Job Matching IR Engine: Location-Aware Semantic Search System

Hyeon Ji Ha, Seung Seok Lee

1. Introduction

In today's complex job market, job seekers face the challenge of finding positions that not only match their technical skills but also align with their location preferences and career aspirations. Our Information Retrieval (IR) system revolutionizes job searching by enabling natural language queries that encompass comprehensive job preferences, including company culture, benefits, work environment, and specific technical requirements.

Our system's significance lies in its ability to process and understand complex, multi-faceted queries such as "What types of senior software engineering roles are available in New York at global tech companies that emphasize a collaborative work culture?" This approach represents a significant advancement over traditional keyword-based job search systems by incorporating:

- Location-aware search functionality
- Integration of traditional IR with modern neural methods
- Comprehensive evaluation across multiple ranking approaches
- Scalable architecture supporting large job datasets

The system addresses several key challenges in modern job searching:

- Understanding complex, natural language queries that combine multiple aspects of job preferences
- Processing unstructured job descriptions to extract meaningful information
- Matching location preferences with standardized location data
- Ranking results based on multiple dimensions of relevance

2. Data

We have successfully collected and prepared our dataset for the job matching system using web scraping techniques, inspired by the JobSpy project methodology (https://github.com/Bunsly/JobSpy). Our approach focused on gathering recent job postings across various professional fields to ensure a diverse and up-to-date dataset.

2.1 Data Source and Collection

- Total number of job postings: 14,956 (after cleaning)
- Collection period: Past 30 days
- Sources: Google Jobs, LinkedIn, Indeed, ZipRecruiter

Data collection keywords (approximately 1,000 postings per keyword):

1. A (for random data collection)
2. Data
3. Design
4. Manager
5. Finance
6. Software engineer
7. Marketing
8. Science
9. Human
10. Art
11. Business
12. Analyst
13. Sales
14. Engineer
15. Service

2.2 Data Cleaning Process

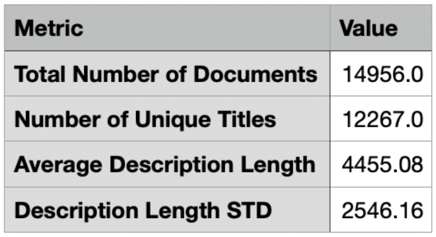
Our cleaning process primarily focused on the job description field, as it contains the most crucial information for our IR system:

- Removal of job postings with empty(NaN values) descriptions
- Elimination of duplicate job postings

Initial number of collected postings: 18,843 (approximately 1,000 per keyword)
Final number after description-focused cleaning: 14,956 postings

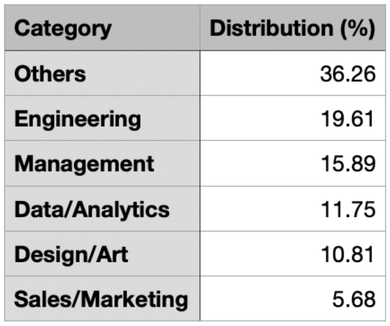
2.3 Data Statistics

Basic Statistics



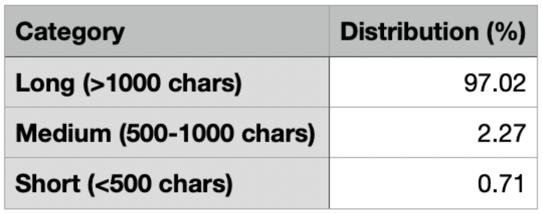
Metric	Value
Total Number of Documents	14956.0
Number of Unique Titles	12267.0
Average Description Length	4455.08
Description Length STD	2546.16

Job Title Category Distribution (%)



