TalentFit: AI-Powered Candidate Ranking and Matching System

Yuhao Huang
Apziva
Confidential - Internal Use Only

February 17, 2026

Contents

1 Executive Summary

The primary objective of this project was to develop a robust and interpretable machine learning system that automatically ranks and matches talented candidates to specific job roles for technology companies. As a talent sourcing company, identifying high-potential candidates efficiently is critical, given the manual and labor-intensive nature of traditional recruitment processes. Imagine a recruiter spending hours scrolling through hundreds of profiles, trying to find that perfect match for an "Aspiring Human Resources" role—it's tedious, time-consuming, and frankly, inefficient.

This project focused on automating and enhancing the candidate ranking pipeline for roles such as "Aspiring Human Resources" professionals. The solution required not only predicting a fitness score for each candidate based on their available information but also implementing a dynamic re-ranking mechanism that learns from user feedback when recruiters star ideal candidates. When a recruiter finds someone who perfectly represents what they're looking for and clicks that star button, the system should take notice and adjust its understanding of what makes an ideal candidate for that specific role.

The approach evolved through multiple meaningful phases, each building upon the last in our quest for better matching quality. We started with exploratory data analysis to understand the candidate pool and what their profiles actually tell us. Then we moved to traditional NLP methods like TF-IDF for baseline matching—it worked, but we quickly discovered its limitations when dealing with synonyms and semantic variations. Static embedding models including Word2Vec, GloVe, and FastText came next, offering a better grasp of semantic relationships between words. We then advanced to contextualized transformer-based models, specifically Sentence-BERT, which understood that context matters when interpreting job titles. The final and most powerful phase involved integrating Large Language Models, specifically Qwen and Llama, with techniques such as LoRA, QLoRA, and RAG to achieve state-of-the-art ranking performance that truly understands what makes a candidate right for a role.

Throughout this progression, we learned something valuable: each advancement in modeling capability led to meaningful improvements in ranking quality, but the journey wasn't always smooth. There were moments when simpler methods outperformed expectations, and times when the latest technology brought unexpected challenges. The LLM-based solution ultimately delivers the most accurate and context-aware rankings, significantly outperforming traditional embedding methods while providing the flexibility to learn and improve from user feedback. It's satisfying to see how far we've come from those early TF-IDF experiments to a system that can genuinely understand the nuance in job descriptions.

2 Business Understanding

2.1 Background

The company specializes in sourcing and managing talent for technology companies. This work requires deep understanding of client requirements, the ability to identify what makes a candidate suitable for specific positions, and efficient methods to discover talented individuals across various platforms. Think about it from a recruiter's perspective: they receive a request for "aspiring human resources" professionals, and then they need to search through candidate databases using keywords, manually review each profile to determine fitness, and build a shortlist of promising candidates. It's a process that demands both precision and patience.

Currently, the sourcing process relies heavily on manual operations. When a recruiter searches for candidates using keywords like "full-stack software engineer," "engineering manager," or "aspiring human resources," they get a list of results—but it's just a list, not a ranked collection of the best matches. The recruiter then manually reviews each profile, spending minutes on each one, trying to determine whether this person would actually be a good fit for the role. When they find an ideal candidate, they can "star" that person, indicating this candidate represents the ideal profile for the role—but this information rarely gets used to improve future searches. The system should then re-rank all candidates based on this feedback, learning from the recruiter's expertise and becoming smarter over time.

The goal is to build an automated system that predicts how fit a candidate is for a given role, ranks them accordingly from most promising to least promising, and improves its ranking over time through user feedback. This reduces manual effort while maintaining or even improving matching quality. Imagine a world where recruiters can trust the system to surface the best candidates first, allowing them to focus on what they do best—connecting with people and closing placements.

2.2 Business Objective

The client required a machine learning pipeline that accomplishes three key objectives, each building toward a more intelligent hiring process. First, the system must predict a fitness score for each candidate based on their available information, ranging from 0 to 1 where higher scores indicate better fit for the role. This score serves as the foundation for all subsequent ranking decisions.

Second, candidates should be sorted by their fitness scores in descending order, presenting the most promising matches first. This is crucial because recruiters typically only review the top 10 to 20 candidates from any search result—if the best candidates aren't at the top, the system has failed in its primary purpose.

