UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Guilherme Drummond Lima

Synthetic Data Fine-Tuning for Effective Team Formation in Enterprises

Belo Horizonte
2024

Guilherme Drummond Lima

Synthetic Data Fine-Tuning for Effective Team Formation in Enterprises

Final Version

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Adriano Veloso

Belo Horizonte
2024

[Ficha Catalográfica em formato PDF]

A ficha catalográfica será fornecida pela biblioteca. Ela deve estar em formato PDF e deve ser passada como argumento do comando `ppgccufmg` no arquivo principal `.tex`, conforme o exemplo abaixo:

```
\ppgccufmg{
      ...
      fichacatalografica={ficha.pdf}
}
```

[Folha de Aprovação em formato PDF]

A folha de aprovação deve estar em formato PDF e deve ser passada como argumento do comando `ppgccufmg` no arquivo principal `.tex`, conforme o exemplo abaixo:

```
\ppgccufmg{
     ...
     folhadeaprovacao={folha.pdf}
}
```

Acknowledgments

First and foremost, I would like to express my deepest gratitude to God, whose guidance and blessings have been my constant source of strength and inspiration throughout this journey.

To my parents, Marcos and Angélica, thank you for your unwavering support, encouragement, and sacrifices. Your belief in me has been my driving force, and I am forever grateful for the love and wisdom you have shared. To my brother, Rodrigo, for the countless insights, shared struggles, and unwavering mutual support throughout our academic journeys. Your companionship has made this path less daunting and more meaningful.

To my family, for the unconditional love and support throughout all my journey. Your belief in me has been my anchor, and I am forever grateful for your unwavering presence in my life.

To my advisor, Professor Adriano, for your exceptional guidance, patience, and mentorship since my early years as an undergraduate student. Your wisdom and dedication have shaped not only this work but also my growth as a researcher and individual.

To the people at Dexco, specially João, Wesllei and Glizia, for providing valuable insights and feedback during this research project. Also for providing all the needed structure to deploy and test our models.

Finally, I would like to extend my heartfelt gratitude to all the professors at DCC for their invaluable knowledge, guidance, and inspiration throughout my academic journey. Additionally, I am deeply thankful to the PPGCC staff for their dedication, support, and assistance, which have been instrumental in making this work possible.

"*The best way to predict the future is to create it.*"
(Peter Drucker)

Resumo

Os algoritmos de busca semântica frequentemente dependem de conjuntos de dados generalistas e abertos para treinamento. Embora essa abordagem produza bons resultados, sua eficácia diminui quando aplicada a tarefas específicas de domínio. Portanto, uma estratégia comum para melhorar o desempenho do modelo nesse contexto é o ajuste fino (fine-tuning) de um modelo pré-treinado com dados especializados. Infelizmente, a aquisição de dados específicos de domínio pode ser proibitivamente cara, pois geralmente requer uma grande quantidade de informações produzidas por especialistas da área. O avanço dos Modelos de Linguagem de Grande Escala (LLMs) tornou possível gerar volumes significativos de dados textuais sintéticos, oferecendo fontes custo-eficazes de dados de treinamento. Essa nova abordagem traz desafios relacionados à qualidade e relevância dos dados.

Neste trabalho, avaliamos a eficácia do fine-tuning com dados sintéticos para Busca Semântica em um cenário real do problema de Formação de Equipes Empresariais. Nesse contexto, buscamos identificar o funcionário mais adequado para uma tarefa específica, com base em informações como habilidades, experiências, objetivos, entre outros aspectos. Avaliamos duas estratégias de geração de dados sintéticos: (1) aumento de dados reais com rótulos sintéticos e (2) criação de perfis sintéticos de funcionários adaptados a tarefas específicas. Para medir o impacto dessas estratégias, ajustamos um modelo pré-treinado utilizando as técnicas de Low-Rank Adaptation (LoRA) e Agregação de Rankings. O desempenho do modelo foi comparado com algoritmos de estado da arte em um conjunto de dados curados por humanos. Nossos experimentos demonstram que um modelo treinado com a combinação das duas estratégias de geração de dados sintéticos supera modelos pré-treinados consolidados na tarefa de Formação de Equipes, melhorando as métricas de ranking testadas em uma média de 30% em comparação com o modelo pré-treinado de melhor desempenho.

Palavras-chave: geração de dados sintéticos, Recuperação de Informação, Agregação de Rankings, fine-tuning, Large Language Models

Abstract

Semantic search algorithms often rely on open general-purpose datasets for training. While this approach yields good results, its effectiveness diminishes when applied to domain-specific tasks. Therefore, a common strategy for improving model performance in this task context is fine-tuning a pretrained model with specialized data. Unfortunately, acquiring domain-specific data can be cost-prohibitive, as it typically requires a high amount of data produced by field experts. The advancement of Large Language Models (LLMs) has made it possible to generate significant volumes of synthetic textual data, which provides large sources of cost-effective training data for domain-specific semantic search. This new approach opens new challenges regarding the quality and relevance of the data.

In this work, we evaluate the effectiveness of synthetic data fine-tuning for Semantic Search in a real-world Enterprise Team Formation problem scenario. In this problem, we aim to retrieve the best employee for a given task, given their information regarding abilities, experiences, objectives, and other aspects. We evaluate two synthetic data generation strategies: (1) augmenting real-world data with synthetic labels and (2) generating synthetic profiles for employees tailored to specific tasks. To measure the impact of these strategies, we fine-tune a pretrained text embedding model using Low-Rank Adaptation (LoRA) and Rank Aggregation techniques. We evaluate the model performance against current state-of-the-art algorithms on a human-curated dataset. Our experiments indicate that training a model that uses a combination of both Synthetic data generation strategies outperforms already established pre-trained models on the Team Formation task, improving the tested ranking metrics by an average of 30% in comparison to the best-performing pre-trained model.

Keywords: Synthetic data generation, Information Retrieval, Rank Aggregation, fine-tuning, Large Language Models

List of Figures

List of Tables

Contents

Chapter 1

Introduction

A semantic search system processes text queries to retrieve and rank related documents. This ranking process is based on extracting underlying semantic relationships between the queries and the content of the documents by using text embeddings (Mikolov et al., 2013; Pennington et al., 2014) to calculate the query-document similarities. This technique enables a more nuanced understanding of user intent and context. Search-based Information Retrieval systems often use this type of algorithm for reranking (Nogueira et al., 2019; Ma et al., 2023a) and vector-based search (Johnson et al., 2019). Recently, with the boom of Large Language Models (LLMs), semantic search systems are being used as tools for Retrieval Augmented Generation (RAG) to improve LLM text generation (Lewis et al., 2020b).

Search systems have evolved significantly since their inception. Early systems, such as Boolean Search, used exact term matching between queries and document terms to calculate similarities, struggles with synonyms, context, and user intent. Later, statistical methods, such as TF-IDF (Salton and McGill, 1983) and BM25 (Robertson et al., 1994), significantly improved retrieval results by introducing ways of calculating the relative relevance of each term in the context of individual documents versus its relevance in the entire corpus, which gave the systems the ability to focus on the most relevant words on each query to return the right results. However, these algorithms still could not capture the semantics of a passage, often resulting in incorrect results for queries with more implicit meaning.

The advent of Word Embeddings allowed search systems to measure semantic similarity between vectors rather than a naive lexical overlapping. The arrival of deep learning and transformer-based models like BERT further revolutionized the field, enabling embeddings to capture the meanings of words and their contextual usage within sentences. Decoder-based LLMs have recently been fine-tuned to perform dense retrieval tasks (Ma et al., 2023a; Xiao et al., 2023; Lee et al., 2025; Wang et al., 2024a), achieving the current state-of-the-art (SOTA) results.

Although semantic search algorithms are typically trained on open general-purpose datasets (Wang et al., 2024a), this widely-used approach demonstrates limited effectiveness when applied to specialized domains. One straightforward solution to this problem

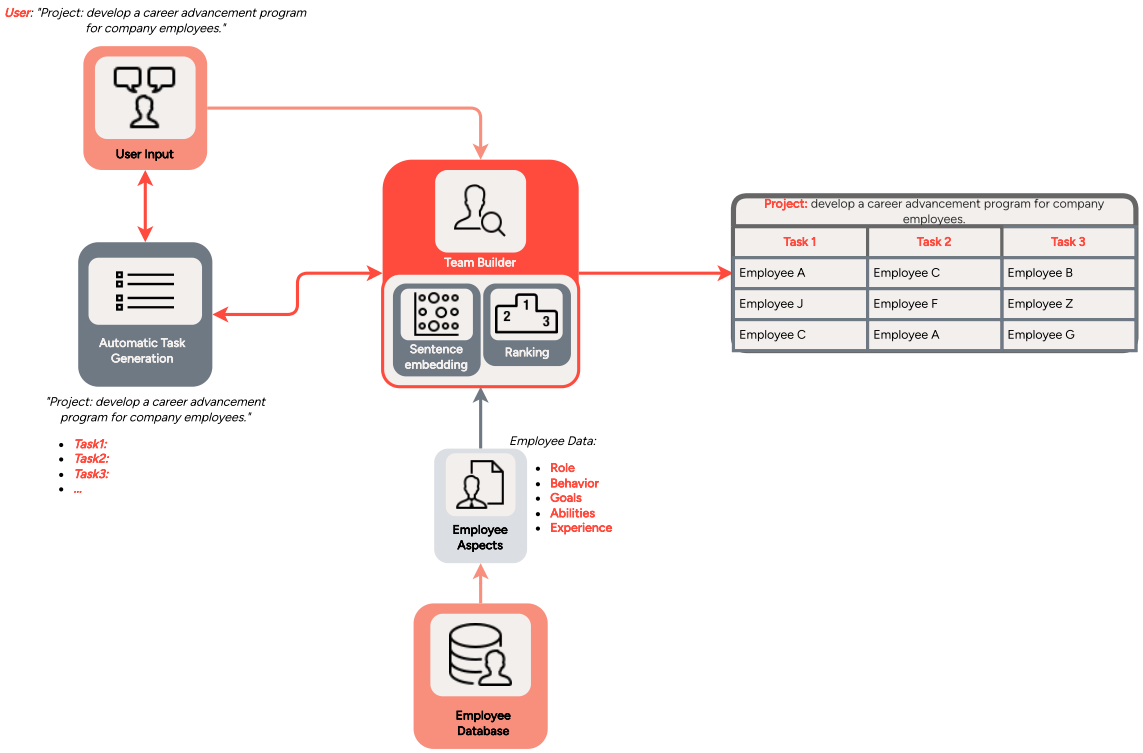

Figure 1.1: Schematic for the full team formation application pipeline. The pipeline starts by a user creating a project. Then, we use an LLM to break it into tasks that can be further edited by the user. To create the team, we extract employee data from an internal database and use a semantic search algorithm to create the rankings for each task.

is to fine-tune pre-trained models on specialized datasets for the specific domain. However, building a training dataset can become expensive, as field specialists often need to generate large amounts of training data for an effective fine-tuning.

With the arrival of LLMs, generating high amounts of high-quality textual data is now possible. This opens possibilities of using synthetic textual data to fine-tune Semantic Search Models for specific contexts. This work will focus on one domain-specific task that uses semantic search: the **Enterprise Team Formation** problem. This problem involves selecting the best employee for a given task based on their abilities, experiences, objectives, and other aspects. Figure 1.1 shows an schematic diagram of our developed Enterprise Team Formation framework. We start by a end-user inputing a project. Then, the system breaks it into a set of tasks that can be edited by the user. The the user is satisfied with the tasks, we then use all available employee data calculate the best rankings for each task.

To create high-quality data for this domain, we need a specialist involved in multiple enterprise contexts who is familiar with all employees' positive and negative aspects. In order to alleviate this dependence on a specialist for labeling the data, we propose two different strategies of synthetic data generation:

1. **Augmenting real-world data with synthetic labels:** In this strategy, we generate synthetic tasks and use a large language model (LLM) to label employees as relevant or irrelevant for those tasks.

2. **Generating synthetic profiles for employees tailored to specific tasks:** For this strategy, for each of the generated tasks, we use an LLM to generate the ideal employee curriculum for that task.

We fine-tune a Qwen2-based (Li et al., 2024) semantic search model (Stella-400M (Zhang, 2024)) using Low-Rank Adapters (Hu et al., 2022). Our experiments compare the fine-tuned models against current SOTA pre-trained models across multiple ranking metrics. The results show that our best fine-tuned model achieved a relative improvement of over 35% in nDCG and 30% in Average Precision compared to strong baselines when tested with real-world data.

1.1 Motivation

The motivation for this study derives from the growing demand for effective solutions in domain-specific semantic search applications, particularly in the enterprise context. We want to fill this gap, especially in team formation, which is a growing demand for enterprises to create high-performance teams. This is a challenging task due to the difficulties of acquiring the data needed to train a model for the enterprise domain.

We also see a growing tendency for synthetic data usage, especially in Natural Language Processing tasks (Dong et al., 2024; Wang et al., 2024a; Yue et al., 2022; Hosseini et al., 2024), due to the availability of LLMs. This new tendency raises the need for research on the effectiveness and reliability of synthetic data in specialized domains. Despite being already proven effective in open-domain applications (Wang et al., 2024a), the use of synthetic data on domain-specific scenarios remains underexplored, specially in an enterprise-related context, where data is sensible and difficult to collect.

The data generation process also remains a challenge. Due to how LLMs are trained, they develop a set of biases, such as high positivity García-Ferrero et al. (2023); Hossain et al. (2020); Truong et al. (2022). This challenge raises questions about how to effectively generate domain-specific data akin to that produced by a field specialist.

By investigating these aspects, this thesis seeks to contribute to the growing body of knowledge on synthetic data applications in NLP and provide actionable insights for enterprises looking to optimize team formation processes using semantic search technology.

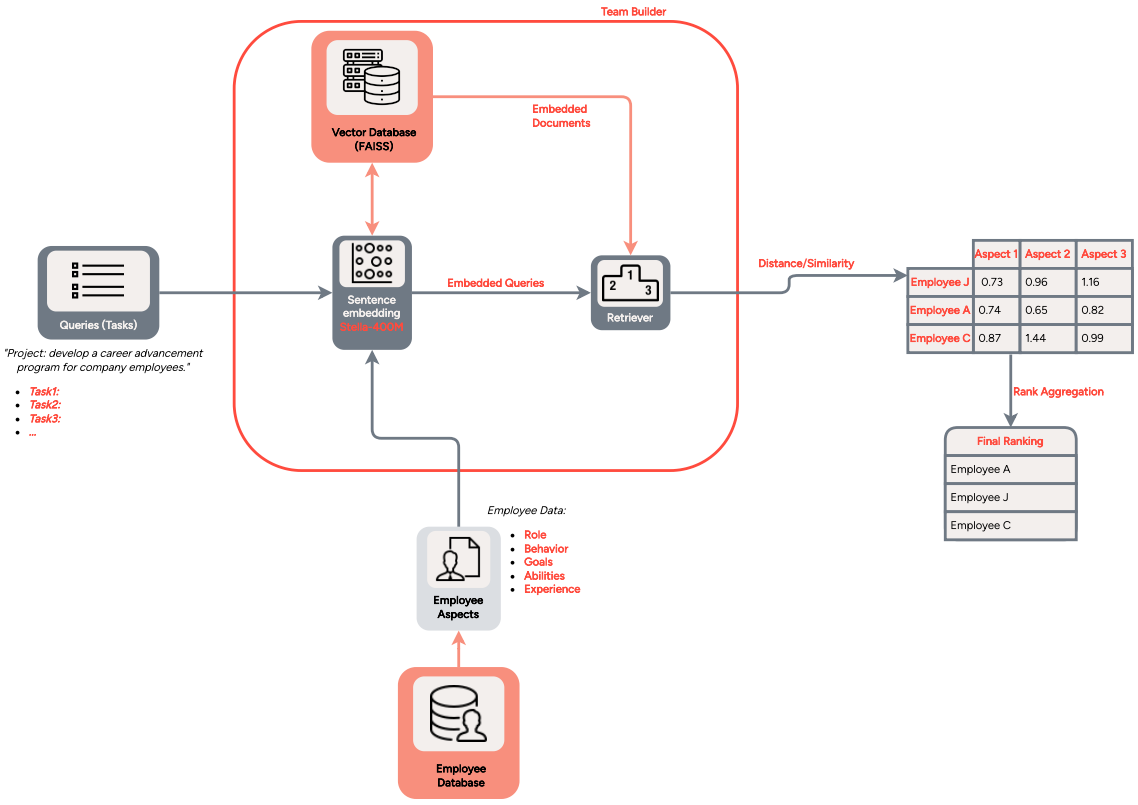

Figure 1.2: The complete semantic search pipeline at inference. First, we use a trained sentence embedding model to create the vector embeddings for both queries (tasks) and documents (employees). the employee vectors are pre-calculated and stored in a FAISS vector database. we We generate multiple rankings (one for each employee aspect) based on the cosine similarity between query and documents. Finally, we use a rank aggregation algorithm to create the final Ranking.

1.2 Problem Statement

We model the Enterprise team Formation problem as a Semantic Search task. This task revolves around finding the most relevant **documents** to a **query** by calculating the semantic similarity between their vector representations in a shared space. To create an effective team for a given project, we assume a different set of tasks that are tied to that project. These tasks can be viewed as queries in the following format:

$$q_{i,j} = \text{“Project: } \{p_i\}\text{. Task: } \{t_{i,j}\}\text{”} \tag{1.1}$$

where each query $q_{i,j}$ is the concatenation of the parent project p_i and a corresponding task $t_{i,j}$.