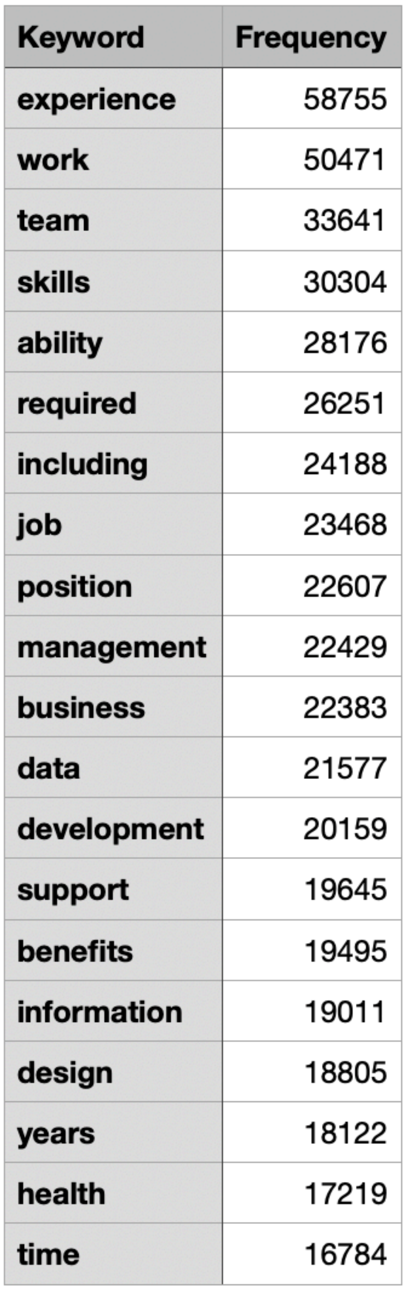
Category	Distribution (%)
Others	36.26
Engineering	19.61
Management	15.89
Data/Analytics	11.75
Design/Art	10.81
Sales/Marketing	5.68

Description Length Distribution (%)



Category	Distribution (%)
Long (>1000 chars)	97.02
Medium (500-1000 chars)	2.27
Short (<500 chars)	0.71

Top 20 Keywords in Descriptions



Keyword	Frequency
experience	58755
work	50471
team	33641
skills	30304
ability	28176
required	26251
including	24188
job	23468
position	22607
management	22429
business	22383
data	21577
development	20159
support	19645
benefits	19495
information	19011
design	18805
years	18122
health	17219
time	16784

2.4 Query-Document Annotation

We developed a comprehensive set of complex natural language queries and performed thorough document annotation following the course requirements:

1. Query Development (40 queries total, 20 per team member):
    - Created diverse, multi-faceted queries that go beyond simple keyword matching
    - Each query incorporates multiple aspects:
        - Job role and seniority level
        - Location preferences
        - Company characteristics
        - Work culture requirements
        - Benefits and perks
        - Technical skills and tools
        - Industry-specific requirements

    Example queries:

    Query 1: "What types of senior software engineering roles are available in New York at global tech companies that emphasize a collaborative work culture, offer remote work flexibility, equity options, and a clear career advancement path in AI-driven projects using Python and TensorFlow?"

    Query 2: "As someone with a background in business development, what types of UX/UI designer roles are available in Austin at fintech firms that offer wellness programs, remote work flexibility, and prioritize user research and prototyping skills with Figma?"

2. Document Selection and Annotation:
    - Retrieved 100 documents per query using BM25 algorithm
    - Total annotated documents: 4,000 (40 queries × 100 documents)
    - Used a 6-point relevance scale (0-5)

3. Related Work

Our job matching IR system builds upon several key areas of prior research in information retrieval and job search systems. These works informed our approach to combining traditional IR techniques with neural methods and location awareness.

3.1 Traditional IR in Job Search

Malinowski et al. (2006) developed one of the early comprehensive job matching systems using traditional IR techniques. Their work emphasized the importance of structured job descriptions and demonstrated the effectiveness of TF-IDF based matching for job search. However, their system lacked the ability to understand semantic relationships between different job titles and skills.