Third and most importantly, when a recruiter stars a candidate as an ideal match, the system must dynamically re-rank the entire list based on this feedback, continuously improving its recommendations. This re-ranking capability is crucial because even the best algorithm cannot perfectly capture the nuanced requirements of a specific role without human guidance. By learning from recruiter feedback, the system becomes increasingly aligned with the client's actual hiring needs over time.

These three objectives work together to create a virtuous cycle: initial ranking provides a good starting point, recruiter feedback refines the understanding, and the improved model leads to better initial rankings in the future. It's human-AI collaboration at its finest.

2.3 Success Criteria

The revised success metrics focus on practical business value rather than abstract performance numbers. The primary objective is to achieve meaningful ranking quality that correlates with actual candidate fitness, meaning top-ranked candidates should genuinely be better fits than bottom-ranked ones. We don't just want the system to assign scores—we want those scores to mean something in terms of actual hiring outcomes.

The secondary objective is to demonstrate measurable improvement in ranking after each starring action, showing that the system learns from feedback. When a recruiter stars a candidate, they are essentially telling the system "this is what I'm looking for"—the system should honor that signal and adjust accordingly.

Additionally, the system must be able to filter out candidates who clearly should not appear in search results for a given role—someone looking for software engineers shouldn't see candidates whose entire background is in finance. We also need to establish reliable cut-off points that work across different positions without losing high-potential candidates, finding that sweet spot between comprehensiveness and precision.

These criteria directly address the client's core concerns about efficiency, accuracy, and the ability to improve over time through human-AI collaboration. At the end of the day, the system needs to save recruiters time while helping them find better candidates than they would on their own.

3 Data Understanding and Preparation

The dataset contains candidate profiles with several attributes that form the building blocks of our matching system. The id field serves as a unique numeric identifier for each candidate, allowing us to track and reference individuals throughout the system. The job title field contains free-text descriptions of each candidate's professional role, and this becomes our primary feature for matching—we look at what roles people have held or are seeking to determine their fitness for new positions. The location field indicates

geographical area, which can be relevant for roles with location constraints or preferences. The connections field shows the number of professional network connections, with 500+ indicating the candidate has over five hundred connections—a proxy for professional network breadth. The target variable is fit, representing how fit the candidate is for the role on a probability scale from 0 to 1, where higher values indicate stronger fit.

The target keywords for our demonstration are "Aspiring human resources" or "seeking human resources," representing typical searches for entry-level HR positions. We chose this specific focus because it represents a well-defined use case with clear matching criteria: we're looking for people who either currently work in HR or are seeking HR positions. This specific focus allows us to develop and validate the ranking system on a constrained but meaningful problem before extending to other roles where matching criteria might be more complex.

During exploratory data analysis, we examined the general distribution of candidates across various dimensions. The EDA process revealed patterns in job title complexity—for instance, some candidates simply list "Human Resources" while others provide detailed descriptions like "Senior Human Resources Manager with 10 years of experience in talent acquisition." We also looked at geographic distribution, seeing candidates spread across different states and even international locations, and connection strength among candidates, with some having just a handful of connections while others max out at 500+. These insights guided our feature engineering decisions and helped us understand what characteristics might indicate a strong match for HR-related positions.

No missing values were found in the dataset, and the data appeared consistent and well-structured—a relief, as data cleaning can often consume significant project time. The primary challenge was developing appropriate text processing pipelines to extract meaningful features from job title descriptions, as these varied significantly in length, format, and content. Some titles were single words, others were paragraphs. Some used industry jargon, others used plain language. Getting the text processing right was foundational to everything that followed.

We performed comprehensive text preprocessing including lowercase conversion to normalize all text, punctuation removal to focus on the core words, and stop word filtering to eliminate common but uninformative words. We also analyzed both unigrams and bigrams, discovering that top unigrams included "human," "resources," "aspiring," "seeking," and "manager," while top bigrams revealed phrases like "human resources," "human resources professional," and "human resources management." These patterns helped us understand what language candidates use to describe themselves in the HR domain.

Location data presented another preprocessing challenge. Some candidates listed specific cities, others just states, and some provided only country information. We standardized this data and categorized it by US regions and countries to enable location-based

filtering when needed.

4 Modeling Approach

When we first started building the matching system, we knew we needed a progressive approach—start with something simple that works, then progressively add sophistication. This philosophy guided our entire development process.

