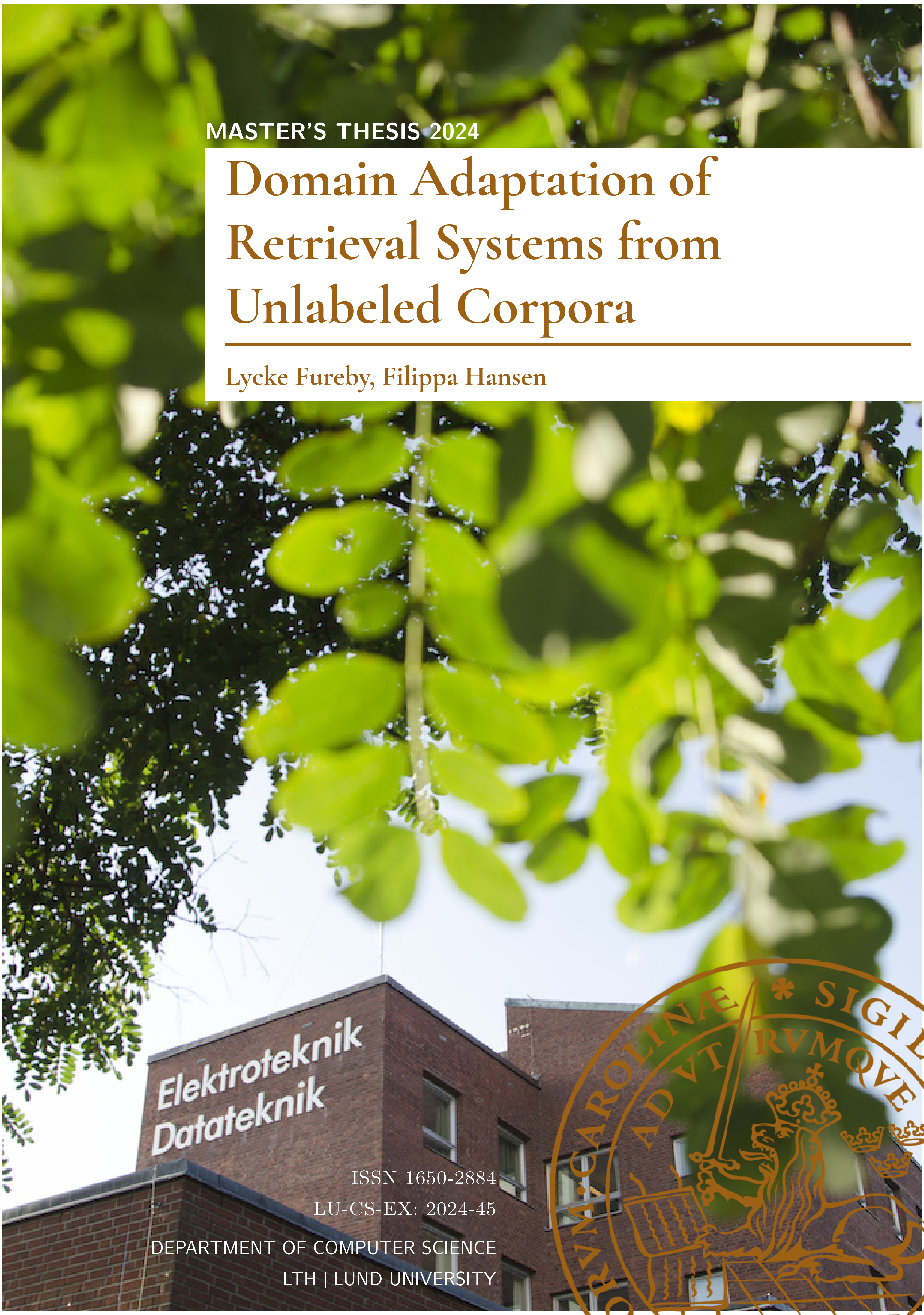


MASTER'S THESIS 2024

Domain Adaptation of Retrieval Systems from Unlabeled Corpora

Lycke Fureby, Filippa Hansen

ISSN 1650-2884
LU-CS-EX: 2024-45
DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-45

Domain Adaptation of Retrieval Systems from Unlabeled Corpora

Användning av Omärkta Korpora för Domänanpassning av Söksystem

Lycke Fureby, Filippa Hansen

Domain Adaptation of Retrieval Systems from Unlabeled Corpora

Lycke Fureby
lycke.f@gmail.com

Filippa Hansen
filippahaansen@gmail.com

June 26, 2024

Master's thesis work carried out at Modulai AB.

Supervisors: Patrik Edén, patrik.eden@cec.lu.se
Svante Sörberg, svante@modulai.io
Dmitrijs Kass, dmitrijs@modulai.io

Examiner: Elin A. Topp, elin_a.topp@cs.lth.se

Abstract

Retrieval Augmented Generation (RAG) is an increasingly popular machine learning technique combining the realm of Large Language Models (LLMs) with an Embedding Model based Information Retrieval System. This enables traceable, fact-based LLM generated answers, which is critical for professional tasks. The retrieval component of a RAG-system is crucial for accuracy, and open-source embedding models provide a solid foundation for successfully implementing such systems. However, for highly domain-specific text, a steep decrease in retrieval performance is observed, reducing RAG's effectiveness across specialized landscapes.