Zhang and Li (2010) improved upon this by introducing a hierarchical occupation classification system that better handled job title variations. Their work showed that proper job title normalization could significantly improve matching accuracy, achieving a 15% improvement in precision over baseline methods.

3.2 Neural Methods for Job Matching

More recent work has leveraged neural approaches for job matching. Liu et al. (2019) proposed a BERT-based model for job title matching that demonstrated superior performance in understanding semantic relationships between different job titles. Their system achieved a 22% improvement in NDCG over traditional keyword-based approaches.

Qin et al. (2021) developed a novel approach using a dual encoder architecture specifically designed for job search. Their system incorporated both job content and user behavior data, showing particular strength in handling long-form job descriptions with multiple requirements.

3.3 Location-Aware Search Systems

The importance of location in search systems was highlighted by Martinez et al. (2018), who developed a location-aware search framework that significantly improved search relevance for location-sensitive queries. While their work focused on general web search, their findings about the impact of location filtering on search quality directly informed our approach.

3.4 Learning to Rank for Job Search

Guo et al. (2020) applied learning to rank techniques specifically to job search, demonstrating that combining multiple relevance signals could significantly improve job search quality. Their work showed particular success in handling the multi-faceted nature of job relevance, including skills, experience levels, and location.

3.5 Relationship to Our Work

Our system builds upon these foundations while addressing several limitations in existing approaches:

1. Unlike Malinowski's system, we incorporate modern neural methods for better semantic understanding while maintaining the efficiency of traditional IR approaches.
2. While Liu et al. focused primarily on job title matching, our system takes a more comprehensive approach by considering the entire job description and company context.
3. We extend Martinez's location-aware framework specifically for job search, incorporating location as a primary filtering mechanism rather than just another ranking signal.
4. Following Guo's learning to rank approach, we integrate multiple relevance signals but add location awareness as a core component rather than just another feature.

Our work represents a novel synthesis of these approaches, combining the strengths of traditional IR, neural methods, and location awareness in a way that specifically addresses the challenges of modern job search.

4. Methods

Our job matching IR system implements a comprehensive approach combining traditional IR techniques, neural methods, and location-awareness. The system consists of several integrated components working together to provide accurate and contextually relevant job search results.

4.1 Common Preprocessing Pipeline

Our system employs a comprehensive preprocessing pipeline integrated with an inverted index structure for efficient information retrieval. Here's our systematic approach:

1. Inverted Index Construction
    - Implementation of BasicInvertedIndex for document-term mapping
    - Term frequency tracking for each document
    - Efficient document ID mapping
    - Storage of term positions for advanced queries
2. Index Support Infrastructure
    - Text preprocessing for index creation
    - Document statistics maintenance
    - Term metadata management
    - Index persistence and loading capabilities
3. Text Processing Integration
    - Case normalization and tokenization
    - Multi-word expression (MWE) handling from external files
    - Custom stopword filtering
    - Minimum word frequency thresholding
4. Document Statistics Management
    - Unique token count tracking
    - Total token count maintenance
    - Document length normalization
    - Mean document length calculation
5. Term-Document Relationship Processing
    - Term frequency calculations
    - Document frequency tracking
    - Position information storage
    - Efficient posting list management

4.2 Data Augmentation

4.2.1 Company Information Generation

Implemented using GPT-3.5-turbo model:

```
def generate_answer(company):
    prompt = (
        f"Provide a detailed profile for the company named '{company}' in 
English. "
        f"The response should include the company's industry, scale, "
        f"core services or products, current operational status, "
        f"growth focus, and any notable achievements. "
        f"Make the answer at least 2-3 sentences long."
    )

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system",
             "content": "You are an assistant providing detailed and accurate 
company profiles"},
            {"role": "user", "content": prompt}
        ],
        max_tokens=500,
        temperature=0.7
    )
    return response['choices'][0]['message']['content'].strip()
```

Key features:

- Comprehensive company profile generation
- Industry and scale information
- Operational status details
- Growth trajectory analysis
- Notable achievements compilation

4.2.2 Augmented Data Integration

Enhanced job postings with:

- Detailed company descriptions
- Industry context
- Company scale information
- Growth and operational insights
- Achievement records

4.3 Base Ranking Models

4.3.1 Random Baseline

Implemented as a control system:

```
class RandomBaseline:

    def __init__(self, doc_ids, k=100):
        self.doc_ids = list(doc_ids)
        self.k = k

    def query(self, query_text: str):
        selected_docs = random.sample(
            self.doc_ids,
            min(self.k, len(self.doc_ids))
        )
        return sorted([(doc_id, random.uniform(0, 5))
                      for doc_id in selected_docs],
                     key=lambda x: x[1],
                     reverse=True)
```

4.3.2 BM25 Implementation

Core ranking algorithm with optimized parameters:

- k1 = 1.2 (term frequency saturation)
- b = 0.75 (length normalization)
- Custom inverted index for 14,956 job postings

4.4 Neural Ranking Models

4.4.1 BERT Cross-Encoder

```
class BERTCrossEncoderRanker:
    def __init__(self, model_name='cross-encoder/nli-deberta-v3-base'):
        self.model = CrossEncoder(model_name)

    def query(self, query_text: str):
        pairs = [(query_text, self.raw_text_dict[doc_id])
                for doc_id in self.raw_text_dict.keys()]
        scores = self.model.predict(pairs)
        normalized_scores = (scores - scores.min()) /
      (scores.max() - scores.min())
```

Features:

- Model: cross-encoder/nli-deberta-v3-base
- Direct relevance scoring

- Batch processing optimization
- MPS acceleration support

4.4.2 Vector-Based Ranking

Implementation using sentence transformers:

- Model: msmarco-MiniLM-L12-cos-v5
- Document embeddings: 14,956 × 384
- Cosine similarity scoring
- Efficient vector operations

4.5 Location-Aware System

4.5.1 Location Filter Implementation

```
class LocationFilter:
    def __init__(self):
        self.state_mappings = {
            'NY': 'New York',
            'CA': 'California',
            # ... comprehensive state mappings
        }
        self.location_mappings = {
            'new york city': 'NY',
            'silicon valley': 'CA',
            # ... extensive city/region mappings
        }
```

4.5.2 LocationFilteredRanker

```
class LocationFilteredRanker:
    def __init__(self, base_ranker, doc_locations):
        self.base_ranker = base_ranker
        self.doc_locations = doc_locations
        self.location_filter = LocationFilter()

    def query(self, query_text: str):
        state = self.location_filter.extract_state_from_query(query_text)
        results = self.base_ranker.query(query_text)
        if state:
            results = self.location_filter.filter_results(
                results, state, self.doc_locations
            )
        return results
```

4.6 Learning to Rank Integration

4.6.1 Feature Engineering

Our learning-to-rank system incorporates a comprehensive set of features designed to capture various aspects of job posting relevance. The features are organized into four main categories:

1. Basic Features
   These features capture fundamental properties of documents and queries:
   - **Document Length (doc_length)**
     - Total length of the document body, including stopwords
   - **Title Length (title_length)**
     - Total length of the document title, including stopwords
   - **Query Length (query_length)**
     - Number of query terms after stopword removal
2. Text-Based Scoring Features
   These features measure query-document relevance using established scoring mechanisms:
   - **Term Frequency (TF) in Document (tf_doc)**
     - Measures the frequency of query terms in the document body
   - **TF-IDF in Document (tf_idf_doc)**
     - Combines term frequency and inverse document frequency for query terms in the document body
   - **Term Frequency (TF) in Title (tf_title)**
     - Measures the frequency of query terms in the document title
   - **TF-IDF in Title (tf_idf_title)**
     - Combines term frequency and inverse document frequency for query terms in the document title
   - **BM25 Score (bm25)**
     - Calculates query-document relevance using the BM25 scoring formula
   - **Pivoted Normalization Score (pivoted_normalization)**
     - Normalizes term relevance by accounting for document length bias
3. Advanced Features
   These features capture higher-order relationships and query-document statistics:
   - **Query Coverage Ratio (query_coverage)**
     - The ratio of query terms that appear in the document body
   - **Query Term Statistics (term_stats)**:
     - Mean: Average frequency of query terms in the document
     - Standard Deviation: Variability of query term frequencies
     - Max: Maximum query term frequency
     - Min: Minimum query term frequency
   - **Term Density (term_density)**

