TU WIEN Informatics

Verbesserung der Produktsuche durch Large Language Models und synthetischer Abfragegenerierung

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Lukas Burtscher, BSc
Matrikelnummer 11925939

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Allan Hanbury

Wien, 1. Dezember 2024

Lukas Burtscher	Allan Hanbury

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Enhancing Embedding-based Product Search using Large Language Models and Synthetic Query Generation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Lukas Burtscher, BSc
Registration Number 11925939

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Allan Hanbury

Vienna, December 1, 2024

Lukas Burtscher	Allan Hanbury

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Lukas Burtscher, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Dezember 2024

Lukas Burtscher

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Diplomarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Herr Allan Hanbury, der meine Diplomarbeit betreut und begutachtet hat. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.

Abschließend möchte ich mich bei meinen Eltern bedanken, welche mir mein Studium durch ihre Unterstützung ermöglicht haben und stets ein offenes Ohr für mich hatten.

Lukas Burtscher

Wien, 18.05.2025

Acknowledgements

At this point, I would like to thank all those who have supported and motivated me during the preparation of this diploma's thesis.

First and foremost, my gratitude goes to Mr. Allan Hanbury, who supervised and reviewed my diploma's thesis. I would like to sincerely thank him for his helpful suggestions and constructive criticism during the creation of this work.

Finally, I want to thank my parents, who made my studies possible through their support and always had an open ear for me.

Lukas Burtscher

Vienna, May 18, 2025

Kurzfassung

E-Commerce-Plattformen haben sich zu integralen Bestandteilen des täglichen Lebens entwickelt und erfordern fortschrittliche Retrieval-Systeme, um eine optimale Benutzererfahrung zu gewährleisten. Traditionelle Methoden zur lexikalischen Term-Übereinstimmung, wie BM25, haben Schwierigkeiten mit semantischen Nuancen und scheitern daran, effektiv zu generalisieren, wenn kontextuelles Verständnis erforderlich ist. Embedding-basierte Retrieval-Modelle (EBR) beheben diese Einschränkungen, sind jedoch durch den Bedarf an umfangreichen, überwachten Datensätzen eingeschränkt.

Diese Arbeit untersucht die Nutzung von Large Language Models (LLMs) zur Generierung synthetischer Produktsuchanfragen, um angereicherte Trainingsdatensätze für EBR-Systeme zu erstellen. Durch die Feinabstimmung von BART-Modellen werden synthetische Datensätze generiert, um eine zweisäulige neuronale Retriever-Architektur zu trainieren. Dieser Ansatz wird mit traditionellen BM25-Methoden und vortrainierten, auf LLM basierenden Retrievern anhand von Metriken wie NDCG und Precision@k verglichen.

Die Ergebnisse zeigen Verbesserungen in der Genauigkeit und Relevanz von Retrievals und demonstrieren die Wirksamkeit synthetischer Daten zur Bewältigung von Herausforderungen bei der Datenknappheit. Diese Arbeit etabliert ein Framework zur Integration von LLMs in moderne Retrieval-Architekturen für Produktsuchen im E-Commerce.

Abstract

E-commerce platforms have become integral components of daily life, requiring advanced retrieval systems to ensure an optimal user experience. Traditional lexical term matching methods, such as BM25, struggle with semantic nuances and fail to generalize effectively for tasks that require contextual understanding. Embedding-based retrieval (EBR) models address these limitations but are hindered by the need for extensive supervised datasets.

This thesis investigates the use of Large Language Models (LLMs) to generate synthetic product search queries, creating enriched training datasets for EBR systems. Using fine-tuned BART models, synthetic datasets are generated to train a two-tower neural retriever architecture. This approach is compared against traditional BM25 methods and pre-trained LLM-based retrievers using metrics such as NDCG and Precision@k.

The findings reveal improvements in retrieval accuracy and relevance, demonstrating the efficacy of synthetic data in addressing challenges of data scarcity. This work establishes a framework for integrating LLMs with modern retrieval architectures in the search for e-commerce products.

Contents

CHAPTER 1

Introduction

In recent years, online shopping has become an integral part of daily life, with major e-commerce platforms like *eBay*, *Amazon*, *Taobao*, and *JD* attracting hundreds of millions of active users each day and facilitating billions of transactions [44, 76, 95]. The scale and frequency of these interactions highlight the need for effective and efficient search and recommendation systems to enhance user experience and satisfaction.

Traditionally, lexical term-matching algorithms such as BM25 [73] have been the cornerstone of retrieval systems due to their efficiency. However, these algorithms often fall short when it comes to retrieving relevant passages that do not share lexical overlap with the user's query, and they cannot be fine-tuned for specific downstream tasks [16, 45]. To address these limitations, embedding-based retrieval (EBR) models have emerged as a leading trend in the industry. EBR models leverage semantic embeddings to capture the meaning of queries and documents, allowing for more accurate and contextually relevant matches [40].

Despite their advantages, EBR systems require large amounts of supervised training data, which can be a significant barrier to their implementation [42, 51]. This thesis aims to overcome this challenge by utilizing Large Language Models (LLMs) to generate synthetic product search queries. By creating a comprehensive dataset of synthetic queries, we can train neural retriever models more effectively. The performance of these models will be evaluated against a zero-shot lexical term-matching baseline, such as BM25, to demonstrate the efficacy of synthetic data in enhancing EBR systems.

Traditional embedding-based retrieval systems utilize a single-tower approach, where one neural network is responsible for learning embeddings for both product information and query text [10]. In contrast, our approach adopts a two-tower model, employing separate neural networks for queries and products. This architecture allows for more precise control over the relevance between query-product pairs [40]. Research indicates

that two-tower models generally achieve higher accuracy and greater system efficiency compared to single-tower models [10, 56].

The innovative approach of leveraging LLMs for synthetic query generation not only addresses the data scarcity issue but also holds the potential to significantly improve the accuracy and relevance of search results in e-commerce platforms. This thesis will explore the generation process, the training of a two-tower retriever on synthetic data, and the comparative evaluation against traditional lexical methods, ultimately contributing to the advancement of recommender systems in the realm of online shopping.

1.1 Problem Statement

In the domain of product search, the performance of retrieval systems is crucial for enhancing user experience on e-commerce platforms. Embedding-based recommender systems have emerged as a state-of-the-art approach. However, their effectiveness is heavily reliant on large amounts of supervised training data, which is often limited in real-world applications. This scarcity of data presents a significant challenge to the deployment and performance of embedding-based recommender systems.

To address this issue, the primary focus of this thesis is to mitigate the challenge of data scarcity in product search by generating synthetic training data. We propose fine-tuning LLMs to create synthetic search queries that correspond to existing product descriptions. These synthetic queries will serve as a rich source of training data, enabling the development of a robust retrieval system.

The objective of this work is to demonstrate that a two-tower neural retriever, trained on synthetically generated data, can outperform traditional BM25 [73] algorithms. By leveraging the advanced capabilities of Transformers [83] and the architectural benefits of two-tower models, we aim to establish a new benchmark in product search performance, ultimately leading to more accurate and relevant search results for users on e-commerce platforms.

1.2 Key Research Questions

- **RQ1a:** How can synthetic query generation using LLMs bridge the data scarcity gap in training embedding-based retrievers?
- **RQ1b:** How accurately do synthetic search queries generated by a fine-tuned BART model correspond to their respective product descriptions, as measured by cosine similarity of embedding vectors, ROUGE scores, and qualitative data quality assessments?
- **RQ2:** What architectural design choices (e.g., layer and normalization strategies) optimize the integration of synthetic data for semantic product retrieval, as measured by Accuracy?

- **RQ3:** Under what conditions do synthetic data retriever outperform or complement traditional lexical methods like BM25, as measured by NDCG@10?

1.3 Contributions

This thesis makes significant advancements in the domain of product search and retrieval systems, contributing to the state-of-the-art in the following key areas:

- **Analysis of Encoder-Decoder Architectures:** Conducts a comprehensive theoretical analysis of transformer-based encoder-decoder architectures, including BART, and explored word embedding methodologies, specifically the Alibaba GTE model. This analysis synthesizes current natural language processing theories to underpin the design and fine-tuning of a two-tower deep retrieval system, thereby enhancing the understanding and application of advanced transformer architectures in information retrieval contexts.
- **Novel Synthetic Query Generation Approach:** Proposes a novel methodology for generating synthetic search queries tailored specifically to e-commerce product descriptions by fine-tuning a BART-based language model. Unlike previous studies that adapted BART for question-answering tasks, this approach customizes the model to handle the unique demands of product search. This results in a substantial dataset of synthetic query-product pairs that enhances the training and performance of deep retrieval models in product search applications.
- **Evaluation of Dataset Quality:** Introduces evaluation strategies for assessing the quality of synthetic positive pair datasets, including both quantitative and qualitative metrics. This analysis examines the reliability and effectiveness of LLM-generated data in retrieval tasks.
- **Development of Advanced Retrieval Models with TensorFlow Recommenders:** Constructs a state-of-the-art deep retrieval model utilizing TensorFlow Recommenders. The model integrates a pre-trained sentence encoder and is fine-tuned on the synthetic dataset, achieving enhanced retrieval performance.
- **Comprehensive Performance Analysis:** Conducts a detailed comparative evaluation of the fine-tuned two-tower retrieval model against BM25 and pre-trained LLM-based retrievers on the *Test QREL (NIST)* dataset (4.1). Performance metrics such as NDCG and *Precision@k* are employed to highlight improvements and trade-offs between advanced neural retrieval models and traditional methods.

Significance of the Study

Addressing these questions will bridge the gap between traditional and modern retrieval techniques, in a narrow application domain. The insights gained from this research

will inform the development of product search systems, leveraging synthetic data and advanced neural architectures.

1.4 Methodology

The methodological approach follows a common data science methodology called CrossIndustry Standard Process for Data Mining (CRISP-DM) [26], in addition to a Literature Review. The steps in the context of this thesis are:

1. **Literature Review**

   Chapter 2 systematically explains the evolution of Information Retrieval from traditional term-matching methods to advanced neural approaches. It highlights the limitations of earlier systems, the development of probabilistic models like BM25, and the transformative impact of deep learning and data augmentation in modern IR systems.

2. **Language Model Understanding**

   Chapter 3 delves into Large Language Models (LLMs), their underlying architecture, advancements, and applications, focusing on the transformative impact of LLMs like GPT and BART on natural language processing (NLP). This gives an introduction in the experiment setup.

3. **Data Understanding and Experiment Setup**

   Chapter 4 outlines the experiment setup for the study, detailing the datasets, preprocessing methods, training environment, evaluation metrics, and baseline model used to support the research.

4. **Modeling**

   In Chapters 5 and 6, we detail the process of fine-tuning BART to generate extensive corpora of synthetic query-product pairs, addressing the challenge of insufficient training data for an EBR. We then outline our approach to constructing a two-tower retriever, including a comprehensive explanation of its architecture and hyperparameters.

5. **Evaluation**

   In Chapter 6, we evaluate our retriever model using a test set comprising relevant and irrelevant query-product pairs. We compare its performance against both a BM25 model and a zero-shot embedding model.

6. **Deployment**

   Deployment is not covered as part of this thesis.

CHAPTER 2

Background

Information Retrieval (IR) is integral to daily human activities through its application in various practical domains, such as web search engines, question-answering systems, personal assistants, chatbots, and digital libraries. The primary aim of IR is to locate and retrieve information pertinent to a user's query. Given that multiple records may be relevant, the results are typically ranked based on their relevance score to the user's query [27].

2.1 Sparse and Dense Retrieval

Traditional text retrieval systems represent queries and documents as sparse vectors where dimensions correspond to vocabulary terms, and values reflect term importance (e.g., frequency) [27]. They largely depend on matching terms between the query and the documents. The representation is high-dimensional and there is no training required for such models. However, these term-based retrieval systems face several limitations, including issues with polysemy, synonymy, and lexical gaps between the query and the documents [15]. BM25 is a typical sparse retriever that will be introduced later.

The integration of deep learning methodologies has significantly advanced traditional text retrieval systems. Neural network-based systems map the input to dense, low-dimensional vectors (e.g. 768 dimensions) via training [27]. For fine-tuning query-document relevance pairs are typically used. Convolutional neural networks [34] and recurrent neural networks [24] have notably enhanced the performance of IR by enabling more sophisticated feature extraction and sequential text modelling. Subsequent developments, such as the Transformer architecture [83] and pre-trained language models, have further revolutionized IR systems. These approaches leverage semantic and contextual representations of queries and documents, addressing lexical mismatches and improving relevance estimation through large-scale language understanding [27].

2.2 BM25

The BM25 (Best Matching 25) algorithm is a probabilistic information retrieval model based on the probabilistic relevance framework [73, 27]. It is widely used for document retrieval tasks due to its effectiveness in ranking documents by their relevance to a given query [27].

The BM25 ranking function is defined as follows:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})} \tag{2.1}$$

where:

1. $\text{score}(D,Q)$ is the relevance score of document D for query Q.
2. q_i represents the i-th query term in Q.
3. $f(q_i,D)$ denotes the frequency of the term q_i in document D.
4. $|D|$ is the length of document D in terms of the number of words.
5. avgdl is the average document length in the corpus.
6. k_1 and b are hyperparameters. k_1 typically ranges between 1.2 and 2.0, and b is usually set around 0.75.
7. $\text{IDF}(q_i)$ is the inverse document frequency of the term q_i.

The term frequency $f(q_i,D)$ is adjusted by the document length to avoid bias towards longer documents. This adjustment is performed using the denominator, which includes the parameters k_1 and b. The parameter b controls the scaling relative to the average document length avgdl. Specifically, k_1 regulates the saturation of term frequency, where higher values of k_1 allow higher frequencies to contribute more to the score until a saturation point is reached. The parameter b modifies the extent of document length normalization, with a value of 1 indicating full normalization by document length and a value of 0 indicating no normalization. The IDF component reduces the impact of common terms while enhancing the importance of rare terms, which is essential for distinguishing relevant documents from irrelevant ones.

2.2.1 Inverse Document Frequency (IDF)

Inverse Document Frequency (IDF) is a statistical measure used in information retrieval and text mining to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of documents in the

corpus that contain the word and is offset by the frequency of the word in the document [73].

The intuition behind IDF is that terms that occur in many documents are less informative than those that occur in a few. For instance, common words like “the”,“is” and “and” are not useful for distinguishing between relevant and non-relevant documents.

IDF can be mathematically defined as:

$$IDF_i = \log\left(\frac{N}{df_i}\right) \tag{2.2}$$

where: N is the total number of documents in the corpus and df_i (document frequency) is the number of documents containing the term i.

The logarithm is used to dampen the effect of terms that are too frequent. Variants of this formula may add 1 to both N and df_i to avoid division by zero and to ensure the IDF value is always positive.

In the BM25 formula, IDF is crucial because it adjusts the weight of a term based on its frequency across the entire document collection. A term with a high IDF score is rare and potentially very informative, while a term with a low IDF score is common and less informative. By combining TF (term frequency) and IDF, BM25 can provide a balanced scoring mechanism that considers both the relevance of a term within a specific document and its importance within the entire corpus [73].

Consider a corpus of 1000 documents. If the term “information” appears in 10 documents, the IDF for “information” would be:

$$IDF_{\text{information}} = \log\left(\frac{1000}{10}\right) = \log(100) = 2$$

If another term “retrieval” appears in 500 documents, its IDF would be:

$$IDF_{\text{retrieval}} = \log\left(\frac{1000}{500}\right) = \log(2) \approx 0.301$$

Thus, “information” has a higher IDF score than “retrieval”, indicating it is more specific and likely more informative.

2.2.2 Example on BM25

Let’s assume we have the following:

1. A corpus with 4 documents.
2. A query Q containing the terms $\{t1, t2\}$.

3. The following term frequencies (TF) for terms $t1$ and $t2$ in each document:

Document	TF($t1$)	TF($t2$)
$D1$	3	0
$D2$	0	2
$D3$	1	1
$D4$	2	3

4. Document lengths:

Document	Length
$D1$	100
$D2$	150
$D3$	200
$D4$	50

5. The document frequencies (DF) for terms $t1$ and $t2$ are:

$$DF(t1) = 3, \quad DF(t2) = 3$$

6. We choose the parameters $k1 = 1.5$ and $b = 0.75$.

The IDF for each term is calculated as follows:

$IDF(t1) = \log\left(\frac{N-DF(t1)+0.5}{DF(t1)+0.5} + 1\right) = \log\left(\frac{4-3+0.5}{3+0.5} + 1\right) = \log\left(\frac{1.5}{3.5} + 1\right) = \log(1.4286) \approx 0.3567$

$IDF(t2) = \log\left(\frac{N-DF(t2)+0.5}{DF(t2)+0.5} + 1\right) = \log\left(\frac{4-3+0.5}{3+0.5} + 1\right) = \log(1.4286) \approx 0.3567$

For document $D1$:

$$\text{BM25}(D1, Q) = IDF(t1) \cdot \frac{tf1_{D1} \cdot (k1+1)}{tf1_{D1} + k1 \cdot (1 - b + b \cdot \frac{\text{Length}}{\text{avgdl}})} +$$

$$IDF(t2) \cdot \frac{tf2_{D1} \cdot (k1+1)}{tf2_{D1} + k1 \cdot (1 - b + b \cdot \frac{\text{Length}}{\text{avgdl}})}$$

The average document length (avgdl) is: $avgdl = \frac{100+150+200+50}{4} = 125$

$$= 0.3567 \cdot \frac{3 \cdot (1.5+1)}{3 + 1.5 \cdot (1 - 0.75 + 0.75 \cdot \frac{100}{125})} + 0.3567 \cdot \frac{0 \cdot (1.5+1)}{0 + 1.5 \cdot (1 - 0.75 + 0.75 \cdot \frac{100}{125})}$$

$$= 0.3567 \cdot \frac{3 \cdot 2.5}{3 + 1.5 \cdot (1 - 0.75 + 0.6)} + 0$$

$$= 0.3567 \cdot \frac{7.5}{3 + 1.5 \cdot 0.85}$$

$$= 0.3567 \cdot \frac{7.5}{4.275}$$

$$= 0.3567 \cdot 1.753$$

$$\approx 0.625$$

Repeating similar calculations for $D2$, $D3$, and $D4$:

For document $D2$:

$$\text{BM25}(D2, Q) = 0.3567 \cdot \frac{0 \cdot (1.5 + 1)}{0 + 1.5 \cdot (1 - 0.75 + 0.75 \cdot \frac{150}{125})} + 0.3567 \cdot \frac{2 \cdot (1.5 + 1)}{2 + 1.5 \cdot (1 - 0.75 + 0.75 \cdot \frac{150}{125})} = 0.479$$

For document $D3$:

$$\text{BM25}(D3, Q) = 0.3567 \cdot \frac{1 \cdot (1.5 + 1)}{1 + 1.5 \cdot (1 - 0.75 + 0.75 \cdot \frac{200}{125})} + 0.3567 \cdot \frac{1 \cdot (1.5 + 1)}{1 + 1.5 \cdot (1 - 0.75 + 0.75 \cdot \frac{200}{125})} = 0.562$$

For document $D4$:

$$\text{BM25}(D4, Q) = 0.3567 \cdot \frac{2 \cdot (1.5 + 1)}{2 + 1.5 \cdot (1 - 0.75 + 0.75 \cdot \frac{50}{125})} + 0.3567 \cdot \frac{3 \cdot (1.5 + 1)}{3 + 1.5 \cdot (1 - 0.75 + 0.75 \cdot \frac{50}{125})} = 1.331$$

Summary of BM25 Scores

Document	BM25 Score
$D1$	0.625
$D2$	0.479
$D3$	0.562
$D4$	1.331

Thus, based on the BM25 scores, the ranking of the documents in response to the query Q would be:

$$D4 > D1 > D3 > D2$$

2.2.3 Improvements to BM25

Several improvements to BM25 have been proposed to address its limitations and enhance its performance [80]. These include *BM25L, BM25+, BM25-adpt* and *BM25T*.

BM25L

BM25's document length normalization tends to favour shorter documents over longer ones. To address this problem, Lv and Zhai introduce the *BM25L* [47] function. Their derivation begins with an adjusted version of BM25 that avoids negative values and differs only in the IDF component:

$$rsv_q = \sum_{t \in q} \log\left(\frac{N+1}{df_t + 0.5}\right) \cdot \frac{(k_1+1) \cdot tf_{td}}{k_1\left((1-b) + b \cdot \frac{L_d}{L_{avg}}\right) + tf_{td}} \tag{2.3}$$

They re-arrange to get

$$rsv_q = \sum_{t \in q} \log\left(\frac{N+1}{df_t + 0.5}\right) \cdot \frac{(k_1+1) \cdot c_{td}}{k_1 + c_{td}} \tag{2.4}$$

where

$$c_{td} = \frac{tf_{td}}{1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)} \tag{2.5}$$

For *BM25L*, Lv and Zhai aim to adjust the c_{td} component to prevent excessive penalization of long documents. They achieve this by adding a positive constant, δ, which shifts the function to better accommodate larger document lengths (i.e., small numbers in the denominator, resulting in more favourable scores for long documents). The final *BM25L* equation is expressed as:

$$\text{BM25L} = \sum_{t \in q} \log\left(\frac{N+1}{df_t + 0.5}\right) \cdot \frac{(k_1+1) \cdot (c_{td} + \delta)}{k_1 + (c_{td} + \delta)} \tag{2.6}$$

where δ is a small positive constant, typically 0.5 [47].

