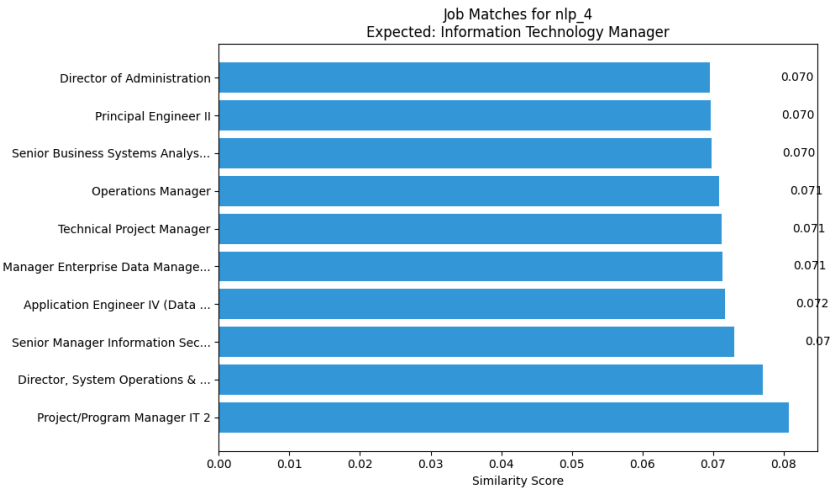
In Figure 1, for the expected job title Systems Engineer, the correct title appears with a similarity score of 0.755. Interestingly, other engineering-related roles such as Staff Software Engineer, Director, Design Verification are ranked slightly higher, even though they have slightly lower relevance. This indicates that the model is good at grouping jobs in the same general field, but it doesn't always recognize when one title is an exact match and should come first.

In Figure 2, which shows cosine similarity results for the expected job title Accountant. One interesting thing to point out, the bottom three entries are essentially the same job title, just written in slightly different ways: Senior Accountant, Sr Accountant, and Senior Accountant appear separately with slightly different similarity scores. This is a title variation issue that we'll talk about more in the error analysis, but basically it is evident that the system treats visually different labels as separate entries, even though they refer to the same role. Overall, the cosine similarity does a pretty good job on identifying roles within the same field, but it doesn't always rank exact title matches at the top.

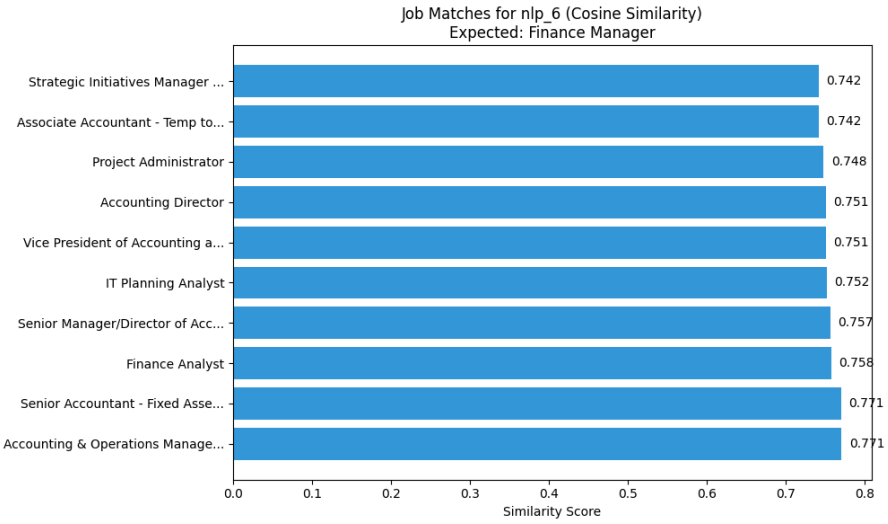
Error Analysis

Our error analysis identified several common error patterns across both matching methods:

- Title variations: The system struggled with title variations, for example, the desired "correct" job titled "Information Technology Manager" did not match with "Project/Program Manager IT", or the correct job "Teacher" did not match with "Teaching faculty".



