Project: Towards a Fair Employment Recommendation System

Intern:

Sinjini Ghosh[1]
M1 MLDM

Supervisors:

Professor Antoine Gourru[1]
Professor Christine Largeron[1]
Bissan Audeh[2]

June 24, 2025

[1]Laboratoire Hubert Curien, Université Jean Monnet
[2]Inasoft HR Creative Solutions

Abstract

E-recruitment is a rapidly emerging domain that mirrors the global shift towards online platforms across various sectors. In this evolving landscape, as hiring practices become more digital, e-recruitment recommendation systems play a crucial role in streamlining the hiring process by matching job seekers to the most suitable jobs and vice versa. This project focuses on developing a system to effectively match optimal candidates with suitable job openings. Our approach tackles a two-fold problem: first, a binary classification task to determine a candidate's suitability for a given vacancy, and second, a ranking problem to present the most relevant candidates to recruiters in a prioritized manner. To achieve this, we employed Sentence Transformers, utilizing sentence pairs where the first sentence represents the candidate profile and the second describes the job vacancy, subsequently training the model for classification.

1 Introduction

E-recruitment signifies the integration of technology to optimize the recruitment cycle. It enables HR departments to automate recruitment tasks, enhance communication with applicants, and reduce hiring costs. E-recruitment in human resource management involves the use of specialized software that manages job postings, processes applications, selects candidates for interviews, and maintains a database for potential future job seekers. E-recruitment recommender systems greatly impact the career opportunities of job seekers as well as the hiring efficiency of companies. An efficient e-recruitment recommender system not only streamlines the process by matching suitable candidates to job openings but also addresses critical challenges such as the two-sided nature of the market, the high-stakes decisions involved, and the often short user histories available. Furthermore, it can incorporate fairness concerns to ensure equitable outcomes in hiring.

The domain of e-recruitment comes with its own set of challenges [7] such as:

1. Data Quality: We have a lot of heterogeneous data from multiple sources that is often disorganized in nature. Hence, it is imperative to organize the data before proceeding towards training the model.

2. Semantic ambiguity: The very nature of this problem involves dealing with a large amount of text. In the text used for training the models, different words might mean the same thing, and certain words might have ambiguous meanings that might introduce noise in the training data. We need a way to make the model understand the semantic context of the text, in spite of varied vocabulary. Simple keyword matching doesn't work, which means it is necessary to use a more nuanced approach for text comparison.

3. Data Imbalance: In candidate application data, the number of positive applications are often significantly lesser than the number of negative applications. Hence, building an unbiased classification and ranking system becomes even more of a challenge.

4. Bias and Fairness: We need to make sure that the e-recruitment recommender system shows no bias in recommendations, and shows the same behaviour irrespective of a sensitive attribute of a candidate, such as gender, or nationality.

This project was done in partnership with *Inasoft HR Creative Solutions* (https://www.inasoft.fr/), a company that specializes in developing software solutions for recruitment that are tailored to the specific needs of various enterprises. The aim of this project was to explore machine learning and NLP (natural language processing) based solutions to build a fair and efficient employment recommendation system.

The first step in this project was to create appropriate datasets for classification and ranking. For training sentence transformers we created three versions of a dataset with candidate-vacancy sentence pairs. The first version comprises raw candidate and vacancy texts. In the second version of the dataset, the candidate text is cleaned using NLP techniques, particularly using an LLM (*phi-4-mini*). The third version of the dataset comprises sentence pairs with the *ESCO (European Skills, Competences, Qualifications, and Occupations)* standardized vocabulary. We explored a couple of different approaches such as using a sentence transformers to make a binary classification of candidate's suitability for a vacancy. Traditional classification models such as Random Forest, XGBoost, LightGBM were also explored. The results for binary classification for both approaches were rather dismal primarily because of the stark imbalance in the labels. However, classification and ranking using sentence transformers is a more promising direction than using standard ML classification models because of the vast amount of text corpus that characterizes the nature of this problem. Hence, the evaluation results for the same have been mentioned in this report.

2 Literature Review

There are many research papers in this field, but two key ones influenced the work in this project.

2.1 Multi-Head Sparse Recruitment System (MUSE) by Guillaume Bied [1]

The MUSE algorithm is an algorithm designed to enhance the efficiency and effectiveness of job matching. It is structured in a two-tier architecture, each one focusing on a different aspect of the recommendation process. MUSE addresses the "cold-start problem" in recommendation systems through its innovative two-tiered architecture. The cold-start problem in recommendation systems refers to the challenge faced when there is little to no interaction history for users or items. This is common in labor market settings where job seekers may not have extensive previous interactions with job ads. As a result, traditional collaborative filtering methods, which rely on historical data, often struggle to provide effective recommendations in these scenarios. The MUSE architecture addresses this problem through an innovative multi-embeddings based architecture. The different tiers of the MUSE algorithm are as follows:

MUSE.0: Candidate Retrieval Stage: This is the first tier of the MUSE architecture. It employs a "two-tower" structure to compute a score for job seeker and job ad pairs based on their embeddings. The score, denoted as Muse.0 $0(x, y)$ is calculated using the dot product of the embeddings for the job seeker x and the job ad y. This stage is crucial for narrowing down the job ads to the top 1,000 candidates, which are then passed to the second tier for more detailed analysis.

Scoring Function:

$$Score(x, y) = \sum_{m \in \{skills, geo, etc.\}} \langle \phi_m(x), \psi_m(y) \rangle$$

- x = The job seeker's profile
- y = The job advertisement
- m = A specific facet (e.g., skills, geography)
- $\phi_m(x)$ = Embedding of job seeker x in facet m
- $\psi_m(y)$ = Embedding of job ad y in facet m
- $\langle \cdot, \cdot \rangle$ = Dot product between embeddings

MUSE.1: Re-ranking Stage: After filtering, MUSE.1 refines the ordering of the top job ads using more complex features. The recommendation score is calculated as follows:

Scoring Function:

$MUSE.1(x, y) = MLP(\phi(x), \psi(x, y), \phi(x) \odot \psi(x, y))$

- $\phi(x)$: Embedding of the job seeker
- $\psi(x, y)$: Contextual embedding for seeker-job ad pair
- $\odot$: Element-wise product
- MLP: Multi-layer perceptron model

In this equation MLP denotes a multi-layer perceptron, and the term $(x)(x, y)$ captures the interactions between the job seeker and job ad embeddings. This allows the model to learn relevant user-item interactions automatically. MUSE.1 is trained to predict whether a job seeker will be hired based on the job ad, minimizing a cross-entropy loss function.

MUSE.2 (Final Recommendations): This tier focuses on providing recommendations based on actual hiring data, further enhancing the quality of matches over time. It utilizes insights from both MUSE.0 and MUSE.1 to improve the recommendation process.

In summary, the MUSE scoring system effectively combines initial filtering through MUSE.0, detailed re-ranking with MUSE.1, and final recommendations from MUSE.2 to provide a robust job recommendation framework. This multi-tiered approach ensures that the system can handle the complexities of job matching, even in cold-start scenarios where interaction data is sparse.

2.2 LLM-Based Generative Adversarial Network (Yingpeng Du et al.) [4]

This paper aims to tackle the problem of incomplete candidate resumes using the semantic and natural language understanding powers of LLMs. In real-world recruitment datasets, a large number of candidates often have resumes with sparse or incomplete information. This can impact the quality of recommendations. Having rich, comprehensive information about candidates and vacancies is crucial in ensuring accurate recommendations. This paper aimed to tackle the problem of incomplete candidate resumes by utilising the power of LLMs. The core of the paper is a novel framework called LGIR (LLM-based GANs Interactive Recommendation). This method addresses two primary challenges:

1. **LLM Fabrication:** Large Language Models have a tendency to generate fabricated or "hallucinated" information, which can degrade the quality of resume enhancement.

2. **The Few-Shot Problem:** Many users have limited interaction data (e.g., jobs they have applied to or viewed), making it difficult for an LLM to generate a high-quality, personalized resume completion.