BM25+

BM25+ [49] is another enhancement to prevent the penalization of long documents. They lower-bound the contribution of a single term occurrence, ensuring that each term occurrence contributes at least a constant amount to the retrieval status value.

$$\text{BM25+} = \sum_{t \in q} \log\left(\frac{N+1}{df_t}\right) \cdot \left(\frac{(k_1+1)\cdot tf_{td}}{k_1\left((1-b)+b\cdot\frac{L_d}{L_{\text{avg}}}\right)+tftd} + \delta\right) \tag{2.7}$$

where δ is typically set to 1 across collections [49].

BM25-adpt

In the work on BM25 [48], Lv and Zhai note that using a global k_1 for all query terms is likely to be less effective than employing term-specific k_1 values. They aim to identify these term-specific k_1 values directly from the index, enhancing the ranking function's transferability across different collections without the need for re-training.

They address this issue by applying information gain and divergence from randomness theory. They begin with the probability of observing at least one occurrence of a term, given zero or more occurrences and the query:

$$p(1|0,q) = \frac{df_r + 0.5}{N+1} \tag{2.8}$$

they derive the probability of seeing one more occurrence as:

$$p(r+1|r,q) = \frac{df_{r+1}+0.5}{df_r+1} \tag{2.9}$$

from which the information gain at any point in the function can be computed as the change from r to $r+1$ occurrences, minus the initial probability:

$$G_q^r = \log_2\left(\frac{df_{r+1}+0.5}{df_r+1}\right) - \log_2\left(\frac{df_{r+1}+0.5}{N+1}\right) \tag{2.10}$$

df_r is more complex. Rather than using term frequency, tf_{td}, values, they define df_r based on the result of the length normalized term frequency. They do that by defining df_r as:

$$df_r = \begin{cases} |D_{t\,|\,c_{td}\geq r-0.5}| & r > 1 \\ dft & r = 1 \\ N & r = 0 \end{cases} \tag{2.11}$$

that is, for the base case of $r = 0$, the number of documents in the collection is used; when $r = 1$, the document frequency is used; in all other cases, $|D_{t|c_{td}\geq r-0.5}|$, the number of documents, $|D_t|$, containing the term, t, that have a length normalized occurrence count, c_{td}, greater than r (once rounded).

To compute k_1, they align the information gain function to the BM25 score function and solve for k_1 giving the term specific $k_1^{'}$.

$$k_1^{'} = \arg\min_{k_1} \sum_{i=0}^{T} \left(\frac{G_q^r}{G_q^1} - \frac{(k_1+1)\cdot r}{k_1 + r} \right)^2 \tag{2.12}$$

They can compute $k_1^{'}$ entirely from the index because all parameters are there – but they suggest pre-computing these values and storing them in the index, one per term. Finally, $k_1^{'}$ gets substituted into the term frequency component of BM25, and the IDF score replaced by G_q^1.

$$\text{BM25-adpt} = \sum_{t \in q} G_q^1 \cdot \frac{(k_1^{'}+1)\cdot tf_{td}}{k_1^{'} \cdot \left((1-b) + b \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} \tag{2.13}$$

2.3 Neural Information Retrieval

Neural-based approaches for semantic retrieval, including those utilizing Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have been proposed to enhance the effectiveness of information retrieval systems [25]. Figure 2.1 shows the typical design for such a dense retriever, where the idea is to independently represent text documents and queries in a continuous vector space, enabling the use of neural network-based similarity measures, such as cosine similarity or dot-product, to rank documents by their relevance to a given query [50, 40, 27]. Moreover, these neural methods can learn complex non-linear representations of text data, improving the performance of retrieval models [50, 34, 23, 44, 95].

Attention-based mechanisms, such as the Transformer architecture [83], have further advanced the ability of IR systems to focus on important parts of the query and documents for matching [90]. Additionally, pre-trained language models like BERT [16], RoBERTa [45] and GTE [41] have significantly enhanced IR systems by offering a better understanding of the semantics and context of natural language queries and documents.

In recent years, word embeddings have become a popular method for representing documents and queries in IR systems due to their ability to capture semantic meaning. These embeddings are dense, continuous representations of words that significantly improve the effectiveness of IR systems. Various approaches have leveraged word embeddings for different IR tasks. For instance, some methods use word embeddings combined with Fisher Vector aggregation to represent documents and queries [13]. Other approaches use bilingual word embeddings for cross-lingual IR [85], and some focus on representing short texts using word embeddings [33]. Additionally, dual embedding spaces have been explored to represent documents and queries more effectively [53], while some techniques use word embeddings to represent context and potential responses in natural language generation [28].

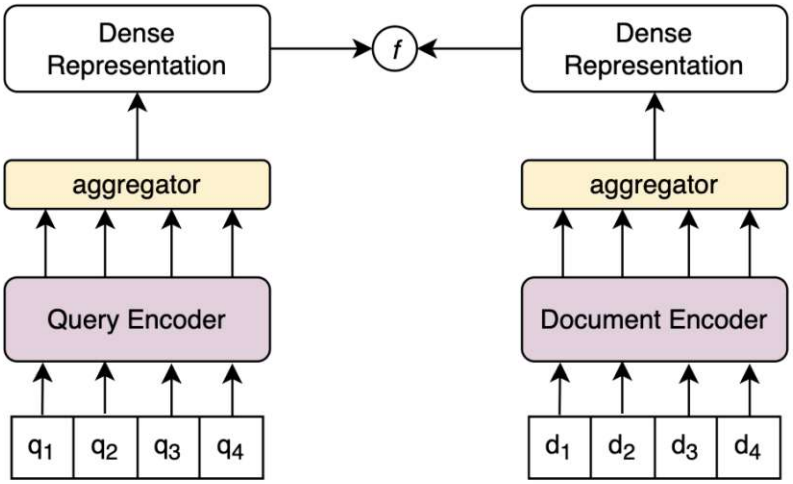

Figure 2.1: Dual-Encoder Architecture

A particularly noteworthy method is the use of pre-trained transformer-based models for text encoding. One approach for question-answering problems, called "DC-BERT" [97], employs two BERT models: an online BERT that encodes the question once, and an offline BERT that pre-encodes and caches document encodings. Another method [93] involves using a neural retrieval component and a cross-attention component to generate responses based on retrieved passages, with data augmentation methods proposed to enhance model performance.

Additionally, some research has explored the use of approximate nearest neighbour search and negative contrastive learning for dense text retrieval. For example, the ANCE [91] method employs a neural network to encode documents and queries, applying approximate nearest neighbour search and negative contrastive learning to rank documents. Other studies [27] have proposed methods for training dense retrieval models efficiently using a combination of hard and soft negative sampling and optimizing the balance between retrieval and generation components. Further advancements include a global weighted self-attention network for web search and optimized training approaches for open-domain question answering using dense passage retrieval, all contributing to improved effectiveness in their respective areas.

2.3.1 Comparison with Lexical Term-Matching Models

As previously explained, models like BM25 have been the backbone of information retrieval systems for decades. They rely on the exact matching of query terms within documents to compute relevance scores. In contrast, contemporary deep learning-based retrieval models utilize neural networks to capture semantic meanings and relationships between queries and documents, enabling a more nuanced understanding of user intent.

Term-matching models cannot handle synonyms or paraphrased expressions, and they also fail to understand context or user intent beyond exact terms. For example, if the user search query is "cheap running shoes" and the product title is "affordable jogging sneaker", BM25 will predict low relevance due to the lack of exact term matches. A language model like BERT will predict a high relevance score by recognizing synonyms such as "cheap" $\approx$ "affordable". Additionally, when dealing with complex queries, BM25 may not effectively differentiate between products when the exact terms are missing. For instance, when searching for "quadcopter with FPV capabilities", a deep learning model understands that "quadcopter" is a type of drone and "FPV" stands for "First-Person-View".

2.4 State of the Art Retrieval Models in Product Search

E-commerce search faces unique challenges compared to web search, such as shorter, less structured text and the importance of considering extensive historical user behaviors [2]. While lexical matching engines, which rely on exact query term matches, are still crucial for their reliability, they struggle to account for user-specific interests and the semantic nuances of queries [27, 51, 42].

With the advent of deep learning, companies like *Amazon* [76] and *JD* [95] have developed two-tower embedding-based retrieval systems to enhance product relevance and personalization. These systems, including one implemented by *Taobao* [40], capture the relationship between query semantics and user behavior, initially showing significant improvements in various metrics. However, EBR systems face issues with the controllability of search relevance, often failing to match query terms precisely, leading to user complaints and problematic search results [25].

To address this, a relevance control module is introduced to filter products based on exact matching signals before they proceed to the ranking stage. Despite improving relevance, this approach often discards about 30% of candidates, wasting computing resources and reducing system performance [40]. The main challenge now is to refine the EBR model to retrieve more relevant products and increase the number of candidates for the ranking stage, thereby enhancing the overall effectiveness of the search system.

2.4.1 Taobao Search

Li et al. [40] introduced a Multi-Grained Deep Semantic Product Retrieval (MGDSPR) system, designed to simultaneously model query semantics and historical behaviour data to retrieve highly relevant products. The general structure of MGDSPR is depicted in Figure 2.2. They begin by defining the problem, followed by a detailed explanation of the two-tower model, which consists of a user tower and an item (product) tower.

They first formulate the retrieval problem by introducing the following notations. Let $\mathcal{U} = \{u_1, ..., u_u, ...u_N\}$ denote a collection of $\mathcal{N}$ users $\mathcal{Q} = \{q_1, ..., q_u, ..., q_N\}$ denote the corresponding queries, and $\mathcal{I} = \{i_1, ..., i_i, ..., i_M\}$ denote a collection of $\mathcal{M}$ items (products). Also, they divide the user u's historical behaviours into three subsets according

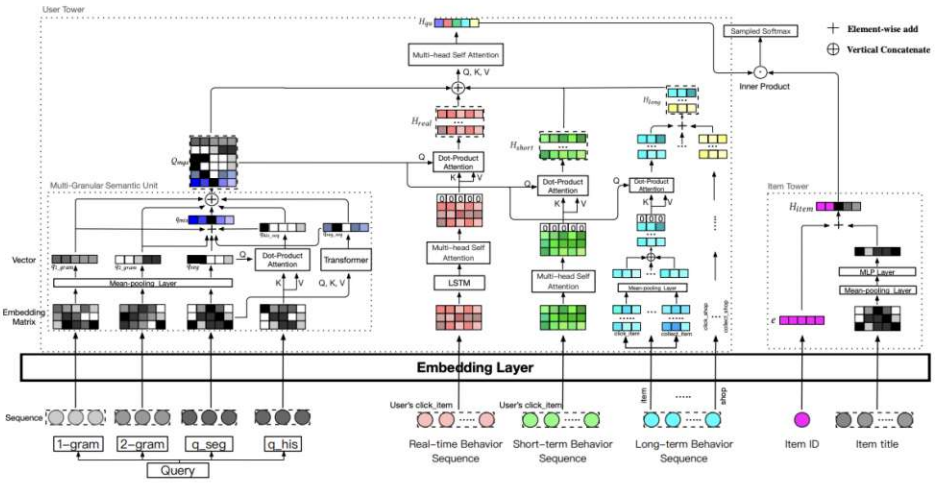

Figure 2.2: General architecture of the proposed Multi-Grained Deep Semantic Product Retrieval model (MGDSPR) [40]

to the time interval from the current time t: real-time (denoted as $\mathcal{R}^u = \{i_1^u, ..., i_t^u, ..., i_T^u$, before the current time step), short-term denoted as $\mathcal{S}^u = \{i_1^u, ..., i_t^u, ..., i_T^u\}$, before $\mathcal{R}^u$ and within ten days) and long-term sequences (denoted as $\mathcal{L}^u = \{i_1^u, ..., i_t^u, ..., i_T^u\}$, before $\mathcal{S}^u$ and within one month), where T is the length of the sequence.

They now define the task. Given the historical behaviors $(\mathcal{R}^u, \mathcal{S}^u, \mathcal{L}^u)$ of a user $u \in \mathcal{U}$, once he/she submits query q_u at time t, their goal is to return a set of items $i \in \mathcal{I}$ that fulfil his/her search request. Typically, they predict the top-K item candidates from $\mathcal{I}$ at time t based on the scores z between the user (query and behavior) and the items, expressed as:

$$z = F\left(\phi(q_u, \mathcal{R}^u, \mathcal{S}^u, \mathcal{L}^u), \psi(i)\right),$$

where $F(\cdot)$ represents the scoring function, $\phi(\cdot)$ is the query and behaviors encoder, and $\psi(\cdot)$ is the item encoder. They utilize a two-tower retrieval model for efficiency, with F instantiated as the inner product function. The following sections describe the design of the user and item towers, respectively.

User Tower

Queries in Taobao search are usually in Chinese and often short. After query segmentation, the average length is less than three words. Therefore, a multi-granular semantic unit is proposed to capture the meaning of queries from multiple semantic granularities and enhance their representation (e.g., individual words, word pairs, or segments). Given a query segmentation result $q_u = \{w_1^u, \ldots, w_n^u\}$, where each $w^u = \{c_1^u, \ldots, c_m^u\}$, and its

historical query $q_{his} = \{q_1^u, \ldots, q_k^u\} \in \mathbb{R}^{k \times d}$, six granular representations $Q_{mgs} \in \mathbb{R}^{6 \times d}$ can be obtained by concatenating the unigram mean-pooling q_{1_gram}, 2-gram mean-pooling q_{2_gram}, word segmentation mean-pooling q_{seg}, word segmentation sequence q_{seg_seq}, historical query words q_{his_seq}, and mixed q_{mix} representations. Mean pooling is a technique to combine multiple vector representations into a single fixed-dimensional vector by averaging them [12]. For example when unigram mean-pooling the word "dress" you compute the embeddings for each character and average them.

User behaviors are captured through their history of items clicked or bought. Taking the user's short-term behaviors S^u as an example, $i_t^u \in S^u$ denotes the user's click on item i at time t, where each item i is described by its ID and side information F (e.g., category, brand, shop). Each input item $i_{ut} \in S_u$ is defined by:

$$e_i^f = W_f \cdot x_i^f \quad i_t^u = \text{concat}(\{e_i^f | f \in F\}) \tag{2.14}$$

where W_f is the embedding matrix, x_i^f is a one-hot vector, and $e_i^f \in \mathbb{R}^{1 \times d_f}$ is the corresponding embedding vector of size d_f.

To retrieve products relevant to the current user's query while preserving personalized characteristics, the multi-granular semantic representation Q_{mgs} and personalized representations $(H_{real}, H_{short}, H_{long})$ are combined as the input to a self-attention mechanism, dynamically capturing the relationship between them. A "[CLS]" token is added at the first position of the input $I = \{[CLS], Q_{mgs}, H_{real}, H_{short}, H_{long}\}$, and the output is regarded as the user tower's representation $H_{qu} \in \mathbb{R}^{1 \times d}$:

$$H_{qu} = \text{Self\_Att}^{first}([[CLS], Q_{mgs}, H_{real}, H_{short}, H_{long}]) \tag{2.15}$$

Item Tower

In [40] they use the item ID and title to obtain the item representation H_{item}. Given the representation of item i's ID, $e_i \in \mathbb{R}^{1 \times d}$, and its title segmentation result $T_i = \{w_{i1}, \ldots, w_{iN}\}$, $H_{\text{item}} \in \mathbb{R}^{1 \times d}$ is calculated as follows:

$$H_{\text{item}} = e + \tanh\left(W_t \cdot \frac{\sum_{i=1}^{N} w_i}{N}\right), \tag{2.16}$$

where W_t is the transformation matrix. They empirically find that applying LSTM [30] or Transformer [83] to capture the context of the title is not as effective as simple mean-pooling since the title is often composed of keywords and lacks grammatical structure.

2.4.2 Ebay Search

This approach to product recommendations [88] involves training a two-tower deep learning model that simultaneously generates user embeddings and item embeddings. The architecture of this model is illustrated in Figure 2.3 and is detailed below.

Inspired by the work of Covington et al. [14], the task of generating recommendations is defined as a classification problem utilizing softmax probability.

$$P(s_i|U) = \frac{e^{\gamma(v_i,u)}}{\sum_{j \in V} e^{\gamma(v_j,u)}}, \tag{2.17}$$

where:

1. $u \in \mathbb{R}^D$ is a D-dimensional vector for the embedding of user U
2. $v_i \in \mathbb{R}^D$ is a D-dimensional vector for the embedding of item s_i
3. γ is the affinity function between user and item
4. V represents all items available on eBay. As V could contain billions of items, it is infeasible to perform a full-size softmax operation. Negative sampling is used to limit the size of V

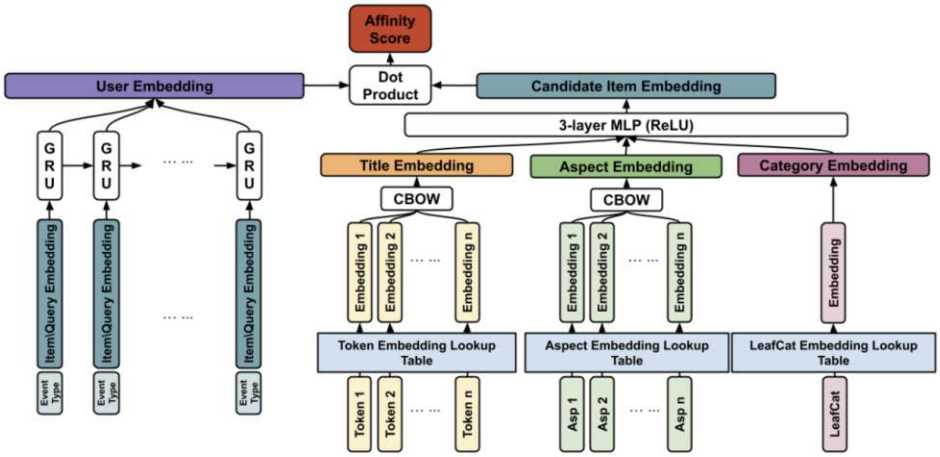

Figure 2.3: Model architecture with recurrent user representation [88]

User Tower

A user's activity on an e-commerce platform includes various actions such as viewing items, making search queries, adding items to the cart, and more. These actions provide valuable signals for generating personalized recommendations. Wang et al. [88] incorporate multi-modal user activities into the model, focusing on item viewing and search queries as representatives of item-based and query-based events.

For item-based events, they map each item s_{z_i} to its embedding v_{z_i} and concatenate it with a 4-dimensional vector e_{z_i} representing the event type. For search queries, each

query is modeled as a "pseudo-item" with the query text replacing the item title and the dominant query category (predicted using a separate model) replacing the item category.

The vector representation for each user event z_i is:

$$E(z_i) = \text{concat}(v_{z_i}, e_{z_i}) \tag{2.18}$$

With Continuous Bag-of-Events Representation, the model aggregates all event embeddings into a single vector by averaging them. An MLP layer with L layers, H hidden dimensions, and ReLU activation functions generates the D-dimensional user embedding u:

$$\tilde{u} = \text{MLP}\left(\frac{1}{n}\sum_{i=1}^{n} E(z_i)\right), \quad u = \frac{\tilde{u}}{\|\tilde{u}\|} \tag{2.19}$$

To incorporate the ordering of user events, the model uses a Gated Recurrent Unit (GRU). The GRU processes the sequence of event embeddings, with update rules defined as:

$$r_t = \sigma(W_r x_t + U_r h_{t-1}), \quad u_t = \sigma(W_u x_t + U_u(r_t \odot h_{t-1})) \quad \tilde{h}_t = \tanh(W x_t + U(r_t \odot h_{t-1}))$$
$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t$$

The user embedding u is generated by averaging the output vectors from all GRU steps:

$$\tilde{u} = \frac{1}{n}\sum_{i=1}^{n} l_i, \quad u = \frac{\tilde{u}}{\|\tilde{u}\|} \tag{2.20}$$

This recurrent representation captures the order of user activities.

Item Tower

For title and aspect features, raw text is tokenized into embeddings of size D_{text} using the Continuous-Bag-of-Words (CBOW) [52] approach. The vocabulary for titles consists of approximately 400K tokens specific to eBay item titles. Aspect features come from a production database with around 100K tokens. Category values are mapped into an embedding space of size D_{category} using a lookup table. All embedding tables are trained from scratch with random initialization [88].