4.1 Traditional NLP Methods: TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) provided our natural starting point. This classical approach vectorizes text documents by calculating the importance of each word based on how frequently it appears in a document versus across the entire corpus. The intuition is straightforward: words that appear frequently in a specific document but rarely across the corpus are more distinctive and therefore more informative.

We applied TF-IDF vectorization to job titles and calculated cosine similarity between candidate profiles and target keywords. The TF-IDF implementation in TF-IDF.py established a baseline for comparison that we could use to measure improvements from more sophisticated methods.

While this method successfully identified exact and near-exact keyword matches, it struggled with semantic understanding. The problem is that TF-IDF treats words as independent tokens—it doesn't understand that "HR Professional" means essentially the same thing as "Human Resources Specialist." For example, a candidate describing themselves as "Human Resources Specialist" would score differently from one using "HR Professional" despite having essentially the same meaning. Similarly, "seeking HR positions" might not match well against "aspiring human resources" even though both clearly indicate the target profile.

This limitation became apparent during initial testing, where semantically similar candidates received widely different scores based purely on word choice. We watched candidates with nearly identical backgrounds receive fitness scores that differed by 30% or more simply because they chose different words to describe themselves. The experience highlighted the need for embedding-based approaches that could capture meaning beyond surface-level text matching.

We also explored traditional word embedding models to address these semantic limitations.

4.2 Static Word Embedding Models

To address the semantic limitations of TF-IDF, we implemented three static word embedding models: Word2Vec, GloVe, and FastText. These models represent words as dense

vectors in a continuous semantic space, where similar words are positioned closer together. The key advantage is that these embeddings capture semantic relationships—so "manager" and "director" end up in similar regions of the vector space, even though they're different words.

Word2Vec was trained on our corpus of job titles to learn domain-specific semantic relationships. By averaging word vectors within each job description, we generated document-level embeddings that captured the overall meaning of each candidate's profile. This approach successfully linked synonyms and related terms, improving matching quality over TF-IDF. If someone described themselves as "people operations," the system could now recognize this as related to "human resources."

GloVe (Global Vectors for Word Representation) leveraged pre-trained embeddings learned from large-scale text corpora. This provided semantic knowledge that extended beyond our relatively small dataset, capturing broader linguistic patterns and relationships. GloVe embeddings had seen millions of documents during training, giving them a richer understanding of word relationships than we could learn from our data alone.

FastText, developed by Facebook, offered the additional advantage of subword information. By representing words as bags of character n-grams, it could handle rare words, misspellings, and morphological variations more effectively than methods working only with full words. This was particularly valuable for job titles, where people often create their own variations—"HRManager," "HR Manager," "HR-Manager" might all refer to the same role.

Despite these improvements, all three models share a fundamental limitation: they generate static embeddings where each word has a single vector representation regardless of context. In practice, words like "manager" in "Sales Manager" and "Product Manager" carry different implications in different contexts, but static embeddings cannot distinguish between these usages. The word "manager" gets the same vector regardless of whether it's preceded by "sales" or "product." This is a meaningful limitation for our use case, as context matters when understanding job titles.

We knew we needed something more sophisticated—models that could understand context.

4.3 Transformer-Based Ranking

Moving beyond static embeddings, we implemented transformer-based models that generate contextualized representations. Unlike Word2Vec or GloVe, transformers consider the entire surrounding context when creating each word's representation, allowing them to capture how meaning changes based on neighboring words. In "Experienced HR Manager seeking new opportunities," transformers understand that "HR" modifies "Manager" and that "seeking new opportunities" indicates someone who is job hunting.

We employed Sentence-BERT (specifically the all-MiniLM-L6-v2 variant) for generating sentence-level embeddings. This model had been fine-tuned specifically for semantic similarity tasks, making it well-suited for our candidate-job matching problem. The self-attention mechanism in transformers enables the model to identify which parts of a job description are most relevant for matching.

This phase demonstrated significant improvement in ranking quality. The transformer models better captured phrase-level semantics and could recognize that "people operations" relates closely to "human resources" even without shared vocabulary. They understood that "seeking" and "aspiring" both indicate someone wanting to enter a field. The contextual awareness was a genuine leap forward from static embeddings.

However, the implementation was still limited to ranking based on predefined keywords rather than truly understanding role requirements. We were still comparing candidates to a fixed set of target phrases, not truly understanding what makes someone good for an HR role. The next evolution would address this limitation directly.