To overcome these issues, the LGIR method incorporates two key innovations: Interactive Resume Completion and a GAN-based alignment method.

2.2.1 Interactive Resume Completion with LLMs (IRC)

To ground the LLM and prevent it from generating irrelevant information, the model uses a strategy called Interactive Resume Completion (IRC). Instead of only using the candidate's self-description, this method also provides the LLM with the job descriptions of positions the user has previously interacted with. This allows the model to infer the user's implicit skills and interests, leading to a more accurate and meaningful resume. The process is guided by a specific prompt that incorporates both the user's resume (T_c) and their interaction history (R_c). The generated resume (G_c) is produced as follows: $G_c = LLMs(prompt_{IRC}, T_c, R_c)$ where $R_c = \{T_{j_k} | \mathcal{R}_{cj_k} = 1\}$ represents the set of job requirements that the user 'c' has interacted with. This approach was shown to be superior to a "Simple Resume Completion" (SRC) method, which only uses the original resume and suffers more from LLM hallucinations.

2.2.2 A GAN-based Aligning Method

To address the few-shot problem, the paper proposes a transfer learning approach using Generative Adversarial Networks (GANs) to refine the resumes generated for users with sparse interaction data. This process involves a Classifier, a Generator, and a Discriminator.

Classifier (C): First, a classifier is trained to distinguish between "high-quality" and "low-quality" generated resumes. Resumes generated for users with many interaction records (many-shot users) are assumed to be high-quality, while those from users with very few records (few-shot users) are considered low-quality. The classifier is defined as: $\mathcal{C}(x) = \sigma(W_2^c \cdot Relu(W_1^c \cdot x))$ It is trained using a cross-entropy loss function, $\mathcal{L}_\mathcal{C}$, on a subset of users with very high or very low interaction counts.

Generator (G) and Discriminator (D): The core of the alignment method is a GAN, where a Generator and a Discriminator are trained in a competitive minimax game.

- The Generator ($\mathcal{G}$) takes the representation of a low-quality resume (as identified by the classifier) and attempts to refine it so that it resembles a high-quality one. Its goal is to "fool" the discriminator. The generator is defined as: $\mathcal{G}(x) = W_2^g \cdot Relu(W_1^g \cdot x)$
- The Discriminator ($\mathcal{D}$) is trained to distinguish between genuine high-quality resume representations and the "refined" representations produced by the generator. It is defined as: $\mathcal{D}(x) = \sigma(W_2^d \cdot Relu(W_1^d \cdot x))$

Adversarial Learning: The two models are trained adversarially. The discriminator aims to maximize its ability to correctly label real and refined resumes, while the generator aims to minimize the probability that its outputs are classified as refined.

- The discriminator's objective is to maximize the loss function $\mathcal{L}_\mathcal{D}$: $max_{\Theta_D}\mathcal{L}_\mathcal{D} = E_{c_{i_1}\sim\overline{T}_\mathcal{C}^\dagger}[log\mathcal{D}(x_{c_{i_1}})] + E_{c_{i_2}\sim\hat{T}_\mathcal{C}^-}[1 - log\mathcal{D}(\mathcal{G}(x_{c_{i_2}}))]$ Here, $\hat{T}_\mathcal{C}^\dagger$ and $\hat{T}_\mathcal{C}$ are the sets of high-quality and low-quality resumes detected by the classifier, respectively.
- The generator's objective is to minimize the loss function $\mathcal{L}_\mathcal{G}$: $min_{\Theta_\mathcal{G}}\mathcal{L}_\mathcal{G} = E_{c_{i_2}\sim\hat{T}_\rfloor^-}[1 - log\mathcal{D}(\mathcal{G}(x_{c_{i_2}})]$

Through this iterative training, the generator becomes adept at improving the representations of low-quality resumes, effectively bridging the gap between few-shot and many-shot users.

2.2.3 Multi-objective Learning for Recommendation

Finally, the refined resume representations are used to make job recommendations. For any given user, their resume representation, z_{c_i}, is either the original high-quality version or the generator-refined version, depending on the classifier's output. $z_{c_i} = \begin{cases} x_{c_i}, & if\mathcal{C}(x) \geq 0.5 \\ \mathcal{G}(x_{c_i}), & if\mathcal{C}(x) < 0.5 \end{cases}$

A matching score, $\hat{R}_{i,k}$, between a user c_i and a job j_k is then calculated using a deep model that captures non-linear relationships. $\hat{R}_{i,k} = g(c_i, j_k) = W^p \cdot [z_{c_i} + x_{j_k}, z_{c_i} - x_{j_k}, z_{c_i} \odot x_{j_k}]$ where $\odot$ denotes the element-wise product.

The entire model is trained using a pairwise loss function, $\mathcal{L}_{rec}$, which aims to ensure that jobs a user has shown interest in receive a higher score than those they have not. $\mathcal{L}_{\tau ec} = max_\Theta \sum_{(i,j_1,j_2)\in D} log\ \sigma(\hat{R}_{i,j_1} - \hat{R}_{i,j_2}) - \lambda||\Theta||^2$

The LGIR method was tested on three large, real-world recruitment datasets (Designs, Sales, and Tech). Experiments demonstrated that LGIR consistently and significantly outperformed all baseline methods. The ablation and few-shot studies confirmed that both the Interactive Resume Completion (IRC) and the GAN-based alignment are crucial to the model's success, with the GAN providing the most significant improvements for users with the fewest interactions.

3 Experimental Contribution

This section of the report talks about the creation of datasets for training a sentence transformer to classify binary labels indicating whether a candidate is fit for a given role.

3.1 Dataset Preparation

The dataset provided by Inasoft was divided into three parts: candidates, vacancies, and applications. There were 78723 unique candidates, 13203 unique vacancies, and 100633 unique applications. The candidates dataset contained solely the data about candidates such as name, gender, nationality, past experiences, and the raw text extracted from their CVs. The vacancies dataset contained data about the vacancies such as the job title, location, description, expected candidate profile etc. The applications dataset contains details about applications, connecting candidates to jobs. The applications dataset contains a field called "status" which indicates the status of the application. This field was used to label the data. The values in the "status" field were as follows:

- Poste Pourvu: Job post deleted.
- Embauche: Hired
- NAE: Non-Accepté Entretien (interview not accepted)
- NAE + 5jrs: Same as above but more than 5 days passed since that status.
- NAE + Vivier: Interview not accepted, but added to talent pool
- Vivier: Added to talent pool
- Vivier + 2jrs: Candidate added to talent pool 2 days ago
- Négatif: Rejected in the very first round itself
- Négatif + 2jrs / +5jrs: Rejected in the very first round, 2 or 5 days ago
- Neg sans msg: Rejected in the very first round without any communication with the candidate
- No News: No updates yet
- NET: Probably Non Entretien Téléphonique (no phone interview), or more generally "not contacted yet"
- NET + 3jrs: No contact made in 3 days
- NET Rému: No contact made yet about remuneration
- NET Localisation: No update about job location
- NET Savoir Etre / Savoir Faire / Cond. Travail: No contact yet about skills, or work conditions

Applications with status Embauche were labeled "1". Applications with status Vivier, Vivier + 2jrs, No news, and Poste pourvu were not considered for training because they were ambiguous. The rest of the applications were labeled as "0". The applications dataset had a column for "Mobilite" which is an important feature as it contains the candidate's desired job location (a region of France). However, this column contained a significant amount of missing values. The values that are not missing are regions, not cities. The Ville column (candidate's current city) had negligible missing values. Hence, a new column Region was created by mapping each Ville to its corresponding region using the Wikidata API. The missing Mobilite values were then filled with this new Region column. This means that if the candidate has specifically mentioned their wish of job location, we would take that into account, otherwise, we will take into account the region where the candidate currently lives.
After pre-processing the primary datasets given by Inasoft, we created three datasets as follows:

- **Dataset 1 The Raw Baseline:** This dataset comprises the raw and unprocessed candidate text directly extracted from the CV versus the full job description.
  Shape of the dataset: 35137 unique sentence pairs.
  Distribution of label:

Label	Count	Percentage
0	34720	98.81%
1	417	1.19%

- **Dataset 2 LLM enhanced:** This dataset comprises the candidate text cleaned and professionally structured using an LLM (phi-4-mini). The vacancy text was already well organized, so pre-processing using LLMs was not deemed necessary.