The item feature embeddings z_i are concatenated and passed through a Multi-Layer Perceptron (MLP) with L hidden layers, H hidden dimensions, and ReLU activation to generate a D-dimensional item embedding v_i:

$$z_i = \text{concat}(z_{\text{title}i}, z_{\text{aspect}i}, z_{\text{category}i}), \quad \tilde{v}_i = \text{MLP}(z_i), \quad v_i = \frac{\tilde{v}_i}{\|\tilde{v}_i\|} \tag{2.21}$$

The item embedding v_i is normalized to unit length [88].

2.5 Data Augmentation for Information Retrieval

Another significant challenge in developing neural models for information retrieval (IR) is the scarcity of domain-specific training data. Creating high-quality datasets manually is arduous as it necessitates obtaining queries from real users. Although there are a lot of datasets from diverse domains ranging from Wikipedia, scientific publications, news and many more [5, 37, 84, 75], according to the BEIR Benchmark [78] in-domain performance cannot predict how well a model will generalize. Many neural retriever that outperformed BM25 on an in-domain evaluation, performed poorly on other datasets. Datasets need diverse pooling strategies for fair assessment [78].

To address this issue, data augmentation methods have been employed to increase the amount of training data, thereby aiding the learning process of data-driven models. In low-resource settings, small-scale language models (LMs) have been utilized to generate synthetic data for various natural language processing (NLP) tasks [18, 35]. Fadaee et al. [18] discuss data augmentation in a low-resource Neural Machine Translation (NMT) context, particularly focusing on generating synthetic data to improve translation performance. They observed increases in BLEU [65] scores of up to 2.9 points over the baseline model, demonstrating that their augmentation approach substantially improved translation accuracy. This is achieved by generating novel contexts for rare words, which allowed the model to better generalize to unseen words and phrases. Kobayashi et al. [35] used contextual augmentation, via a bi-directional language model, to improve the model's accuracy across six benchmark datasets. For example, it improved performance on sentiment analysis tasks (SST2, SST5) and question classification (TREC) over baseline models without augmentation. The improvement is more significant than with synonym-based augmentation. Recent studies demonstrate that large pre-trained language models can generate task-specific synthetic data to address low-resource scenarios, with quality validated through downstream performance improvements. For text classification, models fine-tuned on limited labeled data generate class-conditioned sentences, which are filtered via classifier-based curation to improve accuracy [3]. In relation extraction, GPT-2-generated training examples for biomedical relations enhance BERT-based classifiers by up to 11 F1 points [62]. For commonsense reasoning, generative augmentation with diversity-aware selection improves both in-distribution accuracy and out-of-distribution robustness [92]. Similarly, GPT-2-simulated user-agent dialogues trained on limited human annotations achieve significant gains in low-resource dialogue systems [54]. These results indicate that synthetic data quality is "reasonable" when it measurably mitigates data scarcity and improves task-specific metrics like F1, accuracy, and generalization.

In the realm of information retrieval, dense retrievers have demonstrated comparable performance to BM25 on certain datasets when pre-trained exclusively on documents without relevance labels (unsupervised) [69, 31, 58]. These methods involve creating synthetic pairs from the raw documents. This is done by looking at overlapping spans. If the span is "Vienna" we create a query by taking a snippet from one document, like "Vienna is the capital of...", other documents that include the same span are then positive pairs while others are negative pairs.

In [7], documents are randomly sampled from various collections, and GPT-3's Curie model [8] is employed to generate questions based on these documents. Two prompt templates are utilized: **Vanilla**, which involved randomly choosing pairs of documents and relevant questions from the MS MARCO [5] training dataset, and **Guided by Bad Questions (GBQ)**, which used MS MARCO questions as examples of "bad" questions and manually created more complex "good" questions. The generated questions are filtered based on their relevance to the documents, retaining the top 10,000 pairs for training. BM25 is used for initial retrieval, followed by reranking using the monoT5 [61] model adapted for text ranking. Retrievers fine-tuned on the synthetic data outperformed BM25 and some self-supervised dense retrieval methods. The experiment demonstrated the effectiveness of using large LMs to generate synthetic training data for IR tasks. The synthetic data helped in adapting retrievers to specific domains, addressing the challenge of limited domain-specific training data.

In [42], the core idea involves using a sequence-to-sequence (seq2seq) model for synthetic query generation, addressing the challenge of obtaining labeled training data. The authors utilized BART [39], a pre-trained Transformer-based seq2seq model, for query generation. The model is fine-tuned on MS MARCO [5] positive query-passage pairs to learn query generation from passages. The model is then applied to passages from English Wikipedia to generate synthetic query-passage pairs. For each passage, 10 synthetic queries are generated using nucleus sampling with a probability $p = 0.95$, and the top 5 queries based on likelihood scores are retained, resulting in a dataset named WIKIGQ, comprising 110 million synthetic query-passage pairs. The retrieval models, initialized with BERT-base, are pre-trained on the WIKIGQ dataset. This pre-training involves a two-tower architecture, where query and document embeddings are constructed separately, and the dot product of these embeddings measures similarity. The experiments demonstrate that the synthetic queries significantly improve retrieval performance, with models pre-trained on synthetic data often outperforming those trained on real data in zero-shot settings. Key findings in [7] indicate that models trained on synthetic data can outperform classical retrieval methods, such as BM25, and even surpass models trained on real datasets. Synthetic data enhances both performance and robustness, allowing retrieval models to generalize better to unseen data. For example, in non-Wikipedia domains like ANTIQUE and BIOASQ, models fine-tuned with domain-specific synthetic data displayed substantial improvements, suggesting that synthetic data helps bridge domain gaps and adapts models to target contexts effectively.

Additionally, Ma et al. [51] utilize an encoder-decoder architecture with Transformer layers for question generation [83]. The encoder creates a representation of a passage, while the decoder formulates a plausible question that the passage could answer. The trained model is then applied to target domain passages to produce synthetic question-passage pairs. Inputting target domain passages creates noisy but relevant question-passage pairs. Ma et al. [51] reveal that the augmented data aids models in adapting across different domains and even improves retrieval performance close to that of supervised models in certain cases. The study also introduces a hybrid approach combining BM25 (term-based)

and neural models, which leverages the strengths of both methods to enhance retrieval results. This approach shows significant gains in specialized domains, such as biomedical literature, where traditional models struggle without domain-specific supervision

2.6 Summary

Information retrieval bridges users with relevant data through systems like search engines and chatbots. Traditional sparse retrieval methods, such as BM25, rely on lexical term matching and probabilistic frameworks to rank documents. While effective and efficient, these models struggle with semantic nuances, synonymy, and polysemy. Enhancements like BM25L, BM25+, and BM25-adpt address limitations such as document length bias and term-specific parameter tuning, improving robustness across diverse corpora.

Neural retrieval models, powered by deep learning, have revolutionized IR by encoding queries and documents into dense vector spaces. Architectures like dual encoders, transformers, and pre-trained language models (e.g., BERT, RoBERTa) capture semantic and contextual relationships, overcoming lexical gaps. Applications in e-commerce, such as Taobao's Multi-Grained Deep Semantic Product Retrieval and eBay's two-tower recommendation system, demonstrate how neural models integrate user behavior and multi-modal data to deliver personalized results. These systems balance efficiency (via pre-encoding) and relevance, though challenges like controllability persist.

Data scarcity remains a critical hurdle, particularly in domain-specific and multilingual settings. Synthetic data generation, using large language models (e.g., GPT-3, BART) to augment training corpora, has proven effective. Techniques like query generation and contrastive learning with synthetic pairs enable models to generalize across domains, achieving performance comparable to supervised methods. Hybrid approaches, combining sparse and dense retrievers, further enhance robustness, underscoring the synergy between classical and modern paradigms in advancing IR systems.

CHAPTER 3

Large Language Models

Large Language Models (LLMs) are a subset of artificial intelligence (AI) models designed to understand and generate human language. They employ deep learning techniques, particularly neural networks, to process and produce text that is contextually relevant and coherent [11]. These models are trained on extensive amounts of textual data, enabling them to capture the intricacies of language, including syntax, semantics, and even some aspects of pragmatics. Large Language Models, such as GPT (Generative Pre-trained Transformer) [11], BERT (Bidirectional Encoder Representations from Transformers) [16], and BART (Bidirectional and Auto-Regressive Transformers) [39], have revolutionized the field of natural language processing (NLP). They are typically constructed using transformer architecture [83], which allows them to manage the complexity and variability of human language.

3.1 Transformers

The deep learning field is undergoing a significant transformation due to the rapid evolution of Transformer models. These models have not only set new benchmarks in Natural Language Processing (NLP) but have also expanded their impact across various aspects of artificial intelligence [16, 83].

Transformer models are distinguished by their unique attention mechanisms and ability to process data in parallel, leading to unprecedented accuracy and efficiency in understanding and generating human language [16, 83].

Transformers were initially developed to address the challenge of sequence transduction, or neural machine translation, which involves converting an input sequence into an output sequence [83]. This capability to transform sequences is the reason behind their name, “Transformers”.

At the time of the Transformer model introduction, Recurrent Neural Networks (RNNs) were the preferred approach for handling sequential data, characterized by a specific order in its input.

3.1.1 The shift from RNNs to Transformers

RNNs [30] function similarly to a feed-forward neural network but process the input sequentially, one element at a time.

Transformers were inspired by the encoder-decoder architecture found in RNNs. However, instead of using recurrence, the Transformer model is entirely based on the attention mechanism [83].

Besides improving RNN performance, Transformers have introduced a new architecture to solve many other tasks, such as text summarization, image captioning, and speech recognition.

RNNs face two main problems for NLP tasks:

1. They process input data sequentially, which does not leverage modern GPUs designed for parallel computation, resulting in slow training.
2. They become ineffective when elements are distant from each other due to the potential loss of information over long sequences.

The shift from RNNs like LSTM to Transformers in NLP is driven by these issues and Transformers' ability to address them through the attention mechanism. Attention allows the model to focus on specific words, regardless of their distance and it enhances performance speed by enabling parallel computation.

3.1.2 Model Architecture

When considering a Transformer (Figure 3.1) for language translation as a simple black box, it takes a sentence in one language, such as English, as an input and outputs its translation in German.

The encoder maps an input sequence of symbol representations $(x_1, \ldots, x_n)$ to a sequence of continuous representations $z = (z_1, \ldots, z_n)$. Given z, the decoder then generates an output sequence $(y_1, \ldots, y_m)$ of symbols one element at a time. At each step, the model is auto-regressive [24], consuming the previously generated symbols as additional input when generating the next. The Transformer follows this architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder.

Encoder

The encoder serves as a crucial element within the Transformer architecture, primarily tasked with converting input tokens into contextualized representations. Unlike preceding

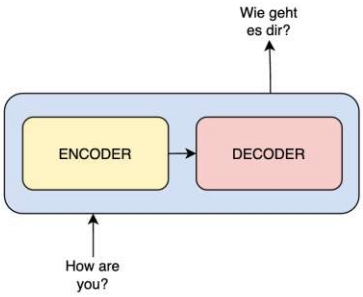

Figure 3.1: Global structure of Encoder-Decoder

models, which processed tokens in isolation, the Transformer encoder captures the context of each token relative to the entire sequence [83]. This structural overview is depicted in Figure 3.2.

The embedding process is performed exclusively in the bottom-most encoder. The encoder initiates by transforming input tokens-whether words or subwords-into vectors through embedding layers [83, 16]. These embeddings capture the semantic meaning of the tokens, converting them into numerical vectors.

Each encoder receives a list of vectors, each with a fixed size of 512 [83]. In the bottom encoder, these vectors represent the word embeddings, while in subsequent encoders, they are the outputs from the encoder directly below them.

Since Transformers do not have a recurrence mechanism like RNNs, they use positional encodings added to the input embeddings to provide information about the position of each token in the sequence [16]. This allows the model to understand the position of each word within the sentence.

It is suggested to use a combination of various sine and cosine functions to create positional vectors, enabling the use of positional encoding for sentences of any length [83]. Each dimension is represented by unique frequencies and offsets of the wave, with values ranging from -1 to 1, effectively representing each position [19].

The Transformer encoder is composed of a stack of identical layers, with the original Transformer model featuring six layers [83]. Each encoder layer transforms input sequences into continuous, abstract representations that encapsulate information learned from the entire sequence. This process includes residual connections around each sublayer, followed by layer normalization. The output of each sub-layer is given by $LayerNorm(x + Sublayer(x))$, where $Sublayer(x)$ is the function implemented by the sub-layer itself.

Within the encoder, the multi-headed attention mechanism employs self-attention. This mechanism allows the model to associate each word in the input with other words, enabling the encoder to focus on different parts of the input sequence as it processes each token. For instance, the model can learn to link the word “are” with “you”.

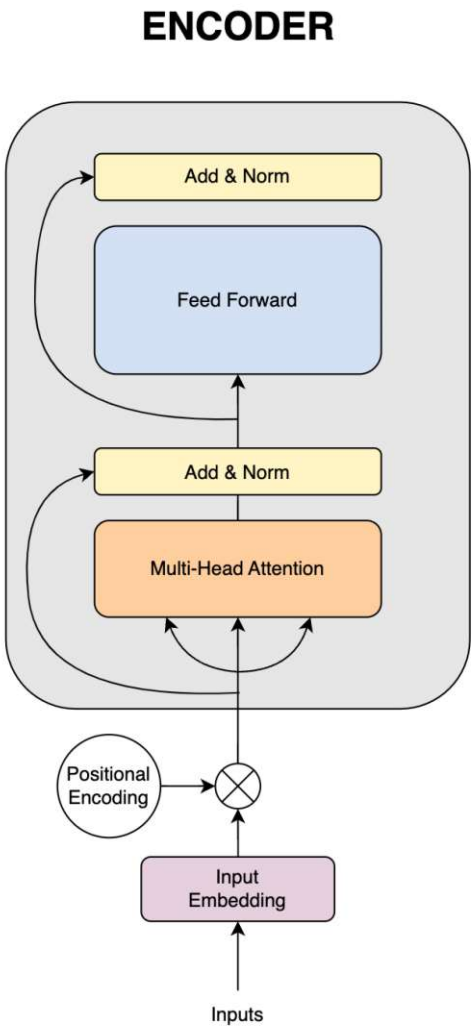

Figure 3.2: Encoder Architecture

As shown in Figure 3.4, the attention scores are computed using three components:

- The query, a vector representing a specific word or token from the input sequence.
- The key, also a vector corresponding to each word or token in the input sequence.
- The value, associated with a key, used to construct the output of the attention layer.

When a query and a key have a high attention score, the corresponding value is emphasized in the output. The self-attention module allows the model to capture contextual information from the entire sequence. Queries, keys, and values are linearly projected h times,

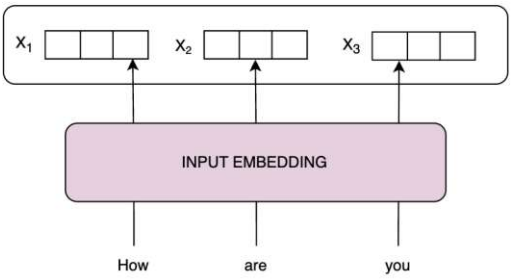

Figure 3.3: Encoder - Input Embedding

and attention is performed in parallel on each projected version, yielding h-dimensional output values.

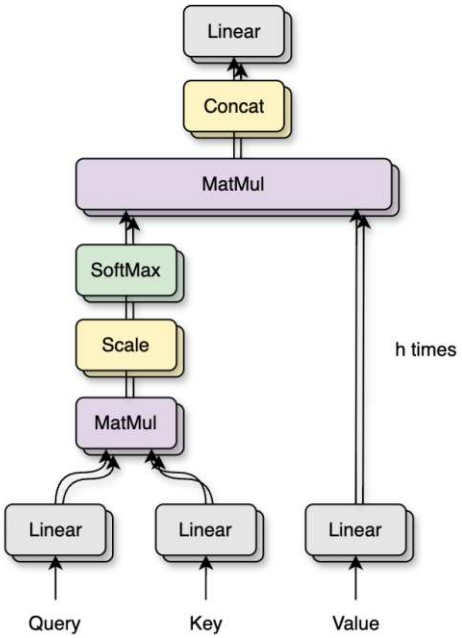

Figure 3.4: Multi-Headed Self-Attention Mechanism

Once the query, key, and value vectors are processed through a linear layer, a dot product matrix multiplication between the queries and keys creates a score matrix. This matrix indicates the emphasis each word should place on others. Higher scores mean greater focus. These scores are scaled down by dividing by the square root of the dimension of the query and key vectors to ensure stable gradients.

A softmax function is then applied to the adjusted scores to obtain attention weights,

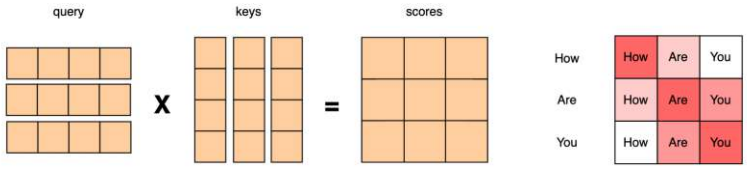

Figure 3.5: Attention mechanism - Matrix Multiplication

resulting in probability values between 0 and 1. The softmax function emphasizes higher scores while reducing lower ones, enhancing the model's ability to determine which words should receive more attention [83]. These weights are multiplied by the value vector, resulting in an output vector where only words with high softmax scores are preserved. This output vector is then fed into a linear layer for further processing, yielding the final output of the attention mechanism.

Each sub-layer in the encoder layer is followed by normalization and residual connections to mitigate the vanishing gradient problem, allowing for deeper models. This process is repeated after the feed-forward neural network as well. The normalized residual output then passes through a pointwise feed-forward network, consisting of two linear layers with a ReLU activation in between.

Once processed, the output loops back and merges with the input of the pointwise feed-forward network, followed by another round of normalization. This ensures everything is well-adjusted for subsequent steps. The final encoder layer output consists of vectors that represent the input sequence with rich contextual understanding, which are then used as input for the decoder in the Transformer model [83].

Decoder

The decoder is responsible for generating text sequences. Similar to the encoder, the decoder comprises sub-layers, including two multi-headed attention layers, a pointwise feed-forward layer, residual connections, and layer normalization after each sub-layer. These components function similarly to the encoder's layers but with a unique twist: each multi-headed attention layer in the decoder has a distinct role [19].

The final stage of the decoder's process involves a linear layer acting as a classifier, followed by a softmax function to calculate the probabilities of different words. The Transformer decoder is specifically designed to generate outputs by decoding the encoded information step by step.

The decoder operates in an auto-regressive manner, starting its process with a start token. It utilizes a list of previously generated outputs as its inputs, along with the encoder outputs that are rich with attention information from the initial input [83].

Figure 3.6 depicts the sequential decoding process, continuing until the decoder produces a token that marks the end of the output generation. The first two layers, comprising output

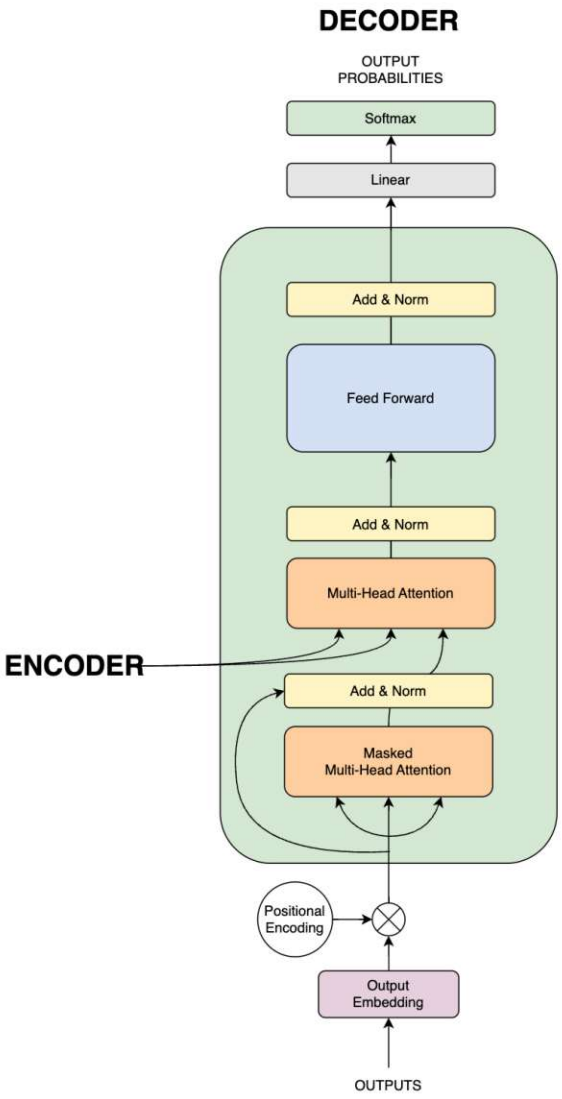

Figure 3.6: Decoder Architecture

embeddings and positional encodings, are similar to those in the encoder. Each layer in the decoder consists of three key components. The masked self-attention mechanism, while similar to the encoder's self-attention, has a critical distinction: it prevents positions from attending to future positions. This ensures that each word in the sequence is not influenced by tokens that come later. For example, when calculating attention scores for the word "are", it is essential that "are" does not access information from "you", the next word in the sequence. This masking enforces that predictions at any given position can only rely on outputs from preceding positions [19].