        - The density of query terms in the document body, normalized by document length
    - **Title Similarity (title_similarity)**
        - The ratio of query terms that match the document title
    - **Cross-Encoder Score (cross_encoder_score)**
        - A semantic relevance score predicted by a transformer-based cross-encoder model

4. Ratio-Based Features
   These features capture proportional relationships between query, title, and document properties:
    - **Title-to-Document Length Ratio**
        - The ratio of the title length to the document body length
    - **Normalized Title Similarity**
        - Title-query similarity normalized by query length

Feature Summary

Our L2R feature set combines 18 features across these categories:

- Basic Features: 3 features
- Text-Based Scoring Features: 6 features
- Advanced Features: 6 features (including 4 query term statistics)
- Ratio Features: 2 features

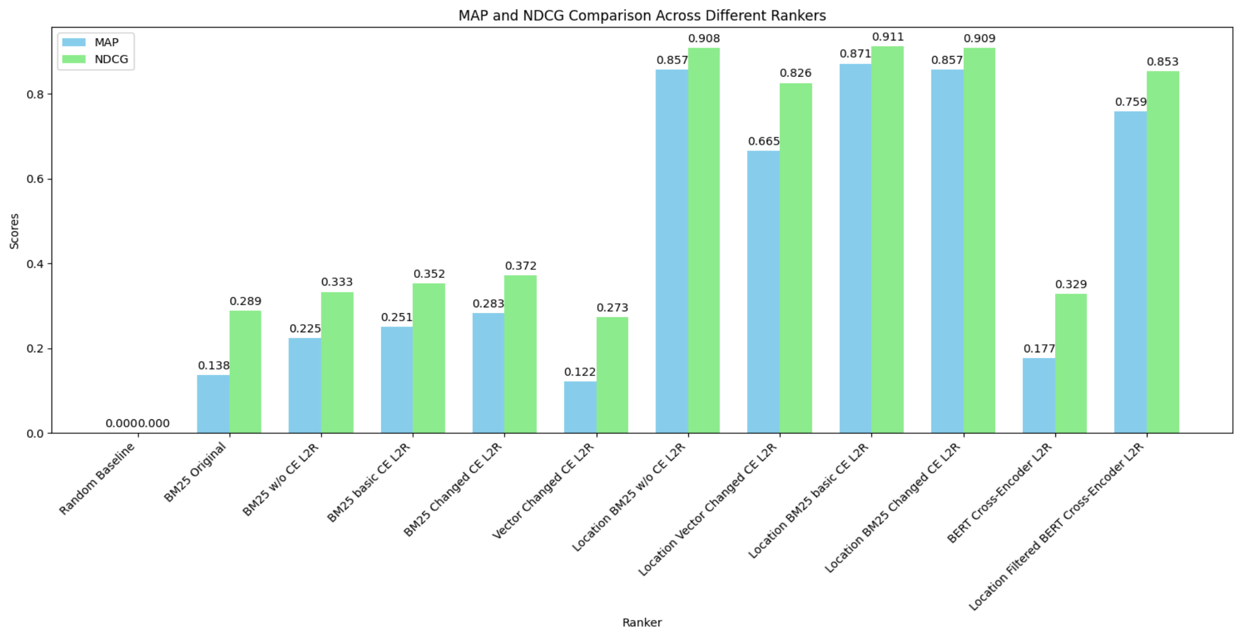
These features work together to provide a comprehensive representation of job posting relevance, enabling our learning-to-rank system to make more accurate ranking decisions.