  Prompt used to organize the candidate text:

  > *Extrayez les champs structurés suivants du profil du candidat :*
  >
  > *Intitulé du poste actuel*
  > *Intitulés des postes précédents*
  > *Formation*
  > *Compétences*
  > *Entreprises précédentes*
  > *Secteurs d'activité des entreprises précédentes*
  >
  > *Soyez concis et concentrez-vous uniquement sur les mots-clés. Voici le profil:*

  Shape of the dataset: 35137 unique sentence pairs.
  Distribution of label:

Label	Count	Percentage
0	34720	98.81%
1	417	1.19%

- **Dataset 3: ESCO Standardized:** The motivation behind creating this dataset was to standardize the vocabulary used in the vacancy and candidate texts. In the original dataset, the words used in candidate and vacancy texts are often not consistent. Different words might be used to describe the same job position or skill. Hence, it was decided to standardize the vocabulary. ESCO stands for European Skills, Competences, Qualifications, and Occupations. ESCO works as a dictionary, describing, identifying and classifying professional occupations and skills relevant for the EU labour market. The ESCO occupations and skills datasets were downloaded from the official website and a graph data structure was created from it to facilitate information retrieval. In this graph, each node represented an occupation or a skill. The edges connect occupations to required skills. For candidates, the ESCO occupation titles were already present in the dataset and were extracted from the dataset. The skills corresponding to the ESCO titles were retrieved from the graph. Unlike candidates, the ESCO titles for vacancies were not explicitly present in the dataset, so the vacancy titles were compared to the ESCO titles using the sentence embedding model 'paraphrase-multilingual-mpnet-base-v2'. Each vacancy in the dataset was compared to all the occupation nodes in the ESCO graph using cosine similarity. The top match was stored. After matching each vacancy to its nearest ESCO title, the associated skills were retrieved from the graph. Each candidate and job title now has a structured set of ESCO-defined skills for structured skill gap analysis. From the ESCO titles and skills, the following type of sentence pairs were made:

  `Sentence 1 (Candidate):`
  J'ai travaillé dans les domaines suivants : {*titles*}*. Je possède les compétences suivantes:* {*skills*}*.*
  `Sentence 2 (Vacancy):`
  Le poste recherché est : {*title*}*. Les compétences requises sont :* {*skills*}*.*

  Shape of the dataset: 32546 unique sentence pairs.
  Distribution of label:

Label	Count	Percentage
0	32149	98.87%
1	397	1.13%

  The number of sentence pairs is lower in this dataset because for some candidates, the ESCO titles were not available in the dataset, so they had to be removed.

3.2 Sentence Transformers Model Architecture [8]

A core task in our project is to measure the semantic similarity between a CV and a job description. While standard transformer models like BERT [3] are powerful, they are not designed for direct, large-scale sentence comparison. Using BERT to compare every CV with every job would require a massive number of computations ($N \times M$ *pairs*), making it inefficient. To solve this, we used Sentence-BERT (SBERT), a modification of the BERT architecture that is optimized for generating semantically meaningful sentence embeddings.

SBERT utilizes a siamese network structure [2], where a single, pre-trained transformer model processes two sentences (e.g., a candidate CV and a vacancy description) to generate independent, fixed-size vector embeddings. To obtain a fixed-size sentence embedding from the variable-length token outputs of the transformer, SBERT employs a mean pooling strategy. This strategy calculates the mean of all token embeddings, weighted by their corresponding attention mask, to produce a single vector representing the entire meaning of the input. The resulting pooled embeddings are then normalized to unit length to make them suitable for comparison by cosine similarity. This approach is highly efficient because the embeddings for all candidates and jobs can be pre-computed, which allows for rapid comparison. For classification tasks, the SBERT architecture is extended with a classification head with a softmax layer. The process is as follows:

1. Each input sentence (sentence1 and sentence2) is passed through the Sentence Transformer to obtain its respective normalized embedding, U and V.
2. These embeddings are then combined to capture information about both the individual sentences and their relationship. The strategy used is to concatenate the two embeddings along with their element-wise difference: classifier input $[U; V; U - V]$
3. The classifier_input vector is then fed into a final linear classifier layer to produce the classification logits.

This architecture is flexible, allowing for different training objectives depending on the task, from simple classification to metric learning.

3.3 Loss Functions

The training process is governed by a composite loss function, UtilityAndFairnessLoss, which is designed to balance two potentially conflicting objectives: task-specific utility and fairness. The total loss is a weighted sum of a utility loss and a fairness loss, controlled by the hyperparameter β. $\mathcal{L}_{total} = \mathcal{L}_{utility} + \beta \cdot \mathcal{L}_{fairness}$

3.3.1 Utility Loss ($\mathcal{L}_{utility}$)

Given below are the utility loss functions we explored for the purpose of binary classification. For training our binary classification model, we primarily used *Binary Cross Entropy loss* as the loss function.

- **Binary Cross-Entropy with Logits Loss (BCE):** This is the loss function used for binary classification. The ground truth labels are binarized based on a minimum relevance threshold; any candidate with a relevance score greater than or equal to the threshold is considered a positive example.

$$\mathcal{L}_{BCE} = -(y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y}))$$

- **Cross-Entropy Loss (CE):** Used for multi-class classification tasks.

$$\mathcal{L}_{CE} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

- **Contrastive Loss:** A metric learning approach for paired sentences. It aims to pull similar pairs (positive, $y = 1$) closer together while pushing dissimilar pairs (negative, $y = 0$) apart by a specified margin m. The loss for a single pair with distance d is:
  $\mathcal{L}_{contrastive} = \frac{1}{2}(y \cdot d^2 + (1-y) \cdot max(0, m-d)^2)$
- **Triplet Loss:** Another metric learning approach that operates on triplets of (anchor, positive, negative). The objective is to ensure the distance between the anchor and the positive, $d(a, p)$, is smaller than the distance between the anchor and the negative, $d(a, n)$, by at least a margin m. The loss for a single triplet is:
  $\mathcal{L}_{triplet} = max(d(a,p) - d(a,n) + m, 0)$
  This loss also supports an adaptive margin where m is determined by the difference in relevance scores between the positive and negative examples.

3.3.2 Fairness Loss ($\mathcal{L}_{fairness}$)

This component of the total loss explicitly penalizes biases in model predictions with respect to a sensitive attribute (sa), such as gender. The framework can optimize for one of the following fairness metrics:

- **Demographic Parity:** This metric aims to ensure that the selection rate is equal across different demographic groups, irrespective of the true labels. It is calculated as the absolute difference in the mean predicted probability for each group:
  $\mathcal{L}_{DP} = |E[\hat{y}|sa = 0] - E[\hat{y}|sa = 1]|$
- **Equal Opportunity:** This metric requires the True Positive Rate (TPR) to be equal across demographic groups. It focuses on fairness for the individuals who qualify for the positive outcome:
  $\mathcal{L}_{EOpp} = |E[\hat{y}|y = 1, sa = 0] - E[\hat{y}|y = 1, sa = 1]|$
- **Equalized Odds:** This is a stricter criterion that requires equality for both the True Positive Rate and the False Positive Rate (FPR) across groups. It is calculated as the sum of the absolute differences in TPR and FPR.

3.4 Evaluation Metrics

The model's performance is assessed using three distinct categories of metrics, depending on the task: classification performance, fairness, and information retrieval performance.