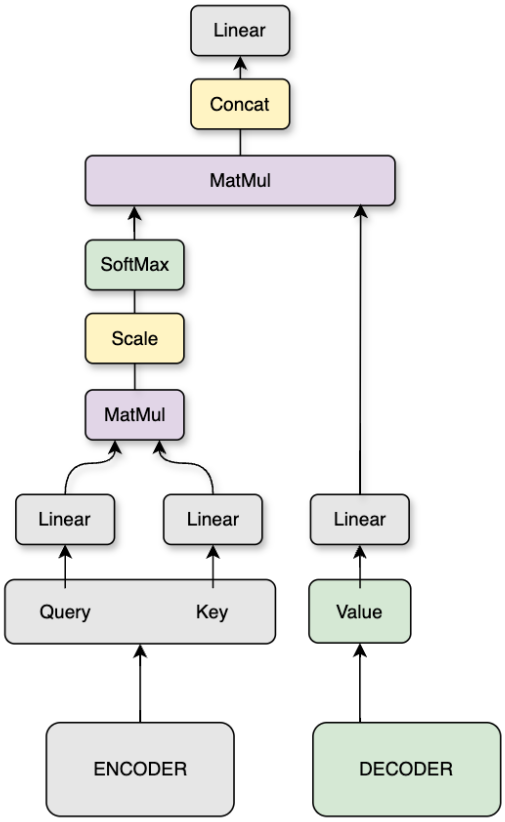

Figure 3.7: Encoder-Decoder Attention

In the second multi-headed attention layer of the decoder (Figure 3.7), we observe a unique interaction between the encoder and decoder components. Here, the outputs from the encoder serve as both queries and keys, while the outputs from the first multi-headed attention layer of the decoder serve as values. This setup aligns the encoder's input with the decoder's, enabling the decoder to identify and emphasize the most relevant parts of the encoder's input.

Like the encoder, each layer of the decoder contains a fully connected feed-forward network, which is applied to each position individually and in the same way across all positions [83].

In the transformer's data processing pipeline, the final stage involves passing through a linear layer that acts as a classifier. The size of this classifier matches the total number of classes, which corresponds to the number of words in the vocabulary. For example, if there are 100 unique classes representing 100 different words, the classifier will produce

an output array of 100 elements.

This output is then passed through a softmax layer, which converts the values into probability scores ranging between 0 and 1. The word with the highest probability is chosen as the model's predicted next word in the sequence. Additionally, a normalization step is applied after each sub-layer (such as masked self-attention, encoder-decoder attention, and feed-forward network) and incorporates a residual connection around it [83].

The output of the final layer is converted into a predicted sequence, usually through a linear layer followed by a softmax function to produce probability distributions over the vocabulary. As the decoder operates, the newly generated output is added to the list of inputs, allowing the decoding process to proceed. This cycle repeats until the model predicts a special token that marks the end of the sequence. The token with the highest probability is selected as the final class, commonly represented by an end token [19].

Figure 3.8 shows the complete architecture.

3.2 BART

This model pre-trains by combining bidirectional and auto-regressive transformers. BART functions as a denoising autoencoder using a sequence-to-sequence model, making it versatile for numerous end tasks. The pre-training process involves two stages: first, the text is corrupted using an arbitrary noising function, and second, the model learns to reconstruct the original text. It utilizes a standard Transformer-based neural machine translation architecture, which, despite its simplicity, generalizes BERT (with its bidirectional encoder), GPT (with its left-to-right decoder), and other recent pre-training methods [39].

A significant advantage of BART's design is its flexible noising capability, allowing for arbitrary transformations to the original text, including changes in length. Lewis et al. [39] evaluate various noising techniques and find that the best performance comes from randomly shuffling the order of the original sentences and employing a novel in-filling scheme. This in-filling method replaces arbitrary length spans of text with a single mask token, extending the original word masking and next sentence prediction objectives of BERT by requiring the model to reason about sentence length and make extensive transformations [39].

The model demonstrates high effectiveness when fine-tuned for text generation and performs well on comprehension tasks. It matches RoBERTa's [45] performance on GLUE [86] and SQuAD [68] with similar training resources and sets new state-of-the-art results in abstractive dialogue, question answering, and summarization tasks. Notably, it improves performance by 6 ROUGE points over previous work on the XSum dataset [57].

Figure 3.9 compares the differences of BERT [16] and GPT [8] to the BART model.

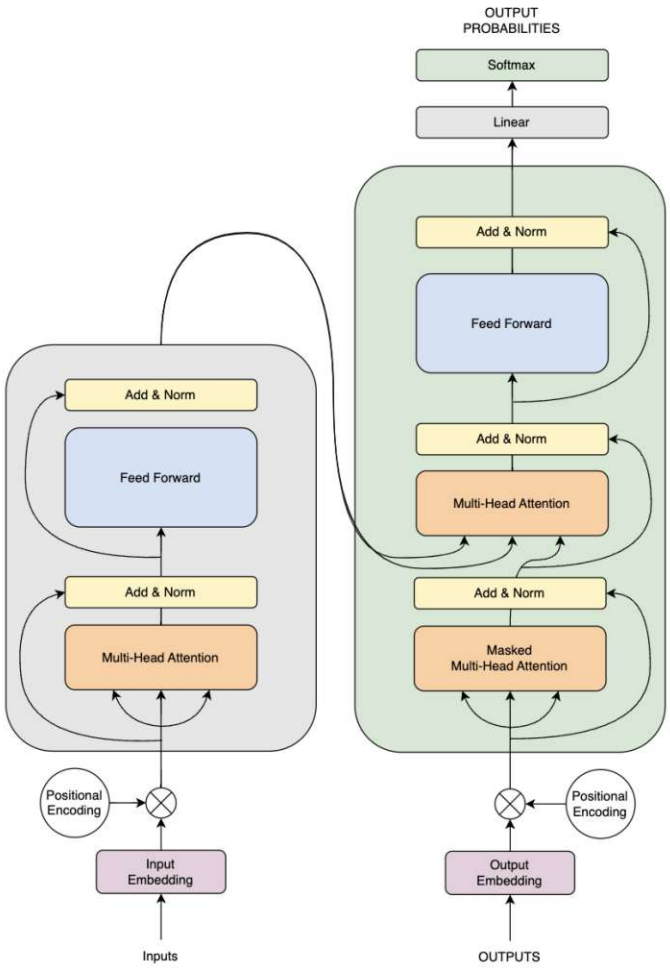

Figure 3.8: Transformer architecture

- **BERT:** Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, which means BERT cannot be easily used for generation [39].

- **GPT:** Tokens are predicted auto-regressively, enabling GPT to be used for generation. However, words can only condition on leftward context, preventing it from learning bidirectional interactions [39].

- **BART:** Inputs to the encoder do not need to be aligned with decoder outputs, allowing arbitrary noise transformations. In this case, a document is corrupted by replacing spans of text with mask symbols. The corrupted document is encoded with a bidirectional model, and the likelihood of the original document is calculated with an auto-regressive decoder. For fine-tuning, an uncorrupted document is input

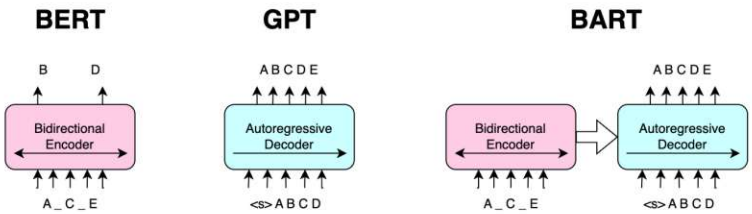

Figure 3.9: A semantic comparison of BART with BERT and GPT

to both the encoder and decoder, and representations from the final hidden state of the decoder are used [39].

3.2.1 Architecture

BART is a denoising autoencoder that reconstructs a corrupted document back to its original form. It is built as a sequence-to-sequence model featuring a bidirectional encoder for the corrupted text and a left-to-right auto-regressive decoder. During pre-training, we optimize the negative log-likelihood of the original document [39].

It employs the standard sequence-to-sequence Transformer architecture as described in [83], with the ReLU activation functions replaced by GeLUs (Gaussian Error Linear Units) [29], and parameters initialized from $N(0, 0.02)$. The activation function is $X\Phi(x)$, where $\Phi(x)$ is the cumulative distribution function of the standard Gaussian [29]. BART's base model includes 6 layers of encoders and decoders, while the large model has 12 layers. The architecture is similar to BERT [16], with two primary differences:

1. Each layer of BART's decoder incorporates cross-attention over the encoder's final hidden layer, akin to the transformer sequence-to-sequence model.

2. BERT includes an additional feed-forward network before word prediction, which BART omits. Overall, BART contains about 10% more parameters than a similarly sized BERT model.

Pre-training BART

BART is trained by corrupting documents and then optimizing a reconstruction loss, specifically the cross-entropy between the decoder's output and the original document. Unlike traditional denoising autoencoders that are designed for specific noising schemes, it can handle any type of document corruption. The transformations used in [39] are summarized in Figure 3.10

1. **Token Masking:** Similar to BERT, random tokens are sampled and replaced with $[MASK]$ tokens.

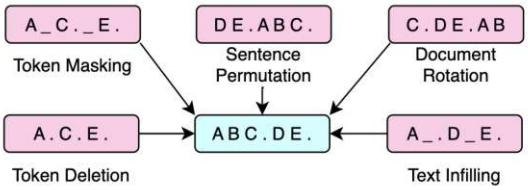

Figure 3.10: Transformations for noising the input documents

2. **Token Deletion:** Random tokens are removed from the input. Unlike token masking, the model must identify the positions of the missing inputs.
3. **Text Infilling:** Several spans of text are sampled, with span lengths drawn from a Poisson distribution ($\lambda = 3$). Each span is replaced by a single $[MASK]$ token. 0-length spans result in the insertion of $[MASK]$ tokens. Text infilling trains the model to predict the number of tokens missing from a span.
4. **Sentence Permutation:** The document is divided into sentences based on full stops, and these sentences are shuffled randomly.
5. **Document Rotation:** A token is chosen at random, and the document is rotated to start with that token. This task trains the model to recognize the beginning of the document.

3.2.2 Fine-tuning BART

The representations generated by BART can be employed for various downstream tasks.

Sequence Classification Tasks

For sequence classification, the input is passed through both the encoder and decoder, and the final hidden state of the last decoder token is input into a new multi-class linear classifier. This is similar to the CLS token in BERT, but an additional token is appended to enable the decoder's representation to attend to the complete input [39].

Token Classification Tasks

For token classification the entire document is processed through the encoder and decoder, with the top hidden state of the decoder used as the word representation for classification [39].

Sequence Generation Tasks

BART's auto-regressive decoder allows for direct fine-tuning for sequence generation tasks such as abstractive question answering and summarization. The encoder receives the input sequence, and the decoder generates outputs auto-regressively, leveraging the denoising pre-training objective [39].

Machine Translation

While previous research [17] demonstrated improvements by using pre-trained encoders, benefits from pre-trained decoders were minimal. [39] proposes utilizing the entire BART model, both its encoder and decoder as a unified pre-trained decoder for machine translation.

To achieve this, the original BART encoder embedding layer is replaced with a newly initialized encoder tailored to handle foreign language inputs. This new encoder can employ a different vocabulary from BART's original model. The system is trained end-to-end, enabling the new encoder to effectively transform foreign words into a format that BART can de-noise into fluent English.

1. **Initial Phase:** Most of BART's parameters are frozen. Only the newly initialized source encoder, BART's positional embeddings, and the self-attention input projection matrix of BART's first encoder layer are updated. The training is guided by backpropagating the cross-entropy loss from BART's output.
2. **Fine-Tuning Phase:** All model parameters are further trained for a small number of additional iterations to refine the translation quality.

3.3 GTE

GTE is a general text embedding model trained using multi-stage contrastive learning [1]. It was developed and is maintained by the Institute for Intelligent Computing at Alibaba Group [41]. The model is widely referenced in current research [41, 55, 96]. On the Hugging Face Massive Text Embedding Benchmark (MTEB) Leaderboard, the GTE_{large} model secured 19th place for English retrieval tasks and is downloaded over 2 million times monthly [2]. MTEB evaluates 309 text embedding models (for the English language) across 58 datasets and 8 different tasks in 112 languages. Despite requiring only 1.62 GB of memory, GTE_{large} supports sequences up to 8192 tokens, making it a resource-efficient option compared to higher-ranked models, while still offering the complexity needed for building a two-tower retriever tailored to our requirements.

The model training consists of unsupervised pre-training and supervised fine-tuning. Both stages employ the learning objective of contrastive learning.

3.3.1 Model Architecture

The model is a deep Transformer encoder [83], which can be initialized with a pre-trained language model such as BERT [16]. It adheres to a standard dual-encoder architecture, applying mean pooling on top of the contextualized token representations produced by the language model [41].

[1] `https://huggingface.co/Alibaba-NLP/gte-large-en-v1.5`, last seen: 10.15.2024
[2] `https://huggingface.co/spaces/mteb/leaderboard`, last seen: 10.15.2024

Formally, given a text sequence $x = (x_1, \ldots, x_n)$ consisting of n tokens, an embedding model E transforms the text into a low-dimensional dense vector $\mathbf{x} = E(x) \in \mathbb{R}^d$. To achieve this, we first use a language model to obtain deep contextualized token representations:

$$\mathbf{h} = LM(x) \in \mathbb{R}^{n \times d} \tag{3.1}$$

Subsequently, we apply mean pooling across the first dimension to derive the final text representation:

$$\mathbf{x} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{h}_i \in \mathbb{R}^d \tag{3.2}$$

These text representations are learned using a contrastive objective, which differentiates between semantically similar and dissimilar text pairs [41]. This training process requires positive and negative pairs in the format (q, d^+, d^-), where q is a query, d^+ is a relevant document, and $D_- = \{d^-, \ldots, d_n^-\}$ is a set of irrelevant documents. A commonly used contrastive objective is the InfoNCE loss [82]:

$$L_{cl} = -\log \frac{\exp(s(q, d^+)/\tau)}{\exp(s(q, d^+)/\tau) + \sum_{i=1}^{n} \exp(s(q, d_i^-)/\tau)} \tag{3.3}$$

In this context, $s(q, d)$ measures the similarity between two text segments q and d by calculating the vector distance between their embeddings, $\mathbf{q} = E(q)$ and $\mathbf{d} = E(d)$. To obtain high-quality text embeddings that can generalize across a variety of tasks, it is essential to use a large and diverse dataset of text pairs from different formats and domains [41]. This dataset is then used to train the model through an enhanced contrastive loss method in multiple stages. The objective is to minimize the distance between similar pairs in the embedding space while maximizing the distance between dissimilar pairs, improving the overall quality of the learned representations.

3.3.2 Pre-training Data

Building upon prior research [60, 59, 87], this model is pre-trained using naturally occurring text pairs extracted from diverse sources. To enhance the generalizability of the embedding model, Zhang et al. [41] utilize a wide range of text pair sources, including web pages (e.g., CommonCrawl, ClueWeb), scientific articles (e.g., arXiv, SemanticScholar), community QA forums (e.g., StackExchange), social media platforms (e.g., Reddit), knowledge repositories (e.g., Wikipedia, DBPedia), and code repositories (e.g., StackOverflow, GitHub). They also take advantage of hyperlinks present in certain datasets for extracting text pairs. In total, approximately 800M text pairs were used for the unsupervised pre-training phase [41].

3.3.3 Fine-tuning Data

During the supervised fine-tuning phase, Zhang et al. [41] employ smaller datasets with human-annotated relevance between two text segments, along with optional hard

negatives generated by an additional retriever to create text triples [41]. To address both symmetric tasks (e.g., semantic textual similarity) and asymmetric tasks (e.g., passage retrieval), they compile data from a wide range of tasks and domains, such as web search (e.g., MS MARCO), open-domain QA (e.g., NQ), natural language inference (e.g., SNLI), fact verification (e.g., FEVER), and paraphrasing (e.g., Quora). In total, they use approximately 3 million text pairs for fine-tuning, drawing from training datasets used in prior work [22, 21, 4].

3.3.4 Training

In the initial phase of unsupervised pre-training, data sources can vary significantly in terms of the number of available training instances. To mitigate this imbalance, a multinomial distribution is employed to sample data batches from different sources, considering the size of each source. Let the complete pre-training dataset D consist of m distinct subsets $D_1, ..., D_m$, with the size of each subset denoted as $N_i = |D_i|$. During each training iteration, the probability of sampling data from the i-th subset D_i is given by:

$$p_i = \frac{n_i^\alpha}{\sum_{j=1}^{m} n_j^\alpha} \tag{3.4}$$

where they set $\alpha = 0.5$ in this work [41]. Additionally, to prevent the model from learning task-specific shortcuts for distinguishing data, all instances within a batch are ensured to come from the same task.

Instead of reusing the contrastive loss objectives outlined in [32], this model adopts an enhanced contrastive learning objective that is bidirectional and extends the pool of negative samples by incorporating both in-batch queries and documents. This approach can be seen as a combination of the loss variants proposed in [66, 72].

Given a batch of positive text pair samples $B = \{(q_1, d_1), (q_2, d_2), \ldots, (q_n, d_n)\}$, the model utilizes an improved contrastive loss in the following form:

$$L_{icl} = -\frac{1}{n} \sum_{i=1}^{n} \log \frac{\exp(s(q_i, d_i)/\tau)}{Z} \tag{3.5}$$

where the partition function is defined as:

$$Z = \sum_{j} \exp(s(q_i, d_j)/\tau) + \sum_{j \neq i} \exp(s(q_i, d_j)/\tau) + \sum_{j} \exp(s(q_j, d_i)/\tau) + \sum_{j \neq i} \exp(s(q_j, d_i)/\tau) \tag{3.6}$$

Here, the first two terms handle the query-to-document contrast, while the last two terms address the reverse (document-to-query) contrast. The model uses cosine similarity as the distance metric:

$$s(q, d) = \frac{\mathbf{q} \cdot \mathbf{d}}{\| \mathbf{q} \|_2 \cdot \| \mathbf{d} \|_2} \tag{3.7}$$

The temperature parameter τ is set to 0.01 [41].

3.3.5 Evaluation

For the unsupervised text retrieval task, the model is evaluated using the BEIR benchmark [77] for zero-shot retrieval. BEIR is a diverse information retrieval benchmark comprising retrieval tasks from various formats and domains. When evaluated on 15 datasets, the base-sized model (built on BERT_{base} with 110M parameters) significantly outperforms other models like SimCE, Contriever, and E5 on the recall@100 metric. The base model's performance is comparable to E5_{large} (330M parameters). For the MTEB benchmark [55], the model is tested across 56 English datasets, covering seven different tasks, including text classification, text retrieval, and text reranking. In the unsupervised evaluation setting, the model outperforms E5 by a considerable margin. Moreover, in the supervised setting, the model surpasses $\text{OpenAI}_{ada-002}$ [41].

CHAPTER 4

Experiment Setup

We provide a detailed description of the datasets employed in our experiments and the preprocessing steps taken to prepare the data for model training. We will also discuss the training environment, including the hardware and software configurations used.

4.1 Datasets

The dataset utilized in this research is derived from the *Text Retrieval Conference (TREC)* for product search [9] and is based on the *ESCI* dataset [70]. This dataset is a comprehensive collection of challenging Amazon search queries and their corresponding results, which has been publicly released to encourage research aimed at enhancing the quality of search results. The dataset is structured around three key tasks essential for improving the user experience in product search:

- **Query-Product Ranking**: Given a user-defined query and the top 40 products retrieved by a commercial search engine, this task focuses on ranking these products such that the more relevant products are positioned higher than the less relevant ones.
- **Multi-class Product Classification**: This task involves classifying a set of products returned for a specific query into one of the following categories: exact match, substitute, complement, or irrelevant.
- **Product Substitute Identification**: Here, the objective is to identify potential substitutes for a given product from a list of candidates.

Table 4.1 provides an overview of the various datasets considered in this thesis.