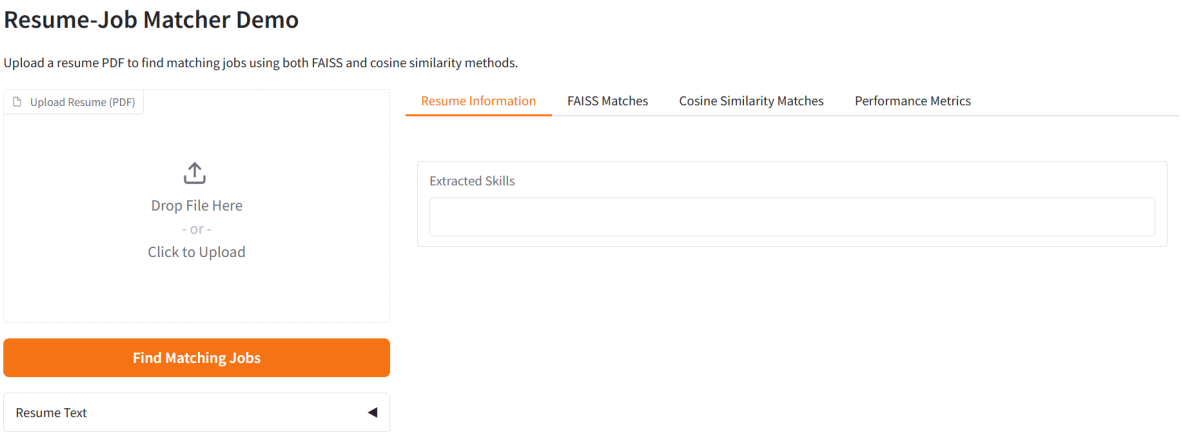
- The system had problems with highly technical and managerial roles as well, due to the model struggling to exactly match the title despite having similar domains.



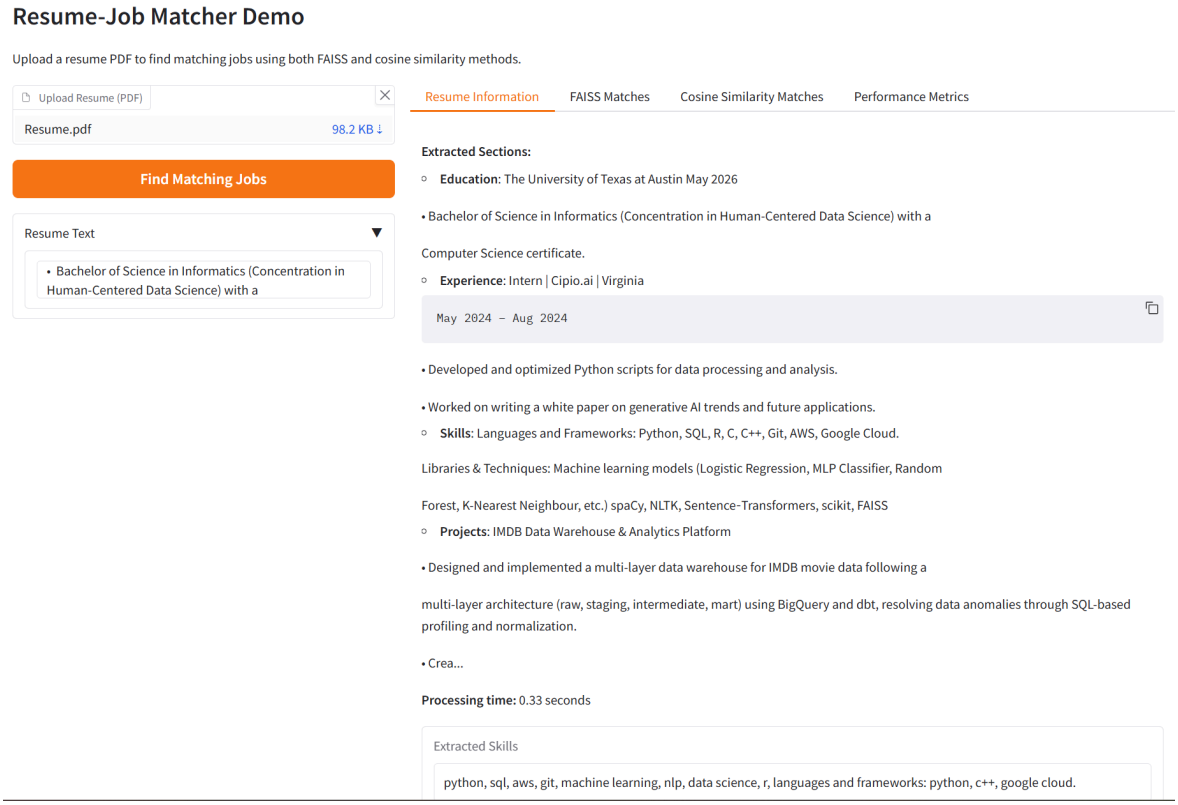
Demo

To demonstrate our resume-job matching system, we used the Gradio python library to develop a web interface. The demo allows users to upload their resume PDF and see matching job recommendations using both FAISS and cosine similarity. It consists of the following five components/steps:

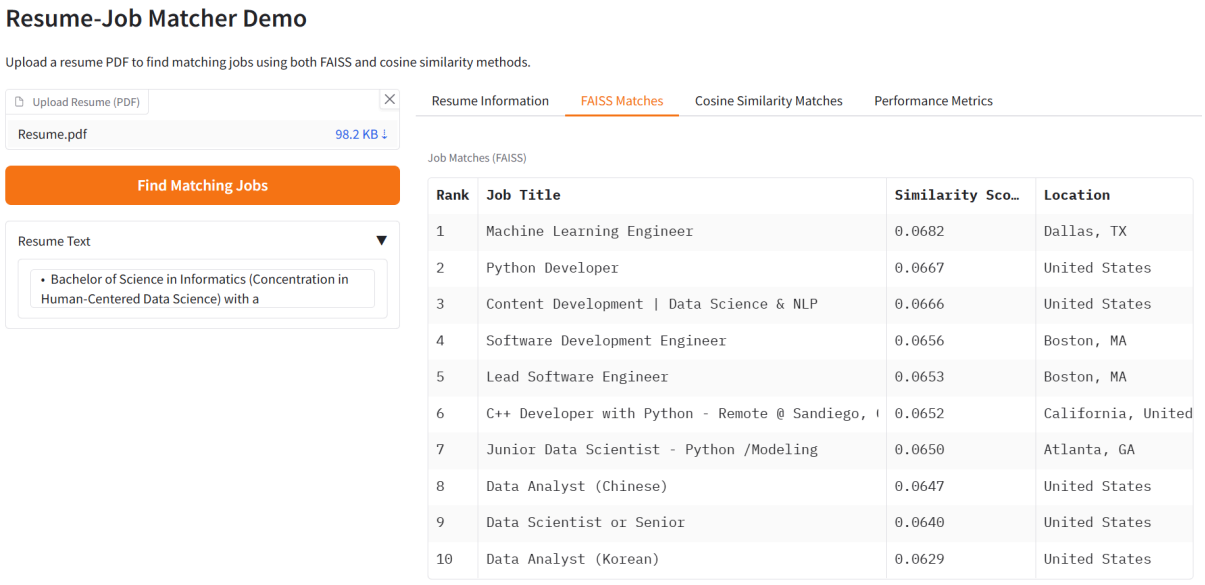
1. Resume Upload: Users upload their resume in PDF format.



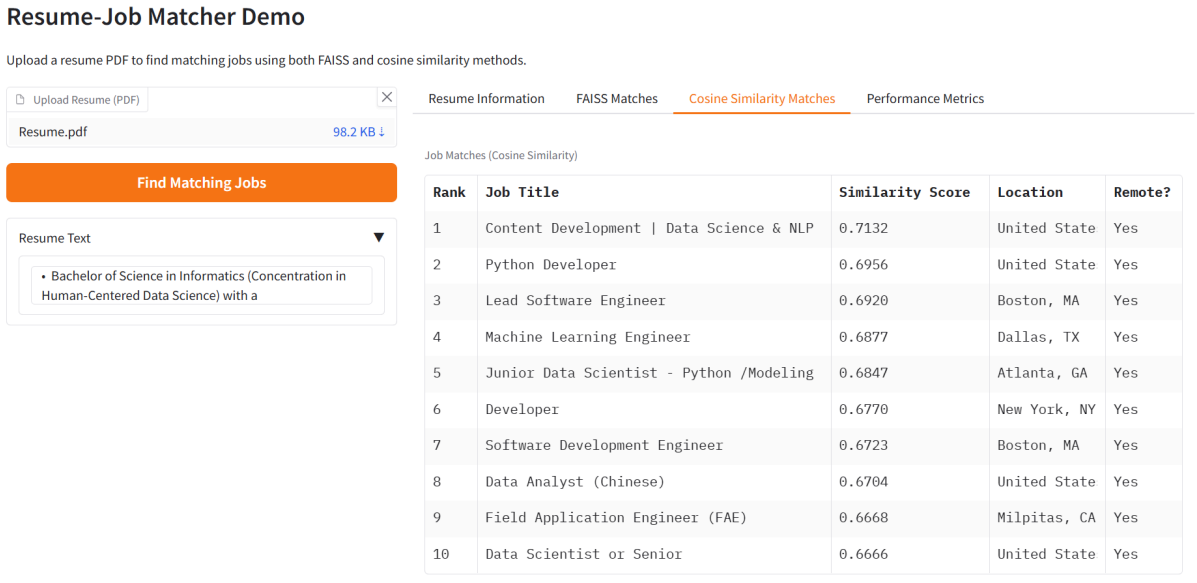
2. Resume Information Extracted: The demo displays extracted sections and skills from the resume, it extracts text using pdfminer.six and identifies key sections like education, experience, and skills using regex patterns.