3.4.1 Classification Metrics

For standard classification tasks (trained with BCE or CE loss), the model is evaluated using the following metrics:

- **Accuracy:** The percentage of correctly classified instances.
  $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- **Precision:** The ratio of true positives to the total number of predicted positives.
  $Precision = \frac{TP}{TP+FP}$
- **Recall:** The ratio of true positives to the total number of actual positives.
  $Recall = \frac{TP}{TP+FN}$

– **F1-Score:** The harmonic mean of precision and recall, providing a single score that balances both concerns.
$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$

3.4.2 Fairness Metrics

To quantify the fairness of the final model, the same metrics used in the fairness loss component; namely Demographic Parity (DP), Equal Opportunity (EOpp), and Equalized Odds (EOdds) are calculated on the test set predictions. This provides a clear insight on the behaviour of the model with respect to the sensitive attribute.

3.4.3 Information Retrieval (IR) Metrics

For ranking-oriented tasks trained with metric learning losses (e.g., Triplet, Contrastive), a specialized IREvaluator is used. This evaluator assesses the quality of the ranked list of candidates generated for each job vacancy using the following metrics:

– **Accuracy@k:** Measures whether at least one relevant candidate is present in the top-k recommended results. It is averaged over all queries.

– **Precision@k & Recall@k:** Standard precision and recall calculated on the top-k items in the ranked list.
$P@k = \frac{|\{\text{relevant}\} \cap \{\text{retrieved at k}\}|}{k}$
$R@k = \frac{|\{\text{relevant}\} \cap \{\text{retrieved at k}\}|}{|\{\text{total relevant}\}|}$

– **Mean Reciprocal Rank (MRR@k):** The average of the reciprocal ranks of the first correctly identified relevant candidate, capped at rank k. For a set of queries Q:
$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$

– **Normalized Discounted Cumulative Gain (NDCG@k):** A measure of ranking quality that rewards placing more relevant candidates higher in the list, using their actual relevance scores ($reli$).
$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)}$

$NDCG@k = \frac{DCG@k}{IDCG@k}$
where IDCG@k is the DCG score of the ideal ranking.

– **Mean Average Precision (MAP@k):** The mean of the Average Precision (AP) scores for each job query. AP rewards ranking relevant items higher.
$AP@k = \frac{\sum_{i=1}^{k} (P@i \times \text{is relevant}(i))}{|\{\text{total relevant}\}|}$
$MAP@k = \frac{\sum_{q \in Q} AP@k(q)}{|Q|}$
where is relevant(i) is 1 if the item at rank i is relevant, and 0 otherwise.

– **Expected Reciprocal Rank (ERR@k):** A metric designed for scenarios with graded (multi-level) relevance. ERR estimates the expected reciprocal rank at which a user will find a sufficiently relevant candidate. It assumes that users are more likely to stop searching after encountering highly relevant items. It is computed as:

$$ERR@k = \sum_{i=1}^{k} \frac{1}{i} \cdot P(i) \cdot \prod_{j=1}^{i-1} (1 - P(j))$$

where $P(i)$ is a function of the relevance grade at rank i.

Choosing the Right Metric

– **Binary Relevance:** When candidate relevance is binary (e.g., suitable or not suitable), use metrics such as Accuracy@k, Precision@k, Recall@k, MRR@k, and MAP@k. In our case, we have a binary relevance.

– **Graded Relevance:** If candidates are assigned graded relevance levels (e.g., highly suitable, moderately suitable, marginally suitable), NDCG@k and ERR@k are more appropriate, as they account for varying degrees of relevance.

4 Results

The experimental results for each dataset across various evaluation metrics are presented in Table 1.

The sensitive attribute for fairness metrics is "gender" or "Civilite", with two unique values.

4.1 Information Retrieval (Ranking Performance)

The ranking performance varies dramatically across the datasets, with the LLM-enhanced data showing a significant advantage.

– **Dataset 2 (LLM Enhanced)** is the clear leader in information retrieval. It is the only dataset to achieve a non-zero score for top-1 accuracy (Acc@1: 0.59, Pre@1: 0.59) and demonstrates substantially better overall ranking quality with the highest NDCG@5 (0.39), MRR@5

Table 1: Performance Results Across Different Datasets (Normalized between 0 and 100)

Metric Type	Metric	Dataset 1: Inasoft Raw	Dataset 2: Inasoft LLM	Dataset 3: Inasoft ESCO
Information Retrieval	Acc@1	0.00	**0.59**	0.00
	Acc@3	0.59	**1.18**	0.62
	Acc@5	**1.18**	**1.18**	0.62
	Pre@1	0.00	**0.59**	0.00
	Pre@3	0.20	**0.39**	0.21
	Pre@5	0.24	**0.24**	0.12
	Rec@1	0.00	**0.07**	0.00
	Rec@3	0.02	**0.37**	0.21
	Rec@5	0.32	**0.37**	0.21
	NDCG@5	0.23	**0.39**	0.14
	MRR@5	0.44	**0.88**	0.21
	MAP@5	0.13	**0.26**	0.07
Classification	Accuracy	85.88	86.07	**89.98**
	F1-Score	**27.39**	24.98	24.28
	Precision	**23.45**	21.55	17.21
	Recall	32.93	29.70	**41.18**
Fairness	DP	14.60	11.33	**1.45**
	EOpp	14.52	**4.98**	13.90
	EOdds	28.14	16.27	**14.07**

(0.88), and MAP@5 (0.26) scores. This indicates that the LLM-based text cleaning and structuring is highly effective for identifying and prioritizing relevant candidates.

– **Dataset 1 (Raw Baseline)** and **Dataset 3 (ESCO Standardized)** show very poor ranking performance. Both fail to place a single correct candidate at the top position (Acc@1 is 0.00). Their overall low scores (e.g., NDCG@5 of 0.23 and 0.14, respectively) suggest that without the LLM enhancement, the model struggles to distinguish and rank relevant applications effectively.

4.2 Classification Metrics

– **Accuracy:** All datasets show high accuracy (85.88% to 89.98%), with Dataset 3 (ESCO) being the highest. However, given the severe class imbalance of the data, this metric is misleading as it largely reflects correct classification of the negative (majority) class.

– **F1-Score, Precision, and Recall:** These metrics reveal a more nuanced picture of performance on the positive (minority) class.

∗ **Dataset 1 (Raw Baseline)** achieves the highest F1-Score (27.39) and Precision (23.45), suggesting it provides the best balance between precision and recall among the three.

∗ **Dataset 3 (ESCO Standardized)** obtains the highest Recall (41.18) by a significant margin. This indicates it is the best at identifying positive candidates, but this comes at the cost of having the lowest Precision (17.21), meaning it produces a high number of false positives.

∗ **Dataset 2 (LLM Enhanced)** shows a trade-off, with its F1-Score (24.98) and Recall (29.70) positioned between the Raw and ESCO datasets.

∗ The low Precision values across all datasets remain a concern, highlighting a general difficulty in avoiding false positive classifications.

4.3 Fairness Metrics

The results show significant differences in fairness outcomes depending on the data preparation method.

- **Demographic Parity (DP):** Dataset 3 (ESCO Standardized) is exceptionally effective at achieving demographic parity, with a DP of 1.45. This is a substantial improvement over both Dataset 1 (14.60) and Dataset 2 (11.33), indicating a much more equitable selection rate across demographic groups.
- **Equal Opportunity (EOpp):** Dataset 2 (LLM Enhanced) performs best on this metric with the lowest EOpp value of 4.98. This suggests it provides the most equal True Positive Rate for qualified individuals across groups.
- **Equalized Odds (EOdds):** Dataset 3 (ESCO Standardized) has the best EOdds value (14.07), followed closely by Dataset 2 (16.27). This indicates that the ESCO data achieves a better balance of True Positive and False Positive rates across groups. Dataset 1 (Raw) performs poorly on all three fairness metrics.

5 Overall Conclusions and Future Directions

The latest results demonstrate that the choice of data preparation has a profound and varied impact on ranking, classification, and fairness. The severe class imbalance remains a core challenge, but these findings provide a clearer path forward.