We model the documents as the available employee data in the enterprise's internal dataset. This data contains employee aspects such as their abilities, past experiences, and

goals. We can create an extensive document that is the textual concatenation of all these aspects, or we can treat them separately, aggregating the multiple generated ranks into a single consensus ranking.

Figure 1.2 shows the complete pipeline for generating the final ranking for a given task. First, we use a Sentence Embedding model to calculate the embeddings for a query and all documents. The document embeddings are pre-calculated and stored in a vector database (e.g. FAISS (Johnson et al., 2019)). Then, we calculate the similarities between query and aspect documents, generating multiple rankings, one for each aspect. These multiple rankings are then aggregated using a Rank Aggregation algorithm to create the final Rank for that given task.

1.3 Context

This project revolves around a partnership with Dexco, a leading Brazilian company specializing in wood paneling, ceramic tiles, and bathroom fixtures. This multinational company has more than 10,000 employees, in which more than 1,000 are employees that occupate strategic roles (Senior-level +). Another challenge we encounter is the wide variety of contexts within the same enterprise such as factories, office, sales, and many others.

For this project, we focus on creating strategic teams, composed of only senior and management-level employees, with the objective of tackling strategic projects.

1.4 Thesis Structure

This thesis is organized in 6 Chapters including this introduction. The remainder of chapters is organized as follows: In Chapter 2, we provide a literature background, presenting all concepts we use thoughout the thesis. Chapter 3 presents relevant prior work that are related to our research. Chapter 4 describes our method and data used during the experiments. In Chapter 5, we present our research questions alongside the experimental results. Finally, Chapter 6 presents a brief summary of all chapters with our conclusions or the study. This chapter also provides possible directions for future work on this theme.

Chapter 2

Background

This chapter introduces the methods and concepts used in this work. We explain all methods and techniques used throughout this thesis. First, discuss the Transformer architecture and its core components. Then, we detail the problem of Semantic Search, introducing its concepts and types of training, such as contrastive learning. We also explain the evaluation metrics we use in this work. After that, we explain parameter-effcient fine-tuning using Low-rank adapters. Finally, we introduce the concept of Rank Aggregation and explain the Condorcet algorithm.

2.1 Transformers

Prior to Transformers, Recurrent Neural Networks (RNNs) and their variants dominated sequence processing tasks. As their name suggests, RNNs process sequence elements sequentially, with each step depending on the previous one. In their architecture, RNNs maintain a hidden state that carries information from previous timesteps to the next one. This design is particularly suited for Natural Language processing tasks, aligning with the sequential nature of text. Later, new architectures such as Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Units (GRUs) (Cho et al., 2014) addressed some innate problems within the recurrent architecture, such as the vanishing gradient problem.

However, despite their dominance, these recurrent architectures had several limitations in regards to scalability. For example, due to the dependency of calculating the previous step to calculate the next one, it was not possible to parallelize both training and inference procedures, which limited these architectures to scale efficiently. Later, Bahdanau et al. (2015) introduced the attention mechanism, which allowed models to focus on different parts of the input sequence simultaneously. This discovery led to the development of Transformers (Vaswani et al., 2017), which leverage self-attention to process sequences in parallel, dramatically improving both training efficiency and model performance.

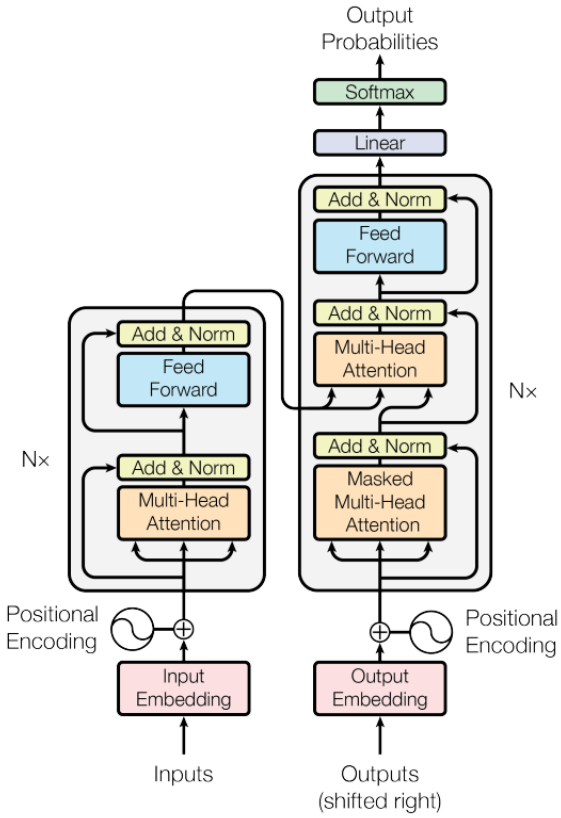

Figure 2.1: Original Transformer architecture - extracted from Vaswani et al. (2017)

The Transformer architecture completely eliminates recurrence, introducing the self-attention mechanism, which allows the model to weigh the importance of every element in a sequence against every other element, capturing both local and long-range dependencies in parallel.

Figure 2.1 illustrates the original Transformer architecture, which consists of an Encoder-Decoder structure with a configurable number of transformer blocks. Subsequent models took different approaches: some, like BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and DistillBERT (Sanh et al., 2020), used only encoder blocks; others, such as the GPT Family (Radford et al., 2018, 2019; Brown et al., 2020), and LLaMA (Touvron et al., 2023), utilized only decoder blocks; while models like T5 (Raffel et al., 2020) and BART (Lewis et al., 2020a) maintained the complete Encoder-Decoder architecture. Below, we examine the core components of a Transformer Block:

Positional Encodings: The self-attention mechanism is permutation-invariant, i.e., it does not consider the position of each token in the text, treating the text sequence as a "bag of words". To encode the order of tokens in the sequence, we add a positional encoding function PE to the original token embedding E to create the Transformer input:

$$\text{Input}_i = E + PE \tag{2.1}$$

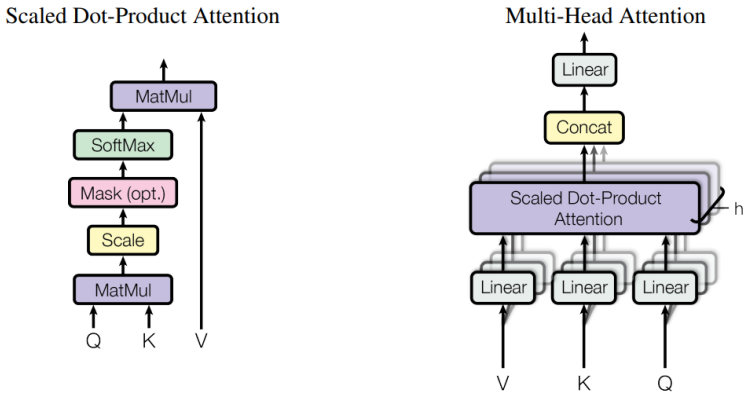

Figure 2.2: Multi-Head Attention mechanism. The mechanism (right) consists of multiple Scaled Dot-Product Attention layers (left) running in parallel. The results are then concatenated. Extracted from Vaswani et al. (2017).

The original Transformer uses Sinusoidal Encoding, which is predefined and deterministic, based on sine and cosine functions of different functions:

$$PE_{(i,2j)} = sin\left(\frac{i}{10000^{2j/d}}\right) \tag{2.2}$$

$$PE_{(i,2j+1)} = cos\left(\frac{i}{10000^{2j/d}}\right) \tag{2.3}$$

where i is the position index, j is the dimension index, and d is the dimension of the embeddings. Later implementations like BERT and GPT use learned positional embeddings. Modern architectures such as Qwen2 (Yang et al., 2024) and LLaMA3 (Grattafiori et al., 2024) adopted Rotary Positional Embeddings (RoPE) (Su et al., 2023b). Instead of directly adding positional information to the token embeddings, RoPE applies a rotation-based transformation to the query and key vectors in the attention mechanism. The rotational structure allows RoPE to encode absolute and relative positions in a way that naturally respects the periodicity and geometric properties of the embedding space.

Multi-Head Attention: Figure 2.2 shows the Multi-Head Attention mechanism. First, we generate the query (Q), key (K), and value (V) matrices. These matrices are the result of individual linear layers with trainable weights W^Q,W^K and W^V, respectively.

After obtaining the Q, K and V matrices, we can apply the Scaled Dot-Product Attention. First, we obtain the **attention scores** by multiplying $Q \cdot K^T$. The intuition behind this operation is to measure the similarity between the query vector and key vectors, which determines the importance of each key-value pair for the query. Then, we scale the scores down by $\sqrt{d_k}$, where d_k is the dimensionality of the key vectors. The resulting scaled score matrix is then passed through a softmax function in order to convert the scores into a probability distribution, ensuring the scores are normalized and sum to 1. These new weights are then aggregated to the value vectors (V), assigning higher

importance to values corresponding to keys that are more relevant to the query. The complete Scaled Dot-Product Attention equation is given as:

$$\textit{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) V \tag{2.4}$$

The Multi-Head Attention mechanism computes the self-attention operation in parallel, using separate learned projections for Q, K, and V in each head. Each head operates in a subspace of the input space, capturing diverse aspects of the relationships between tokens. Then, all head outputs are concatenated and passed through a final output Linear layer W^O, producing the attention output. This can be summarized by the following equations:

$$\text{MultiHead}(Q, K, V) = \text{concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.5}$$

$$\text{head}_i = \textit{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.6}$$

After computing the Multi-Head Attention, we add a residual connection from the input, followed by a layer-normalization step. The result is the input for the feed-forward layer.

Feed Forward: The resulting matrix after the attention mechanism is processed through a Feed-Forward layer with dropout and ReLU activation function. The fully-connected layer is applied individually to each position. The Feed-forward layer can be represented as:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2 \tag{2.7}$$

where x is an individual attention output, W_1 and W_2 are weight matrices of the two linear transformations, and b_1, b_2 are bias terms. Finally, to create the output to the next block, we perform a new residual connection followed by a layer-normalization step.

2.1.1 Large Language Models

The architectural breakthrough of Transformers allowed the construction of large pre-trained models that scale to billions of parameters. These models are often fine-tuned on instruction-following using Reinforcement Learning from Human Feedback (RLHF) (Christiano et al., 2017). Scaling, combined with instruction-tuning, has been shown to enable emergent capabilities in generative tasks that smaller models do not possess.

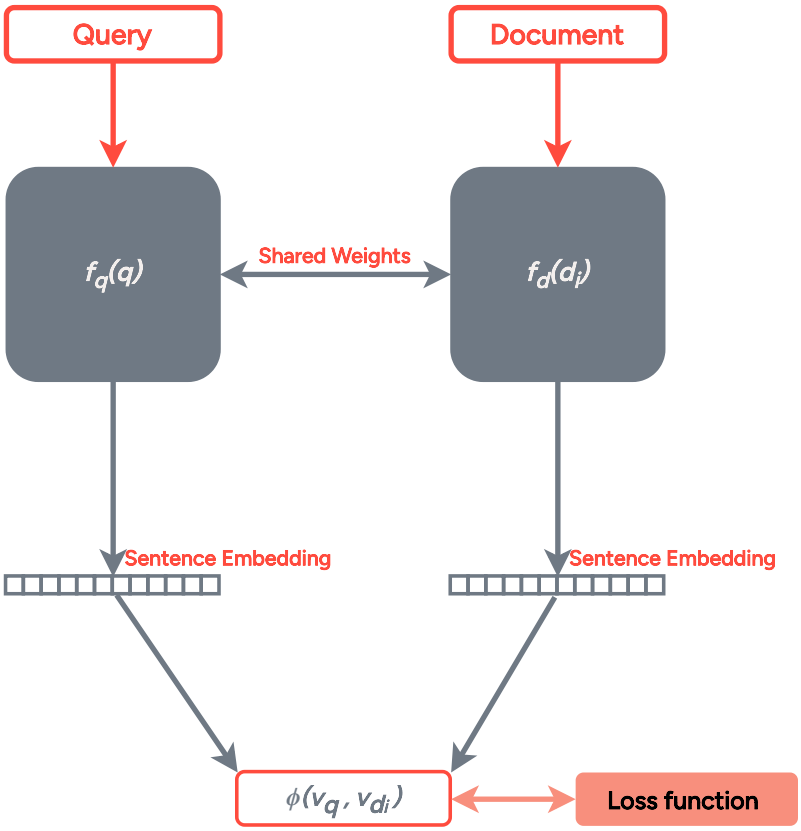

Figure 2.3: Siamese dual-encoder (SDE) architecture. In this setup, queries and documents share the same encoder (usually a neural network).

Recent developments have introduced techniques like Chain-of-Thought (Wei et al., 2022) prompting and Few-Shot Learning (Brown et al., 2020), which enable these models to perform complex reasoning tasks without task-specific fine-tuning.

2.2 Semantinc Search

Semantic Search can be defined as the process of finding the most relevant items to a query by computing the similarity between their representations in a shared semantic space. This often involves embedding both the query and documents into a high-dimensional vector space and using mathematical operations to compute similarity. Formally, given a query text q and a set of N documents $D = \{d_1, d_2, ..., d_n\}$, we map each document d_i into a vector representation v_{d_i} using an embedding function f_d:

$$v_{d_i} = f_d(d_i) \tag{2.8}$$

Similarly, we get the query vector representation v_q of the original query q by using a embedding function f_q:

$$v_q = f_q(q) \tag{2.9}$$

Often, f_q and f_d are the **same function** (e.g., a siamese dual-encoder model) but can be distinct in some cases (e.g., an asymmetric Dual-Encoder). In our implementation, we use the Siamese dual-encoder (SDE) architecture, as it is simpler to train (shared weights) and generally outperforms other dual-encoder architectures (Dong et al., 2022). Figure 2.3 illustrates the fundamental components of the SDE architecture.

After getting the query and document embeddings, as shown in Figure 2.3, we measure the similarity between the query embedding and all document embeddings. the most common approach is to use the cosine similarity function:

$$\phi(v_q, v_{d_i}) = \frac{v_q \cdot v_{d_i}}{||v_q||\ ||v_{d_i}||} \tag{2.10}$$

The training objective in semantic Search can be approached in two main ways. The first approach directly optimizes the cosine similarity between positive query-document pairs using a cosine loss function, which aims to maximize positive query-document similarity scores. However, this method fails to account for negative examples, potentially leading to suboptimal embedding space organization. Alternatively, **contrastive learning** approaches, such as InfoNCE (van den Oord et al., 2019) loss, optimize the relative distances between positive and negative pairs. InfoNCE encourages positive pairs to have higher similarity scores while simultaneously pushing away negative examples in the embedding space. This contrastive approach creates more discriminative embeddings by maintaining clear separation between relevant and irrelevant documents. The loss is defined as:

$$\mathcal{L}\text{InfoNCE} = -\log \frac{\exp(\phi(v_q, v_{d^+})/\tau)}{\exp(\phi(v_q, v_{d^+})/\tau) + \sum_{d^- \in \mathcal{N}} \exp(\phi(v_q, v_{d^-})/\tau)} \tag{2.11}$$

where τ is a temperature parameter that controls the separation between positive and negative examples, v_{d^+} represents the positive document embedding, and $\mathcal{N}$ is the set of negative examples.

Recently, state-of-art Decoder-based Semantic Search models (Lee et al., 2025; Li et al., 2024; Lei et al., 2025) utilize instruction-based fine-tuning (Asai et al., 2023; Su et al., 2023a; Wang et al., 2024a), where the input (more commonly, the queries) is differentiated by concatenating an instruction template to the original text. This allows the models to learn different different types of tasks simultaneously. One basic example would be the following:

$$q_{\text{inst}} = \text{“Instruct: \{template\} \textbackslash n Query: ”} + q \tag{2.12}$$

Where {template} can be any text similarity task. One example could be *“Given a web search query, retrieve relevant passages that answer the query.”*, or *“Retrieve semantically similar text.”*.

2.3 Ranking Metrics

We usually use ranking metrics to measure the retrieval performance when evaluating Semantic Search algorithms. This work focuses on three key metrics: Average Precision (AP), Hit Rate, and Normalized Discounted Cumulative Gain (NDCG).

Average Precision measures the precision at each relevant document position and averages these values, capturing precision and recall aspects in a single metric. For a query q with n relevant documents, AP is calculated as:

$$AP = \frac{1}{n} \sum_{k=1}^{N} P(k) \cdot rel(k) \tag{2.13}$$

where $P(k)$ is the precision at cutoff k, $rel(k)$ indicates if the item at rank k is relevant ($rel(k) = 1$) or irrelevant ($rel(k) = 0$), and N is the number of retrieved documents.