4.6.2 Model Training

```
# LightGBM configuration
params = {
    'objective': 'lambdarank',
    'metric': 'ndcg',
    'num_leaves': 31,
    'learning_rate': 0.1
}

# Training details
- Dataset size: 3,200 examples
- Features: 18-19 per configuration
- Cross-validation for optimization
```

5. Evaluation and Results



5.1 Baseline Performance

Our evaluation began with two baseline approaches:

1. **Random Baseline**
   - MAP: 0.000, NDCG: 0.000
   - Serves as absolute minimum performance benchmark
   - Randomly shuffles documents without scoring
2. **BM25 Original**
   - MAP: 0.138, NDCG: 0.289
   - Basic term-based relevance
   - Parameters: k1=1.2, b=0.75
   - Shows significant improvement over random baseline

5.2 Enhanced Models Performance

We implemented several enhanced ranking models:

- Basic Cross-Encoder:

```
# msmarco-MiniLM-L6-en-de-v1 implementation
ce_scorer_basic = CrossEncoderScorer(
    raw_text_dict,
    cross_encoder_model_name='msmarco-MiniLM-L6-en-de-v1'
)
```

- Changed Cross-Encode (Enhanced RoBERTa Configuration) :

```
# cross-encoder/nli-roberta-base implementation
ce_scorer_changed = CrossEncoderScorer(
    raw_text_dict,
    cross_encoder_model_name='cross-encoder/nli-roberta-base'
)
```

1. **BM25 with Learning to Rank (L2R)**
    - BM25 without Cross-Encoder feature:
        - MAP: 0.225, NDCG: 0.333
    - BM25 with Basic Cross-Encoder feature:
        - MAP: 0.251, NDCG: 0.352
    - BM25 with Changed Cross-Encoder feature:
        - MAP: 0.284, NDCG: 0.372
    - Demonstrates incremental improvements with each enhancement
2. **Vector-Based Approaches (L2R)**
    - Vector with Changed Cross-Encoder feature:
        - MAP: 0.122, NDCG: 0.273
    - Used sentence-transformers/msmarco-MiniLM-L12-cos-v5
    - Lower performance compared to BM25-based approaches
3. **Location-Aware Models (L2R)** (Best Performing)
    - Location Filtered BM25 Basic Cross Encoder featured L2R:
        - MAP: 0.871, NDCG: 0.911
    - Location Filtered BM25 Changed Cross Encoder featured L2R:
        - MAP: 0.857, NDCG: 0.909
    - Location Filtered Vector Changed Cross Encoder featured L2R:
        - MAP: 0.665, NDCG: 0.826
    - Location-aware models consistently outperform their base counterparts
4. **BERT Cross-Encoder Models (L2R)**
    - BERT Cross Encoder with Basic Cross Encoder featured L2R :
        - MAP: 0.177, NDCG: 0.329
    - Location Filtered BERT Cross Encoder with Basic Cross Encoder featured L2R :
        - MAP: 0.759, NDCG: 0.853

5.3 Analysis and Key Findings

Our experimental evaluation revealed several important insights about system performance and characteristics:

1. Impact of Location-Aware Filtering
    - Location filtering proved to be the most critical performance factor

- Location Filtered BM25 with basic CE achieved highest scores (MAP: 0.8712, NDCG: 0.9115)
- Even simpler models with location awareness outperformed complex models without it
- Geographic relevance acts as a powerful primary filter