Type	Num Records	Format
Collection	1,661,907	tsv: docid, Title, Description
Query to Query ID	30,734	tsv: qid, query
Test Queries	186	tsv: qid, query
Train QREL (ESCI)	392,119	tsv: qid, 0, docid, relevance label
Dev QREL (ESCI)	169,952	tsv: qid, 0, docid, relevance label
Test QREL (NIST)	115,490	tsv: qid, 0, docid, relevance label
Training Triples (Query, Positive, Negative Pairs)	20,888	json: qid, query, positive passages, negative passages

Table 4.1: TREC Datasets

1. *Collection:* Comprises raw product documents, each annotated with a unique ID, title, and descriptive metadata.

2. *Query to Query ID / Test Queries:* Assigns a unique identifier to textual search queries, enabling systematic referencing in evaluation datasets.

3. *QREL Datasets:* Provides graded relevance judgments (0–3) for query-document pairs, indicating their semantic alignment.

4. *Training Triples:* Encodes labeled query-product pairs (relevant vs. irrelevant) in JSON format for supervised model training.

Our work focuses on information retrieval and synthetic data generation, and since product descriptions are crucial for those tasks, we chose to explore the *Collection* more in detail.

4.2 Data Exploration

The title length distribution (Figure 4.1) reveals a prominent concentration of titles within the 0–250 character range, underscoring a preference for brevity and conciseness in product naming conventions. Kernel density estimation (KDE) highlights a distinct mode at 200 characters, though this global maximum corresponds to a narrow interval (196–200 characters) encompassing over 46,000 products. This granularity discrepancy between the histogram binning and KDE smoothing suggests the need for careful interpretation of distributional trends. While outliers with titles exceeding 250 characters exist, their scarcity reinforces the dominance of succinct titles.

In contrast, description lengths exhibit a heavy-tailed distribution, with most entries clustered below 2,000 characters but extending to extreme values (50,000 characters). This asymmetry reflects diverse descriptive practices: while many products use concise summaries, a non-trivial subset employs exhaustive detail, likely for technical or feature-rich items. Such variability poses scalability challenges for computational models, as excessively long descriptions may necessitate resource-intensive processing and memory allocation.

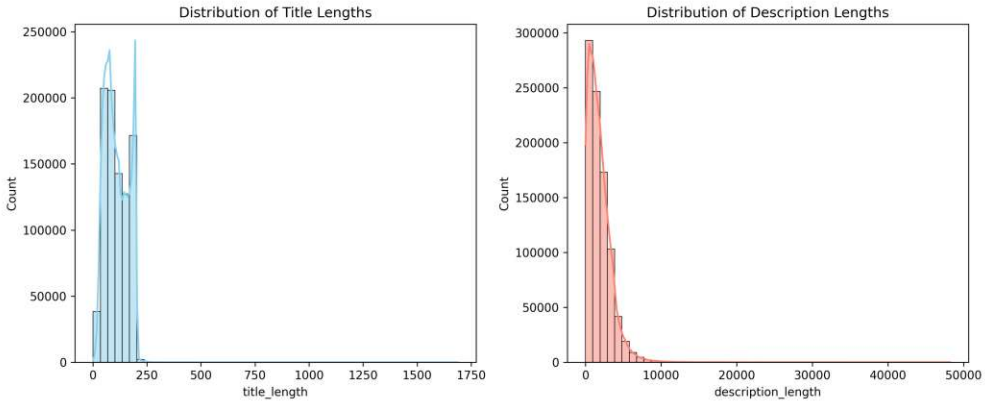

Figure 4.1: Distribution of Text Lengths

Notably, some titles and descriptions are repeated multiple times across different products. To analyze these repetitions, we identify the top 10 most frequent phrases in both titles and descriptions, as presented in Tables 4.2 and 4.3, respectively.

The most common title is *"Hanes Men's Sweatshirt, EcoSmart Fleece Hoodie, Cotton-Blend Fleece Hooded Sweatshirt, Plush Fleece Pullover Hoodie"*, appearing 83 times in the dataset. This high frequency suggests that certain popular products are listed multiple times, possibly due to variations in size, color, or other attributes not captured in the dataset.

We observe that the most common description is *"From the manufacturer"*, occurring 9,510 times. This generic phrase likely serves as a placeholder or introduction to the manufacturer's details, but without substantive product-specific information.

Other frequently repeated descriptions include:

- *"Product Description"* with 6,607 occurrences.
- Variations like *"Product Description Read more"*, *"Product Description Read more Read more"*, and so on, suggesting truncated or placeholder content.
- A lengthy, templated description appearing 1,622 times, which seems to be a standard marketing blurb reused across multiple products.

The prevalence of such generic or repetitive descriptions implies that many product entries lack unique descriptive content, which could impact the quality of any textual analysis performed on the dataset.

4.2.1 Text Preprocessing

There are 38,556 missing values in the product title and 222,913 in the product description fields. All records with both fields undefined need to be removed.

Title	Count
Hanes Men's Sweatshirt, EcoSmart Fleece Hoodie, Cotton-Blend Fleece Hooded Sweatshirt, Plush Fleece Pullover Hoodie	83
Gloria Vanderbilt Women's Amanda Classic High Rise Tapered Jean	82
Crocs Unisex-Adult Classic Clogs (Best Sellers)	66
Women's Totally Shaping Pull-On Skinny Jeans (Standard and Plus)	65
Hanes Men's Sweatshirt, EcoSmart Fleece Crewneck Sweatshirt, Cotton-Blend Fleece Sweatshirt, Plush Fleece Pullover Sweatshirt	63
Levi's Men's 511 Slim Fit Jeans (Regular and Big & Tall)	62
adidas Women's Adilette Comfort Sandals Slide	52
Crocs Unisex-Child Classic Clogs	51
Amazon Essentials Women's Classic-Fit Short-Sleeve Crewneck T-Shirt, Pack of 2	49
Timberland Men's White Ledge Mid Waterproof Ankle Boot	46

Table 4.2: Top 10 Titles

Description	Count
From the manufacturer	9510
Product Description	6607
Product Description Read more	3081
Product Description Read more Read more	2366
Product Description Read more Read more Read more	1800
From the manufacturer Read more	1726
From the manufacturer Previous page Outfit Your Superfan Disney designs Make everyday a Disney day with designs for the whole family - exclusive to Amazon Outfit Your Occasions Women's looks Dresses and more, in fresh styles to match all your moments. Outfit Your Vibe New styles for him Easygoing or buttoned up, find the tops and bottoms you need for all that you do. Outfit Your Little One The baby scene Keep them in cuteness. Shop new styles, available in adorable. Next page 1 DISNEY X AE 2 WOMEN 3 MEN 4 KIDS/BABY	1622
Product Description This pre-owned or refurbished product has been professionally inspected and tested to work and look like new. How a product becomes part of Amazon Renewed, your destination for pre-owned, refurbished products: A customer buys a new product and returns it or trades it in for a newer or different model. That product is inspected and tested to work and look like new by Amazon-qualified suppliers. Then, the product is sold as an Amazon Renewed product on Amazon. If not satisfied with the purchase, renewed products are eligible for replacement or refund under the Amazon Renewed Guarantee. Product Description This pre-owned or refurbished product has been professionally inspected and tested to work and look like new. How a product becomes part of Amazon Renewed, your destination for pre-owned, refurbished products: A customer buys a new product and returns it or trades it in for a newer or different model. That product is inspected and tested to work and look like new by Amazon-qualified suppliers. Then, the product is sold as an Amazon Renewed product on Amazon. If not satisfied with the purchase, renewed products are eligible for replacement or refund under the Amazon Renewed Guarantee.	1130
From the manufacturer Previous page Next page	996
From the manufacturer Read more Read more	980

Table 4.3: Top 10 Descriptions

Our analysis further identified repetitive phrases in product descriptions (Table 4.3), which we deemed irrelevant for retrieval tasks. To enhance data quality, we curated an exclusion list of non-informative phrases and removed all matching products from the dataset (exclusion criteria available in the GitHub repository[1]). We then merged product titles and descriptions into a unified *product_text* column through concatenation. Following this preprocessing pipeline, the refined Collection dataset comprises 980,974 products, stored in TSV format as *p_collection.tsv*.

We merge the *Training Triples* and *Train QREL* datasets to generate a unified set of high-confidence positive pairs, stored as *query_product.tsv*. To ensure training quality, we exclude pairs with relevance scores ≤ 1, retaining only those with the highest relevance scores (2 and 3). This filtering strategy minimizes noise from marginally relevant examples, enabling the BART model to focus on robust query-product relationships during fine-tuning. The final curated dataset contains 226,438 high-relevance pairs.

[1] https://github.com/lukasthekid/enhanced-product-search-llm

Baseline Preprocessing Stage

As introduced in several studies [51, 42, 27, 41], BM25 serves as a good baseline model. But before feeding the model we build a text-preprocessing pipeline involving several key steps to clean and standardize the text data, which is crucial for improving the performance of NLP models [81].

Firstly, we perform stop word removal. Stop words, such as "and", "the", and "is", are common words that carry little meaningful information for many NLP tasks [81]. Removing these words helps reduce noise and improves the efficiency of the model. We use the *nltk* [6] module in Python to remove english stopwords.

Next, we apply lemmatization, a process that reduces words to their base or root form (lemma). For example, the word "running" is reduced to "run". Lemmatization helps in normalizing the text, ensuring that different forms of a word are treated as a single entity. We use the *WordNetLemmatizer* from the *nltk* module.

Tokenization is another essential step in our preprocessing pipeline. It involves breaking down the text into individual words or tokens. For instance, the phrase "Hello world!" is tokenized into ["Hello", "world", "!"]. Tokenization helps in analyzing the text at a granular level, making it easier to process [81].

We also convert all tokens to lowercase. This step ensures uniformity in the text data, preventing the model from treating words like "Hello" and "hello" as different entities. Lowercasing is a common practice in NLP to maintain consistency [81].

Finally, we remove punctuation marks such as commas, periods, and exclamation marks from the token list. Punctuation often does not contribute to the meaning of the text in a way that is useful for many NLP tasks, and its removal helps in reducing the dimensionality of the text data [81].

4.3 Training Environment

For smaller tasks, such as exploring datasets, text preprocessing, we utilize a MacBook Pro 2021 equipped with 32 GB RAM and an M1 Max Chip. We also build and evaluate our baseline model on the MacBook, since it does not require a GPU. For computationally intensive tasks, such as training large language models (LLMs), we leverage the TU.it dataLab[2] GPU Cluster. This setup includes access to a NVIDIA A100 GPU with 80GB of GPU memory and 64 CPU cores. Additionally, we use a NVIDIA-Tesla V100 GPU with 32GB of memory for running Jupyter Notebooks on the cluster. The maximum job time is 7 days.

[2] https://www.it.tuwien.ac.at/services/netzwerk-und-server/datalab, last seen: 10.18.2024

4.4 Evaluation Metrics

Here we will discuss the metrics that we use during development, tuning and evaluation.

4.4.1 Discounted Cumulative Gain (DCG)

The inclusion of position-based relevance considerations in this metric underscores its importance in our context. It assigns greater significance to relevant items positioned higher in the list. Normalized Discounted Cumulative Gain (NDCG) is especially pertinent to our case because it accommodates scenarios where relevance is a dynamic measure [89].

NDCG prioritizes highly relevant items appearing at the forefront of the list, aligning well with the primary objective of a recommender system to present the most relevant items first. Its "normalized" nature entails scaling the scores between 0 and 1, facilitating comparisons of performance across various queries or systems [89].

Furthermore, when evaluating synthetic queries, we can assess the product rankings achieved using these queries. The objective of these queries is to retrieve the relevant product from which the query is derived, making this metric applicable to this aspect of the work as well.

The DCG at a particular rank 'p' is calculated as:

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)} \tag{4.1}$$

where rel_i is the relevance of the item at rank i. The denominator is a discount factor that gives less weight to items at lower ranks.

Ideal Discounted Cumulative Gain (IDCG): This is the maximum possible DCG through position p, if all relevant items were ranked first. It's calculated in the same way as DCG, but with the items sorted in descending order of relevance [89].

The NDCG at a particular rank p is then calculated as:

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{4.2}$$

This gives a value between 0 and 1, where 1 means the items are perfectly ranked according to relevance [89].

4.4.2 Precision@k

Precision@k [79], also denoted as $P@k$, is a common evaluation metric in information retrieval that measures the accuracy of the top k retrieved documents. It is defined as

the proportion of relevant documents among the top k retrieved documents. Formally, it can be expressed as:

$$P@k = \frac{|\text{Relevant Documents} \cap \text{Retrieved Documents@}k|}{k} \tag{4.3}$$

where:

- Relevant Documents is the set of documents that are relevant to the query.
- Retrieved Documents@k is the set of top k documents retrieved by the system.
- $|\cdot|$ denotes the cardinality of a set.

The metric $P@k$ ranges from 0 to 1, where 1 indicates perfect precision (i.e., all k retrieved documents are relevant) and 0 indicates no relevant documents among the top k retrieved.

4.4.3 Cosine Similarity

Cosine similarity is utilized to measure the similarity between two non-zero vectors [67]. In the context of word embeddings, it assesses the similarity between two pieces of text, such as a product description and a generated query.

Cosine similarity quantifies the cosine of the angle between two vectors in an n-dimensional space. For two vectors $\mathbf{A}$ and $\mathbf{B}$, the cosine similarity is defined as:

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{4.4}$$

where $\mathbf{A} \cdot \mathbf{B}$ represents the dot product of vectors $\mathbf{A}$ and $\mathbf{B}$. $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ denote the magnitudes (or Euclidean norms) of vectors $\mathbf{A}$ and $\mathbf{B}$. θ is the angle between the two vectors.

The value of cosine similarity ranges from -1 to 1. A value of 1 indicates that the two vectors are identical, 0 indicates that the vectors are orthogonal (i.e., no similarity), and -1 indicates that the vectors are diametrically opposed [67].

Calculation Steps:

1. Convert the product description and generated query into their respective word embeddings. This involves using a pre-trained model, such as Sentence-BERT [71], to transform the text into numerical vectors.
2. Calculate the dot product of the two vectors.
3. Compute the magnitudes (Euclidean norms) of both vectors.

4. Divide the dot product by the product of the magnitudes to obtain the cosine similarity score.

4.4.4 ROUGE Score

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [43] comprises a collection of metrics designed to evaluate tasks involving automatic text generation, especially summarization. It compares a machine-generated summary or text to one or more reference summaries or texts produced by humans. In our case, we only use this metric for tuning our BART model, thus this score is not as relevant as the others. ROUGE scores quantify the overlap between the generated text and reference text based on n-grams, sequences of words, and pairs of words. Specifically, ROUGE-1 assesses the overlap of unigrams, while ROUGE-2 evaluates the overlap of bigrams. ROUGE-L measures the longest common subsequence between the generated text and the reference text. By computing ROUGE scores for the generated queries and comparing them to the reference queries, one can quantitatively measure the model's performance. This evaluation aids in determining how effectively the generated queries encapsulate the essence and intent of the product descriptions [43]. We calculate this score between ground truth queries from the $QREL_{DEV}$ (4.1) dataset and machine-generated queries.

4.5 Implementing the Baseline Model

Our implementation is based on the *ATIRE BM25* described in [80] and is available on GitHub[3].

When the BM25 class is initialized with a corpus of documents, several attributes are established to store important information about the corpus. These attributes include the size of the corpus (*corpus_size*), the average document length (*avgdl*), document frequencies (*doc_freqs*), inverse document frequencies (*idf*), and document lengths (*doc_len*). If a tokenizer is provided, the corpus is tokenized using the *_tokenize_corpus* method, which utilizes parallel processing to expedite the tokenization process.

The *_initialize* method processes the corpus to compute essential statistics. It iterates over each document in the corpus, counting the total number of documents (*corpus_size*) and the length of each document. These lengths are stored in *doc_len*, and the average document length is calculated (*avgdl*). For each document, the method calculates the frequency of each term and stores these frequencies in *doc_freqs*. Additionally, the method tracks how many documents each term appears in (*nd*), which is crucial for calculating the inverse document frequency (IDF).

The *BM25Okapi* class, which extends the BM25 class, introduces specific parameters for the BM25 formula, such as *k1*, *b*, and ϵ. It overrides the *_calc_idf* method to calculate the IDF for each term. The IDF is computed using the formula:

[3] https://github.com/lukasthekid/enhanced-product-search-llm, last seen: 10.18.2024

$$\text{idf}(t) = \log\left(\frac{N - n_t + 0.5}{n_t + 0.5}\right) \tag{4.5}$$

where N is the total number of documents and n_t is the number of documents containing the term t. This calculation adjusts the weight of terms based on their distribution across the corpus.

If any IDF values are negative (which can occur for very common terms), they are adjusted to a small positive value using ϵ to prevent skewing the scoring process.

To score documents against a query, the *get_scores* method is utilized. This method initializes a score array for all documents. For each query term, it calculates a score for each document using the BM25 formula:

$$\text{score}(q, d) = \sum_{t \in q} \text{idf}(t) \cdot \frac{f(t, d) \cdot (k1 + 1)}{f(t, d) + k1 \cdot (1 - b + b \cdot \frac{|d|}{\text{avgdl}})} \tag{4.6}$$

where $f(t, d)$ is the term frequency of term t in document d, $|d|$ is the document length, avgdl is the average document length, and $k1$ and b are tuning parameters.

Scores for each query term are summed to obtain a final score for each document.

The *get_top_n* method receives a search query, a set of documents contained in a pandas *DataFrame*, and a parameter k that specifies the number of recommendations to be returned. Within this method, the score is computed for each query-product pair, appended to the *DataFrame*, and then sorted in descending order, ensuring that the products with the highest scores are listed first.

4.5.1 Hyperparameter Tuning

To perform hyperparameter tuning, we utilize the *ParameterGrid* from *scikit-learn* [64]. This tool allows us to define a grid of parameters and generates all possible combinations of parameter values, making it particularly useful for grid search. By exhaustively searching through a specified parameter space, we aim to find the optimal set of parameters for our model. As previously mentioned, the hyperparameters we use for our BM25 retriever are $k1$, b, and ϵ. We employ the following grid:

$$k1 : [1.2, 1.5, 1.8] \quad b : [0.6, 0.75, 0.9] \quad \epsilon : [0.1, 0.25, 0.5]$$

Since this retriever does not need any training, we only use the validation dataset ($QREL_{DEV}$). Remember that this set matches one query to multiple products with a relevance score between 0 and 3. For each parameter combination, we randomly sample 10 queries from that dataset and evaluate. For each query we retrieve the ground truth ranking from the dataset (e.g. one query matches to 10 products). We then take the

same set of possible products and calculate the BM25 scores on that given query and compare the ranking. In each run, we evaluate 10 retrievals. If the NDCG score achieved is higher than the previous best, we save the parameter combination as the new best outcome. The best NDCG score achieved during tuning is 0.7686, with the parameters $b = 0.9, \epsilon = 0.1$ and $k1 = 1.2$. The best *Precision@10* achieved is 0.78.

CHAPTER 5

Synthetic Query Generation

To generate synthetic data we fine-tune BART [39], as this has already been used in similar tasks [51, 42]. BART is particularly effective when fine-tuned for text generation due to its encoder-decoder (seq2seq) architecture [39].

We load the pre-trained $BART_{large}$ model from Huggingface[1]. It uses 12 layers in each encoder and decoder and a hidden size of 1024. For fine-tuning we use *PyTorch 2.4.0+cu121* [63]. While the mechanisms behind the BART model are highlighted in Chapter 3, here we will focus on the implementation that are also on GitHub. This model is developed on the TU.it dataLab Slurm Cluster by using batch jobs.