Hit Rate (HR@k) is a simpler metric that measures whether at least one relevant document appears in the top k results. It is particularly useful when evaluating systems where finding any relevant result quickly is important:

$$HR@k = \begin{cases} 1, & \text{if at least one relevant document in top } k \\ 0, & \text{otherwise} \end{cases} \tag{2.14}$$

nDCG incorporates both the relevance and position of documents in the ranked list, with a logarithmic discount factor reducing the weight of lower-ranked items. For a ranking of k documents, nDCG is computed as:

$$nDCG@k = \frac{DCG@k}{IDCG@k} \tag{2.15}$$

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel(i)} - 1}{\log_2(i+1)} \tag{2.16}$$

$$IDCG@k = \sum_{i=1}^{|REL_k|} \frac{2^{rel(i)} - 1}{\log_2(i+1)} \tag{2.17}$$

where REL_k represents the list of relevant documents, ordered by relevance, and $IDCG@K$ is the normalization factor derived from the ideal ranking order, ensuring the metric ranges from 0 to 1. This metric is particularly valuable when documents have graded relevance levels rather than binary relevance judgments.

2.4 Low-rank Adapters

Low-rank Adapters (LoRA) (Hu et al., 2022) is a particular type of parameter-efficient fine-tuning where, instead of updating the entire model's parameters, we decompose each weight matrix into a product of two lower-rank matrices. These adapter matrices are trained while keeping the original model parameters frozen, effectively reducing the number of trainable parameters. Basically, we freeze all of the pretrained network weights and inject two trainable rank decomposition matrices A and B into each of the network's Linear layers. Figure 2.4 shows the comparison between a standard Linear projection layer and a LoRA adaptation of that, now frozen, Linear projection. As the figure shows, we replicate the layer input to the adapter. The adapter output is the multiplication between the input and the two matrices. Finally, we add the adapter result to the original output of the frozen model.

Formally, given a Linear layer with weights $W_0 \in \mathbb{R}^{d \times k}$, an input tensor x, we have the following:

$$h = (W_0 x + b_0) + \frac{\alpha}{r} B_0 A_0 x \tag{2.18}$$

where h is the final layer output, b_0 the frozen layer's bias, α is a scaling parameter, r is the rank parameter, the internal dimension of $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$. Generally, the rank $r \ll min(d, k)$ and α, which is a constant in r, can be viewed as how much the adapter will influence the final result.

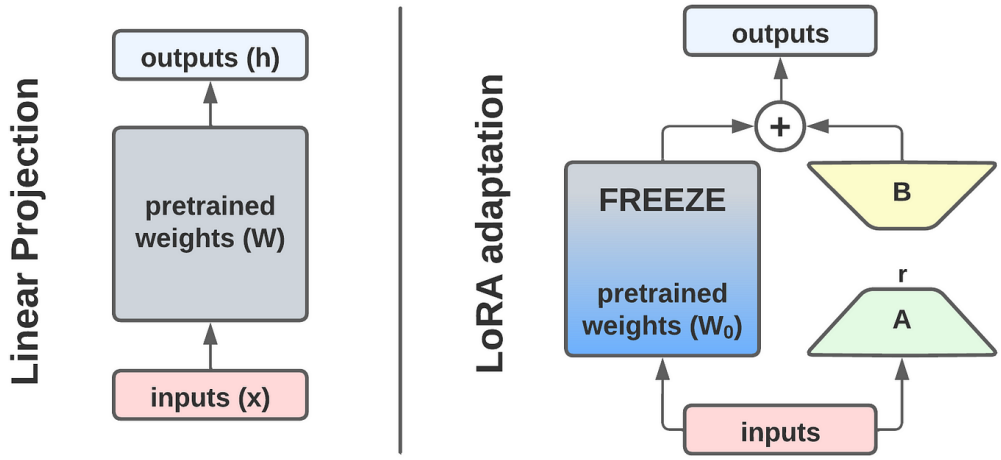

Figure 2.4: Comparison between a standard Linear projection layer (left), and a Linear projection layer with Low-Rank adapters (right). On the Low-Rank Adaptation setup, we freeze the original layer and train two matrices A and B - Extracted from https://pub.towardsai.net

2.4.1 LoRA variants

Several variants of LoRA have been proposed to enhance its capabilities and efficiency. AdaLoRA (Zhang et al., 2023) adaptively allocates parameter budgets across different layers based on their sensitivity to fine-tuning, optimizing the rank of decomposition matrices dynamically. QLoRA (Dettmers et al., 2023) extends the approach by using 4-bit quantization for the frozen model parameters while maintaining full precision for the LoRA parameters, significantly reducing memory requirements. DoRA (Double Low-Rank Adaptation) (Liu et al., 2024) introduces an additional low-rank decomposition to model both positive and negative parameter updates separately, improving performance on certain tasks.

2.5 Rank Aggregation

When dealing with multiple textual features, we may generate a set of different rankings for a given query. Rank Aggregation algorithms leverage this set of rankings, aggregating them into a single final consensus rank (Dwork et al., 2001; Wang et al., 2024b). Formally, given a set of N items to be ranked $U = \{u_1, u_2, ..., u_N\}$, we define an arbitrary ranking $R^t = \{u_i > u_j > ... > u_k\}, i \neq j \neq k$, with $R^t(u_i)$ denoting the

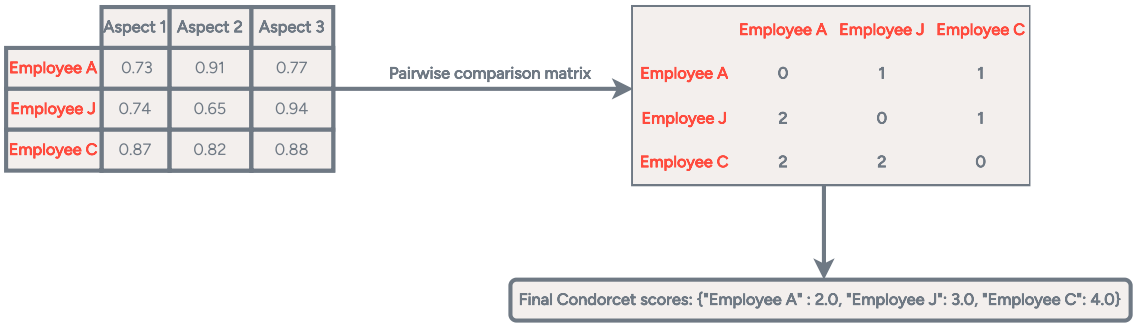

Figure 2.5: Example of a Condorcet rank aggregation using a similarity measure. In this case, Employee C had the biggest Condorcet score, getting a better rank than A and J ($C > J > A$).

position of item u_i in the ranking. Thus, if $R^t(u_i)$ ¡ $R^t(u_j)$, then u_i is more relevant than u_j under R^t. Given a set of M different basic rankings $\mathcal{R} = \{R^1, R^2, R^3, ...R^M\}$, we denote an aggregated consensus ranking R^* as $R^* = f(\mathcal{R})$, where f is a Rank Aggregation function. In this work, we focus on the Condorcet method (de Condorcet, 1785), which has interesting properties such as granting the choice of the overall best item among all ranks when possible (Young and Levenglick, 1978).

2.5.1 Condorcet

The Condorcet algorithm is a Rank Aggregation method derived from the theory of voting systems, where the winner of an election is based on the ranked preferences of voters. The method is based on pairwise comparisons between items. For each pair of items, we calculate how many times one item "won" against each other item, creating a pairwise matrix $\mathcal{M}$ of size (N, N). To calculate the Condorcet scores for each item, we simply sum the column values of each line ($R^*(u_i) = \texttt{sum}(\mathcal{M}[i])$). Figure 2.5 shows an example of Condorcet aggregation using a pairwise comparison matrix. The pair $\mathcal{M}[i, j]$ shows how many times $u_i > u_j$ across all basic rankings. In the end, the item with the highest Condorcet score is the best item, also known as **Condorcet winner**.

A Condorcet winner does not always exist. For example, in cases of cyclic preferences (A beats B, B beats C, C beats A), no single candidate is preferred by a majority in all pairwise comparisons. However, when a set of rankings has a Condorcet winner, it is desirable to choose the Condorcet winner because it ensures the final ranking result aligns closely with the majority preference in direct comparisons, making the winner more representative. When a Rank Aggregation algorithm always chooses the Condorcet

winner (when one exists), we say that it satisfies the **Condorcet property** (Young and Levenglick, 1978; Young, 1988).

2.5.2 Other Rank Aggregation Techniques

Rank Aggregation is an extensive research field that has made progress in recent years (Wang et al., 2024b). Rank Aggregation techniques can be separated into unsupervised and supervised methods. Unsupervised methods, such as Condorcet, Borda Count (de Borda, 1781), and Kemeny-Young (Kemeny, 1959; Young, 1977), rely solely on the input rankings to produce a consensus ranking without learning from historical data. These methods are particularly useful in scenarios where labeled training data is scarce or unavailable.

In contrast, supervised methods require high-quality training data but can provide a better performance (Wang et al., 2024b). These methods often generate the consensus rank R^* using a weighted fusion of $\mathcal{R}$. Some examples of supervised Rank Aggregation methods are the supervised variants of Condorcet (Wu, 2013) and Borda Count (Liu et al., 2007) and neural network-based methods such as CSRA (Yu et al., 2020).

Chapter 3

Related Work

This chapter explores the literature on Semantic Search, Model Improvement through Synthetic Data, and Domain-specific Language Modeling. Semantic Search uses embeddings, natural language understanding, and retrieval techniques to match user queries with the most relevant information or documents, going beyond keyword-based search to capture contextual and semantic meaning. Synthetic Data Generation, meanwhile, aims to improve model performance on specific tasks and improve model generalization through the use of synthetically generated examples. Finally, Domain-specific Language Modeling aims to efficiently adapt pre-trained models to specific domains without losing generalization on general-purpose tasks. Next, we briefly overview these topics and discuss previous work in each area.

3.1 Evolution of Semantic Search Models

Early Information Retrieval methods relied exclusively on individual word occurrence and statistics across the document corpus. The TF-IDF (Salton and McGill, 1983) and BM25 (Robertson et al., 1994) algorithms are the two more famous algorithms.

TF-IDF (**Term Frequency-Inverse Document Frequency**) quantifies the relevance of a term t in a document d relative to a corpus D. It is computed as the product of two components:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \cdot \text{IDF}(t, D), \tag{3.1}$$

where

$$\text{TF}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{3.2}$$

represents the term frequency, i.e., the normalized frequency of t in a document d, and

$$\mathrm{IDF}(t, D) = \log\left(\frac{|D|}{1 + |\{d \in D : t \in d\}|}\right) \tag{3.3}$$

is the inverse document frequency, which decreases the weight of terms appearing frequently across many documents in the corpus. TF-IDF effectively balances the importance of a term within a specific document against its prevalence across the entire corpus.

Meanwhile, BM25 (**Best Matching 25**) builds upon TF-IDF by introducing a probabilistic framework and addressing limitations of term saturation and document length. The BM25 relevance score for a document d with respect to a query q is given by:

$$\mathrm{BM25}(d, q) = \sum_{t \in q} \mathrm{IDF}(t, D) \cdot \frac{f_{t,d} \cdot (k_1 + 1)}{f_{t,d} + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\mathrm{avgdl}}\right)}, \tag{3.4}$$

where:

- $f_{t,d}$ is the frequency of term t in document d,
- $|d|$ is the length of the document d,
- avgdl is the average document length in the corpus,
- k_1 controls term frequency saturation, and
- b adjusts for document length normalization.

BM25 improves upon TF-IDF by ensuring that term frequency contribution saturates and by accounting for the impact of document length, making it more effective in practical search scenarios.

With the advent of word embeddings (Mikolov et al., 2013) and Deep Learning, a set of neural ranking models emerged. The Dual Embedding Space Model (DESM) (Mitra et al., 2016) uses two weight matrices learned by the Continuous Bag-of-Words (CBOW) model from word2vec: the input matrix (IN) and the output matrix (OUT). The CBOW model is trained by maximizing the conditional probability of a word being chosen given the context of neighboring words. The OUT matrix is typically discarded after training, but the paper argues that both matrices contain valuable information. The query is represented in the IN space, while the document is represented in the OUT space. The relevance score is calculated by aggregating the cosine similarities between all query-document word pairs. The document similarity is defined by the centroid of the normalized word vectors in the document.

Another model that utilizes word embeddings for Semantic Search is the Duet model (Mitra et al., 2017). This model addresses the limitations of traditional search models, which rely on exact term matching (i.e., BM25), and recent models that use word

embeddings, arguing that combining both approaches can be more effective. The proposed model consists of two Deep Neural Networks: one operating on local text representations (exact term matching) and the other on distributed representations (embeddings). Joint training of these two networks results in superior performance compared to each network individually and other baseline models. The Duet model consists of a local model and a distributed model. The local model uses the interaction matrix to capture the exact matching of terms and their position in the documents. Meanwhile, the distributed model uses character n-gram embeddings and a convolution and pooling architecture to capture semantic similarity. The two networks are jointly trained to maximize the likelihood of the correct ranking (relevant document above irrelevant documents) for a given query. The combination of the scores from each model produces a final score for ranking.

After the emergence of transformers and BERT, a set of BERT-based models was created, establishing the new state-of-art in Semantic Search at that time. For example, the monoBERT (Nogueira and Cho, 2019) model approaches ranking as a binary classification problem, assessing the relevance of each document individually regarding the query. The query q is fed as sentence A while the document d_i is fed as sentence B. The last hidden layer $[CLS]$ vector is then passed through a single layer MLP to obtain the probability of relevance of d_i in regards to q. The final rank is obtained by sorting the items by their predicted probability score. MonoBERT is trained using the cross-entropy loss:

$$L_{mono} = -\sum_{j \in J_{pos}} log(s_j) - \sum_{j \in J_{neg}} log(1 - s_j) \tag{3.5}$$

where J_{pos} and J_{neg} are the set of indexes of the relevant and irrelevant candidates, respectively.

Later, a new variation of the monoBERT was created, the duoBERT (Nogueira et al., 2019). duoBERT extends monoBERT to a pairwise approach, estimating the probability of a document being more relevant than another. The approach is similar to monoBERT's, using the query as sentence A, a candidate document d_i as sentence B, and another candidate document d_j as sentence C. Then, the $[CLS]$ token vector, like monoBERT, is fed to an MLP. duoBERT's loss is slightly different:

$$L_{duo} = -\sum_{i \in J_{pos}, j \in J_{neg}} log(p_{i,j}) - \sum_{i \in J_{neg}, j \in J_{pos}} log(1 - p_{i,j}) \tag{3.6}$$

where $p_{i,j}$ is pairwise probability score of d_i being more relevant then d_j. To obtain the final scores, the results are aggregated so that each document gets an individual score s_i. These two models were proposed as a multi-stage reranking pipeline, where the monoBERT is applied to the top N BM25 ranking results, and the duoBERT is applied to the top $k \ll N$ results from monoBERT.

With the arrival of LLMs, many researchers tried to adapt their architectures to semantic search. RankLLaMA and RepLLaMA (Ma et al., 2023a) are some of the early examples of adapting LLMs to text retrieval in a multi-stage retrieval pipeline. RepLLaMA adopts the same bi-encoder architecture proposed in DPR (Dense Passage Retrieval) (Karpukhin et al., 2020) but initializes the base model with LLaMA. The architecture consists of a dual-encoder network with some adaptations for LLaMA's Decoder-only architecture. First, an `eos` token `</s>` is appended to the input queries and documents. This token has a similar role as BERT's $[CLS]$ token, encoding the general meaning of the whole passage. Therefore, a vector embedding produced by RepLLaMA can be viewed as:

$$V_T = \text{Decoder}(\text{`}t_1 \sqcup t_2 \sqcup \ldots \sqcup t_k \texttt{</s>}\text{'})[-1] \tag{3.7}$$

where Decoder() represents the LLaMA model's last hidden layer representations of tokens $t_1 ... t_k$`</s>`. Then, the `</s>` vector representation is used to calculate the similarity between queries and documents. RepLLaMA is trained using the InfoNCE contrastive loss.

Meanwhile, RankLLaMA is trained as a pointwise reranker, similar to monoBERT. Basically, we pass a query and a document together as input, and the model's output is a score that indicates the probability of relevance of document d within query q. RankLLaMA's input prompt and similarity calculations are defined as follows:

$$\text{input} = \text{`query: } \{q\} \text{ document: } \{d\}\texttt{</s>}\text{'} \tag{3.8}$$

$$\text{Sim}(q, d) = \text{Linear}(\text{Decoder}(\text{input})[-1]) \tag{3.9}$$

where Linear() is the linear projection of the `</s>` vector to a one-dimensional scalar score. Similar to RepLLaMA, RankLLaMA uses a contrastive loss function for training. Experimental results using the MS MARCO and BEIR datasets showed that RepLLaMA and RankLLaMA outperformed smaller models in both retrieval and reranking tasks. The complete RepLLaMA–RankLLaMA pipeline achieved state-of-the-art results, demonstrating strong zero-shot effectiveness.

RepLLaMA and RankLLaMA, along with other LLM-based pioneer approaches, paved the way for multiple LLM-based models that further achieved state-of-the-art results in Semantic Search tasks.

The bge-en-icl (Li et al., 2024) model leverages in-context learning (ICL) to generate more adaptable text representations by incorporating task-specific examples directly into the query prompt. Unlike approaches that modify the model's architecture, this model retains the original architecture, utilizing the inherent ICL capabilities of LLMs and using the [EOS] token from the final layer as the vector representation.

The model was trained using a dynamic sampling method that provides a variable number of examples for the query during training, enhancing ICL capabilities while maintaining zero-shot performance. This training strategy, combined with batch examples to improve the model's ability to distinguish between examples and inputs, enables bge-en-icl to achieve state-of-the-art (SOTA) results on the MTEB and AIR-Bench benchmarks.