- **Critical Role of LLM for Ranking:** The most significant finding is the success of the LLM-enhanced dataset in the information retrieval task. Unlike the other methods, it produced meaningful ranking results (MRR@5 of 0.88), addressing a critical deficiency from previous experiments. This strongly suggests that LLM-based text structuring is essential for the system's core function of prioritizing candidates.
- **Classification Trade-offs:** No single dataset excels at classification. The Raw data provides the best-balanced performance (highest F1-score), while the ESCO data maximizes recall at the cost of precision. This highlights a classic precision-recall trade-off that must be considered in the context of the desired application behavior (e.g., is it better to miss fewer good candidates or to present fewer bad ones?).
- **Dataset Preparation and Fairness Trade-offs:**
  - **ESCO standardization (Dataset 3)** is highly successful in promoting fairness, achieving the best scores for Demographic Parity and Equalized Odds. However, this comes at the cost of poor ranking performance and the lowest precision.
  - **LLM enhancement (Dataset 2)** offers a compelling middle ground. It is essential for effective ranking, provides the best Equal Opportunity, and maintains competitive classification and fairness scores elsewhere. This suggests it may be the most promising overall approach.

5.1 Proposed Future Direction: Leveraging the Talent Pool (Vivier) with the ESCO Standardized Dataset

A promising opportunity for future work lies in utilizing the "vivier" (talent pool) data, which was previously excluded from training due to its ambiguous nature. The ESCO standardized dataset, with its structured skill and occupation vocabulary, provides an ideal framework for this task. This approach directly addresses the severe class imbalance by introducing a new, larger intermediate class and creates a practical tool for recruiters.

We propose two potential implementations:

- **Multi-Class Classification**: The problem can be reframed from the binary to a three-class classification problem with the following labels:
  - **Class 0**: Rejected
  - **Class 1**: Hired
  - **Class 2**: Talen Pool (Vivier)

  This would allow the model to make more granular predictions and identify candidates who are promising but may not be a perfect fit for the immediate role. Training this model would require switching from a Binary Cross-Entropy loss function (BCE) to a standard Cross-Entropy (CE) loss. This approach could potentially improve the performance in classification by creating a more balanced label distribution.
- **Skill-Based Ranking for Talent Sourcing from the Vivier:** A more direct application is to build a ranking system for the talent pool. For each vacancy, its ESCO-defined skill requirements can be compared against the ESCO skill profiles of all candidates in the "vivier". Using the pre-computed sentence embeddings

for candidates and jobs, we can efficiently calculate a similarity score (for example, cosine similarity) for every candidate in the talent pool against the vacancy. This would generate a ranked list of the most suitable candidates from the existing pool, allowing recruiters to source talent proactively. This method leverages both the ESCO dataset in structured skill analysis and the efficiency of the Sentence Transformer architecture. This approach provides a method for mitigating data imbalance and also allows recruiters to tap into the unexplored list of potential hires. However, since we do not have ground truth labels for the Vivier, evaluation for this approach would have to qualitative initially, demonstrating the relevance of the top-ranked candidates to a human evaluator.

6 Architecture and Training Process of JinaEmbeddings Model [5], [9]

JinaEmbeddings are a series of text embedding models designed for creating embeddings of text in the vector space. They build upon transformer encoders with key modifications and a multi-stage training paradigm. Their performance is observed to be better than that of Sentence BERT models in certain areas which will be discussed in this section.

6.1 General Architecture

JinaEmbeddings models are based on transformer encoder architectures with certain adaptations to improve performance.

1. **Backbone Transformer Encoder:** The base model is fften XLM-RoBERTa or BERT. A significant improvement that the Jina Embeddings models V2 and V3 make over SBERT is the use of "Attention with Linear Biases (ALiBi)" and "Rotary Position Embeddings (RoPE)" repectively.

   Attention with Linear Biases (ALiBi - v2) [5], used in Jina Embeddings v2 model, applies a linear bias $m \cdot (j-i)$ to attention scores QK^T before softmax, allowing extrapolation to longer sequences:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T + \text{ALiBiBias}}{\sqrt{D_k}}\right)V$$

where $\text{ALiBiBias}_{i,j} = -m \cdot |i-j|$.

- Attention(...): This denotes the output of the attention mechanism.
- Q: The **Query matrix**. Each row represents a query vector for a token in the input sequence. The query vector determines how much a token "attends" to other tokens.
- K: The **Key matrix**. Each row represents a key vector for a token in the input sequence. Key vectors are what query vectors are compared against to determine attention.
- V: The **Value matrix**. Each row represents a value vector for a token in the input sequence. These are the actual "information" vectors that are weighted and summed based on the attention weights.
- QK^T: The dot product of the Query matrix Q and the transpose of the Key matrix K. This calculates the raw **attention scores** or "compatibility" between every query and every key.
- ALiBiBias: The **Attention with Linear Biases matrix**. This matrix of biases is directly added to the raw attention scores (QK^T).
- $\text{ALiBiBias}_{i,j}$: An element in the ALiBiBias matrix, representing the bias applied to the attention score between the i-th query token and the j-th key token.
- $-m$: A negative scalar multiplier or "slope" that determines the strength of the linear bias. This value is typically different for each attention head. The negative sign ensures that increasing distance results in a larger (more negative) penalty.
- $|i-j|$: The absolute difference between the index of the query token (i) and the key token (j). This represents the **relative distance** between the two tokens in the sequence. A larger $|i-j|$ means the tokens are further apart.
- $\sqrt{D_k}$: The square root of the dimension of the key vectors (D_k). This is a scaling factor used to prevent the dot products QK^T from becoming too large, which could lead to very small gradients after the softmax function.
- softmax(...): The softmax function. It normalizes the scaled and biased attention

scores into a probability distribution, ensuring that the attention weights for each query sum to 1.
- V: The Value matrix, as described above. The final attention output is a weighted sum of the value vectors.

Rotary Position Embeddings (RoPE - v3) [9], used in Jina-Embeddings-v3 model, encodes absolute positions with rotation matrices, embedding relative dependencies directly in self-attention:

$$(RoPE(q,i))^T(RoPE(k,j)) = q^T R_{j-i} k$$

(simplified form) where R is a rotation matrix.

- $RoPE(\cdot,\cdot)$: The **Rotary Position Embedding function**. This function takes a vector (query or key) and its absolute position, and applies a rotation to it.
- q: A single **query vector** (before rotation) for a specific token.
- k: A single **key vector** (before rotation) for a specific token.
- i: The **absolute position index** of the query token in the sequence.
- j: The **absolute position index** of the key token in the sequence.
- $RoPE(q,i)$: The query vector q after being rotated according to its absolute position i.
- $RoPE(k,j)$: The key vector k after being rotated according to its absolute position j.
- $(RoPE(q,i))^T(RoPE(k,j))$: The dot product between the rotated query vector and the rotated key vector. This is the attention score between the query and key after RoPE is applied. The rotation is designed such that this dot product inherently encodes relative positional information.
- R_{j-i}: A **rotation matrix** that depends only on the *relative distance* $(j-i)$ between the key and query tokens. The mathematical property of RoPE ensures that the dot product of rotated vectors can be expressed in terms of the original vectors and a rotation matrix based on their relative positions.
- $q^T R_{j-i} k$: This shows the simplified form, indicating that the attention score between q and k is effectively modified by a rotation matrix R_{j-i} that depends on their relative distance.