4.4 Large Language Model Integration

The most advanced and powerful phase involved integrating Large Language Models, specifically Qwen and Llama, into our ranking pipeline. LLMs offer unprecedented language understanding capabilities and can grasp nuanced requirements that traditional NLP approaches miss entirely. They can understand that "5 years of recruiting experience" indicates seniority, that "startup environment" suggests a particular work style, and that "PHR certification" is a specific credential worth noting.

Our LLM journey began with prompt-only approaches, where we designed structured prompts describing the task and role requirements, then asked the LLM to score candidates directly. We learned that prompt engineering mattered enormously—small changes in wording could significantly impact ranking quality. "Evaluate how suitable this candidate is for an HR role" produced different results than "Rate this person's fitness for an HR position on a scale of 0-1."

To achieve better task alignment, we fine-tuned the LLMs using parameter-efficient methods. LoRA (Low-Rank Adaptation) introduced trainable low-rank matrices into the frozen base model, allowing us to adapt the model to the ranking task without full fine-tuning. This dramatically reduced computational requirements while maintaining strong performance—we could train on a single GPU instead of requiring a cluster. The key insight was that we didn't need to retrain all the model's parameters, just a small number of task-specific adapters.

Building on LoRA, we implemented QLoRA which combines quantization with adapter fine-tuning. QLoRA enables fine-tuning on resource-constrained environments by reducing model precision from 16-bit to 4-bit, making it possible to train large models on

consumer hardware. This was essential for our development workflow, as we didn't have access to enterprise-grade GPU clusters.

For inference, we leveraged Google Colab's GPU resources to handle the computational demands of LLM processing. This removed local hardware limitations and allowed rapid experimentation with different model configurations. We could try different prompts, different models, and different configurations quickly without waiting for local hardware.

The final implementation utilized RAG (Retrieval-Augmented Generation), combining the knowledge of pre-trained language models with retrieval from candidate databases. We built an HR knowledge base containing information about what makes a good HR professional—what skills matter, what experience is valuable, what qualities distinguish strong candidates. When evaluating a candidate, the system retrieves relevant knowledge and uses it to inform the scoring.

Extensive prompt engineering was conducted throughout this phase, focusing on clarity in role descriptions, structured output formats, and context enrichment through relevant examples. We also implemented scoring and pairwise optimization, comparing candidates relative to each other rather than scoring in isolation. This relative ranking approach proved more stable and aligned better with human judgment—the system would say "candidate A is a better fit than candidate B" rather than assigning potentially inconsistent absolute scores.

The LLM-based solution represents a significant advancement over previous methods. By leveraging deep language understanding and learning from human feedback, the system achieves ranking quality that approaches human-level judgment while maintaining the scalability required for practical deployment. It's genuinely exciting to see how far we've come from those early TF-IDF experiments.

5 Re-ranking Mechanism

The dynamic re-ranking system addresses a fundamental challenge in automated hiring: the gap between algorithmic assumptions and actual role requirements. No algorithm, regardless of sophistication, can perfectly anticipate the specific nuances each recruiter values. Maybe they prioritize candidates with startup experience, or maybe they care more about specific certifications. These preferences are hard to capture in advance.

The starring mechanism bridges this gap by incorporating human expertise directly into the ranking loop. When a recruiter stars a candidate as an ideal match, the system extracts that candidate's features and adjusts ranking weights accordingly. The starred candidate becomes an exemplar—a concrete example of what the recruiter is looking for.

This creates a feedback loop where each starring action refines the system's understanding of what constitutes an ideal candidate for that specific role. Over time, the

ranking becomes increasingly aligned with the recruiter's implicit preferences. The system learns that when someone stars a candidate with "HR Coordinator" experience in "California," it should prioritize similar candidates in future searches.

The re-ranking process works like this: when a candidate is starred, we extract their job title and other relevant features, add these to our understanding of the ideal candidate profile, and then recompute similarity scores for all candidates using this updated profile. Candidates who share characteristics with the starred candidate receive boosted scores, bringing them higher in the ranking.

This approach offers several advantages over static ranking. First, it requires minimal additional effort from recruiters, as the starring action is a natural part of existing workflows—no extra forms to fill out, no explicit preference statements to make. Second, it provides continuous improvement without requiring model retraining from scratch—we're not rebuilding the model, just adjusting the ranking parameters. Third, it personalizes rankings to each recruiter's specific needs and context, recognizing that different recruiters may have different priorities even for the same job requisition.