Stella-400M (Zhang, 2024) achieves a high rank on the MTEB benchmark, with only 400M parameters. The model is a distillation of gte-Qwen2-1.5B-instruct (Li et al., 2023) and uses a dual-encoder architecture with mean-pooling over the last hidden layer. In this work, we use Stella as our base model. Therefore, we dive into its architecture in Chapter 4.

The LENS-d8000 model (Lexicon-based EmbeddiNgS) (Lei et al., 2025) was developed to generate lexicon-based text embeddings using large language models (LLMs). Lexicon-based embeddings are basically sparse vector representations of the passage, where each dimension corresponds to a token in the vocabulary. It is trained in a Masked Language Modeling way, where we predict the likelihood of a token being relevant to each position. Research shows that lexicon-based embeddings can be used as a complement of dense embeddings (Chen et al., 2024) and even outperform dense embeddings on some tasks (Déjean et al., 2023; Formal et al., 2021).

However, this approach has some drawbacks, such as the high dimensionality. This high dimensionality is caused by a set of things, such as tokenizer subword fragmentation ("`education`" into "`edu`" and "`cation`"), token redundancy ("What", "what" and "_what" as distinct tokens), and uncommon tokens that rarely appear on the corpus. To address this, LENS clusters the original tokens, using their centroid embeddings in order to replace the original token embeddings of the LM head.

LENS's training follows a single-stage training procedure and relies exclusively on publicly available data, using an approach similar to bge-en-icl. Its architecture uses Mistral-7B as a base model, applying bidirectional attention and LoRA. For training, the standard InfoNCE Loss is used. LENS achieves state-of-the-art results on the MTEB benchmark on both single and combined (LENS+bge-en-icl) setups.

Another model that reaches the top ranks in MTEB is the NV-Embed-v2 (Lee et al., 2025) model. It presents architectural innovations, namely the latent attention layer and the removal of the causal attention mask. It also performs a two-stage training procedure. The latent attention layer is a new way of obtaining the grouped embeddings that depart from the classic mean pooling or last token embedding. For the causal attention mask removal, the researchers found empirically that removing the attention mask during contrastive learning led to enhanced results.

NV-Embed-v2's training is divided into two phases. The first phase uses contrastive training with instructions on retrieval datasets, applying in-batch negatives and hard negative examples.

The second phase combines retrieval and non-retrieval data (classification, clustering, textual similarity), disabling in-batch negatives to improve accuracy in non-retrieval tasks. The paper states that this approach improved the performance on both retrieval and non-retrieval tasks. Currently, NV-Embed-v2 is the overall best model on the MTEB benchmark for text embedding-related tasks.

3.2 Improving Language Models with Synthetic Data

Following the rapid advancement of LLMs, synthetic textual data have been increasingly used to improve the performance of various NLP tasks. Synthetic data has been used directly in the training of Semantic Search models to improve retrieval performance, used as feedback for reward-based training, and even to prevent Language Models from hallucinating. Next, we briefly review some of these works.

Wang et al. (2024a) explore the use of synthetic data to improve the performance of text embeddings on general semantic search tasks. The main idea of the paper is to simplify text embedding's training pipeline by using synthetic data generated by LLMs. Basically, they use proprietary LLMs to generate a diverse corpus of synthetic text embedding-related tasks across 93 different languages using a brainstorming prompt. Then, they use this task corpus as input for another set of prompts to generate a user query, a positive document, and a negative document for a specific task. In total, they generated 500k synthetic examples.

For training, they perform an instruction-based fine-tuning approach to a Mistral-7b (Jiang et al., 2023) model using Contrastive Learning on the generated pairs. To represent the sentence embedding, an $[EOS]$ token is appended to the end of each text example. Then, the query-document similarity is measured by taking the last layer $[EOS]$ vector. In the training process, they apply the standard InfoNCE Loss. Their approach surpassed previous state-of-art models on the MTEB benchmark (Muennighoff et al., 2023), outperforming the baseline models by 2.4 points.

Another example of synthetic data's application in Language Modeling is the Online AI Feedback (OAIF)(Guo et al., 2024), which tries to solve the *off-policy* problems from Direct alignment from preferences (DAP) methods. The most popular DAP method is Direct Preference Optimization (DPO), which is an alternative to the standard RLHF on adjusting a Language Model to human preferences. In short, DPO simplifies the process of learning preferences by optimizing the Language Model directly to adhere to

human preferences without having to rely on an explicit reward model or reinforcement learning. However, the training of DAP models usually uses datasets that are labeled prior to the training itself, causing the model to not get feedback on its own generations during training. To overcome this off-policy issue, the authors propose the OAIF methods.

The OAIF method uses an LLM as an annotator to provide online feedback on each training iteration. Basically, the OAIF involves sampling two responses from the current policy, getting online feedback from an LLM annotator that indicates which response is preferable, and finally, updating the model based on the online AI feedback. This method shows promising results, outperforming both DAP and RLHF methods.

Another method that utilizes Synthetic data to align LLMs to human preferences is the SynPO (Dong et al., 2024). SynPO is an iterative process that induces the LLM to create high-quality synthetic data to train itself over each iteration. The process starts with an initial policy model π_{θ_0}, a small Supervised Fine-Tuning (SFT) dataset to guide the generation of synthetic preference data, a list of keywords $\mathcal{K}$ and a prompt generator $\mathcal{G}$. At each iteration t, $\mathcal{G}$ creates new prompts $\{x_i\}_{i=0}^{m}$ using the keywords sampled from $\mathcal{K}$. The generator (which is the LLM itself) is initially trained on the SFT dataset. At each iteration, the LLM is trained as a response improver $\mathcal{R}_t$ to identify discrepancies between the model's response and gold standard responses in the SFT data.

With all components ready, the iteration starts by generating responses from the previous model $\pi_{\theta_{t-1}}$ for the synthetic prompts x_i. These responses are then improved by $\mathcal{R}_t$. The original and improved responses are used as preference pairs in the training of the next model π_{θ_t}. The preference pairs are filtered, maintaining only the ones with a significant difference in quality. The model optimizes the SimPO (Meng et al., 2024) objective at each iteration.

SynPO demonstrated significant performance improvements on Llama3-8B and Mistral-7B, substantially enhancing their instruction-following capabilities. In evaluations using AlpacaEval 2.0 and ArenaHard benchmarks, the method achieved win rate improvements of over 22.1%. Simultaneously, SynPO also improved the general performances of both LLMs, achieving 3.2% to 5.0% performance improvements on the Open LLM Leaderboard (Beeching et al., 2023).

Another use for synthetic data in LLMs is for reducing hallucinations. *SYNTRA* (Jones et al., 2024) is a method that creates synthetic tasks where model hallucinations are frequent and easy to measure and optimizes the LLM's system message (instead of the model weights) using this task using prefix-tuning (Li and Liang, 2021). Then, this optimized message is transferred to real-world tasks.

Evaluating hallucinations in LLM outputs is challenging, expensive, and prone to errors, which makes direct optimization difficult. *SYNTRA* proposes the use of tasks where hallucination is easy to evaluate automatically, and LLMs tend to hallucinate frequently on them. The task the paper uses as an example is the name retrieval task:

given a list of names, the LLM's task is to retrieve the first n names that start with a specific character. To train the model, a continuous postfix is appended to the system message, and it is optimized using prefix-tuning. This postfix is a set of embeddings that are learned through gradient descent.

The method is then used in two LLMs (Vicuna (Chiang et al., 2023) and Orca (Mukherjee et al., 2023)) and evaluated in three realistic tasks: web search (MS MARCO (Bajaj et al., 2018)), meeting summarization (QMSum (Zhong et al., 2021)), and automated clinical report generation (ACI-Bench (wai Yim et al., 2023)). Results show, on average, a reduction of 29% (7.5 points) in the hallucination rate of Orca and a 7% (2.5 points) decrease in Vicuna. For the ACI-bench, the decrease is even bigger, reducing the hallucination rate by 16 points on Orca and 8 points on Vicuna.

3.3 Domain-specific Language Modeling

Adapting Language Models to specific domains is a critical challenge in natural language processing. This involves techniques that refine pre-trained models to excel in specialized contexts. Next, we highlight some of these approaches.

Mixture-of-Domain-Adapters (MixDA) (Diao et al., 2023) is a method for efficiently adapting pre-trained Language Models to specific domains without doing the full model fine-tuning. MixDA aims to mitigate the high computational costs associated with full fine-tuning or continual pretraining while retaining the prior knowledge of the pre-trained model. The main idea is to decouple the feed-forward networks (FFNs) in the Transformer architecture into two parts: the original pretrained FFNs to preserve the knowledge of the original domain and domain-specific adapters to inject domain-specific knowledge in parallel. A mixture-of-adapters gate mechanism is used to fuse the knowledge from different domain-specific adapters dynamically.

MixDA employs a two-step adapter tuning strategy: (1) tuning domain-specific adapters on unlabeled data and (2) tuning task-specific adapters on labeled data. Experimental results demonstrate that MixDA achieves superior performance in in-domain, out-of-domain, and knowledge-intensive tasks. The model was tested against five baselines: Houlsby (Houlsby et al., 2019), Pfeiffer (Pfeiffer et al., 2020), LoRA (Hu et al., 2022), Prefix-Tuning (Li and Liang, 2021) and full fine-tuning. The models were tested on in-domain tasks requiring general-domain knowledge, out-of-domain tasks requiring domain-specific knowledge, and knowledge-intensive tasks requiring commonsense knowledge. Overall, MixDA outperformed all tested baselines across all task types, achieving an average improvement of 2.5% on in-domain tasks, 4.8% on out-of-domain tasks, and

5.0% on knowledge-intensive tasks (all over the best-performing baseline).

Question Value Estimator (QVE) (Yue et al., 2022) addresses the problem of adapting question-answering (QA) models to new domains where annotated training data is scarce. The proposed solution is a novel method for selecting automatically generated synthetic questions that maximize the QA model's performance in the target domain. The process involves generating synthetic question-answer pairs and selecting the highest-quality ones for training the QA model. In order to perform the selection, a Question Value Estimator (QVE) is used, which evaluates the utility of each synthetic question for training the QA model. The QVE is trained using direct feedback from the QA model in the target domain, meaning it learns to select the synthetic questions that most contribute to improving the QA model's accuracy in the target domain.

The modeling pipeline is divided into three main steps: synthetic data generation, QVE training, and the selection and use of synthetic data. First, a question generation (QG) model is pretrained on a source domain with large amounts of QA data. This QG model is then fine-tuned using the annotations available from the target domain. This enables the QG model to generate synthetic questions based on contexts from the target domain. However, the generated synthetic QA pairs may be of low quality, highlighting the need for a selection process.

The QVE model is then trained to separate the good and bad generated pairs. QVE takes a synthetic QA example (context, question, and answer) as input and produces a score indicating the potential of that example to improve the QA model's performance in the target domain. QVE is a BERT-based model that concatenates the context, query, and answer and encodes in the following way:

$$h = BERT[\texttt{<CLS>}q\texttt{<ANS>}a\texttt{<SEP>}c] \tag{3.10}$$

where h is the hidden representation derived from the `<CLS>` token.

The training of the QVE is formulated as a reinforcement learning problem where the model receives synthetic examples and assigns a score to each question, indicating its potential value. The QVE then selects questions for QA model training using Bernoulli sampling. The selected questions are used to train the QA model, and the improvement in QA performance (measured in terms of exact match (EM) gain) serves as the reward for training the QVE. This reward guides the optimization of the QVE, enabling it to estimate and select the value of each question accurately. The QVE can achieve performance comparable to fully supervised models with only 15% of the human annotations in the target domain.

Another interesting example involving synthetic data and domain-specific tasks is the MathGenie model (Lu et al., 2024). MathGenie is a model designed to generate synthetic and reliable math problems to enhance the mathematical reasoning of LLMs. To

achieve this goal, it combines iterative solution augmentation, question back-translation, and solution filtering based on verification.

Iterative Solution Augmentation: Instead of directly augmenting the questions, MathGenie starts with existing solutions to math problems. These solutions are iteratively augmented by a model (M_{text}) fine-tuned with specific prompts to create new solutions. This process is repeated multiple times, generating multiple augmented solutions. The iteration ensures that the augmented solutions gradually deviate from the original solutions, increasing diversity. The initial solutions come from datasets such as GSM8K and MATH.

Question Back-Translation: The augmented solutions are then back-translated into new math questions by a model ($M_{\text{backtrans}}$) trained on inverted question-solution pairs. This step leverages the constraints and logical relationships in the mathematical solutions to create a diverse and high-quality set of new problems. Question back-translation is more reliable than direct question augmentation because it takes advantage of the constraints present in the solutions.

Verification-Based Solution Filtering: The new questions generated do not have reliable reference solutions, so MathGenie employs a model (M_{code}) to generate integrated code solutions and verify their correctness. The M_{code} model generates integrated code solutions and also verifies them through verification justifications. This model is trained on data with integrated code verification justifications. The verification uses natural language interspersed with code to ensure accuracy. Solutions verified as correct are retained, improving the quality of the generated data. The initial filtering also removes questions whose solutions lead to different answers.

Then, models such as LLaMA-2, CodeLLaMA, Mistral, InternLM2, and Mixtral-8x7B are fully fine-tuned and tested on in-domain and out-of-domain datasets. Overall, the MathGenieLM-InternLM2 model achieves 87.7% accuracy on GSM8K and 55.7% on MATH, representing the best overall performance among open-source language models.

Chapter 4

Data and Method

In this chapter, we present the data, the proposed method, and its implementation details. Then, we present the base model and our fine-tuning technique using Low-Rank Adaptation and contrastive learning. We also explain our contrastive learning setup, including the loss function used and how we set up the positive and negative examples. After that, we focus on how we generate synthetic data for both synthetic labeling and synthetic curriculum generation techniques. Later, we show how we aggregate the multiple rankings generated by the different aspects in the dataset using the Condorcet and Borda Count methods.

Both training procedures and rank aggregation techniques were implemented in Python. For the fine-tuning script, we use Pytorch 2.4.1 and Huggingface Transformers 4.43.3. For the LoRA implementation, we use PEFT 0.12.0. The rank aggregation algorithm was written using Numpy 1.26.4.

4.1 Data

This section presents the real-world data utilized in this work. It includes employee information from Dexco, specifically focusing on individuals in senior management roles within the enterprise, which are strategically important to the company. This dataset includes 835 employees. We also create a task-employee human feedback dataset from scratch for our proposed methods to use as an evaluation dataset.

4.1.1 Employee data

We collect multiple textual features for each employee, which are referred to as *aspects* in this work. These aspects refer to abilities, past working experiences, behavior, and goals. This data was collected by Dexco's Human Resources team using a set of forms and internal databases. Next, we detail each of these aspects.

- Role: The employee's role in the company alongside a description of their responsibilities.
- Behavior: The employee's most recent behavior evaluation. The direct boss of each employee performs this evaluation yearly.
- Goals: The established goals for each employee. This aspect has a mixture of personal goals and goals established by the bosses.
- Abilities: Relevant professional abilities that the employee has.
- Experience: The employee's previous professional experiences within or outside the company.

This dataset includes 835 employees with strategic roles, such as coordinators and managers. Although these individuals represent a small portion of the company, optimizing team performance in these roles has an outsized impact on the organization's overall success. In total, there are 280 Coordinators, 247 Senior employees, 222 Supervisors, 83 Managers, and 3 Directors.

4.1.2 Human Feedback

To validate our model's performance on real-world applications, we have established a human-curated evaluation dataset alongside our synthetic training data. We built this curated dataset using two distinct applications where domain experts could build teams for their desired project or label the relevance of specific employees for a given task.