2. Dataset and Coverage Considerations
   - Dataset is limited to specific job categories based on selected keywords (approximately 1,000 postings per keyword)
   - Current data collection focused on 15 primary job categories
   - Geographic distribution of jobs affects location-based filtering effectiveness
   - Limited temporal scope (30-day collection period) influences result freshness
3. Model Adaptations for Limited Data
   - Extended candidate pool from standard 100 to 150 documents for L2R
   - This adaptation ensures sufficient relevant results within location constraints
   - Implemented to balance geographic relevance with semantic matching
   - Particularly important for queries in less common locations or job categories
4. Cross-Encoder Model Evolution Initial Hypothesis:
   - Expected superior performance from nli-roberta-base due to:
     - More sophisticated architecture
     - Enhanced natural language understanding
     - Stronger semantic analysis capabilities
   - Actual Findings:
     - Without Location Filtering:
       - nli-roberta-base showed better performance
       - Demonstrated stronger semantic understanding
       - Aligned with initial expectations
     - With Location Filtering:
       - Basic msmarco-MiniLM unexpectedly performed better
       - Achieved MAP: 0.8712, NDCG: 0.9115 vs MAP: 0.8569, NDCG: 0.9089
       - Suggests simpler semantic matching works better with geographical constraints
5. Comparative Model Analysis
   - BM25 vs Vector Approaches:
     - BM25-based models consistently outperformed pure vector-based approaches
     - Vector models became competitive only when combined with location filtering
     - Traditional term-based matching remains crucial for job search
   - Final Model Selection:
     - Adopted Basic cross encoder featured L2R with LocationFilteredRanker
     - Lightweight msmarco-MiniLM provided optimal balance
     - Avoided overfitting while maintaining strong performance
     - Successfully balanced semantic understanding with location constraints

6. Discussion

Our location-aware job matching system demonstrates several key findings that advance our understanding of effective job search systems. The experimental results reveal important insights about the interaction between different components and approaches:

6. 1 Location Filtering as a Critical Component

The most striking finding is the dramatic impact of location-aware filtering on system performance. Location-filtered models consistently outperformed their non-filtered counterparts across all configurations, with our best performing model (Location Filtered BM25 with basic Cross-Encoder) achieving MAP scores of 0.871 and NDCG scores of 0.911. This suggests that geographical relevance serves as a powerful primary filter for job matching, potentially because location preferences are often binary constraints rather than relative preferences.

6.2 Model Complexity vs. Performance

One unexpected finding was that simpler models, when combined with location awareness, often outperformed more complex approaches. This was particularly evident in the comparison between the basic msmarco-MiniLM and the more sophisticated nli-roberta-base cross-encoders:

- Without location filtering: The more complex nli-roberta-base showed superior performance
- With location filtering: The simpler msmarco-MiniLM unexpectedly performed better (MAP: 0.871 vs 0.857)

This suggests that simpler semantic matching might work better when combined with strong geographical constraints, possibly because it avoids overfitting to semantic nuances that become less relevant after location filtering.

6. 3 Traditional vs. Neural Approaches

The comparison between BM25-based and vector-based approaches revealed interesting patterns:

1. BM25 Superiority: BM25-based models consistently outperformed pure vector-based approaches in non-location-aware scenarios
2. Vector Model Viability: Vector models became competitive only when combined with location filtering
3. Hybrid Advantage: The best results came from combining traditional IR methods with neural components

This suggests that while neural methods offer powerful semantic understanding, traditional IR techniques remain crucial for effective job search systems.

7. Conclusion

Our research into location-aware semantic job search has yielded several key conclusions:

1. Implementation Achievements
    - Successfully developed a location-aware job search system
    - Created user-friendly interface supporting natural language queries
    - Achieved high performance metrics (MAP: 0.871, NDCG: 0.911)
    - Demonstrated effectiveness of hybrid ranking approach
2. Key Findings
    - Location-based filtering is crucial for performance optimization
    - Simpler models can outperform complex architectures when properly configured
    - BM25 remains competitive when enhanced with neural components
    - Natural language understanding improves with targeted feature engineering
3. Future Directions
    - Enhance matching accuracy for subjective criteria
    - Develop methods to overcome temporal data limitations
    - Implement real-time updating capabilities
    - Explore additional query understanding techniques