In addition to this, **Gated Linear Units (GLU - v2)** may replace standard FFNs for enhanced efficiency. For GEGLU:

$$\text{GEGLU}(x) = (xW_1 + b_1) \odot \text{GELU}(xW_2 + b_2)$$

- $\text{GEGLU}(x)$: The output of the **Gated Enhanced Gated Linear Unit** for input x.
- x: The input vector or tensor to the GLU layer, typically the output from a previous transformer sub-layer (e.g., attention output).
- W_1, W_2: **Weight matrices** that transform the input x. These are trainable parameters learned during model training.
- b_1, b_2: **Bias vectors** that are added after multiplication by the weight matrices. These are also trainable parameters.
- $(xW_1 + b_1)$: A linear transformation of the input x. This forms the "main" branch or linear projection.
- $\text{GELU}(\dots)$: The **Gaussian Error Linear Unit** activation function. This is a non-linear activation function applied to the second linear transformation. It introduces non-linearity and is a common alternative to ReLU in modern transformers, known for its smoothing properties.
- $(xW_2 + b_2)$: Another linear transformation of the input x. This forms the "gating" branch, which then passes through the GELU activation.
- $\odot$: The **element-wise product** (also known as the Hadamard product). The output of the first linear branch is multiplied element-wise by the output of the second branch (after GELU). This gating mechanism allows the network to selectively control the flow of information, enhancing efficiency and expressiveness.

2. **Pooling Layer:** Typically mean pooling, aggregating token embeddings into a single fixed-size sentence embedding. The pooling layer is the same as SBERT.

6.2 Training Process (Multi-stage Paradigm)

JinaEmbeddings models employ a sophisticated multi-stage training for high performance across tasks and long contexts.

1. **Stage 1: Pre-training (Masked Language Modeling - MLM) [5]:**

- **Objective:** Train on massive corpus (e.g., C4, CulturaX) to predict masked tokens:

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in \text{masked tokens}} \log P(w_i|\text{context})$$

- **Long Context Integration:** Progressive training on longer sequences (up to 8192 tokens) to facilitate extrapolation.

2. **Stage 2: Fine-Tuning with Text Pairs (Contrastive Learning) [5]:**
   - **Objective:** Learn to map similar texts to proximate points using InfoNCE (Information Noise Contrastive Estimation) loss. For query q and positive passage p^+ from batch B:

$$\mathcal{L}_{\text{InfoNCE}} = -\log \frac{e^{\text{sim}(q,p^+)/\tau}}{\sum_{p_j \in B} e^{\text{sim}(q,p_j)/\tau}}$$

     where $\text{sim}(q,p)$ is cosine similarity and τ is temperature.
   - **Bidirectional Loss:** Often beneficial to calculate loss in both directions.
3. **Stage 3: Hard Negative Fine-Tuning [5]:**
   - **Objective:** Refine distinction between relevant and irrelevant documents, especially challenging hard negatives.
   - **Triplet Loss or Margin-based Loss:** Push hard negatives further away from anchors:

$$\mathcal{L}_{\text{hard\_negative}} = \max(0, \text{sim}(q,p^-) - \text{sim}(q,p^+) + \epsilon)$$

     where $\text{sim}(q,p^-)$ is similarity to a hard negative, and $\text{sim}(q,p^+)$ to a positive.
4. **Task-Specific LoRA Adapters (JinaEmbeddings v3) [9]:** Small, trainable matrices added to transformer layers while main weights are frozen. Enable efficient adaptation for different downstream tasks (retrieval, classification, clustering).

7 Differences Between Sentence-BERT and JinaEmbeddings

Both SBERT and JinaEmbeddings aim to produce high-quality sentence embeddings, but they differ in architectural enhancements, training strategies, and optimizations, leading to performance variations, particularly when compared to models like "all-Mini-LM-v6".

7.1 How JinaEmbeddings Improves over "all-Mini-LM-v6"

"all-Mini-LM-v6" is a distilled SBERT model known for speed and size. JinaEmbeddings (especially v2 and v3) improve upon it for retrieval and ranking:

1. **Longer Context Understanding:**
   - **"all-Mini-LM-v6" Limitation:** Limited to ∼512 tokens, leading to truncation and information loss for long documents.
   - **JinaEmbeddings Improvement:** ALiBi and RoPE enable processing up to 8192 tokens, providing a more complete understanding of long documents, crucial for domains like legal or scientific research.
2. **Superior Retrieval and Ranking through Hard Negative Mining:**
   - **"all-Mini-LM-v6" Training:** Relies on contrastive learning, but might lack sufficiently challenging negative samples.
   - **JinaEmbeddings Improvement:** The explicit **hard negative fine-tuning** stage forces the model to learn fine-grained distinctions between truly relevant and deceptively similar (but irrelevant) documents. This leads to:
     * **Better Recall:** Retrieves more truly relevant documents.
     * **Higher Precision:** Pushes irrelevant, superficially similar documents down the ranking.
     * **Nuanced Understanding:** Develops a deeper semantic understanding beyond simple keyword matching.
3. **Architectural Enhancements (GLU, LoRA):**
   - **"all-Mini-LM-v6" Simplicity:** Prioritizes speed and size with standard components.
   - **JinaEmbeddings Sophistication:** GLU (v2) enhances efficiency and expressiveness. LoRA adapters (v3) offer flexible, efficient adaptation for specific tasks without full model fine-tuning.
4. **Multilingual Capabilities:** JinaEmbeddings v3 is extensively trained on diverse multilingual data, outperforming models not specifically designed for cross-lingual tasks.

8 Reranker: Architecture, Differences, and Relevance Score [10]

A reranker is a model that computes a relevance score using a document and a query. The score computed is high for a document with a contextually relevant query document pair and low for the opposite.

8.1 How a Reranker is Built

Rerankers typically use a **cross-encoder architecture**, differing from the bi-encoder approach of Embeddings.

1. **Input:** Takes a *pair* of texts (query and candidate document) concatenated with special tokens: `[CLS]` Query `[SEP]` Document `[SEP]`.
2. **Transformer Encoder:** This concatenated sequence is fed into a single transformer encoder. Self-attention directly models interactions between query and document tokens.
3. **Output Layer (Scoring Head):** The final hidden state of the `[CLS]` token is passed through a simple feed-forward network to output a single scalar **relevance score**:

$$s_{\text{relevance}} = \text{Linear}(\text{TransformerOutput}([\text{CLS}]\text{Query} \\ [\text{SEP}]\text{Document}[\text{SEP}]))$$

8.2 Training Process for a Reranker

Rerankers are trained to assign high scores to relevant pairs and low scores to irrelevant ones.

1. **Data:** Query-document pairs with relevance labels (binary or graded), often from datasets like MS MARCO.
2. **Loss Function: Pairwise Loss (Ranking Loss)** is the loss function used in training the Jina reranker. It is also the most common loss function used for training rerankers. For a query q, positive d^+, and negative d^-, aims for $\text{score}(q, d^+) > \text{score}(q, d^-)$ by margin margin:

$$\mathcal{L}_{\text{pairwise}} = \max(0, \text{score}(q, d^-) \\ - \text{score}(q, d^+) + \text{margin})$$

3. **Fine-tuning:** Pre-trained transformer backbone weights are updated to specialize in query-document relevance.

8.3 Logic and Math Behind the "Relevance Score" in Rerankers

The relevance score from a Reranker is a learned scalar value quantifying a document's relevance to a query, derived from a deep, contextualized analysis of their interaction.

Logic: The cross-encoder's strength lies in its ability for token-level interaction. When query and document tokens are fed together into a single transformer, the self-attention mechanism computes attention weights across all tokens, including between query tokens and document tokens. This allows the model to learn intricate relationships directly.

Mathematical Formulation: Let query be $Q = [q_1, \ldots, q_L]$ and document $D = [d_1, \ldots, d_M]$. The input sequence is:

$$X = [[\text{CLS}], q_1, \ldots, q_L, [\text{SEP}], d_1, \ldots, d_M, [\text{SEP}]]$$

1. **Token Embeddings:** Each token is converted to an input embedding e_k.
2. **Transformer Encoder Processing:** These embeddings $E_X = [e_1, \ldots, e_{L+M+3}]$ pass through N transformer layers. Crucially, self-attention allows query tokens to attend directly to document tokens and vice-versa. The output of the last layer is $h^{(N)}$.
3. **Relevance Score Calculation:** The representation of the `[CLS]` token from the last layer, $h^{(N)}_{\text{CLS}}$, which encapsulates the combined meaning and interaction, is fed through a scoring head:

$$s_{\text{relevance}} = \mathbf{W}_s \cdot h^{(N)}_{\text{CLS}} + b_s$$

where $\mathbf{W}_s \in \mathbb{R}^{1 \times D_m}$ and $b_s \in \mathbb{R}$ are learnable parameters.