The first application, shown in Figure 4.1, uses the pretrained Stella-400M model to generate the rankings for each task. The user gives a potential project and its related tasks as input (with the option to autocomplete tasks using GPT-4o). For each of the

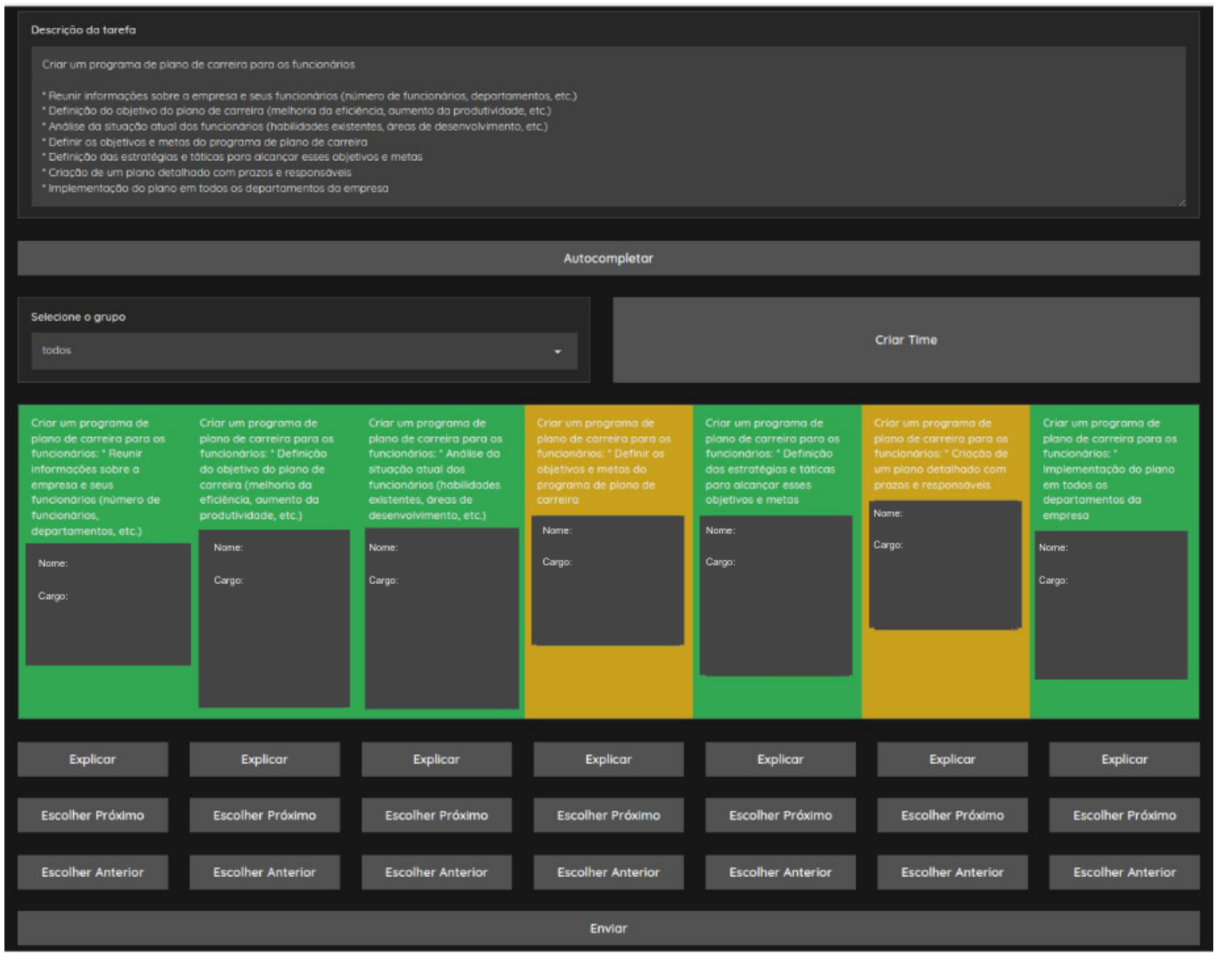

Figure 4.1: Team Builder application. First, the user inputs a project and its associated tasks (with the option to use an LLM to generate them). Then, we generate the best-ranking team with the top-1 of each ranking. The user can navigate each ranking, "correcting" the algorithm. The results are then stored to create pairs for the evaluation dataset. All employee information was omitted in this image.

tasks, the algorithm creates a rank with the most suitable employees for each task. In the application, only the first employee of each rank is shown to the user. If the domain specialist disagrees with the algorithm's decision, she can navigate through the complete ranking to get the most suitable employee for each task.

The interaction data was transformed into query-document pairs, with selected employees marked as relevant and replaced employees marked as irrelevant. Although this data collection approach was methodologically sound, the domain experts found it overwhelming to manage multiple rankings simultaneously while also having to provide relevant and creative projects. These limitations resulted in a small dataset of approximately 50 labeled pairs.

To address these data collection limitations, we developed an alternative approach using a more straightforward application, shown in Figure 4.2. This refined approach presents evaluators with randomly selected tasks from our synthetic dataset and associated

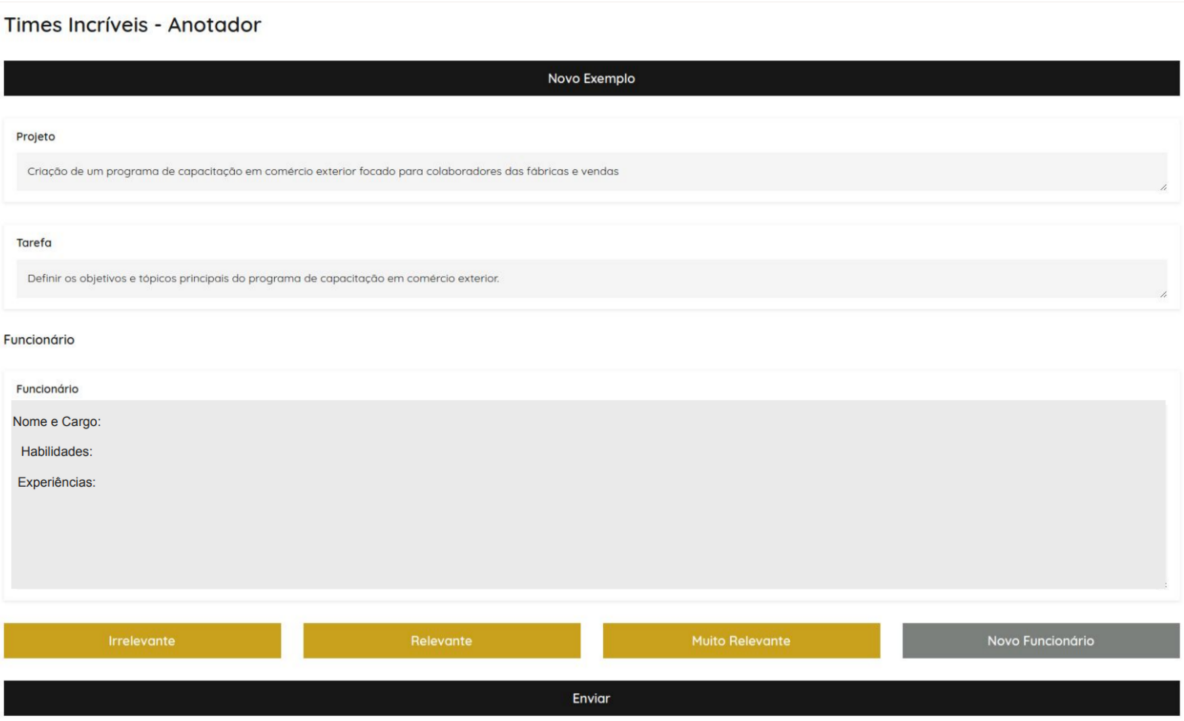

Figure 4.2: Annotator aplication. Here, we present the user a task and an employee. The user labels this pair as relevant/irrelevant. All generated pairs are used on our evaluation dataset. All employee information was omitted in this image.

employee candidates. For each task, the candidate employee is selected through a balanced sampling strategy: 50% probability of random selection from the employee pool and 50% probability of selection from the top 10 candidates ranked by our base model. Then, the domain specialist has to label the chosen employee as relevant/irrelevant. They can also label multiple employees for the same task by choosing to get a new sample employee. This new approach greatly improved the amount of data collected, resulting in a dataset of over 500 labeled pairs.

4.2 Model

We use Stella-400M (Zhang, 2024) as our pretrained base model and use LoRA (Hu et al., 2022) for a parameter-efficient fine-tuning for the Enterprise Team Formation task. Stella-400M, derived from gte-Qwen2-1.5B-instruct (Li et al., 2023), uses *Knowledge Distillation* (Hinton et al., 2015), a neural network compression technique where a more compact model, known as the student, tries to reproduce the behavior of a larger model, known as the teacher. For text embedding models, Distillation loss is calculated by maxi-

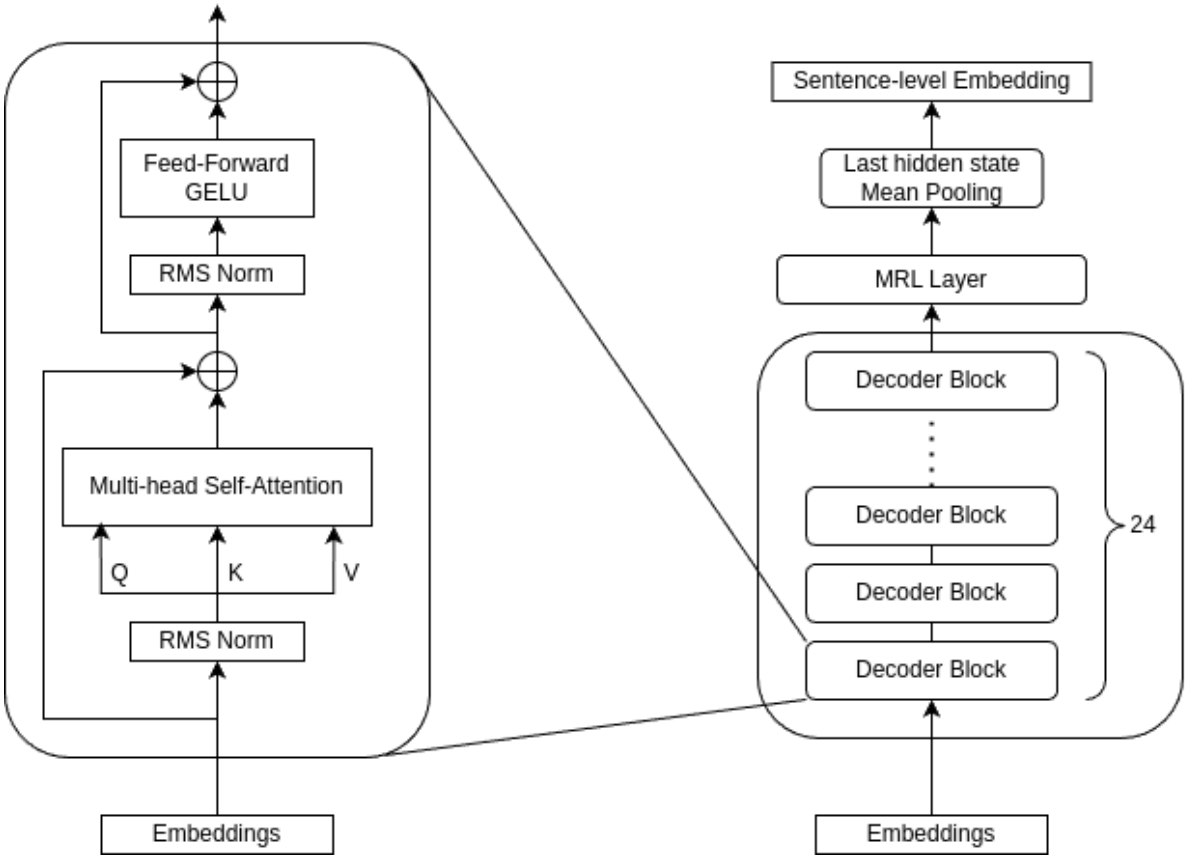

Figure 4.3: Stella-400M architecture. The zoom (left) shows the detailed decoder block, while the right side shows the complete architecture, with 24 Transformer Decoder layers.

mizing the similarity between the embeddings generated by the frozen teacher model and the training student model.

Stella-400M is a Decoder-based Language Model with around 440 Million parameters. Its architecture is shown in Figure 4.3. Its Embedding layer incorporates Rotary Positional Embeddings (RoPE) (Su et al., 2023b), with a vocabulary size of 30,528 tokens and a hidden dimension size of 1024. The model has 24 stacked Transformer blocks with the standard multi-head self-attention (Vaswani et al., 2017), where the Query, Key, and Value vectors are projected into one single linear layer (3×1024). The MLP block has a 1024 hidden size expanding to 8192 in the up-projection, with a GELU activation.

To get the sentence-level Embedding, we pass through a Matryoshka Representation Learning (MRL) (Kusupati et al., 2024) Layer, a technique that scales embedding dimensions to multiple dimensions. In our case, we keep the embedding dimensions at 1024. Finally, to get the sentence-level Embedding, we perform a mean-pooling operation on the Last hidden state of the token embeddings.

We choose Stella-400M due to it being a very cost-effective model, obtaining one of the top rankings on the MTEB Benchmark (Muennighoff et al., 2023) for sentence embedding tasks while having a low number of parameters.

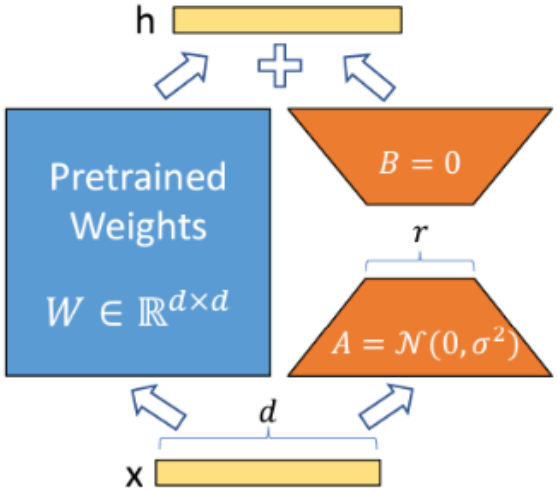

Figure 4.4: LoRA-modified layer. On this setup, we freeze the original model and train two matrices A and B - extracted from Hu et al. (2022).

4.2.1 LoRA

Our Fine-Tuning stage uses Low-Rank Adaptation (Hu et al., 2022) to the Linear layers of our pre-trained model. With this technique, we freeze the 400 Million model and train two adapter matrices A and B (Figure 4.4), reducing the memory and computation needed to fine-tune the model. The dimensionality of these low-rank matrices is determined by rank r, which we set to 16 in our experiments. This optimization reduces the number of trainable parameters from approximately 440 million to 8 million.

We apply the adapter matrices to all linear layers on each Decoder. For the Attention Block, we apply the adapters on both QKV (QKV_proj) and output (o_proj) projections. The QKV_proj layer projects the input tensor into three different spaces simultaneously: one for the query, one for the key, and one for the value. Its equation with the adapters is the following:

$$[Q; K; V] = (W_{qkv}x + b_{qkv}) + B_q A_q x \tag{4.1}$$

Where, considering a batch size B and sequence length S, we have $[Q; K; V] \in \mathbb{R}^{B \times L \times 3072}$ corresponds to the concatenated Query, Key, and Value tensors, $x \in \mathbb{R}^{B \times L \times 1024}$ is the input embedding, $W_{qkv} \in \mathbb{R}^{3072 \times 1024}$ are the frozen QKV pretrained weights, $b_{qkv} \in \mathbb{R}^{3072}$ is the pretrained bias, $A_q \in \mathbb{R}^{16 \times 1024}$ is the trainable LoRA down-projection and $B_q \in \mathbb{R}^{3072 \times 8}$ is the trainable LoRA up-projection.

Similarly, the output projection has the following equation:

$$y = (W_{out}x + b_{out}) + B_o A_o x \tag{4.2}$$

Where $y \in \mathbb{R}^{B\times L\times 1024}$ is the output tensor, $x \in \mathbb{R}^{B\times L\times 1024}$ is the input from attention operation, $W_{out} \in \mathbb{R}^{1024\times 1024}$ are the frozen output pretrained weights, $b_{out} \in \mathbb{R}^{1024}$ is the pretrained bias, $A_o \in \mathbb{R}^{16\times 1024}$ is the trainable LoRA down-projection and $B_o \in \mathbb{R}^{1024\times 16}$ is the trainable LoRA up-projection.

For the MLP Block, we have the following equations:

$$u = W_{up}x + B_{up}A_{up}x \tag{4.3}$$

$$y = W_{down}(\text{GELU}(u)) + B_{down}A_{down}(\text{GELU}(u)) \tag{4.4}$$

Where $u \in \mathbb{R}^{B\times L\times 8192}$ is the intermediate tensor, $x \in \mathbb{R}^{B\times L\times 1024}$ is the attention output, $W_{up} \in \mathbb{R}^{8192\times 1024}$ are the frozen up-projection pretrained weights, $b_{up} \in \mathbb{R}^{8192}$ is the pretrained up-projection bias, $A_{up} \in \mathbb{R}^{16\times 1024}$ is the trainable LoRA down-projection, $B_{up} \in \mathbb{R}^{8192\times 16}$ is the trainable LoRA up-projection, $y \in \mathbb{R}^{B\times L\times 1024}$ is the output tensor, $W_{down} \in \mathbb{R}^{1024\times 4096}$ are the frozen down-projection pretrained weights, $b_{down} \in \mathbb{R}^{1024}$ is the pretrained down-projection bias, $A_{down} \in \mathbb{R}^{16\times 4096}$ is the trainable LoRA down-projection, and $B_{down} \in \mathbb{R}^{1024\times 16}$ is the trainable LoRA up-projection.

4.2.2 Contrastive Learning

In our training, we use a Contrastive Learning approach. The queries represent the anchor embedding, while the employee data (documents) represent the positive (for the relevant example) or negative (for irrelevant/random in-batch examples) embeddings.

Given a positive pair of query-document (q^+, d^+), we apply a simple instruction template to the queries:

$$q^+_{instr} = \text{“Instruct: \{template\} \textbackslash n Query: ”} + q^+ \tag{4.5}$$

where *“template”* represents the text embedding related task. In our case, we use the text as an instruction template: *“In an enterprise context, given a project and an associated task, retrieve relevant employees that fill that role.”*.