5.1 Fine-tuning BART

Fine-tuning refers to adapting a pre-trained model to a specific task using task-relevant data. For this we use our previously generated positive pairs dataset (examples in Table 5.1). We first transform our relevant query product pairs into a Huggingface Dataset, tokenize the dataset and split it into training and test sets using a 95/5 split. We allocate 95% of the dataset for training and validation across multiple epochs, while the remaining 5% is reserved as data for hyperparameter tuning and testing (50:50). 10% of the training data is set aside for validation purposes. So in total, we train on 190,591 positive pairs, use 21,177 pairs for epoch evaluation, 5,661 pairs for hyperparameter tuning and 5,484 pairs for testing. The tokenization process includes truncation and padding to ensure uniform input sizes, which is critical for training transformers efficiently. Training arguments, such as the number of epochs, learning rate, and batch size, are defined using *Seq2SeqTrainingArguments*. Training is carried out using *Seq2SeqTrainer*, which is configured with the model, training arguments, datasets, tokenizer, and a custom callback for resource tracking. The training process is timed and logged, with the custom

[1] `https://huggingface.co/facebook/bart-large`, last seen: 10.07.2024

callback function logging CPU and memory usage at regular intervals, as well as elapsed time and training metrics. This information is saved to a log file at the end of training. The sequence-to-sequence training is tailored for text generation tasks like summarization or translation. Our task of search query generation is similar to summarization, as it involves capturing the most important product attributes in a query.

query	product_text	relevance
bmouo kids case for new ipad 10.2	LTROP Kids Case with Adorable Robot Back Design - Fits to New iPad 9th/8th/7th Generation 10.2-inch(2021/2020/2019) iPad 10.2** Case, Kids Case for iPad 10.2-inch 2021(9th Generation)/2020 (8th Generation) /2019 (7th Generation), Shock-Proof Handle Stand Shoulder Strap iPad 10.2 Case for Kids Compatible Model: iPad 9th Generation 2021 10.2-inch (Model: A2602 A2603 A2604 A2605) iPad 8th Generation 2020 10.2-inch (Model : A2428, A2429, A2430, A2270) iPad 7th Generation 2019 10.2-inch (Model : A2197, A2200, A2198).	3
duffle bag xlarge size	Duffel Bag, Length 50 In, Width 30 In, Depth 30 In, Diameter 30 In, Material Canvas, Color Black Duffel Bag, Length 50 In, Width 30 In, Depth 30 In, Diameter 30 In, Material Canvas, Color Black From the manufacturer Zippered Canvas Duffle Bag Available in Black or OD Outdoor Everything T exsport - Authentic Adventure Gear- has been proud to provide outdoor enthusiasts with family camping equipment for over 70 years!	2
crew socks unicorn	From the brand Merry Christmas Previous page Funny Slouch Socks For Women Visit the Store Funny Food Socks Visit the Store Gifts for Her and Him Visit the Store Funny Crew Socks Visit the Store Funny Kids Socks Visit the Store SOCKFUN socks combine a unique design style with high quality, premium materials ensure our novelty socks are durable, comfortable, and breathable. Let our exquisitely designed socks fill your everyday life with fun, laughter and love.	3

Table 5.1: Positive Query Product Pairs

For tokenization, the product descriptions are tokenized with a *max_input_length* of 1024, while the target search queries are tokenized with a *max_output_length* of 64. The fine-tuning process involves updating the model's parameters to minimize the loss between the predicted and actual target sequences.

1. **Forward Pass**: The Encoder processes the tokenized product descriptions and passes them to the Decoder, which generates the predicted search queries token by token. During training, the Decoder uses the previous actual token as input for the next prediction, which facilitates faster convergence.

2. **Loss Calculation**: Cross-entropy loss [39] is used to measure the difference between the predicted token probabilities and the actual tokens. Padded tokens are ignored during loss computation (masked tokens) to avoid their impact. The loss function is expressed as:

$$Loss = -\sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i) \tag{5.1}$$

   where y_i is the actual token (one-hot-encoded) and $\hat{y}_i$ is the predicted probability for that token.

3. **Backpropagation**: Gradients of the loss with respect to each model parameter are computed, and the parameters are updated using the AdamW [46] optimizer along with the calculated gradients. The parameters are updated according to the following equation:

$$\theta_{new} = \theta_{old} - \eta \cdot \nabla_\theta \mathcal{L} \tag{5.2}$$

where θ denotes the model parameters, η is the learning rate, and $\nabla_\theta \mathcal{L}$ represents the gradient of the loss with respect to the parameters. AdamW combines Adam with weight decay, which applies a penalty proportional to the magnitude of the weights to the loss function, reducing the risk of overfitting [46]. A learning rate scheduler is also employed to start with a higher learning rate and decrease it gradually for more stable training.

The model is trained for 10 epochs, where each epoch represents a full pass through the training dataset. After each epoch, the model is evaluated on the validation set. The dataset is divided into batches for efficient computation, using a batch size of 32, which balances memory usage and convergence speed. Evaluation is also performed after every epoch to monitor model performance on unseen data, with the model generating search queries for the product descriptions in the validation set. Checkpointing is implemented, saving the model weights at the end of each epoch, and the best weights are loaded based on the lowest validation loss. The model is trained on an A100 GPU, with an initial learning rate of $2 \cdot 10^{-5}$. The entire fine-tuning process took 31 hours to complete. Figure 5.1 illustrates the performance metrics throughout the training process.

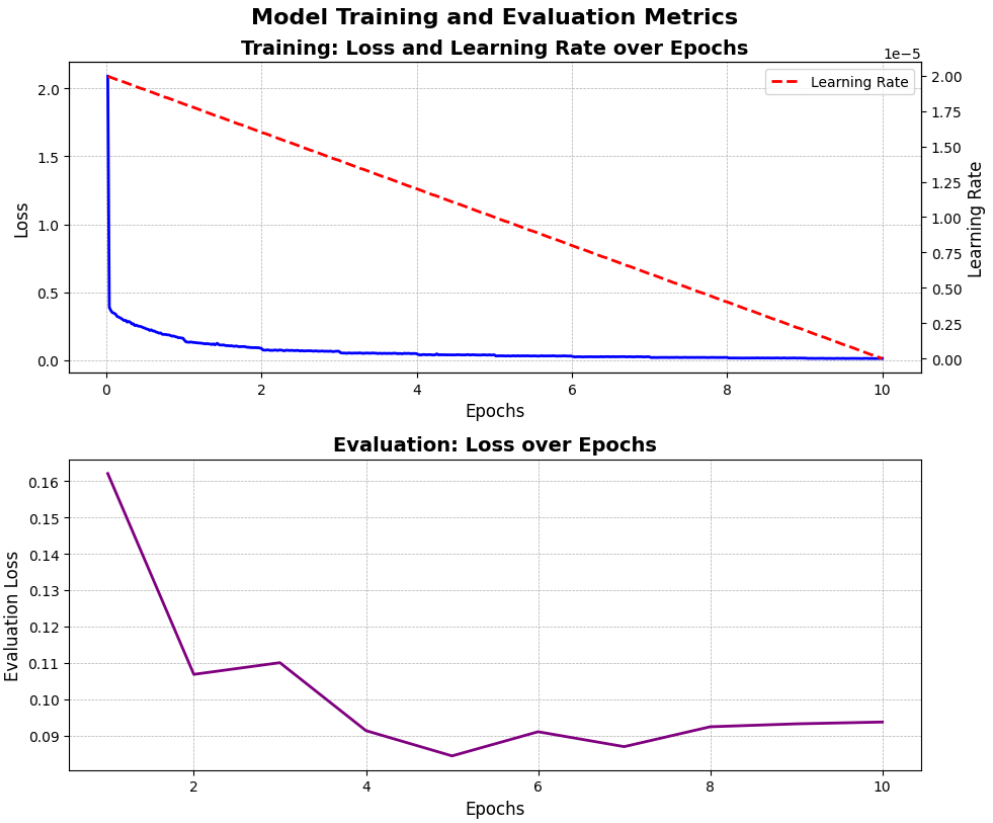

Figure 5.1: BART Training and Evaluation Metrics

5.1.1 Hyperparameter Tuning

To understand the parameters that can be chosen during inference. We present a listing of the most important ones:

1. **num_beams:** Specifies the number of beams used during the beam search. A higher number of beams typically results in more accurate and diverse queries as the model considers more potential sequences. However, it also increases computational complexity and time. For query generation, using a higher value helps ensure that the generated queries are of high quality and relevant to the product description.

2. **no_repeat_ngram_size:** Ensures that no n-grams of the specified size are repeated in the generated sequence. For example, if set to 2, the model will avoid repeating bigrams (two-word sequences). This is particularly useful in query generation to avoid redundancy and make the queries more diverse and natural-sounding.

3. **top_k:** Sets the number of highest-probability candidates to consider at each step in the generation process. For instance, if top_k is set to 50, only the top 50 token candidates are considered for the next token in the sequence. This helps generate more focused and relevant queries by avoiding low-probability tokens that could make the queries nonsensical.

4. **top_p:** Defines the cumulative probability threshold. For example, if top_p is set to 0.95, the model considers only the smallest set of tokens whose combined probability is 95%. This approach allows for more dynamic and context-sensitive query generation, balancing diversity and relevance.

5. **temperature:** Lower values (e.g., 0.7) make the model more confident and conservative by sharpening the probability distribution, often leading to more predictable and repetitive queries. Higher values (e.g., 1.0 or above) make the model more random and creative by flattening the distribution, encouraging more diverse outputs. For query generation, an optimal temperature ensures a balance between creativity and relevance.

6. **length_penalty:** Influences the preference for longer or shorter sequences. A value greater than 1.0 penalizes shorter sequences and encourages longer ones, while a value less than 1.0 does the opposite. In query generation, adjusting the length penalty helps generate queries of appropriate length, ensuring they are neither too brief nor unnecessarily verbose.

7. **num_return_sequences:** Sets the number of unique query sequences to generate per input description. Generating multiple sequences can provide a variety of queries for the same product description, which is useful for applications like search engine optimization or improving the diversity of search results.

In our approach, we choose three parameters—**num_beams**, **top_p**, and **temperature**—which yield a total of 36 possible combinations. Evaluating all these combinations on our A100 GPU takes 29 hours and 38 minutes, even though we run three evaluations in parallel. Due to this significant computational cost, we do not explore further parameter combinations.

For each configuration, we generate queries for 5,661 products. To optimize the quality of the generated queries, we evaluate each parameter set using metrics such as semantic similarity and ROUGE scores. The parameters are tuned using a test dataset that the model has not seen during training (including epoch evaluation). For each product in this dataset, a single ground truth query is provided, which we use to compute similarity scores against the generated queries. Specifically, we calculate the average semantic similarity between the product text and the generated queries using cosine similarity, and we compute an overall average of ROUGE-1, ROUGE-2, and ROUGE-L scores by comparing each generated query to its corresponding reference query. These metrics are then averaged across all products for each parameter combination.

Finally, we calculate a weighted score using the formula:

$$1.5 \cdot (\text{average ROUGE score}) + (\text{average cosine similarity score}) \tag{5.3}$$

According to our evaluation metrics, the best query generation performance is achieved with the configuration: **num_beams: 5**, **top_p: 0.9**, and **temperature: 0.8**.

5.2 Negative Sampling

As we begin to utilize our model for generating relevant query-product pairs, it becomes necessary to develop a strategy for creating irrelevant pairs as well. This process is known as negative sampling [94]. In our methodology, we employ random sampling: we first generate a positive query-product pair and then randomly select another product to pair with the same query as an irrelevant match. We label these pairs with a 1 for relevance and a 0 for irrelevance.

This method may not yield optimal results since it can produce negative pairs that are too straightforward for the model to differentiate effectively. In future work, hard negative sampling techniques could be utilized [94]. However, the implementation of more sophisticated approaches is beyond the scope of this diploma thesis.

5.3 Generating Synthetic Datasets

A key objective of this thesis is also to produce a substantial synthetic dataset of positive and negative query-product pairs. To achieve this, we leverage our fine-tuned BART model to generate large-scale datasets. These datasets consist of query-product pairs annotated with a relevance label, where relevant pairs are assigned a label of 1 and

non-relevant pairs are assigned a label of 0. Such datasets are critical for training and evaluating information retrieval systems, including search engines on e-commerce platforms.

The product collection is divided into four equal parts, each representing 25% of the product descriptions. This division facilitates parallel processing and optimizes computational resource usage. Each partition is used to generate a separate dataset, resulting in four large datasets. For each product description within a partition, up to five synthetic queries are generated using the fine-tuned BART model. The model accepts the product description as input and produces likely search queries that users might employ to locate the product. These generated queries are paired with their corresponding product descriptions and labelled with a relevance score of 1, representing relevant query-product pairs. Additionally, for each generated query, we randomly select a product description from an unrelated product and pair it with the query, labelling these pairs with a relevance score of 0, indicating non-relevant pairs. This approach implements negative sampling, which is crucial for training models to differentiate between relevant and non-relevant items. The positive and negative pairs are combined into a single *DataFrame*. Furthermore, a separate dataset containing only positive pairs is created, using 40% of the product collection. This dataset includes the generated queries and their respective product descriptions and can be utilized for tasks where negative sampling is unnecessary or for additional fine-tuning. Any duplicate pairs produced by the model are removed.

So, our primary objective is to generate synthetic queries for each product in the *Collection* dataset. To accomplish this, we create five tab-separated files, each containing approximately 920,000 positive and 920,000 negative pairs. Processing each subset requires 10 hours on an A100 GPU, and we designate these subsets as Synthetic Pairs 1 to 5. However, we do not utilize this extensive dataset of around 9 million pairs for training due to memory constraints on our GPU Cluster. Additional details are provided in Table 5.2.

For training our dense retriever model, we use a different dataset consisting solely of positive pairs. This dataset, named *Synthetic Positive Pairs*, includes 1.45 million pairs and takes 7 hours to generate. We produce 1,450,042 queries across 392,390 products, averaging 3.7 queries per product. Further information can be found in Table 5.3.

5.4 Data Quality Evaluation

In this section, we assess the quality of our synthetically generated data. As discussed in [20], conventional metrics such as BLEU and ROUGE are often inadequate for this task, as they prioritize sentence-level matches rather than capturing the broader distribution of generated text. To address these limitations, recent studies [20, 36, 74] have incorporated TF-IDF analysis, embedding similarity measures, and human reviews of sampled data to provide a more comprehensive evaluation. Following this methodology, we adopt a similar approach to evaluate the quality of our generated dataset.

query	product_text	relevance
mindspace superfood	Sun Potion Lion's Mane (Organic) - Mind Power Mushroom (100g) Mind Power Mushroom Lions Mane Mushroom Powder - Superfood Extract Organic Dietary Supplement - Adaptogenic Nootropic Herb for Memory, Mental Clarity, Athletic Recovery, and Health - Non GMO Vegan Mind Power Mushroom Brain + Nervous System Tonic This organically grown Lion's Mane Mushroom powder may support: Memory Mental Clarity Enhanced Concentration...	1
iphone xr thin case	Ergodyne Skullerz Odin Anti-Fog Safety Glasses - Black Frame, Clear Lens Full-frame safety glasses with anti-fog technology provide an increased level of coverage from dangerous tools to everyday dust and debris. Scratch-resistant lenses filter 99.9 percent of harmful UVA, UVB and UVC rays for safe, outdoor use. Full-frame design features a flexible nylon frame and strong polycarbonate temple construction for increased durability...	0
ergonomic desk chair without wheels	KOLLIEE Armless Office Chair Mesh Ergonomic Small Desk Chair Armless Adjustable Swivel Black Computer Task Chair No Armrest Mid Back Home Office Chair for Small Spaces KOLLIEE Mesh Office Chair, a professional sleek office chair provides sophisticated support for all-day comfort. Installation Video Showed on Detail Page We provide an installation video on the "Images" part. You can see all steps of assembling in the second half of the video...	1
bff gifts for boys from girls	lenoup Rainbow Cell Phone Ring Stand Holder, Blue Multicolor Ring Grip Kickstand,360 Rotation Metal Finger Ring for Almost All Phones, Pad. Zinc Alloy Instruction: 1. Wipe away dirt dust on the back of your phones or tablets and dry it completely. 2. Peel off the back of paper.Don't touch the adhesive surface by your hands. 3. Please confirm the position to stick the phone ring as the adhesive is not reusable....	0
nerf guns good and small	NERF Rival 50-Round Refill Pack Whether you're on Team Red or Team Blue, the Nerf Rival competition is going to be fierce – and you'll need every round you've got! Each high-impact round in this 50-round Refill Pack works with Nerf Rival blasters (sold separately). If it's precision and power you want, you need the 50-round Refill Pack! Nerf Rival and all related properties are trademarks of Hasbro...	1

Table 5.2: Synthetic Pairs Dataset

query	product_text
black light for cars interior	ABALDI El Wires Car Interior LED Lights, Ambient Lighting Kits for Car Decoration(5M/16FT, Blue) Package included: – 5m EL wire*1 – El wire line driver*1 – Plastic triangle scraper*1 Features: – Wire lenth: 5m/16ft – Wire Diameter: 2.3mm – Sewing Edge: 5mm – Can offer 360 degrees of illumination – Energy saving and environmental-friendly – Flexible and water resistant, can be bent into any shape and cut into any length. but you will need to seal the new...
zip up nightgowns for women	Ekouaer Sleepwear Women Snap/Button Front Housecoats Cotton Dusters Short Sleeve House Dress S-XXL From the brand Previous page Women House Dress Button Down Nightshirt Women Satin Robe Couple Sleepwear for Christmas Christmas Pajamas Set Christmas Pajamas for women Womens Waffle Knit Loungewear Button Down Pajamas Set Zipper Front Robe for Women Long Nightgown for Women Sleepwear for Men/Women...
work uniform men	Chef Works Men's Hartford Chef Coat Brimming with confidence, the Hartford stands ready to tackle everyday tasks. From the grandest project to the tiniest minutia, Hartford is your day-in-and-day-out hero. It feels executive, even if the price point doesn't. Brimming with confidence, the Hartford stands ready to tackle everyday tasks. From the grandest project to the tiniest minutia, Hartford is your day-in-and-day-out hero. It feels executive, even if the price point doesn't. From the manufacturer...

Table 5.3: Synthetic Positive Pairs Dataset

First, we begin by comparing the unique vocabularies found in the *queries* and *product_text*. The **Query Vocabulary Size** comprises 53,701 unique words, indicating a moderate variety in the types of products that users are searching for. In contrast, the **Product Text Vocabulary Size** includes 1,821,431 unique words, suggesting that product descriptions are quite extensive and provide detailed information across a wide range of products.

The **Vocabulary Diversity Ratio** stands at 0.029, indicating that for every unique term in the queries, there are approximately 34 unique terms in the product descriptions. This low ratio highlights that the language used in product descriptions is considerably more extensive and varied compared to that in queries. In the context of recommendation systems, this suggests that users typically use more concise and direct language, whereas

product descriptions tend to be more elaborate. This discrepancy could affect the precision of information retrieval.

To accommodate this difference in language complexity, we have set the maximum output length of our model to 64 tokens. This decision is based on our analysis of the *Query ID to Query* dataset, where the longest query contains 29 tokens. By allowing up to 64 tokens for synthetic queries, we aim to generate longer queries that can capture more detailed product information.

5.4.1 TF-IDF Evaluation

Next, we randomly (with state) select 100,000 observations from our *Synthetic Positive Pairs* dataset and apply TF-IDF vectorization to both the *query* and *product_text* columns. We then calculate the cosine similarity between each query and its corresponding product description to assess their semantic alignment. The resulting average cosine similarity of approximately 0.1784 suggests a low level of semantic relevance between queries and product descriptions. This indicates some alignment in the usage of significant terms, yet there remains substantial potential for enhancing the synthetic queries to more accurately capture the content and context of the associated product texts.

5.4.2 Semantic Similarity

To go beyond simply matching words, we use our pretrained embedding model, $GTE_{large-en-v1.5}$, to convert the text into numerical vectors. We then calculate the cosine similarity for each pair of texts and create a distribution to visualize these similarities (Figure 5.2). Due to memory limitations on our cluster, we randomly (with state) select 20% of the total data.

On average, the cosine similarity is 0.644. This means that, typically, the pairs of texts share a moderate to high level of meaning. This result suggests that our method for generating synthetic query-product pairs is effective in producing relevant matches. The median similarity score is 0.652, which is very close to the average, indicating that the data is fairly evenly distributed.

Looking at the 75th percentile, the similarity score is 0.709. This high value shows that a large portion of the pairs have strong semantic alignment. Most of the synthetic pairs clustered around similarity scores between 0.6 and 0.7. This peak is in line with our expectations for positive query-product pairs, where we aim for high but not perfect similarity.

There is a slight tail in the distribution extending to lower similarity values (below 0.5), which means that some pairs are less relevant. However, this portion is small, indicating that our generation process rarely produces low-quality pairs.

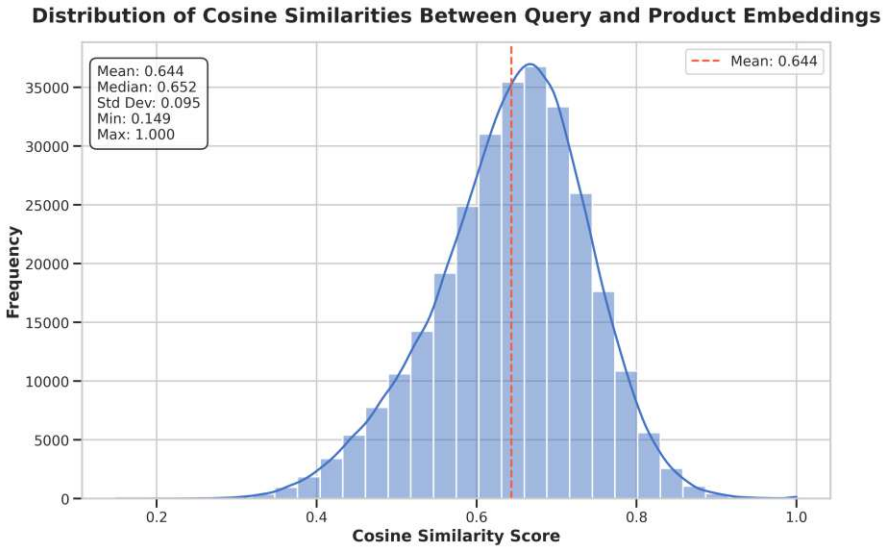

Figure 5.2: Synthetic Pairs Cosine Similarity Distribution

5.4.3 Qualitative Observations

Analysis Criteria

To evaluate relevance, we apply the following qualitative criteria:

1. **Query-Product Semantic Similarity:** The product description should contain key terms from the query and align semantically.
2. **Specificity Matching:** Products should match specific terms in the query (e.g., "sds max chisel" should retrieve tools with similar specifications).
3. **User Intent Fulfillment:** The product should align with the intended use suggested by the query, avoiding unrelated items.