This score is learned during training to reflect relevance. Optional sigmoid activation can normalize it to a $[0, 1]$ range, or softmax for pairwise comparisons during training.

8.3.1 Difference between relevance score in rerankers and cosine similarity in embeddings models

The core difference lies in the **interaction mechanism** and the **nature of the comparison**:

- **Cosine Similarity (Bi-Encoders):** An *indirect* comparison. Query and document are encoded independently. Similarity is a geometric calculation ($\frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|_2 \|\mathbf{v}\|_2}$) between two separate

vectors. No token-level interaction during encoding.

– **Reranker's relevance score (Cross-Encoders):** A *direct* comparison. Query and document are processed as one sequence. The score is a learned output from a model that has seen them *together*, allowing it to capture intricate contextual and semantic relationships through direct token-level interactions. It's a direct prediction of relevance, not a geometric distance.

9 Results with Jina-AI reranker

We used the pre-trained Jina-AI reranker on our datasets to rank candidates according to their suitability for a vacancy and the Jina embeddings v2 model to perform binary classification. The results are given in Table 2. They are all normalized between 0 to 100.

Comparison with Table 1 (Sentence Transformer Results) It is important to note that the scores in Table 1 are normalized between 0 and 100, whereas the Jina-Reranker results in Tables 2 are normalized between 0 and 1. To make a fair comparison, the values in Table 1 can be divided by 100.

After normalization, we observe that:

– **Ranking Metrics:** Jina-Reranker consistently outperforms the Sentence Transformer models across all datasets, particularly in ranking metrics such as NDCG, MRR, and Recall. For example, the NDCG@5 score for Dataset 2 (LLM-enhanced) using Jina-Reranker is **0.2808**, which is higher than the corresponding normalized score from Table 1 (0.39/100 = 0.0039), showing a significant performance boost.

– **Precision and Recall:** Precision@k and Recall@k scores also improve substantially with Jina-Reranker. Sentence Transformers struggled with top-1 accuracy in Table 1 (near zero), whereas Jina-Reranker achieves high Accuracy@1 scores (1.00 across all datasets).

– **LLM-Enhanced Dataset Remains Best:** Both models (Sentence Transformers and Rerankers) consistently show the LLM-enhanced dataset as the top performer, but the performance gap between datasets is more clearly distinguished in the reranker results.

These comparisons further confirm the effectiveness of rerankers for ranking-focused tasks, thanks to their cross-encoder architecture which allows deeper token-level interactions between candidate profiles and job descriptions.

10 Learning-to-Rank

Learning-to-Rank (LTR) is an important researhc topic in machine learning that applies machine learning techniques to solve ranking problems. While traditional machine learning algorithms are primarily designed for classification and regression tasks, they can be adapted for ranking within the LTR framework.

In the LTR setting, the training data $\mathcal{T}$ consists of a set of queries $\mathcal{Q}$. Each query $q \in \mathcal{Q}$ is associated with a list of result documents. For each document, there is a feature vector $\mathbf{x}_d$ (where d denotes a document) and a corresponding relevance label y_d. Hence, the training data is represented as:

$$T = \{(\mathbf{x}_q, \mathbf{y}_q) | q \in Q\}.$$

The goal in a learning-to-rank algorithm is to find a ranking model Φ that can predict the relevance scores $\mathbf{s}$ for all documents in a query:

$$\mathbf{s} = \Phi(\mathbf{x})$$

The generic form of a ranking model is represented with Φ. In practice, Φ can be implemented in different forms. For example, RankingSVM uses a linear model; LambdaMART uses a tree-based model; RankNet uses a neural network model. Though the implementation can be different, all the learning-to-rank algorithms learn Φ using loss functions as their objectives. A loss function is in general defined based on relevance labels $\mathbf{y}$ and predicted scores $\mathbf{s} = \Phi(\mathbf{x})$:

$$\mathcal{L}(\Phi) = \frac{1}{|T|} \sum_{(\mathbf{x},\mathbf{y}) \in T} l(\mathbf{y}, \Phi(\mathbf{x})) \quad (1)$$

where l is the loss function for a single query that takes the labels and scores as input and output a real value as the loss.

$$l : (\mathbf{y}, \mathbf{s}) \to \mathbb{R} \quad (2)$$

Metric	Jina-AI (Pre-trained)	Qwen-3 (Pre-trained)	all-MiniLM-L6-v2
Classification Metrics			
Accuracy (%)	**87.85**	52.67	85.88
F1-Score (%)	22.33	24.74	**27.39**
Precision (%)	14.70	15.34	**23.45**
Recall (%)	46.42	**64.18**	32.93
Information Retrieval Metrics			
Accuracy@1 (%)	31.31	**36.92**	0.00
Accuracy@3 (%)	61.68	**67.29**	0.59
Accuracy@5 (%)	74.30	**78.97**	1.18
Precision@1 (%)	31.31	**36.92**	0.00
Precision@3 (%)	26.32	**27.57**	0.20
Precision@5 (%)	21.50	**23.74**	0.24
Recall@1 (%)	19.32	**23.70**	0.00
Recall@3 (%)	45.27	**47.01**	0.02
Recall@5 (%)	58.00	**60.99**	0.32
NDCG@5 (%)	44.81	**49.22**	0.23
MRR@5 (%)	47.13	**52.87**	0.44
MAP@5 (%)	38.02	**42.39**	0.13

Table 2: Comparison of Re-ranker models with SBERT model on the Inasoft Raw Dataset (Valued between 0 and 100)

The goal of a learning-to-rank algorithm is to find the optimal $\hat{\Phi}$ that minimizes the overall loss:

$$\hat{\Phi} = \arg \min_{\Phi \in \mathcal{R}(\Phi)} \mathcal{L}(\Phi)$$

in the space of ranking models $\mathcal{R}(\Phi)$.

11 Types of Ranking Loss Functions

Ranking loss functions can broadly be categorized into the following approaches:

1. **Pointwise Approaches:** These methods treat each document independently, converting the ranking problem into a classification or regression task. The loss is computed on individual document scores.
2. **Pairwise Approaches:** These approaches define loss functions over pairs of documents. The goal is to correctly order every possible pair of documents for a given query.
3. **Listwise Approaches:** These methods consider the entire list of documents for a query. Loss functions are optimized to directly align the predicted ranking with the ground truth ranking of the complete list, often by optimizing ranking metrics like Normalized Discounted Cumulative Gain (NDCG) or Mean Average Precision (MAP).

12 The Challenge of Optimizing Ranking Metrics

Ideally, Learning to Rank (LTR) algorithms should directly optimize the ranking metrics used for evaluation (e.g., NDCG, MAP, MRR, etc.). However, this is a significant challenge because these metrics depend on the ranks of documents, which are determined by a sorting operation. This sorting makes the metrics non-differentiable (either flat or discontinuous), preventing the use of standard gradient-based optimization methods.

12.1 LambdaRank: A Heuristic Solution

To bridge the gap between smooth loss functions and non-smooth ranking metrics, algorithms like **LambdaRank** were introduced. LambdaRank is a method that incorporates ranking metrics into the training process.

The core idea is to re-weight the document pairs during training based on the change in the desired ranking metric. For a pairwise logistic loss function, instead of treating all mis-ordered pairs equally, LambdaRank scales the gradient for each pair (i, j) by the absolute difference in NDCG (ΔNDCG) that would result from swapping their positions.