To train the embedding model, we employ the InfoNCE loss function (van den Oord et al., 2019), which operates on both positive and negative examples, where negative samples can be either hard negatives or in-batch examples. The loss is defined as:

$$\min \mathbb{L} = -log\left(\frac{\phi(q^{+}_{instr}, d^{+})}{\phi(q^{+}_{instr}, d^{+}) + \sum_{n_i \in N}(\phi(q^{+}_{instr}, n_i))}\right) \tag{4.6}$$

where ϕ represents a similarity measure, in our case, the cosine similarity, and $n_i \in N$ represents a negative document. The main idea of this loss function is to maximize the similarity between the positive pairs ($\phi(q^{+}_{instr}, d^{+})$) while minimizing the similarity of negative pairs ($\phi(q^{+}_{instr}, n_i)$). To calculate the similarity ϕ, we first extract the sentence-level embeddings of queries and documents by performing a mean pooling over the last hidden state of our model. Then, we apply the cosine similarity function to the resulting embeddings to get the similarity score ϕ. The cosine similarity equation is defined as:

$$\phi(q, d) = \frac{h_q \cdot h_d}{||h_q||\ ||h_d||} \tag{4.7}$$

where h_q and h_d represent the sentence-level embeddings for a query and a document, respectively.

4.3 Synthetic Data

To fine-tune our model, we test two different approaches to generating synthetic examples. In the first approach, we use an LLM to generate synthetic labels for a set of query-document pairs. In the second approach, we leverage all queries, which are tasks within projects, and generate an ideal employee for each task. For both approaches, we first need to generate diverse projects and tasks relevant to the company's context. For all data generation tasks, we use OpenAI's GPT-4o-mini API (OpenAI, 2024b,a).

4.3.1 Synthetic Tasks

To generate the synthetic tasks, we build a two-step generation process, illustrated in Figure 4.5. First, we prompt the LLM to brainstorm a pool of projects. To ensure the relevance and diversity of projects, we provide a contextualizing text to the prompt, presenting the enterprise areas and main corporate activities. We also sample a set of abilities present in our employee dataset to ensure that the LLM generates projects for the various kinds of abilities present in the company. This brainstorming process is done

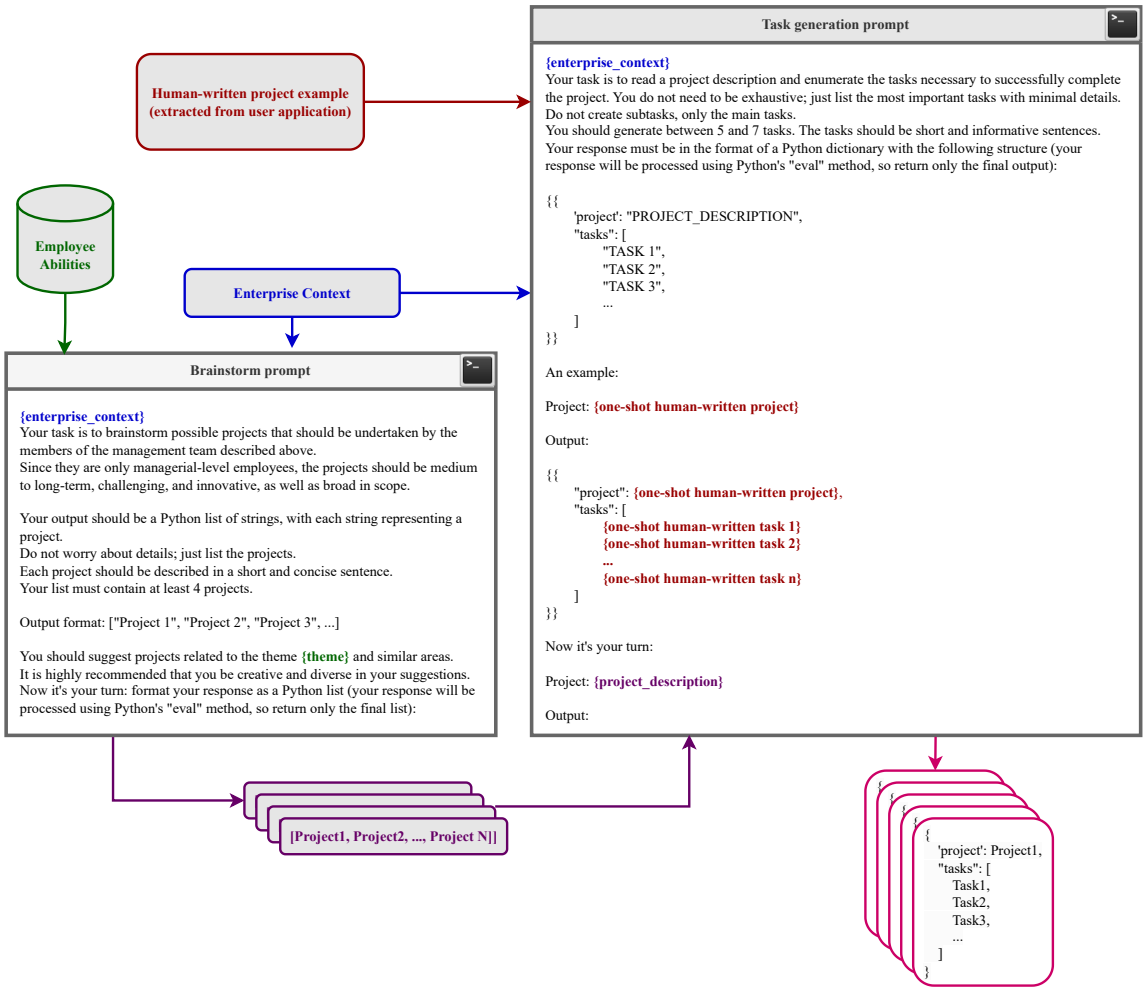

Figure 4.5: Two-stage prompt for the synthetic tasks creation. On the left, we brainstorm a set of projects; on the right, we break the projects into tasks. The colored texts correspond to the contextual information provided to each prompt.

multiple times, with different abilities each time. We ensure the output format is a Python list that can be used in further steps.

For the second step, we feed the generated projects into a second prompt to extract a set of tasks for each generated project. We guide the generation process by presenting a one-shot example from a human-written project (extracted from our user application). Finally, the LLM outputs a JSON containing a project and its associated tasks.

Machine Feedback prompt

{enterprise_context}

You are responsible for the department of coordinators and managers. Your team consists of professionals from various areas, including manufacturing, sales, and human resources.
Your task is to evaluate whether an employee is relevant or not for a specific task in a project.
You will receive the description of a project and a related task, along with the resume of an employee. Critically analyze the information to determine if the employee is suitable to perform the described task.

Please follow these guidelines in your response:

1. Briefly summarize the Project and Task, highlighting the importance of the task for the project's success.
2. Analyze the employee's competencies directly related to the task, considering both practical and theoretical experience. Ask yourself: "How relevant are the employee's skills and experiences to the task in question?"
3. Discuss the positives and negatives of the resume concerning the task, dividing your analysis into sections:
    - Competency Analysis
    - Strengths
    - Weaknesses
    - Conclusion
4. The last character of your response must be a score of 0 or 1, where:
    - 0: irrelevant/insufficient
    - 1: relevant/sufficient

Be rigorous in your evaluation and justify your decision clearly and concisely.

Now it's your turn:

Project: **{project_description}**
Task: **{task_description}**
Resume: **{employee_resume}**

Your response:

Figure 4.6: Machine-feedback prompt for the synthetic labeling process. We extract labels generated by an LLM to create synthetic relevance pairs between generated tasks and real employees.

4.3.2 Synthetic Labeling

The synthetic labeling process comprises two key steps: task and employee sampling, followed by the application of a Chain-of-Thought (CoT) prompt (Wei et al., 2022) to generate the labels.

Using the synthetic tasks generated previously, we use Stella-400M to generate the 20 best-ranked employees for each task. To create the task-employee pairs, we first randomly sample the tasks. The associated employee has a 50% chance of being sampled from the top 20 and a 50% chance of being sampled from the whole database.

After generating the pairs, we feed them to the LLM with the prompt in Figure 4.6. Similar to the task generation process, we add the enterprise context to the start of the prompt. Then, we guide its generation process with strict analysis guidelines such as discussing the positives and negative aspects of that employee regarding the task, followed by a competency analysis. In the conclusion, the LLM must give the employee a score of 0 (irrelevant/insufficient) or 1 (relevant/sufficient). Using this process, we generated more than 30,000 labeled task-employee pairs.

Machine Feedback base prompt (Eureka inital prompt)

{enterprise_context}

You are responsible for the department of coordinators and managers. This team consists of people from various sectors, such as manufacturing, sales, human resources, and others.
Your task is to determine whether an employee is relevant for a specific task in a project.

You will receive as input a project description, an associated task, and an employee's resume. You must indicate whether the employee is suitable for that task.

You should explain your decision, but the last character of your response must be a score of 0 or 1, where:

0: irrelevant / insufficient
1: relevant / sufficient

You must be very strict in your decision. Consider both the strengths and weaknesses of the resume in relation to the task.

Now it's your turn:

Project: **{project_description}**
Task: **{task_description}**
Resume: **{employee_resume}**

Your response:

Figure 4.7: Initial handcrafted machine feedback prompt. This is the starting point for the Eureka optimization process.

4.3.2.1 Prompt Optimization

To get to the prompt in Figure 4.6, we perform a prompt optimization step using the EUREKA framework (Ma et al., 2023b). Briefly, EUREKA is an evolutionary search that leverages an LLM to optimize prompts for specific tasks. We model the relevance labeling as a prompt refinement task, providing a base handcrafted prompt (Figure 4.7) as input. To effectively guide this optimization, we provided comprehensive evaluation metrics, including accuracy, precision, and recall scores from the best-performing prompt, along with representative examples spanning true positives, true negatives, false positives, and false negatives. The complete review prompt is highlighted in Figure 4.8.

At each generation, we use GPT-4o to generate candidate prompts (individuals) using the review prompt in Figure 4.8. Then, we evaluate each individual as a labeling prompt for our labeling LLM (GPT-4o-mini) by (re-)labeling our evaluation dataset. For each individual, we calculate its relevance classification metrics in the evaluation dataset. The best prompt is then passed to the next generation's review prompt. If neither individual outperformed the best prompt, it is replicated to the next generation. Algorithm 1 summarizes all these steps:

Eureka's Review Prompt

You are Eureka, a prompt optimization algorithm for LLMs. Your task is to improve a prompt for evaluating an employee's relevance to a given task.

As input, you will receive the current prompt along with some performance metrics based on test data.
You must modify the prompt in a way that improves the metrics compared to the previous generation.

You may reason to enhance your analysis. However, your final prompt must always be enclosed within ```.

Here is an example template:

Response:
[YOUR REASONING]

Conclusion:
```[YOUR FINAL PROMPT]```

Now it's your turn. You will receive the best prompt from the previous generation and its metrics. Your mission is to refine it to improve its metrics:

Prompt:
```
{best_prompt}
```

metrics:
accuracy: {acc}, precision: {prec}, recall: {rec}

examples:
- True Positive: ```{tp}```
- True Negative: ```{tn}```
- False Positive: ```{fp}```
- False Negative: ```{fn}```

Response:

Figure 4.8: Review prompt for the Eureka optimization process.

Algorithm 1 EUREKA for labeling prompt optimization

Input: `LLM`, fitness function F, initial prompt `prompt`
Output: S_{Eureka}
Hyperparameters: Search iteration N, number of samples K, elite size R
begin
  for N *Iterations* **do**
    // Sample K labeling prompts from `LLM`
    $S_i, ..., S_K \sim$ `LLM(prompt)`
    // Evaluate candidates
    $s_i = F(S_i), ..., s_k = F(S_K)$
    // Reflection step
    `prompt` := `prompt` : $\texttt{Reflection}_{i=1}^{R}(S_{best_i}, s_{best_i})$ where $best = R - argmax$
  end
  $S_{Eureka} := S_{best_i}$
end

In the algorithm, the fitness function F is a function that labels the evaluation dataset and calculates the accuracy, f1, precision, and recall metrics for a generated prompt S_k. The metric we use to optimize the prompt in the reflection step is the f1_score. The `Reflection` function use the best prompt globally to create the updated `prompt` for the next iteration.

In the next chapter, we evaluate the effectiveness of this prompt optimization technique, comparing models trained on synthetic data generated by both base and EUREKA-
```

Synthetic Curriculum prompt

{enterprise_context}

You are responsible for the department of coordinators and managers. This team includes people from diverse sectors such as manufacturing, sales, human resources, and more.
Your task is to generate an ideal resume for a specific task related to a project.

The generated resume must follow this format:

"
Position: IDEAL_POSITION

Objective: IDEAL_OBJECTIVE

Behavior: IDEAL_BEHAVIOR

Experience: IDEAL_EXPERIENCE (previous professional experiences relevant to the position)

Skills: IDEAL_SKILLS (skills relevant to the position)
"

Now it's your turn:

Project: **{project_description}**
Task: **{task_description}**

Ideal resume:

Figure 4.9: Prompt for the synthetic curriculum prompt. We use our generated synthetic tasks and generate an ideal curriculum for each task, creating similarity pairs that will be used during training.

optimized prompts.

4.3.3 Generating Synthetic Curriculumns

Our initial idea with the Synthetic curriculums was to generate both positive and hard negative examples for each task. However, we found that the LLMs struggle to generate negative examples, as stated in previous work (García-Ferrero et al., 2023; Hossain et al., 2020; Truong et al., 2022). Therefore, we changed our prompt to generate only positive examples, and during training, we used the in-batch examples as negatives, which is shown to be a strong alternative to labeled hard negative examples (Chen et al., 2020; Ye et al., 2019; Doersch and Zisserman, 2017).

Figure 4.9 shows the curriculum prompt. As the previous prompts, we also give an enterprise context, followed by the task description. The curriculum format we provide follows the aspects that are present in the real employee data. Then, we provide the project and task description for the LLM to generate the ideal curriculum. We generated one synthetic curriculum for each of the generated tasks, totaling more than 32,000 curriculums.

4.4 Rank Aggregation

Our employee data has different textual features, which we call aspects. Thus, our model generates one ranking for each aspect data at inference time. For example, the Abilities aspect creates a rank of employees different from the Experiences aspect. This approach necessitates an aggregation method to combine these individual rankings into a final rank.

A standard practice to skip this aggregation process is concatenating all text and performing the ranking process using the embeddings generated with the full text. However, this approach presents two significant limitations when applied to our employee dataset. First, the concatenated text may often exceed the model's maximum length, which leads to truncation and information loss. Second, forcing the model to compress multiple aspects into a single fixed-dimension embedding reduces the representation quality, as the same embedding space must encode significantly more information. In our approach, we use the Condorcet aggregation method (de Condorcet, 1785) to address these limitations and improve the quality of the ranking process.

4.4.1 Condorcet

The Condorcet method operates by treating the comparison of candidates (in our case, employees) as a pairwise ranking problem. Each aspect of the employee's profile (e.g., skills, experience, and projects) is individually encoded into an embedding, and pairwise comparisons are made between candidates based on their scores for each aspect. Algorithm 2 shows the entire process of calculating the pairwise matrix to generate the Condorcet Scores.

Algorithm 2 Condorcet Ranking calculation algorithm (assuming a similarity measure)

Input: `input_matrix`: Matrix of size (`num_candidates`, `num_aspects`) containing the final rank for each aspect in each column
Output: `ranking`: Array of Condorcet rankings

```
begin
    Initialize condorcet_scores as a zero array of size num_candidates
    Initialize pairwise_matrix as a zero matrix of size (num_candidates,
     num_candidates)
    for i ← 0 to num_candidates - 1 do
        for j ← i + 1 to num_candidates - 1 do
            pairwise_matrix[i, j] ← sum(input_matrix[i] > input_matrix[j])
            pairwise_matrix[j, i] ← sum(input_matrix[i] < input_matrix[j])
        end
    end
    for i ← 0 to num_candidates - 1 do
        condorcet_scores[i] ← sum(pairwise_matrix[i])
    end
    Return condorcet_scores
end
```

The algorithm has a time complexity of $O(n^2 \cdot m)$ where n is the number of employees and m is the number of aspects. As we have a low number of employees in our dataset (835), the quadratic scalability regarding items to be ranked (documents) does not hurt the overall ranking performance. However, for semantic search tasks that have a high amount of documents (i.e., web search), it is recommended to use aggregation algorithms that scale better in regards to the number of documents, such as the Borda Count algorithm (de Borda, 1781).

Chapter 5

Experiments

This chapter presents a comprehensive empirical evaluation of our proposed methodology through a series of experiments. We begin by comparing our three synthetic data fine-tuning strategies — label-based, curriculum-based, and their combination — against already established models. Subsequently, we evaluate the effectiveness of our prompt optimization strategy by comparing the models derived from both EUREKA-optimized and handcrafted prompts. We then investigate the performance implications of two distinct adapter configurations to determine optimal architectural setups. Finally, to analyze the impact of each aspect, we conduct ablation studies where we remove one aspect at a time from training.

For convenience, we structure these experiments around the following research questions:

(RQ1) To what extent do models fine-tuned on synthetic data demonstrate superior performance compared to current state-of-the-art approaches in Enterprise Team Formation tasks?

(RQ2) Which synthetic data generation strategy yields better performance when evaluated on real-world datasets?

(RQ3) How does the implementation of Rank Aggregation methodology impact model performance metrics?

(RQ4) What is the relationship between prompt engineering quality and model performance outcomes?

(RQ5) How does the performance of specialized aspect-specific adapters compare to that of a unified global adapter architecture?

(RQ6) What are the potential performance implications of removing individual aspects from the model architecture?