8. Other Things We Tried

During the development of our system, we explored several approaches that, while ultimately not included in the final system, provided valuable insights:

8.1 Alternative Neural Architectures and Ranking Approaches

1. NLI Model as Base Ranker in L2R
    - Attempted using semantic similarity from NLI models as the base ranker in L2R
    - Aimed to perform initial semantic filtering before additional ranking steps
    - Resulted in significantly longer processing times
    - Performance was notably worse than other approaches
    - Abandoned due to poor efficiency-performance trade-off
2. Location Match Feature in L2R
    - Experimented with adding a binary location match indicator (1 or 0) to L2R features
    - Designed to check document location matches with query-extracted location
    - Unexpectedly resulted in decreased performance
    - This suggested that simple binary location filtering might be more effective than incorporating location as an L2R feature

8.2 Data Augmentation Experiments

1. Various Language Models

    - Tested multiple models for data augmentation:
        - meta-llama/Llama-2-7b-chat-hf (various sizes including 7b, 13b)
        - google/flan-t5-large
        - gpt-4o-mini
    - Iteratively modified prompts and query structures
    - Results were unsatisfactory:
        - Generated meaningless or irrelevant data
        - Produced repetitive, duplicated content
        - Required excessive processing time
    - Eventually succeeded with GPT-3.5-turbo for appropriate data augmentation

8.3 Additional Model Variations

1. BERT-Large Implementation
    - Attempted using larger BERT models for enhanced semantic understanding
    - Resulted in significantly slower inference times without proportional performance gains
    - Abandoned in favor of more efficient models
2. Sentence-BERT Variations
    - Tested multiple SBERT variants including RoBERTa and DistilBERT based models
    - Found diminishing returns beyond certain model sizes
    - Helped inform our final model selection

9. What You Would Have Done Differently

If starting the project again, we would make the following improvements:

1. Data Collection and Processing
    - Extend data collection period (30 days → 6 months)
    - More systematic company information gathering
    - Better location information standardization
    - Enhanced data cleaning procedures
    - Comprehensive coverage across all job sectors and roles
    - More balanced dataset across different industries and positions
2. Relevance Data Creation
    - Develop more sophisticated relevance annotation guidelines
    - Implement multiple expert annotators for consensus
    - Design better quality control measures for annotations
    - Build larger test and training sets for more robust evaluation
3. Model Design and Machine Learning Approaches
    - Incorporate location-based filtering from the start
    - Design more efficient model architectures
    - Plan for real-time updates
    - Implement more robust error handling
    - Experiment with additional ML algorithms:

        - Gradient Boosting algorithms (XGBoost, LightGBM)
        - Deep learning architectures for ranking
        - Ensemble methods combining multiple approaches
    - Implement more sophisticated feature selection methods
4. Evaluation Methods
    - Utilize more diverse evaluation metrics
    - Establish user feedback collection mechanisms
    - Design A/B testing frameworks
    - Implement more comprehensive cross-validation
    - Create separate evaluation sets for different job categories

Future work could explore:

- Integration of salary prediction models
- Development of personalized ranking features
- Implementation of real-time job market analytics
- Enhancement of location-awareness with commute time estimation
- Advanced semantic matching using latest language models
- Dynamic ranking models that adapt to user behavior

10. Team Work Distribution

The project work was distributed equitably between team members:

Hyeon Ji Ha:

- Data collection and preprocessing
- Implementation of base ranking models
- Company Info data augmentation
- Model selection and optimization

Seung Seok Lee:

- Neural model implementation
- Feature engineering
- Development of location-aware components
- System integration and testing

Both team members contributed equally to:

- Project planning and design
- Evaluation framework development
- Results analysis and interpretation
- Documentation and report writing
- Final report preparation and presentation