Since we conduct a manual quality assessment, we select only 100 pairs at random from the *Synthetic Positive Pairs* dataset, using a fixed seed for reproducibility. Below, we provide several examples of relevant matches and mismatches, and then summarize our observations based on the review of these 100 pairs.

High Relevance Examples

- **Query:** "humidifier without filter"
  Product Description: "Raydrop Cool Mist 2.2L Humidifiers for Bedroom..."
  This pair demonstrates a strong match as the product addresses the key requirement of being a humidifier, with specifications that likely match user expectations.

- **Query:** “step in bike trailer coupler”
  Product Description: “Burley Bee, 1 and 2 Seat Lightweight, Kid Bike Trailer...”
  The product description directly matches the query, suggesting a high likelihood of relevance. This alignment with specific user needs increases retrieval quality.

Potential Misalignment Examples

- **Query:** “open when letters kit”
  Product Description: “BUSINESS AND LEGAL 500 LETTER TEMPLATES: A BOOK...”
  Although the product includes “letters,” it does not appear to fulfill the implied intent of the query, which likely refers to personal or gift-style “open when” letters. Thus, it may be marked relevant but does not fully meet user expectations.
- **Query:** “garden gate”
  Product Description: “Zippity Outdoor Products WF29012 Black Metal Gate...”
  The product aligns with the garden gate theme, but additional details such as size or material (e.g., “metal gate”) could provide a clearer alignment with user intent.

The analysis of the sample pairs reveals several insights:

- **Relevance Variation**: Not all query-product pairs labeled as relevant truly meet the relevance criteria. Some pairs are fully relevant, others partially, and some are irrelevant.
- **Specificity Matters**: Queries with specific attributes (e.g., "SDS Max," "pink") require precise product matches. Mismatches in key attributes lead to irrelevance.
- **Synthetic Generation Limitations**: The synthetic process may not capture nuanced differences between similar products (e.g., "SDS Max" vs. "SDS Plus") or may overlook essential attributes.

The synthetically generated query-product pairs exhibit a mixture of relevance levels. While some pairs are appropriate and valuable for product search applications, others are not sufficiently aligned with the query intent. For the dataset to be effective in training and evaluating product search models, it is crucial to ensure that the synthetic generation process accurately captures query nuances and matches products accordingly.

5.5 Summary

In this chapter, we presented a comprehensive methodology for generating synthetic query-product pairs to enhance the training and evaluation of information retrieval systems within e-commerce platforms. Leveraging the robust capabilities of the BART

(Bidirectional and Auto-Regressive Transformers) model, specifically the pre-trained $BART_{large}$ variant, we successfully fine-tuned the model to generate relevant and diverse search queries based on extensive product descriptions.

The fine-tuning process, utilizing a substantial dataset of over 190,000 positive pairs, demonstrated the effectiveness of BART's encoder-decoder architecture in capturing the intricate relationships between product attributes and user search intents. Through meticulous hyperparameter tuning, we optimized the model to balance relevance and diversity in the generated queries, achieving notable improvements in semantic similarity and ROUGE scores. The selection of optimal parameters, such as $num\_beams = 5$, $top\_p = 0.9$, and $temperature = 0.8$, underscored the importance of fine-grained adjustments in enhancing query quality.

The generation of large-scale synthetic datasets, encompassing approximately nine million pairs, underscores the scalability of our approach. Despite computational constraints limiting the immediate utilization of the entire dataset, the creation of a dedicated subset of 1.45 million positive pairs for the dense retriever model illustrates the practical applicability of our methodology in real-world scenarios.

Data quality evaluations, encompassing vocabulary analysis, TF-IDF assessments, semantic similarity measurements, and qualitative reviews, provided a multifaceted understanding of the synthetic data's effectiveness. While quantitative metrics indicated a moderate to high level of semantic alignment, qualitative assessments revealed variations in relevance, particularly in cases requiring precise attribute matching. These findings emphasize the inherent challenges in synthetic data generation, such as capturing subtle contextual nuances and ensuring comprehensive alignment with user intents.

CHAPTER 6

Two-Tower Retriever Model

A dual encoder architecture consists of two independent encoders, each responsible for transforming an input (such as text) into an embedding, with the model being optimized based on similarity measures in the embedding space. This approach has demonstrated strong performance across various tasks, including information retrieval and question answering [23, 32]. A notable advantage of this model is its ease of deployment, as the embedding index can be updated dynamically for new or modified documents and passages without requiring the encoders to be retrained [23]. This contrasts with generative neural networks used in question answering, which need retraining when presented with new data, making dual encoders more adaptable to changes.

There are several valid configurations for dual encoders. As outlined in Table 6.1, earlier research highlights two primary types: Siamese Dual Encoders (SDE) and Asymmetric Dual Encoders (ADE). In SDEs, both encoders share parameters, while in ADEs, some or none of the parameters are shared [23]. In practice, asymmetry between the two encoders is often required, particularly when the inputs to the two towers differ. In our approach, we use a Dual Encoder architecture with shared initial layers but separate subsequent layers. This design allows each tower to learn representations specific to queries or products, with distinct dense layers capturing domain-specific features for each.

Model	Architecture
DPR [32]	Asymetric
DensePhrases [38]	Asymetric
SBERT [71]	Siamese
ST5 [60]	Siamese

Table 6.1: Existing off-the-shelf dual encoders

We leverage *TensorFlow* [1] to construct a deep retrieval model, designed to capture complex patterns through its architecture. In general, deeper models can learn more intricate relationships than shallower models. Our model is built upon the $GTE_{LARGE-EN-V1.5}$ sentence encoder, which generates pretrained embeddings for synthetic search queries and product information—formed by concatenating product titles with descriptions. Neural networks are then constructed on top of this embedding layer to fine-tune the model specifically for product search. In contrast, a shallow model, such as one with a single layer and no activation function, might capture only the simplest associations between search queries and products.

However, the complexity of deeper models brings certain challenges. First, they are computationally intensive, requiring greater memory and processing resources for both training and inference. Additionally, deeper models generally require more training data to avoid overfitting and to ensure they learn generalizable patterns rather than merely memorizing examples. Finally, training deep models can be challenging, necessitating careful attention to hyperparameters such as regularization and learning rate to ensure effective learning.

Optimizing the architecture of a real-world recommender system involves a nuanced balance of intuition and precise hyperparameter tuning. Key factors, including model depth and width, activation functions, learning rates, and optimizers, can significantly influence performance. This complexity is further compounded by the gap between offline and online performance metrics, underscoring that the choice of optimization objective can be as crucial as the model itself.

6.1 Training Process

Although developing and optimizing larger models presents challenges, the substantial performance improvements they offer often justify the effort. In this study, we demonstrate the construction of deep retrieval models using *TensorFlow Recommenders* [1]. To accelerate training and address memory constraints, we initially precompute embeddings for our query-product pairs and store them separately. This pre-processing step, which requires approximately six hours on a V100 GPU, facilitates a more efficient training process.

Our objective is to learn embeddings such that the inner product (or cosine similarity) between a query embedding and its corresponding relevant item embedding is maximized, while the similarity with irrelevant items is minimized.

- Let $\mathcal{Q}$ denote the set of all possible queries.
- Let $\mathcal{P}$ denote the set of all possible products.
- Each query $q \in \mathcal{Q}$ is represented by features $\mathbf{x}_q \in \mathbb{R}^{d_q}$.

[1] `https://www.tensorflow.org/recommenders` last seen: 10.24.2014

- Each product $p \in \mathcal{P}$ is represented by features $\mathbf{x}_p \in \mathbb{R}^{d_p}$.
- The embedding dimension is denoted as d_e.

We define the Query Tower as a function $f_q : \mathbb{R}^{d_q} \to \mathbb{R}^{d_e}$, parameterized by θ_q, which maps query features to embeddings:

$$\mathbf{e}_q = f_q(\mathbf{x}_q; \theta_q) \tag{6.1}$$

Similarly, the Product Tower is defined as a function $f_p : \mathbb{R}^{d_p} \to \mathbb{R}^{d_e}$, parameterized by θ_p, which maps product features to embeddings:

$$\mathbf{e}_p = f_p(\mathbf{x}_p; \theta_p) \tag{6.2}$$

To ensure the embeddings lie on the unit hypersphere, we perform L2 normalization:

$$\mathbf{e}_q \leftarrow \frac{\mathbf{e}_q}{\|\mathbf{e}_q\|_2}, \quad \mathbf{e}_p \leftarrow \frac{\mathbf{e}_p}{\|\mathbf{e}_p\|_2} \tag{6.3}$$

The similarity between a query and a product is then measured using the dot product of their embeddings:

$$s(q, p) = \mathbf{e}_q^\top \mathbf{e}_p \tag{6.4}$$

Because the embeddings are L2-normalized, the similarity score $s(q, p)$ lies within the interval $[-1, 1]$.

We approach the task by identifying the appropriate product p for each given query q. The candidate pool includes all products from the positive pairs dataset. Our retrieval model functions implicitly, meaning that any query-product pair not present in the dataset is considered a negative pair that the model should not retrieve. In the *TensorFlow* retrieval framework, we generate pairs by combining each query with every candidate product and apply a cross-entropy loss function. This system manages both positive and negative pairs, effectively utilizing a binary cross-entropy approach. The true labels are represented as one-hot encoded vectors, where a value of 1 indicates a positive pair and 0 denotes a negative pair.

For a specific positive pair consisting of query i and candidate j, the dot product $s(i, j)$ is expected to be higher than the scores of any negative pairs. We then apply softmax normalization across all candidates to determine the probability of the positive candidate:

$$P(j|i) = \frac{\exp(s(i, j))}{\sum_k \exp(s(i, k))} \tag{6.5}$$

The loss function is structured to push the model to increase the probability of the correct candidate j, thereby reducing the cross-entropy loss defined as $\mathcal{L} = -\log P(j|i)$.

6.2 Architecture

To identify the most effective architecture, we tested various layer sizes and activation functions on a smaller portion of our dataset. Specifically, we selected the first 100,000 entries from the *Synthetic Positive Pairs* dataset. This subset was divided into 80% for training and 20% for epoch evaluation. For each query, we retrieved potential matches from the pool of products within the *Synthetic Positive Pairs* dataset and then verified whether the relevant pair was included in the retrieved list. Each architectural configuration underwent training for five epochs, and the best-performing model achieved a top-100 accuracy of 0.201.

As illustrated in Figure 6.1, we utilize our $GTE_{LARGE-EN-V1.5}$ model to generate precomputed embeddings with dimensions of (1,1024). These embedding vectors are then fed into the respective towers. Both the query and product towers share an identical architecture, consisting of a Dense layer with 128 units and an ELU activation function, followed by a Linear layer of the same size. Finally, we normalize the vectors. In our approach, the magnitude of the vectors is not critical, and by normalizing them, the dot product effectively becomes the cosine similarity metric, as previously described. Our model combines the *QueryModel* and *ProductModel* and defines the training objective using *tfrs.tasks.Retrieval*. This task handles the loss function and the evaluation with top-k accuracy. This metric measures how often the true candidate is among the top K candidates for a given query. Additionally, we use an optimizer (Adagrad) with an exponential decay learning rate starting at 0.1.

6.2.1 Training Evaluation

With the retriever architecture now in place, we proceed to examine our entire *Synthetic Positive Pairs* dataset. Our embedding analysis revealed that some of the generated question-product pairs were irrelevant (see Figure 5.2). To mitigate this, we initiate a filtering process to remove potentially irrelevant pairs. As discussed in Section 5.3, we discovered that **25%** of the pairs had a similarity score below 0.585. The **median** similarity score was 0.652, closely matching the mean score of 0.6436. Additionally, **75%** of the pairs scored below 0.709 in similarity.

To systematically assess the effects of various similarity thresholds, we developed several *SYNGET* (Synthetic Fine-Tuned GET Retriever) models, each incorporating different cutoff values for the *Synthetic Positive Pairs* dataset:

- *SYNGET-U*: This model retains all pairs without applying any filtering.
- *SYNGET-0.6*, *SYNGET-0.7*, and *SYNGET-0.8*: These models exclude pairs with similarity scores below 0.6, 0.7, and 0.8, respectively.

This approach also limits the dataset size for each model. The unrestricted model is trained on 1,160,065 synthetic pairs, whereas *SYNGET-0.8* is trained on only 42,300

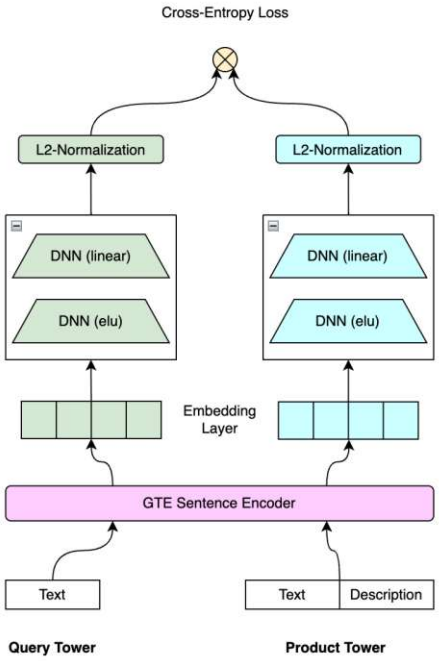

Figure 6.1: Two Tower Architecture

pairs (Table 6.2). When evaluating the models on the *Test QREL (NIST)* set, we will assess their performance on unseen, non-synthetic pairs.

Our goal is to determine how varying levels of pair filtration influence performance. All models are trained on an NVIDIA A100 GPU over 50 epochs. Each model utilizes 80% of the synthetic data for training and the remaining 20% for epoch validation. We do not use a separate test set, as the models are evaluated against each other using real-world labeled query-product pairs. The training performance of each model was analyzed to identify the optimal cutoff value that achieves a balance between relevance and comprehensiveness in the retrieved pairs. Table 6.2 details the dataset sizes employed by each model during training. We hypothesize that the model with the highest cutoff may overfit the synthetic data, as the pool of possible candidates diminishes when applying stricter filters to the dataset.

Model	Training Pairs	Evaluation Pairs	Candidates
SYNGET-0.8	42 300	10 576	35 177
SYNGET-0.7	330 861	82 716	214 006
SYNGET-0.6	817 160	204 290	362 648
SYNGET-U	1 160 065	290 017	392 390

Table 6.2: SYNGET retriever development datasets

In a production setting, the retriever is required to select from a vast pool of products rather than being limited to a small subset with known relevance. To address this, we introduce an additional model, *SYNGET-0.8 AC*, which applies the same filtering approach but retrieves from the full pool of 980,974 candidates. This approach is intended to help the retriever identify highly relevant pairs across the entire dataset, as training exclusively on the smaller subset could potentially hinder the model's performance.

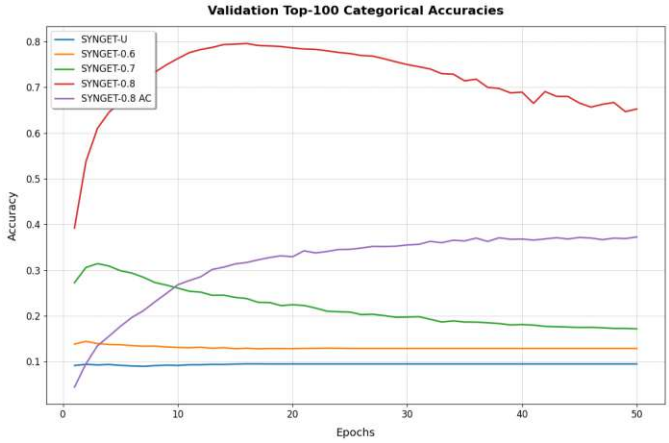

Figure 6.2: SYNGET Accuracy Comparison

Figure 6.2 presents the top-100 accuracy of each model on the evaluation dataset after each epoch. The *SYNGET-U* model incorporates all synthetic pairs without any filtering. As shown in the figure, this model consistently exhibits the lowest validation accuracy across all 50 training epochs. This observation suggests that the inclusion of noisy or irrelevant pairs in the training dataset significantly hinders the model's ability to learn meaningful patterns and effectively retrieve relevant items.

Accuracy	**SYNGET-0.8**	**SYNGET-0.8 AC**
Top-100	0.65	0.38
Top-50	0.53	0.285
Top-10	0.28	0.13
Top-5	0.19	0.08

Table 6.3: Top-k accuracy of SYNGET-0.8 models after 50 epochs

Among all models, *SYNGET-0.8* implements the strictest filtering criteria and demonstrates superior performance, achieving the most stable validation accuracy with a peak around 0.8. Despite a minor performance drop post epoch 30, it's worth noting that this model's training is limited to a smaller, potentially more distinguishable candidate set. In comparison, *SYNGET-0.8 AC* exhibits lower top-100 accuracy but shows consistent improvement throughout all 50 epochs, culminating in a score of 0.38. This model's approach involves learning relevant pairs from the complete candidate pool while considering unknown combinations as irrelevant. As shown in Table 6.3, the model trained on

the restricted candidate pool achieves evaluation metrics approximately double those of the *AC* model.

6.3 Evaluation

We now evaluate our Two Tower Retriever against the BM25 baseline. The evaluation is conducted on an unseen set of 115,490 query-product relevance pairs corresponding to 186 distinct queries. The candidate pool comprises our full collection dataset, which consists of 980,974 products. No thresholds are applied, ensuring that all models face the same level of difficulty. As previously mentioned, our evaluation focuses on the *NDCG@k* metric in conjunction with the *Precision@k* metric. This combination is particularly suitable as we utilize categorical relevance scores ranging from 0 to 3. For *precision*, any score above 0 is classified as relevant, given the majority of pairs are labeled as 0 (non-relevant). The *NDCG* metric further incorporates the ranking order, resulting in higher scores when the top-k retrieved items are ranked by relevance, with items scoring 3 appearing first. In addition to the BM25 baseline, we assess the performance of $GTE_{large-en-v1.5}$ and $MiniLM_{L6-v2}$[2] without any fine-tuning.

Model	NDCG@10	Precision@5	Precision@10
Okapi BM25	0.664	0.688	0.581
GTE Large v1.5	0.617	0.680	0.597
MiniLM L6 v2	0.584	0.647	0.576
SYNTGET-0.8 AC	0.660	0.711	0.635
SYNGET-0.8	0.564	0.626	0.564
SYNGET-0.7	0.220	0.329	0.314
SYNGET-0.6	0.169	0.262	0.259
SYNTGET-U	0.165	0.239	0.243

Table 6.4: Model Evaluation on Test Set

6.3.1 Analysis of the Retriever Evaluation

Table 6.4 and Figure 6.3 present the retrieval performance of several models. In Section 3.3 we stated why GTE was chosen for fine-tuning. Therefore, we decide to include the pretrained GTE and the MiniLm-L6-v2 models in our comparison. In the referenced MTEB leaderboard these models rank at place 19 and 139 while a BM25 algorithm ranks at place 182 [3], so they seem to be relevant models for our evaluation.

At a first glance, Okapi BM25 appears as a strong baseline, achieving an *NDCG@10* of 0.664 and a *Precision@5* of 0.688, though its *Precision@10* drops to 0.581. This suggests that while BM25 is effective at ranking and retrieving the most relevant items at the

[2] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2, last seen: 02.20.2025

[3] https://huggingface.co/spaces/mteb/leaderboard, last seen: 10.15.2024

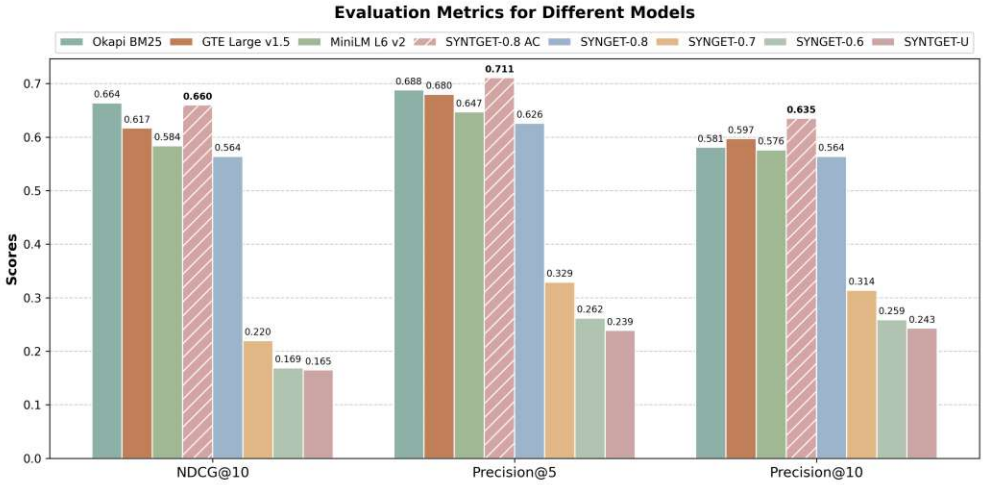

Figure 6.3: Evaluation Metrics for Different Models

very top, its performance is somewhat diminished as more items are considered in the precision calculation.