(RQ7) How does contrastive fine-tuning influence the geometric properties and semantic organization of the embedding space?

First, we evaluate the overall impact of our three proposed synthetic data approaches on model performance. We compare models trained only on synthetic curriculums, only on synthetic labels, and on a combination of both (RQ2), using the same training arguments (LoRA $r = 16$, LoRA $\alpha = 16$ and batch size of 24). To validate our results, we compare our models against current SOTA algorithms in text-embedding tasks (RQ1). We also compare against classic unsupervised models such as BM25. To measure the impact of the Rank Aggregation algorithm (RQ3), we train the model separately that is trained on only one big aspect, which is the concatenation of all employee aspects.

Next, we summarize each of the baselines we tested:

- TF-IDF (Salton and McGill, 1983): scores documents based on term frequency and inverse document frequency (see Chapter 3.1).

- BM25 (Robertson et al., 1994): expands on TF-IDF by using saturation and document length normalization (see Chapter 3.1).

- e5-large-v2 (Wang et al., 2022): a RoBERTa-based Dual-Encoder model that is trained on unlabeled text pairs using contrastive learning. Then, the model is fine-tuned on small, high-quality labeled datasets.

- Sentence-T5-large (Ni et al., 2021): builds upon T5 (Raffel et al., 2020), applying contrastive learning on a Dual-Encoder setup. In this approach, the decoder is discarded, and the sentence embedding is defined as the average pooling of the encoder output.

- OpenAI-text-embedding-3_{small} (OpenAI, 2023): proprietary model from OpenAI (no architectural information available to the public).

- OpenAI-text-embedding-3_{large} (OpenAI, 2023): proprietary model from OpenAI (no architectural information available to the public).

- bge-large-en-v1.5 (Xiao et al., 2023): RoBERTa-based, similar to e5. Uses mean pooling of the last hidden layer to generate the sentence embeddings. BGE was

trained on large-scale datasets, including BEIR, MS MARCO, and other retrieval benchmarks.

- gte-Qwen2-1.5B-instruct (Li et al., 2023): built upon the Qwen2-1.5B LLM, trained on a vast multilingual text corpus. The model incorporates instruction-tuning and bidirectional attention mechanisms.

- Stella-400M (Zhang, 2024): distillation of gte-Qwen2-1.5B-instruct, uses a Dual-Encoder setup and mean pooling (see Chapter 4.2).

The overall results are shown in Table 5.1. We evaluate the models on a series of ranking metrics (see Chapter 2.3). As the table shows, the synthetic fine-tuning is highly effective, achieving a performance improvement of over 30% across all ranking metrics in comparison to the best-performing baseline. We also see that the best strategy overall for fine-tuning was the full data approach, where we use a combination of both curriculum and synthetic labeling data.

	Avg. Prec	nDCG@1	nDCG@5	Hit@5	AUC
Unsupervised Models					
BM25 (Robertson et al., 1994)	0.081	0.020	0.076	0.183	0.670
TF-IDF (Salton and McGill, 1983)	0.169	0.101	0.172	0.305	0.683
Supervised Models					
e5-large-v2 (Wang et al., 2022)	0.141	0.060	0.117	0.243	0.740
Sentence-T5-large (Ni et al., 2021)	0.160	0.044	0.144	0.280	0.742
OpenAI-text-embedding-3_{small} (OpenAI, 2023)	0.288	0.165	0.305	0.481	0.747
OpenAI-text-embedding-3_{large} (OpenAI, 2023)	0.296	0.160	0.303	0.460	0.782
bge-large-en-v1.5 (Xiao et al., 2023)	0.195	0.105	0.188	0.322	0.703
gte-Qwen2-1.5B-instruct (Li et al., 2023)	0.236	0.143	0.220	0.340	0.734
Stella-400M (Zhang, 2024)	0.284	0.150	0.285	0.481	0.758
Ours					
Stella-LoRA$_{\text{Condorcet}}$ + full data	**0.386**	0.217	**0.418**	**0.620**	**0.798**
with synthetic curriculums only	0.296	0.140	0.323	0.544	0.758
with synthetic labels only	0.358	**0.260**	0.393	0.581	0.771
Concat Model	0.352	0.220	0.364	0.565	0.734

Table 5.1: Overall results of model performance on our evaluation dataset.

Separately, the synthetic curriculum approach had marginal improvements over the Stella-400M baseline, with an average improvement of 10% on the ranking except for $NDCG@1$, where there was a performance decrease of around 9%. In contrast, the synthetic labeling data shown significant improvements over the base model, achieving an averege improvement of 39% over Stella-400M , with a 73% improvement on $NDCG@1$. In conclusion, while both data approaches seem to be complementary for the final re-

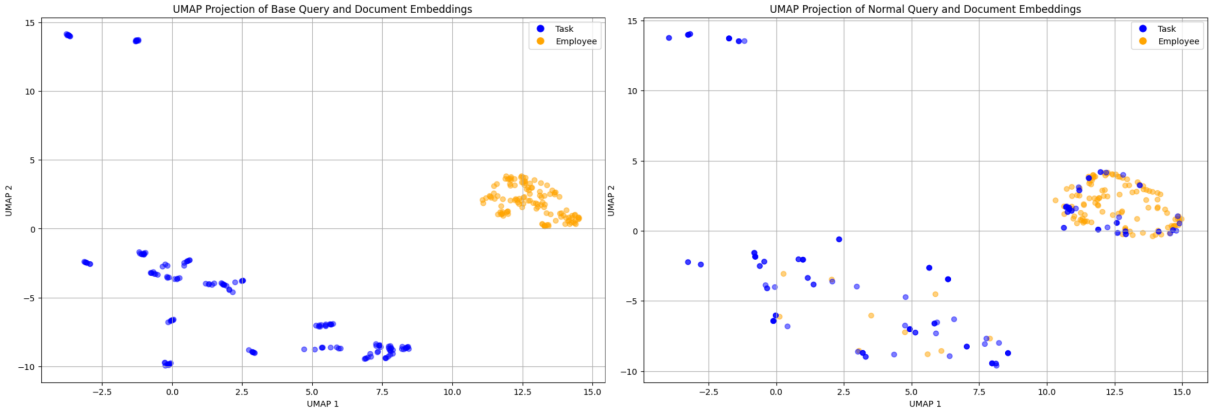

Figure 5.1: UMAP representation of our evaluation dataset embeddings. On left, we have the embeddings of the pre-trained model; on right, we have the embeddings of the fine-tuned model.

sult, the synthetic labeling has a bigger performance improvement when comparing both approaches separately.

In Table 5.1, we can also measure the impact of the Rank Aggregation approach. When comparing the "full data" approach against the Concat Model, we see an average improvement of around 6% on the tested ranking metrics. This shows considerable improvements of the Condorcet algorithm over the classic text concatenation approach.

To visualize the effect of our fine-tuning on the embedding space (RQ7), we compare UMAP (McInnes et al., 2020) representations before and after the contrastive fine-tuning. UMAP builds a high-dimensional weighted graph that captures the structure of the data and then optimizes a low-dimensional representation.

Figure 5.1 shows the UMAP representations of the queries and employees (full data) before and after the fine-tuning procedure. As we can see in the figure, before the training, queries and documents we separated into two clusters. After fine-tuning, we can se a "pairing" behavior between queries and documents, where these two clusters now have elements from the other cluster close to some of its entities.

5.1 Prompt Optimization

As mentioned in 4.3.2.1, we use the Eureka framework to optimize a base handcrafted prompt for the Synthetic Labeling approach. To optimize the prompt, we evaluate the candidate prompts on our eval dataset, choosing the best performing prompt at each generation.

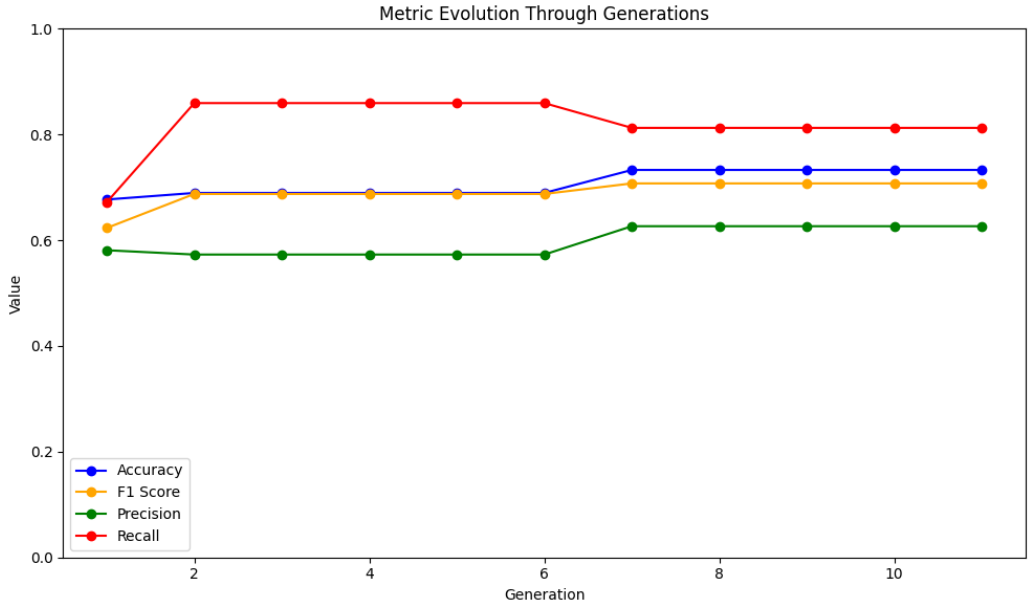

Figure 5.2: Evolution of metrics of the Eureka optimization.

Figure 5.2 shows the metrics of the Eureka optimization throughout 10 generations. We use the F1 score as the main optimization metric. If no prompt got improved upon the current best prompt, it is repeated in the next iteration. As we can see in the figure, we had only two evolutions over the current best, one in generation 2, and another in generation 7.

To measure the impact of this optimization procedure (RQ4), we create two Synthetic Labeling sets, one with data generated by the base handcrafted prompt, and another with data generated by the best Eureka-generated prompt.

We then train separate models on each dataset and compare their performance at both the aspect level and the aggregation level. Table 5.2 presents the ranking metrics for both models. We highlight with grey the aspect which had better perfomance when comparing the eureka/base pairs. For the Role and Behavior aspects, the original prompt had better performance overall. However, for the Goals, Abilities, and Experience aspects, the Eureka-optimized model performed better. When aggregating all aspects, the Eureka-optimized model had better results over all ranking metrics, outperforming the base prompt model by arond 10% on average. By these results, we can conclude that aligning the prompt with the human-curated data can provide benefits to the quality of synthetic data generated, impacting positively the fine-tuning results.

	Avg. Prec	nDCG@1	nDCG@5	Hit@5	AUC
Baseline					
Stella-400M	0.284	0.150	0.285	0.481	0.758
Concat Model	0.352	0.220	0.364	0.565	0.734
Synth Label Model - Eureka-optimized prompt					
Role	0.343	0.210	0.351	0.520	0.786
Behavior	0.292	0.202	0.299	0.460	0.774
Goals	0.350	0.252	0.337	0.460	0.775
Abilities	**0.375**	0.259	0.390	0.580	0.716
Experience	0.359	0.202	**0.396**	0.600	**0.802**
Condorcet (Aggregation)	0.358	**0.260**	0.393	0.581	0.771
Synth Label Model - Original prompt					
Role	0.372	0.258	0.385	0.540	0.794
Behavior	0.299	0.163	0.318	0.522	0.796
Goals	0.343	0.252	0.336	0.484	0.793
Abilities	0.331	0.200	0.352	0.561	0.691
Experience	0.353	0.182	0.385	**0.620**	0.797
Condorcet (Aggregation)	0.346	0.225	0.350	0.521	0.782

Table 5.2: Comparison between the model trained using data from the Eureka-optimized prompt and the model trained using data from the base handcrafted prompt. Rows in grey represent the winning approach.

5.2 Global Adapter and Specialized Adapters

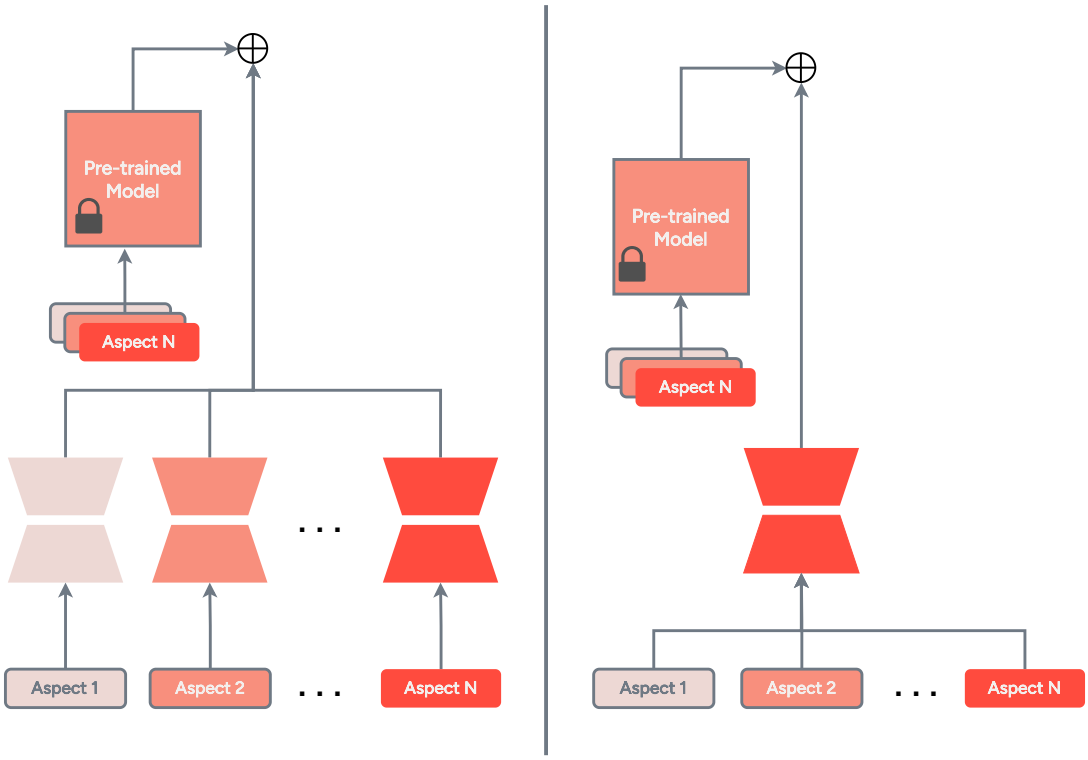

Figure 5.3: (left) Specialized adapters. (right) Global adapter.

We also investigate two different methods of fine-tuning our base model. The first method uses multiple adapters, one for each aspect, while the the second method uses one single trained adapter that is used by all aspects at inference time. Figure 5.3 shows the difference between the two tested approaches.

For the Specialized Adapters, we have to train each adapter separately, using each aspect as the document passages during training. Then, each trained adapter can only be used with its corresponding aspect it was trained on.

The Single Adapter approach performs the training only once. During the adapter training, we concatenate all aspects' text for each employee, creating a long passage that is all text related to that employee. The resulting adapter can further be used globally by all aspects.

Table 5.3 shows aspect-level and aggregation-level metrics for both strategies on the Synthetic Labeling data. The grey coloring shows the winning aspect/aggregation between the model pairs based on the average ration across all evaluation metrics. As expected, when looking at the aspect level, except for the Role, all other aspects had

	Avg. Prec	nDCG@1	nDCG@5	Hit@5	AUC
Baseline					
Stella-400M	0.284	0.150	0.285	0.481	0.758
Concat Model	0.352	0.220	0.364	0.565	0.734
Aspect Models - Specialized adapter					
Role	0.343	0.210	0.351	0.520	0.786
Behavior	0.292	0.202	0.299	0.460	0.774
Goals	0.350	0.252	0.337	0.460	0.775
Abilities	0.375	0.259	0.390	0.580	0.716
Experience	0.359	0.202	0.396	0.600	**0.802**
Condorcet (Aggregation)	0.358	**0.260**	0.393	0.581	0.771
Aspect Models - Global adapter					
Role	0.357	0.259	0.374	0.561	0.785
Behavior	0.262	0.150	0.271	0.443	0.755
Goals	0.254	0.161	0.248	0.380	0.749
Abilities	0.305	0.200	0.305	0.463	0.671
Experience	0.367	0.200	0.377	0.562	0.770
Condorcet (Aggregation)	**0.385**	0.250	**0.407**	**0.605**	0.779

Table 5.3: Comparison between a model trained with specialized adapters and a model trained with a global adapter. Rows in grey represent the winning approach.

better overall performance when using specialized adapters over a single global adapter. However, surprisingly, when we aggregate the rankings, the global adapter aggregation achieves better results, achieving a marginal gain of around 3.5%, on average.