The re-ranking mechanism represents a practical implementation of human-AI collaboration, where algorithms handle initial screening at scale while humans guide refinement through intuitive feedback. It's not about replacing human judgment—it's about augmenting it with the scale and consistency that algorithms provide.

6 Results and Performance

The progressive methodology yielded significant improvements across each phase, and it was rewarding to see each advancement translate into better ranking quality.

TF-IDF provided a functional baseline that proved the concept could work. It could identify candidates with exact keyword matches, giving us something to improve upon. The semantic limitations were clear, but it established a foundation.

Static embeddings improved semantic understanding substantially. Word2Vec, GloVe, and FastText each brought different strengths—FastText was particularly good with misspellings and unusual formulations, while GloVe's pre-trained knowledge helped with general language understanding. But they remained limited by their static nature, unable to distinguish how context changes meaning.

Transformers added contextual awareness that was genuinely impressive. Sentence-BERT could understand that "seeking HR position" and "aspiring HR professional" were very similar despite having no words in common. This was a major leap forward in ranking quality.

The LLM-based approaches delivered the most substantial improvements. Through careful prompt engineering, we learned how to get the models to reason about candidate fitness in ways that matched human intuition. LoRA and QLoRA fine-tuning adapted the

models to our specific task, making them much more accurate than generic prompting. RAG integration brought in domain knowledge about what makes a good HR professional, adding another layer of understanding.

The final system successfully ranks candidates based on their similarity to ideal profiles, filters out clearly unsuitable matches, and improves with each starring action. The system has learned to recognize that HR roles value certain keywords, certain experience levels, and certain career trajectories—and it applies this knowledge to surface the most promising candidates.

The complete implementation is documented across multiple Jupyter notebooks, each representing a phase of development. Prompt_only.ipynb contains our zero-shot prompting experiments, exploring different ways to get the LLM to evaluate candidates. Lora.ipynb and Lora2.ipynb document our LoRA fine-tuning experiments, showing how we adapted the base models to our ranking task. QLoRA.ipynb explores quantized fine-tuning, demonstrating that we could achieve good results with reduced computational requirements. RAG.ipynb shows our retrieval-augmented generation setup, including the HR knowledge base construction. RAG_Score.ipynb presents our scoring and ranking optimization work, including the pairwise comparison approach that improved ranking consistency.

Each notebook tells a story of experimentation, discovery, and incremental improvement—the essence of what makes machine learning projects rewarding to work on.

7 Conclusion and Future Work

This project successfully developed an intelligent candidate ranking system that progresses from traditional NLP techniques to state-of-the-art LLM-powered matching. The solution addresses the core business requirements of ranking candidates by fitness, filtering irrelevant profiles, and learning from user feedback through an intuitive re-ranking mechanism.

The journey from TF-IDF to LLM-powered ranking taught us valuable lessons about the progression of NLP capabilities. Each approach had its place—TF-IDF for speed and simplicity, static embeddings for semantic understanding, transformers for contextual awareness, and LLMs for deep comprehension. The key was matching the right technique to the right problem, knowing when to add complexity and when simpler methods sufficed.

The final LLM-based approach delivers context-aware rankings that significantly outperform traditional methods while providing the flexibility to adapt to different roles and recruiter preferences. By leveraging retrieval-augmented generation with domain-specific knowledge, the system can understand what makes someone a good fit for specific roles—not just in terms of keyword matching, but in terms of genuine qualifications and potential.

This represents a meaningful step toward reducing manual effort in talent sourcing while maintaining or improving match quality. Recruiters can trust that the system surfaces the best candidates first, allowing them to focus on what they do best—building relationships and helping candidates and clients find each other.

Looking ahead, several exciting directions await. Multi-language support would enable global candidate sourcing, opening up talent pools beyond English-speaking regions. Integration of additional candidate features such as skills and experience would enable more nuanced matching beyond job titles alone. Real-time ranking updates as new information becomes available would make the system even more responsive to changing requirements. Deployment on scalable cloud infrastructure would enable production-level inference performance, supporting real-world deployment at scale.

The foundation is solid, the approach is proven, and the path forward is clear. This system has the potential to transform how talent sourcing works—not by replacing human recruiters, but by empowering them with intelligent tools that make their work more effective and more rewarding.