In contrast, the neural models built upon synthetic positive pairs reveal how alterations in training data quality, affected by varying similarity cutoff thresholds, influence overall performance. Notably, the *SYNTGET-0.8 AC* model shows competitive results with an *NDCG@10* score of 0.660, a *Precision@5* of 0.711, and a *Precision@10* of 0.635. The fact that this model, which uses a strict filtering criterion (a cutoff of 0.8) on the synthetic dataset and is evaluated on the full candidate pool, performs almost on par with BM25 in terms of NDCG and even surpasses it in precision metrics is indicative of the benefits of high-quality, filtered training data. This performance implies that a more rigorous filtration process, which removes lower similarity pairs, can lead to a model that more accurately identifies highly relevant query-product pairs when compared to versions that use less strict filtering.

On the other hand, models with less stringent filtering thresholds, such as *SYNGET-0.7* and *SYNGET-0.6*, exhibit significantly poorer performance. Their lower metrics across the board indicate that including a larger volume of synthetic pairs with lower similarity scores introduces noise that hampers the model's ability to accurately discern relevance in unseen, real-world data. The unfiltered model, *SYNGET-U*, which is trained on the full spectrum of synthetic data without any similarity-based exclusion, also underperforms considerably. This underscores a critical insight: the quality of synthetic pairs is paramount. Without filtering, the inclusion of irrelevant or low-quality training pairs can severely undermine model performance.

CHAPTER 7

Conclusion

7.1 Summary

This thesis explores the enhancement of embedding-based product search systems by leveraging Large Language Models (LLMs) for synthetic query generation. Traditional lexical term-matching algorithms, such as BM25, often struggle with retrieving relevant documents in cases where queries and documents lack lexical overlap. Embedding-based retrieval (EBR) models address this challenge but are heavily reliant on large volumes of supervised training data, which is often unavailable.

The core contribution of this work is the fine-tuning of LLMs, specifically BART, to generate synthetic queries based on product descriptions. This approach creates a rich dataset of query-product pairs, facilitating the training of a two-tower neural retriever model. The two-tower architecture, which encodes queries and products separately, enables precise matching and demonstrates superior performance compared to traditional single-tower models.

Experiments conducted in this thesis compare the performance of the proposed synthetic-data-trained retriever against BM25 and zero-shot LLM-based retrievers. Results indicate improvements in retrieval performance when it comes to NDCG and precision, showcasing the potential of synthetic data to overcome data scarcity challenges.

This research contributes to advancements in the field of information retrieval by providing insights into the integration of LLMs and synthetic data for product search tasks.

7.2 Discussion

While our experiments demonstrate that filtering synthetic query-product pairs can improve retrieval performance, the fine-tuned retriever still falls short of consistently

surpassing a classical baseline like BM25. Several factors may contribute to these limitations, both stemming from the nature of the synthetic data and the inherent complexity of the product search domain.

Firstly, the generation of synthetic data relies on a language model (BART) trained to produce plausible queries for given products. Although these queries can appear coherent and product-related, they often fail to capture the nuanced relevance criteria that real users apply. This discrepancy can manifest in subtle ways: queries may be syntactically plausible but semantically misaligned with actual user intent, resulting in a fine-tuned retriever that struggles when confronted with authentic, more diverse user queries. The presence of irrelevant or weakly aligned synthetic pairs—even those passing a relatively high similarity threshold—can dilute the signal in the training data, leading the model to learn patterns that do not generalize well.

Secondly, the product descriptions themselves are rich, varied, and domain-specific, encompassing extensive vocabularies and technical specifications. Synthetic queries tend to focus on broad attributes or frequently occurring terms, often failing to reflect the complexity and diversity of user preferences and niche product attributes. Consequently, the retriever may learn representations that work reasonably well for queries resembling the synthetic training examples but break down under more specialized, less frequently encountered search intents. This gap between the synthetic and real-world distributions inevitably constrains the achievable improvements.

Thirdly, the embedding-based retriever's reliance on semantic vector representations can inadvertently magnify discrepancies introduced during synthetic data generation. While lexical methods like BM25 excel at matching surface-level terms, embedding-based methods aim to model deeper semantic relationships. If the synthetic data does not faithfully encapsulate genuine semantic distinctions—due to noise in generation, imperfect filtering thresholds, or insufficient negative sampling strategies—then the resulting embeddings may cluster around superficial signals rather than truly meaningful differentiations. This leads to weaker retrieval performance when presented with complex or ambiguous queries that demand a richer semantic understanding.

Finally, despite careful filtering of synthetic pairs, the chosen cutoff thresholds may not be universally optimal. Overly conservative filtering risks discarding valuable training examples, shrinking the effective data size and potentially limiting model generalization. Conversely, lenient filtering allows too many low-quality pairs, reducing training efficacy. Striking the right balance remains non-trivial, and suboptimal filtering strategies may constrain the model's ability to outperform robust, well-established baselines.

7.3 Future Work

This thesis demonstrated the potential of synthetic query generation using Large Language Models and its application to enhancing embedding-based retrieval systems. While the

results are promising, several avenues for future research remain unexplored and warrant further investigation.

1. Future research could investigate alternative approaches to synthetic data generation, utilizing a wider array of large language models (e.g., GPT-series, T5) and employing more advanced fine-tuning methodologies. Additionally, integrating targeted data augmentation strategies that emphasize specific linguistic attributes, such as idiomatic expressions or industry-specific terminology, may enhance the model's robustness and its capacity for domain adaptation.

2. This thesis evaluated the quality of synthetic data using indirect metrics like improvements in retrieval performance. Future studies should aim to develop more comprehensive evaluation metrics that directly measure semantic fidelity, linguistic diversity, and contextual relevance of synthetic queries. Such metrics would provide a more nuanced understanding of the effectiveness of synthetic data across various retrieval tasks.

3. Leveraging large language models to generate challenging negative samples presents a promising avenue for enhancing synthetic data generation in retrieval tasks. By creating hard negatives, models can improve their ability to distinguish between relevant and irrelevant information, thereby increasing the overall accuracy and robustness of retrieval systems.

4. Combining embedding-based models with traditional retrieval techniques, such as BM25, offers a fertile ground for innovative research. Future work could explore the development of hybrid systems that dynamically integrate the strengths of both lexical and semantic retrieval methods. Utilizing synthetic data in these hybrid models could further boost performance, particularly in scenarios where individual retrieval approaches may fall short.

List of Figures

List of Tables

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. *ArXiv*, 2015. Software available from tensorflow.org.

[2] Qingyao Ai, Daniel N. Hill, S. V. N. Vishwanathan, and W. Bruce Croft. A zero attention model for personalized product search. CIKM '19, page 379–388, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Ateret Anaby-Tavor, Boaz Carmeli, Esther Goldbraich, Amir Kantor, George Kour, Segev Shlomov, Naama Tepper, and Naama Zwerdling. Do not have enough data? deep learning to the rescue! *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7383–7390, Apr. 2020.

[4] Akari Asai, Timo Schick, Patrick Lewis, Xilun Chen, Gautier Izacard, Sebastian Riedel, Hannaneh Hajishirzi, and Wen-tau Yih. Task-aware retrieval with instructions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 3650–3675, Toronto, Canada, July 2023. Association for Computational Linguistics.

[5] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. Ms marco: A human generated machine reading comprehension dataset. *ArXiv*, 2018.

[6] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[7] Luiz Bonifacio, Hugo Abonizio, Marzieh Fadaee, and Rodrigo Nogueira. Inpars: Data augmentation for information retrieval using large language models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, 2022.

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *ArXiv*, 2020.

[9] Daniel Campos, Surya Kallumadi, Corby Rosset, Cheng Xiang Zhai, and Alessandro Magnani. Overview of the trec 2023 product product search track. *ArXiv*, 2023.

[10] Aldo Gael Carranza, Rezsa Farahani, Natalia Ponomareva, Alex Kurakin, Matthew Jagielski, and Milad Nasr. Synthetic query generation for privacy-preserving deep retrieval systems using differentially private language models. *ArXiv*, 2024.

[11] Daixuan Cheng, Yuxian Gu, Shaohan Huang, Junyu Bi, Minlie Huang, and Furu Wei. Instruction pre-training: Language models are supervised multitask learners. *ArXiv*, 2024.

[12] Jianpeng Cheng and Dimitri Kartsaklis. Syntax-aware multi-sense word embeddings for deep compositional models of meaning. *ArXiv*, 2015.

[13] Stéphane Clinchant and Florent Perronnin. Aggregating continuous word embeddings for information retrieval. In Alexandre Allauzen, Hugo Larochelle, Christopher Manning, and Richard Socher, editors, *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, pages 100–109, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[14] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 191–198, New York, NY, USA, 2016. Association for Computing Machinery.

[15] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, USA, 1st edition, 2009.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[17] Sergey Edunov, Alexei Baevski, and Michael Auli. Pre-trained language model representations for language generation. In Jill Burstein, Christy Doran, and Thamar

Solorio, 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 4052–4059, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[18] Marzieh Fadaee, Arianna Bisazza, and Christof Monz. Data augmentation for low-resource neural machine translation. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 567–573, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[19] Josep Ferrer. How transformers work: A detailed exploration of transformer architecture. `https://www.datacamp.com/tutorial/how-transformers-work`. Accessed: 2024-01-06.

[20] Simone Filice, Jason Ingyu Choi, Giuseppe Castellucci, Eugene Agichtein, and Oleg Rokhlenko. Evaluation metrics of language generation models for synthetic traffic generation tasks. *ArXiv*, 2023.

[21] Luyu Gao and Jamie Callan. Unsupervised corpus aware language model pre-training for dense passage retrieval. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2843–2853, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[22] Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[23] Daniel Gillick, Alessandro Presta, and Gaurav Singh Tomar. End-to-end retrieval in continuous space. *ArXiv*, 2018.

[24] Alex Graves. Generating sequences with recurrent neural networks. *ArXiv*, 2014.

[25] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, page 55–64, New York, NY, USA, 2016. Association for Computing Machinery.

[26] Mark Haakman, Luís Cruz, Hennie Huijgens, and Arie Deursen. Ai lifecycle models need to be revised. *Empirical Software Engineering*, 26:95, 09 2021.

[27] Kailash A. Hambarde and Hugo Proença. Information retrieval: Recent advances and beyond. *IEEE Access*, 11:76581–76604, 2023.

[28] Matthew Henderson, Rami Al-Rfou, Brian Strope, Yun hsuan Sung, Laszlo Lukacs, Ruiqi Guo, Sanjiv Kumar, Balint Miklos, and Ray Kurzweil. Efficient natural language response suggestion for smart reply. *ArXiv*, 2017.

[29] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *ArXiv*, 2023.

[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[31] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *ArXiv*, 2022.

[32] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics.

[33] Tom Kenter and Maarten de Rijke. Short text similarity with word embeddings. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 1411–1420, New York, NY, USA, 2015. Association for Computing Machinery.

[34] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[35] Sosuke Kobayashi. Contextual augmentation: Data augmentation by words with paradigmatic relations. *ArXiv*, 2018.

[36] Huan Yee Koh, Jiaxin Ju, Ming Liu, and Shirui Pan. An empirical survey on long document summarization: Datasets, models, and metrics. *ACM Computing Surveys*, 55(8):1–35, December 2022.

[37] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019.

[38] Jinhyuk Lee, Mujeen Sung, Jaewoo Kang, and Danqi Chen. Learning dense representations of phrases at scale. *ArXiv*, 2021.

[39] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising

sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *ArXiv*, 2019.

[40] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. Embedding-based product retrieval in taobao search. *CoRR*, abs/2106.09297, 2021.

[41] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023.

[42] Davis Liang, Peng Xu, Siamak Shakeri, Cícero Nogueira dos Santos, Ramesh Nallapati, Zhiheng Huang, and Bing Xiang. Embedding-based zero-shot retrieval through query generation. *CoRR*, abs/2009.10270, 2020.

[43] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[44] Shichen Liu, Fei Xiao, Wenwu Ou, and Luo Si. Cascade ranking for operational e-commerce search. KDD '17, page 1557–1565, New York, NY, USA, 2017. Association for Computing Machinery.

[45] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[46] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *ArXiv*, 2019.

[47] Yuanhua Lv and Cheng Xiang Zhai. When documents are very long, bm25 fails. In *SIGIR'11 - Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'11 - Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 1103–1104, United States, 2011. Association for Computing Machinery. 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011 ; Conference date: 24-07-2011 Through 28-07-2011.

[48] Yuanhua Lv and ChengXiang Zhai. Adaptive term frequency normalization for bm25. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, page 1985–1988, New York, NY, USA, 2011. Association for Computing Machinery.

[49] Yuanhua Lv and ChengXiang Zhai. Lower-bounding term frequency normalization. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, page 7–16, New York, NY, USA, 2011. Association for Computing Machinery.

[50] Hao Ma, Haixuan Yang, Irwin King, and Michael R. Lyu. Learning latent semantic relations from clickthrough data for query suggestion. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, page 709–718, New York, NY, USA, 2008. Association for Computing Machinery.

[51] Ji Ma, Ivan Korotkov, Yinfei Yang, Keith Hall, and Ryan McDonald. Zero-shot neural passage retrieval via domain-targeted synthetic question generation. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1075–1088, Online, April 2021. Association for Computational Linguistics.

[52] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *ArXiv*, 2013.

[53] Bhaskar Mitra, Eric Nalisnick, Nick Craswell, and Rich Caruana. A dual embedding space model for document ranking. *ArXiv*, 2016.

[54] Biswesh Mohapatra, Gaurav Pandey, Danish Contractor, and Sachindra Joshi. Simulated chats for building dialog systems: Learning to generate conversations from instructions. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1190–1203, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[55] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022.

[56] Sheshera Mysore, Andrew McCallum, and Hamed Zamani. Large language model augmented narrative driven recommendations. *ArXiv*, 2023.

[57] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *ArXiv*, 2018.

[58] Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, Johannes Heidecke, Pranav Shyam, Boris Power, Tyna Eloundou Nekoul, Girish Sastry, Gretchen Krueger, David Schnurr, Felipe Petroski Such, Kenny Hsu, Madeleine Thompson, Tabarak Khan, Toki Sherbakov, Joanne Jang, Peter Welinder, and Lilian Weng. Text and code embeddings by contrastive pre-training. *ArXiv*, 2022.

[59] Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, Johannes Heidecke, Pranav Shyam, Boris Power, Tyna Eloundou Nekoul, Girish Sastry, Gretchen Krueger, David Schnurr, Felipe Petroski Such, Kenny Hsu, Madeleine Thompson, Tabarak Khan, Toki Sherbakov, Joanne Jang, Peter Welinder, and Lilian

Weng. Text and code embeddings by contrastive pre-training. *CoRR*, abs/2201.10005, 2022.

[60] Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith Hall, Daniel Cer, and Yinfei Yang. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1864–1874, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[61] Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. Document ranking with a pretrained sequence-to-sequence model. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 708–718, Online, November 2020. Association for Computational Linguistics.

[62] Yannis Papanikolaou and Andrea Pierleoni. Dare: Data augmented relation extraction with gpt-2. *ArXiv*, 2020.

[63] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[64] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[65] Matt Post. A call for clarity in reporting bleu scores. *ArXiv*, 2018.

[66] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *ArXiv*, 2021.

[67] Faisal Rahutomo, Teruaki Kitasuka, and Masayoshi Aritsugi. Semantic cosine similarity. *ResearhGate*, 10 2012.

[68] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *ArXiv*, 2016.

[69] Ori Ram, Gal Shachaf, Omer Levy, Jonathan Berant, and Amir Globerson. Learning to retrieve passages without supervision. *ArXiv*, 2022.

[70] Chandan K. Reddy, Lluís Màrquez i Villodre, Francisco B. Valero, Nikhil S. Rao, Hugo Zaragoza, Sambaran Bandyopadhyay, Arnab Biswas, Anlu Xing, and Karthik Subbian. Shopping queries dataset: A large-scale esci benchmark for improving product search. *ArXiv*, abs/2206.06588, 2022.

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

[72] Ruiyang Ren, Shangwen Lv, Yingqi Qu, Jing Liu, Wayne Xin Zhao, QiaoQiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. PAIR: Leveraging passage-centric similarity relation for improving dense passage retrieval. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2173–2183, Online, August 2021. Association for Computational Linguistics.

[73] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389, 01 2009.

[74] Haoxiang Shi, Sumio Fujita, and Tetsuya Sakai. Towards consistency filtering-free unsupervised learning for dense retrieval. *ArXiv*, 2023.

[75] Ian Soboroff, Shudong Huang, and Donna K. Harman. Trec 2018 news track overview. In *Text Retrieval Conference*, 2018.

[76] Daria Sorokina and Erick Cantu-Paz. Amazon search: The joy of ranking products. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 459–460, New York, NY, USA, 2016. Association for Computing Machinery.

[77] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: Heterogenous benchmark for zero-shot evaluation of information retrieval models. In *Proceedings of the 2021 Neural Information Processing Systems (NeurIPS-2021): Track on Datasets and Benchmarks*, 2021.

[78] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *ArXiv*, 2021.

[79] Kai Ming Ting. *Precision and Recall*, pages 781–781. Springer US, Boston, MA, 2010.

[80] Andrew Trotman, Antti Puurula, and Blake Burgess. Improvements to bm25 and language models examined. In *Proceedings of the 19th Australasian Document Computing Symposium*, ADCS '14, page 58–65, New York, NY, USA, 2014. Association for Computing Machinery.

[81] Alper Kursat Uysal and Serkan Gunal. The impact of preprocessing on text classification. *Information Processing Management*, 50(1):104–112, 2014.

[82] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *ArXiv*, 2019.

[83] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[84] Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. Trec-covid: Constructing a pandemic information retrieval test collection. *ArXiv*, 2020.

[85] Ivan Vulić and Marie-Francine Moens. Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, page 363–372, New York, NY, USA, 2015. Association for Computing Machinery.

[86] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *ArXiv*, 2019.

[87] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *ArXiv*, abs/2212.03533, 2022.

[88] Tian Wang, Yuri M. Brovman, and Sriganesh Madhvanath. Personalized embedding-based e-commerce recommendations at ebay. *ArXiv*, 2021.

[89] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Tie-Yan Liu, and Wei Chen. A theoretical analysis of NDCG type ranking measures. *CoRR*, abs/1304.6480, 2013.

[90] Marco Wrzalik and Dirk Krechel. Cort: Complementary rankings from transformers. *ArXiv*, 2021.

[91] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *ArXiv*, 2020.

[92] Yiben Yang, Chaitanya Malaviya, Jared Fernandez, Swabha Swayamdipta, Ronan Le Bras, Ji-Ping Wang, Chandra Bhagavatula, Yejin Choi, and Doug Downey. Generative data augmentation for commonsense reasoning. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1008–1025, Online, November 2020. Association for Computational Linguistics.

[93] Yinfei Yang, Ning Jin, Kuo Lin, Mandy Guo, and Daniel Cer. Neural retrieval for question answering with cross-attention supervised data augmentation. *ArXiv*, 2020.

[94] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Optimizing dense retrieval model training with hard negatives. *ArXiv*, 2021.

[95] Han Zhang, Songlin Wang, Kang Zhang, Zhiling Tang, Yunjiang Jiang, Yun Xiao, Weipeng Yan, and Wenyun Yang. Towards personalized and semantic retrieval: An end-to-end solution for e-commerce search via embedding learning. *CoRR*, abs/2006.02282, 2020.

[96] Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, et al. mgte: Generalized long-context text representation and reranking models for multilingual text retrieval. *arXiv preprint arXiv:2407.19669*, 2024.

[97] Yuyu Zhang, Ping Nie, Xiubo Geng, Arun Ramamurthy, Le Song, and Daxin Jiang. Dc-bert: Decoupling question and document for efficient contextual encoding. *ArXiv*, 2020.