5.3 Ablation Test

	Avg. Prec	nDCG@1	nDCG@5	Hit@5	AUC
Baseline					
Stella-400M	0.284	0.150	0.285	0.481	0.758
Model performance on removed aspect					
Stella-LoRA$_{\text{Condorcet}}$ − *Role*	0.381	**0.260**	0.407	0.60	0.771
Stella-LoRA$_{\text{Condorcet}}$ − *Behavior*	0.352	0.245	0.379	0.58	0.762
Stella-LoRA$_{\text{Condorcet}}$ − *Goals*	0.381	0.245	0.403	0.58	0.784
Stella-LoRA$_{\text{Condorcet}}$ − *Abilities*	0.382	0.250	**0.412**	**0.62**	**0.779**
Stella-LoRA$_{\text{Condorcet}}$ − *Experience*	0.373	0.256	0.397	0.58	0.773
Condorcet (Aggregation)	**0.385**	0.250	0.407	0.605	**0.779**
Concat Model	0.352	0.220	0.364	0.565	0.734

Table 5.4: Ablation results. At each step, we remove an aspect from training.

Finally, in order to laverage the individual importance of each aspect to model performance, we conduct an ablation study. At each test, we remove one aspect from

the training and aggregation process and calculate the metrics for each resulting model. We then compare the ablation results against the full aggregation and the Concatenation models.

Table 5.4 shows the ablation results. For this experiment, we used the Global adapter model introduced in the previous section. The table shows that removing the *Abilities* aspect, the overall performance of the model increases marginally when compared to the full Aggregation model. This suggests that the *Abilities* aspect does not contribute with useful information to the model and may even introduce noise or redundancy. The other aspects presented a performance decrease overall when removed from training, with some exceptions regarding the $nDCG@1$ when discarding *Role* and *Experience* aspects, which had a slight increase in performance in comparison to the full aggregation model.

Chapter 6

Conclusions

In this thesis, we present a Synthetic data Fine-tuning approach to improve the performance of the Enterprise team Formation task. We design this task as a Semantic search problem, where the projects and tasks are modeled as queries, and the employee with their aspects (e.g., skills, experience, and behavior) are modeled as documents. We evaluate our model on a curated human-labeled dataset and conduct a series of experiments in order to validate our proposed approach. In this chapter, we summarize the obtained results and present some possible directions for future work.

6.1 Main Results

The core contribution of this work lies in the development of two synthetic data fine-tuning methodologies for enterprise team formation:

1. **LLM-Augmented Synthetic Labeling:** We leverage large language models (LLMs) to annotate pairs of synthetic tasks and real employee profiles.
2. **Synthetic Curriculum Generation:** We use LLMs to generate idealized employee curriculum tailored to synthetic tasks.
3. **Hybrid:** A combination of the both methods described above.

The data generation method starts by the creation of synthetic tasks. In this step, we use a two-stage generation process. First, we prompt the LLM to brainstorm a pool of projects, giving employee abilities as context for the LLM. In the second step, we expand each project, breaking them down into a series of tasks. With the tasks ready, we use another prompt to generate an ideal curriculum for each task. For the synthetic labeling process, we sample task-employee pairs, and label the relevance of each pair using an LLM and a Chain-of-Thought prompt.

This labeling prompt is optimized using the evolutionary algorithm Eureka. Briefly, Eureka leverages an LLM to optimize prompts for specific tasks, aiming to optimize a desired metric. We model the relevance labeling as a prompt refinement task, optimizing the prompt performance over our human-curated evaluation dataset.

During the modeling phase, we fine-tuned the Stella-400M language model using Low-Rank Adapters (LoRA). Since our employee dataset contains multiple distinct aspects that need to be evaluated, we developed a Rank Aggregation approach to combine the generated rankings. Specifically, we implemented the Condorcet algorithm to unify these rankings into a single consensus Rank. We also test different approaches to LoRA, where we test multiple specialized adapters (one for each aspect), or one single global adapter (trained on the concatenation of all aspects).

We test our fine-tuning approach against a series of well-estabilished semantic search methods on a series of ranking metrics. Our method obtained the best results across all tested models, outperforming the current SOTA algorithms on the team formation task, which suggests the effectiveness of our proposed method. Experimental results demonstrate that combining both Synthetic Labeling and Synthetic Curriculum approaches yields better performance, achieving the highest overall metrics in the tested ranking metrics. Notably, while the hybrid method proved most effective, the synthetic labeling approach alone achieved the strongest individual performance when compared to the Synthetic curriculum approach.

We also measure the effectiveness of the Eureka framework. In this experiments, we train two sparated models, one using data generated by the Eureka-optimized prompt, and another using data generated by the base handcrafted prompt. We compare both models on the aspect and aggregation level. Results show that the Eureka-optimized prompt outperformed the handcrafted prompt across most employee aspects, and also had a performance increase of around 10% in the tested ranking metrics when aggregating the Ranks.

Another experiment we conducted revolves around the Low-Rank Adapter setup. We investigate two different setups:

1. **Specialized Adapters:** We train one adapter for each aspect separately, using each aspect as the document passages during training.
2. **Global Adapter:** We train one single global adapter with the concatenation of all aspects. The resulting adapter is used by all aspects at inference time.

As expected, when viewing the results of each individual aspect rankings, the specialized model had a better performance across most aspects. However, for the aggregated rankings, the global adapter model had a better metrics overall.

Finally, we conduct an ablation study to measure the effectiveness of each aspect. In this test, we train the full model multiple times, excluding one different aspect from

training at each time. The results show that the model trained without the *Abilities* aspect had an improvement over the full model, which is a sign that this aspect contains redundant or noisy information.

6.2 Future Work

We are currently planning to apply this method in another enterprise context that revolves around software development and data science. This environment can be more challenging, as the employees more similar in terms of skills and experiences, which makes the task of finding the best fit for a certain task more difficult. This additional case study can make the method more robust as we show it can work on a multitude of scenarios.

We also map a series of improvements on the modeling side. The Eureka optimization approach proved effective for the Synthetic labeling approach. We plan to extend this optimization to the Synthetic Curriculums approach. However, modeling the fitness function for the evolutionary optimization in this case is not as trivial as the labeling method. The Rank aggregation also presented a big improvement on model performance. We plan to extend the experiments on this side, testing different algorithms and compare them to the Condorcet algorithm.

We also aim to expand the investigations on the model adapters. we plan to test evolutions and variations of LoRA, such as MixDA (see Chapter 3.3). Furthermore, we plan to analize new configurations of adapters, such as adapters for pairs or triplets of aspects.

We are also mapping architectural improvements to our model. The NV-Embed-v2 (3.1), for example, introduces the latent attention layer, that substitutes the final average pooling at the end of the dual-encoder setup with a novel attention-based layer to create the sentence-level embeddings. Another potential improvement could be the use of lexical-based embeddings to complement the sentence embeddings. The LENS model showed state-of-the-art results when combining clustering-based lexical embeddings with standard sentence embeddings.

Bibliography

Asai, A., Schick, T., Lewis, P., Chen, X., Izacard, G., Riedel, S., Hajishirzi, H., and Yih, W.-t. (2023). Task-aware retrieval with instructions. In Rogers, A., Boyd-Graber, J., and Okazaki, N., editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 3650–3675, Toronto, Canada. Association for Computational Linguistics.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., Rosenberg, M., Song, X., Stoica, A., Tiwary, S., and Wang, T. (2018). Ms marco: A human generated machine reading comprehension dataset.

Beeching, E., Fourrier, C., Habib, N., Han, S., Lambert, N., Rajani, N., Sanseviero, O., Tunstall, L., and Wolf, T. (2023). Open llm leaderboard. `https://huggingface.co/spaces/open-llm-leaderboard-old/open_llm_leaderboard`.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., and Liu, Z. (2024). M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand. Association for Computational Linguistics.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., Stoica, I., and Xing, E. P. (2023). Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

de Borda, J.-C. (1781). Mémoire sur les élections au scrutin. *Histoire de l'Académie Royale des Sciences*, 12.

de Condorcet, M. (1785). *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Imprimerie Royale, Paris.

Déjean, H., Clinchant, S., Lassance, C., Lupart, S., and Formal, T. (2023). Benchmarking middle-trained language models for neural search. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 1848–1852, New York, NY, USA. Association for Computing Machinery.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Diao, S., Xu, T., Xu, R., Wang, J., and Zhang, T. (2023). Mixture-of-domain-adapters: Decoupling and injecting domain knowledge to pre-trained language models' memories. In Rogers, A., Boyd-Graber, J., and Okazaki, N., editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5113–5129, Toronto, Canada. Association for Computational Linguistics.

Doersch, C. and Zisserman, A. (2017). Multi-task self-supervised visual learning. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2070–2079.

Dong, Q., Dong, L., Zhang, X., Sui, Z., and Wei, F. (2024). Self-boosting large language models with synthetic preference data.

Dong, Z., Ni, J., Bikel, D., Alfonseca, E., Wang, Y., Qu, C., and Zitouni, I. (2022). Exploring dual encoder architectures for question answering. In Goldberg, Y., Kozareva, Z., and Zhang, Y., editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9414–9419, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Dwork, C., Kumar, R., Naor, M., and Sivakumar, D. (2001). Rank aggregation methods for the web. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 613–622, New York, NY, USA. Association for Computing Machinery.

Formal, T., Piwowarski, B., and Clinchant, S. (2021). Splade: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 2288–2292, New York, NY, USA. Association for Computing Machinery.

García-Ferrero, I., Altuna, B., Alvez, J., Gonzalez-Dios, I., and Rigau, G. (2023). This is not a dataset: A large negation benchmark to challenge large language models. In Bouamor, H., Pino, J., and Bali, K., editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8596–8615, Singapore. Association for Computational Linguistics.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., et al. (2024). The llama 3 herd of models.

Guo, S., Zhang, B., Liu, T., Liu, T., Khalman, M., Llinares, F., Rame, A., Mesnard, T., Zhao, Y., Piot, B., Ferret, J., and Blondel, M. (2024). Direct language model alignment from online ai feedback.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Hossain, M. M., Kovatchev, V., Dutta, P., Kao, T., Wei, E., and Blanco, E. (2020). An analysis of natural language inference benchmarks through the lens of negation. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9106–9118, Online. Association for Computational Linguistics.

Hosseini, M. J., Petrov, A., Fabrikant, A., and Louis, A. (2024). A synthetic data approach for domain generalization of NLI models. In Ku, L.-W., Martins, A., and Srikumar, V.,

editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2212–2226, Bangkok, Thailand. Association for Computational Linguistics.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2022). LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. (2023). Mistral 7b.

Johnson, J., Douze, M., and Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Jones, E., Palangi, H., Ribeiro, C. S., Chandrasekaran, V., Mukherjee, S., Mitra, A., Awadallah, A. H., and Kamar, E. (2024). Teaching language models to hallucinate less with synthetic tasks. In *The Twelfth International Conference on Learning Representations*.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Kemeny, J. G. (1959). Mathematics without numbers. *Daedalus*, 88(4):577–591.

Kusupati, A., Bhatt, G., Rege, A., Wallingford, M., Sinha, A., Ramanujan, V., Howard-Snyder, W., Chen, K., Kakade, S., Jain, P., and Farhadi, A. (2024). Matryoshka representation learning.

Lee, C., Roy, R., Xu, M., Raiman, J., Shoeybi, M., Catanzaro, B., and Ping, W. (2025). Nv-embed: Improved techniques for training llms as generalist embedding models.

Lei, Y., Shen, T., Cao, Y., and Yates, A. (2025). Enhancing lexicon-based text embeddings with large language models.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2020a). BART: Denoising sequence-to-sequence pre-training

for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7871–7880.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020b). Retrieval-augmented generation for knowledge-intensive nlp tasks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Li, C., Qin, M., Xiao, S., Chen, J., Luo, K., Shao, Y., Lian, D., and Liu, Z. (2024). Making text embedders few-shot learners.

Li, X. L. and Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. In Zong, C., Xia, F., Li, W., and Navigli, R., editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Li, Z., Zhang, X., Zhang, Y., Long, D., Xie, P., and Zhang, M. (2023). Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. (2024). Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

Liu, Y.-T., Liu, T.-Y., Qin, T., Ma, Z.-M., and Li, H. (2007). Supervised rank aggregation. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 481–490, New York, NY, USA. Association for Computing Machinery.

Lu, Z., Zhou, A., Ren, H., Wang, K., Shi, W., Pan, J., Zhan, M., and Li, H. (2024). MathGenie: Generating synthetic data with question back-translation for enhancing mathematical reasoning of LLMs. In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2732–2747, Bangkok, Thailand. Association for Computational Linguistics.

Ma, X., Wang, L., Yang, N., Wei, F., and Lin, J. (2023a). Fine-tuning llama for multi-stage text retrieval.

Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. (2023b). Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv: Arxiv-2310.12931*.

McInnes, L., Healy, J., and Melville, J. (2020). Umap: Uniform manifold approximation and projection for dimension reduction.

Meng, Y., Xia, M., and Chen, D. (2024). SimPO: Simple preference optimization with a reference-free reward. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Mitra, B., Diaz, F., and Craswell, N. (2017). Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 1291–1299, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

Mitra, B., Nalisnick, E., Craswell, N., and Caruana, R. (2016). A dual embedding space model for document ranking.

Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. (2023). Mteb: Massive text embedding benchmark.

Mukherjee, S., Mitra, A., Jawahar, G., Agarwal, S., Palangi, H., and Awadallah, A. (2023). Orca: Progressive learning from complex explanation traces of gpt-4.

Ni, J., Ábrego, G. H., Constant, N., Ma, J., Hall, K. B., Cer, D., and Yang, Y. (2021). Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models.

Nogueira, R. and Cho, K. (2019). Passage re-ranking with bert.

Nogueira, R., Yang, W., Cho, K., and Lin, J. (2019). Multi-stage document ranking with bert.

OpenAI (2023). Text embedding 3 model. `https://openai.com/index/new-embedding-models-and-api-updates/`. Accessed: 2024-12-29.

OpenAI (2024a). Gpt-4 technical report.

OpenAI (2024b). Gpt-4o system card.

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Pfeiffer, J., Vulić, I., Gurevych, I., and Ruder, S. (2020). MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. *OpenAI Blog*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Robertson, S., Walker, S., Jones, S., Hancock-Beaulieu, M., and Gatford, M. (1994). Okapi at trec-3. pages 0–.

Salton, G. and McGill, M. J. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Su, H., Shi, W., Kasai, J., Wang, Y., Hu, Y., Ostendorf, M., Yih, W.-t., Smith, N. A., Zettlemoyer, L., and Yu, T. (2023a). One embedder, any task: Instruction-finetuned text embeddings. In Rogers, A., Boyd-Graber, J., and Okazaki, N., editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1102–1121, Toronto, Canada. Association for Computational Linguistics.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. (2023b). Roformer: Enhanced transformer with rotary position embedding.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Truong, T. H., Otmakhova, Y., Baldwin, T., Cohn, T., Lau, J. H., and Verspoor, K. (2022). Not another negation benchmark: The NaN-NLI test suite for sub-clausal negation. In He, Y., Ji, H., Li, S., Liu, Y., and Chang, C.-H., editors, *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 883–894, Online only. Association for Computational Linguistics.

van den Oord, A., Li, Y., and Vinyals, O. (2019). Representation learning with contrastive predictive coding.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

wai Yim, W., Fu, Y., Abacha, A. B., Snider, N., Lin, T., and Yetisgen, M. (2023). Aci-bench: a Novel Ambient Clinical Intelligence Dataset for Benchmarking Automatic Visit Note Generation. *Scientific Data*, 10(1):586.

Wang, L., Yang, N., Huang, X., Jiao, B., Yang, L., Jiang, D., Majumder, R., and Wei, F. (2022). Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*.

Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., and Wei, F. (2024a). Improving text embeddings with large language models. In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916, Bangkok, Thailand. Association for Computational Linguistics.

Wang, S., Deng, Q., Feng, S., Zhang, H., and Liang, C. (2024b). A survey on rank aggregation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., Chi, E., Le, Q. V., and Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Wu, S. (2013). The weighted condorcet fusion in information retrieval. *Information Processing Management*, 49(1):108–122.

Xiao, S., Liu, Z., Zhang, P., and Muennighoff, N. (2023). C-pack: Packaged resources to advance general chinese embedding.

Yang, A., Yang, B., Hui, B., Zheng, B., et al. (2024). Qwen2 technical report.

Ye, M., Zhang, X., Yuen, P. C., and Chang, S.-F. (2019). Unsupervised embedding learning via invariant and spreading instance feature.

Young, H. (1977). Extending condorcet's rule. *Journal of Economic Theory*, 16(2):335–353.

Young, H. P. (1988). *Condorcet's Theory of Voting*, volume 16.

Young, H. P. and Levenglick, A. (1978). A consistent extension of condorcet's election principle. *SIAM Journal on Applied Mathematics*, 35(2):285–300.

Yu, Y., Liang, C., Ruan, W., and Jiang, L. (2020). Crowdsourcing-based ranking aggregation for person re-identification. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1933–1937.

Yue, X., Yao, Z., and Sun, H. (2022). Synthetic question value estimation for domain adaptation of question answering. In Muresan, S., Nakov, P., and Villavicencio, A., editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1340–1351, Dublin, Ireland. Association for Computational Linguistics.

Zhang, D. (2024). Stella english 400m v5. `https://huggingface.co/dunzhang/stella_en_400M_v5/tree/main`. Accessed: 2024-12-29.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. (2023). Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Zhong, M., Yin, D., Yu, T., Zaidi, A., Mutuma, M., Jha, R., Awadallah, A. H., Celikyilmaz, A., Liu, Y., Qiu, X., and Radev, D. (2021). Qmsum: A new benchmark for query-based multi-domain meeting summarization.