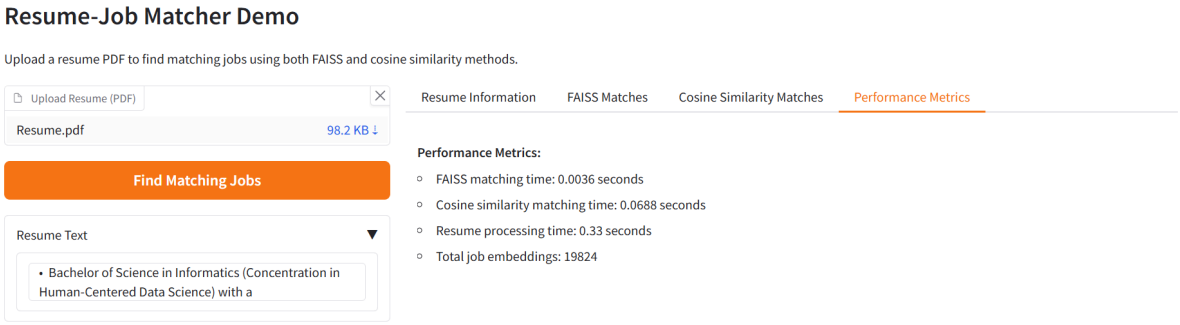
3. FAISS matches: Shows the top job matches using FAISS similarity scores, by matching the resume embedding with the job embedding database.



4. Cosine Similarity matches: Show the top job matches using cosine similarity, by matching the resume embedding with the job embeddings database



5. Performance Metrics: Displays processing times and other performance metrics/statistics.



Conclusion

In conclusion, both cosine similarity and FAISS of our Resume-Job Matcher have strengths in different aspects of the matching task. FAISS performed better in terms of Top-1 accuracy, it correctly ranks the expected job as the top match 20% of the time, while cosine similarity achieved only 10%. However, cosine similarity significantly outperformed FAISS in Top-5 accuracy, it identifies the correct job within the top 5 results in 50% of cases compared to FAISS's 30%. Both methods achieved equal Top-10 accuracy at 60%, which suggests that they both have similar overall effectiveness for broader candidate matching.

In terms of limitations as mentioned above, the system struggled with variations in job titles. For example, it failed to match the correct title “Information Technology Manager” with the semantically similar “Project/Program Manager IT,” and similarly missed matching “Teacher” with “Teaching Faculty.” Moreover, our evaluation was based on a relatively small test set of 20 resumes, which really restricts the generalizability of our evaluation. Another limitation is that the system relies solely on textual similarity, we didn’t take other important factors into account, such as required years of experience, industry context, or education level.

Hence, for future work we could incorporate these additional features, which would significantly improve our matching accuracy, especially for more nuanced roles. Moreover, we could also try to expand the test dataset and incorporate feedback from real users, it would help improve the system's evaluation and make it more reflective of real-world performance. We could also scale things up by building a data pipeline that updates the job postings regularly and adds new jobs to the database, furthermore, we could build a system that applies on behalf of the resume to the top-n matching job postings.

Code

For replicability, the code is available at the following GitHub repository:

https://github.com/ShahDevarsh0209/Resume-Parser-and-Job-Matcher

Appendix

- What is FAISS indexing and how is it helping you scale?
    - By precomputing and caching each job vector's squared norm and using BLAS/SIMD routines, FAISS turns what would be a brute-force search into a series of matrix operations that can process millions of vectors per second. This indexing lets us deliver real-time resume-job similarity queries quickly even as our database grows into millions of postings.
- Why not build a larger, separate dataset for proper testing?
    - The test datasets had to be labeled for their intended job and it would have taken too much time to add more datasets. However, it might have been possible to automate it using AI tools like ChatGPT.

- Are you using the same embedding model for both resumes and jobs, or different models?
    - Yes, we are using the same embedding model for both resume and job descriptions.
- How would you handle ties (same similarity scores)?
    - If there was a tie, we would either return both/all of the jobs or randomly shuffle/rank them
- How do you evaluate cases where multiple jobs might be equally relevant?
    - When several postings score nearly identically, we treat them as equally valid recommendations. Over time, we can refine our evaluation by grouping them under a single recommendation umbrella.

References

- https://www.learndatasci.com/glossary/cosine-similarity/
- https://faiss.ai/index.html
- https://github.com/facebookresearch/faiss
- https://sbert.net/
- https://huggingface.co/sentence-transformers
- https://www.kaggle.com/datasets/arshkon/linkedin-job-postings?select=postings.csv
- https://www.kaggle.com/datasets/snehaanbhawal/resume-dataset