The LambdaRank loss for NDCG is:

$$\ell(y, s) = \sum_{y_i > y_j} \Delta\text{NDCG}(i, j) \cdot \log_2(1 + e^{-\sigma(s_i - s_j)})$$

where $\Delta\text{NDCG}(i, j)$ is the change in NDCG when documents i and j are swapped.

Despite its practical success, especially in tree-based models like LambdaMART, LambdaRank lacked a theoretical foundation. The underlying loss function it was truly optimizing remained unknown, raising concerns about its convergence and whether it was genuinely optimizing NDCG.

13 The LambdaLoss Framework

The 2018 paper, *"The LambdaLoss Framework for Ranking Metric Optimization,"* [11] addressed the theoretical gaps in LambdaRank. It introduced **LambdaLoss**, a probabilistic framework that provides a well-defined loss function for LambdaRank.

13.1 Core Idea of LambdaLoss

LambdaLoss defines a loss as the negative log-likelihood of observing the ground-truth relevance labels y given the predicted scores s. It treats the ranked list (π) as a hidden variable and models the probability over all possible permutations:

$$\ell(y, s) = -\log_2 P(y \mid s) = -\log_2 \sum_{\pi \in \Pi} P(y \mid s, \pi) \cdot P(\pi \mid s)$$

The key insight is that this loss, which is generally difficult to optimize, can be minimized using an Expectation-Maximization (EM) algorithm. The paper shows that LambdaRank's procedure is equivalent to an EM algorithm optimizing a specific instance of this LambdaLoss function. This provides the missing theoretical justification for LambdaRank and guarantees convergence to a local minimum.

More importantly, the framework allows for the design of new, metric-driven loss functions that have a clear and direct connection to specific ranking metrics. The paper derives several such losses, including a tighter bound for NDCG, named **NDCG-Loss2**.

13.2 Optimization via Expectation-Maximization (EM)

This loss function is generally not convex and is difficult to optimize directly because of the summation over all possible permutations (Π) inside the logarithm. The paper shows that it can be minimized by the well-known Expectation-Maximization (EM) algorithm, which is an iterative procedure guaranteed to converge to a local minimum.

The EM procedure for LambdaLoss works in two steps at each iteration t:

1. **E-Step (Expectation):** First, the algorithm estimates the distribution of the hidden ranked list, $P(\pi|s)$, based on the current ranking model $\Phi^{(t)}$.
2. **M-Step (Maximization):** Next, it finds a new model $\Phi^{(t+1)}$ that minimizes the loss over the "complete data," where the distribution of π is now known from the E-step. This step is computationally easier because the difficult summation is removed from inside the logarithm.

In practice, calculating the full distribution over all permutations is computationalyy expensive. The paper uses a simplification called a **hard assignment**, where it considers only the single ranked list $\hat{\pi}$ obtained by sorting documents according to the current scores s. The paper then proves that with this hard assignment and a specific definition of the likelihood $P(y|s, \pi)$, the M-step's loss function becomes identical to the LambdaRank loss.Therefore, the iterative LambdaRank procedure is effectively an EM algorithm optimizing a well-defined LambdaLoss function.

14 Optimizing Top-K Metrics with LambdaLoss@K

In practice, system performance is often evaluated using top-K metrics like NDCG@K, which focus only on the highest-ranked results. The original LambdaLoss paper proposed a heuristic for top-K metrics, but it was found to be theoretically unsound and ineffective for neural ranking models.

The 2022 paper "On Optimizing Top-K Metrics for Neural Ranking Models" [6] introduced **LambdaLoss@K**, a novel and theoretically sound loss designed specifically to optimize NDCG@K for neural models.

14.1 The Innovation of LambdaLoss@K

The core innovation is a **correction multiplier**, $\mu_{i,j}$, which appropriately weighs document pairs where at least one item is ranked beyond the cut-off K, instead of setting their weight to zero. The modified pairwise weight, $\delta_{i,j} \odot K$, is defined as:

$$\delta_{i,j} \odot K = \begin{cases} \delta_{i,j}\mu_{i,j} & \text{if } \pi_i > K \text{ or } \pi_j > K \\ \delta_{i,j} & \text{else} \end{cases}$$

The multiplier $\mu_{i,j}$ is based on a modified cost version of the NDCG@K metric and is defined as:

$$\mu_{i,j} = \frac{1}{1 - \frac{1}{D(\max(\pi_i, \pi_j))}}$$

where D is the rank discount function (e.g., $D(r) = \log_2(1 + r)$). This sound formulation contrasts with LambdaRank@K, which is not effective for neural models.

14.2 Key Findings and Limitations

Experiments show that LambdaLoss@K outperforms common losses like Softmax and GumbelApprox-NDCG on benchmark datasets. A surprising finding was that optimizing with LambdaLoss@K can even improve performance on full-list NDCG, suggesting that focusing on the top of the list during training is a beneficial strategy.

The authors identify future work, including extending the framework to other top-K metrics like Recall@K or Precision@K and evaluating its performance on different neural network architectures.

15 How This Can Help Our Project

The principles from these papers are highly relevant for our project of building a recommendation system for job candidates. The goal is a classic top-K ranking problem, where recruiters focus on a small number of top candidates.

1. **Adopt a Top-K Loss Function:** To fine-tune our reranker models, we should use a loss function specifically designed for top-K optimization. The research clearly shows that directly optimizing for a top-K metric with a sound loss function like LambdaLoss@K yields superior results.
2. **Structure Training Data:** To use these advanced loss functions, our data must be in the LTR format. For each job vacancy (query), we need a corresponding list of candidates (documents) with graded relevance labels (e.g., 0 to 4). This structure is essential for the loss function to learn the fine-grained preferences required for high-quality ranking.

By leveraging the theoretically sound and empirically validated LambdaLoss@K framework, we can train our candidate recommendation model to optimize for what truly matters: ranking the most relevant candidates at the top of the list.

References

[1] Guillaume Bied. *Designing Recommender Systems for the Labour Market*. Phd thesis, Université Paris-Saclay, 2024.

[2] Davide Chicco. *Siamese Neural Networks: An Overview*, pages 73–94. Springer US, New York, NY, 2021.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[4] Yingpeng Du, Di Luo, Rui Yan, Xiaopei Wang, Hongzhi Liu, Hengshu Zhu, Yang Song, and Jie Zhang. Enhancing job recommendation through llm-based generative adversarial networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8363–8371, Mar. 2024.

[5] Michael Günther, Jackmin Ong, Isabelle Mohr, Alaeddine Abdessalem, Tanguy Abel, Mohammad Kalim Akram, Susana Guzman, Georgios Mastrapas, Saba Sturua, Bo Wang, Maximilian Werk, Nan Wang, and Han Xiao. Jina embeddings 2: 8192-token general-purpose text embeddings for long documents, 2024.

[6] Rolf Jagerman, Zhen Qin, Xuanhui Wang, Michael Bendersky, and Marc Najork. On optimizing top-k metrics for neural ranking models. In *Proceedings of the 45th International*

ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '22, page 2303–2307, New York, NY, USA, 2022. Association for Computing Machinery.

[7] Mashayekhi, Yoosof and Li, Nan and Kang, Bo and Lijffijt, Jefrey and De Bie, Tijl. A challenge-based survey of e-recruitment recommendation systems. *ACM COMPUTING SURVEYS*, 56(10):33, 2024.

[8] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[9] Saba Sturua, Isabelle Mohr, Mohammad Kalim Akram, Michael Günther, Bo Wang, Markus Krimmel, Feng Wang, Georgios Mastrapas, Andreas Koukounas, Andreas Koukounas, Nan Wang, and Han Xiao. jina-embeddings-v3: Multilingual embeddings with task lora, 2024.

[10] The Jina-AI team. Maximizing search relevance and rag accuracy with jina reranker https://jina.ai/news/maximizing-search-relevancy-and-rag-accuracy-with-jina-reranker/.

[11] Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. The lambdaloss framework for ranking metric optimization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 1313–1322, New York, NY, USA, 2018. Association for Computing Machinery.