Fine-tuning open-source embedding models on domain-specific corpora could significantly enhance RAG's applicability by adapting the retrieval system to specific domains. For this purpose, various approaches to domain adaptation of dense retrieval systems were explored, aiming to optimize retrieval performance on a corpus consisting of highly technical car repair and diagnosis manuals. The investigated techniques included multiple text-splitting strategies, synthetic data generation, LLM-based data processing, hard negative mining, and several different fine-tuning methods. Evaluation was conducted on both synthetic and human-annotated data.

Peak performance was reached for a system utilizing an LLM for mining of hard negatives, creating an expanded dataset used for staged fine-tuning of the embedding model with Multiple Negatives Ranking loss followed by Online Contrastive loss. Overall, fine-tuning the embedding model on synthetic data was shown to yield statistically significant improvements in performance across both synthetic and human annotated queries. Moreover, the fine-tuned model outperformed a much larger base embedding model.

Keywords: RAG, open-source, fine-tuning, LLM, Embedding model, Information Retrieval

Acknowledgements

First of all, we would like to thank Modulai AB for giving us the opportunity to do our thesis with them and for their warm welcome. We would also like to direct a big thank you to our supervisors at Modulai, Svante Sörberg and Dmitrijs Kass who has helped us tackle problems during the way as well as providing valuable insights. A special thank you goes to our supervisor at Lund University, Patrik Edén, for the time and effort that he has put in while helping us remain within the scope of a Master Thesis.

Contents

1 Introduction **9**
- 1.1 Background . . . 9
- 1.2 Research Objective . . . 11
  - 1.2.1 Research Questions . . . 11
- 1.3 Scientific contribution . . . 11
- 1.4 Related work . . . 12
- 1.5 Outline . . . 13

2 Domain Adaptation of Dense Retrieval Systems **15**
- 2.1 Embedding models . . . 15
  - 2.1.1 The transformer . . . 16
  - 2.1.2 Bidirectional Encoder Representations from Transformers (BERT) . . . 19
  - 2.1.3 Sentence Transformers . . . 19
  - 2.1.4 Bi-Encoders . . . 20
  - 2.1.5 Cross-Encoders . . . 20
- 2.2 Data Processing and Generation . . . 21
  - 2.2.1 Parsers and Chunking . . . 21
  - 2.2.2 Synthetic Query Generation . . . 22
  - 2.2.3 Prompt Engineering . . . 23
  - 2.2.4 Contrastive Learning and Hard Negative Mining . . . 23
- 2.3 Fine-tuning an open-source Embedding Model . . . 24
  - 2.3.1 Loss Functions for Embedding Model Fine-tuning . . . 24
  - 2.3.2 Hyperparameters . . . 26
  - 2.3.3 Zero-shot Learning (ZSL) . . . 27
- 2.4 Vector Database . . . 27
- 2.5 Evaluating Retrieval System Performance . . . 28
  - 2.5.1 Metrics . . . 28

2.5.2 Bootstrapping . . . 29

3 Method **31**

3.1 Selecting the Domain-specific Dataset . . . 32

3.1.1 CHARM - Car repair manuals . . . 32

3.1.2 The Sub-datasets . . . 32

3.2 Data Processing . . . 34

3.2.1 Summarizing the Paths . . . 34

3.2.2 Document Chunking . . . 35

3.2.3 Synthetic Query Generation . . . 36

3.3 The Embedding Model . . . 37

3.3.1 Selecting an Embedding-model . . . 37

3.3.2 Fine-tuning the Embedding Model . . . 38

3.3.3 Staged fine-tuning . . . 38

3.3.4 Mining Hard Negatives . . . 38

3.3.5 Knowledge Distillation and Margin Calculations . . . 39

3.3.6 Fine-tuning on hard negatives . . . 39

3.4 Final Prompts . . . 40

3.4.1 Path Summarization Prompt . . . 40

3.4.2 Synthetic Query-Answer Labeling Prompt . . . 40

3.4.3 Prompt for LLM-based Hard Negative Mining . . . 41

3.5 Vector database . . . 42

3.6 Experimental Set-up . . . 42

3.7 Evaluation . . . 43

3.7.1 Evaluation Measures . . . 43

3.7.2 Zero-Shot Learning (ZSL) . . . 44

3.7.3 Standard Testing . . . 44

3.7.4 Benchmarking Against bge-large-en-v1.5 . . . 45

4 Results **47**

4.1 Retrieval Performance . . . 47

4.1.1 Synthetic Test Datasets . . . 47

4.1.2 Human Annotated Test Datasets . . . 50

4.2 Contrastive Learning and Hard Negative Mining . . . 53

5 Discussion **57**

5.1 Reflections . . . 57

5.2 Issues and Limitations . . . 58

5.2.1 The CHARM dataset . . . 58

5.2.2 Choice of Chunking method . . . 58

5.2.3 Hard Negative Mining and Margin MSE Loss . . . 59

5.3 Future Work and Applications . . . 59

5.3.1 Re-labeling passages . . . 59

5.3.2 Increased Computational Power . . . 59

5.3.3 Applications . . . 60

6 Conclusion **61**
6.1 Answering the Research Questions 61
6.2 Final Conclusions 62

Appendix A Division of Work **69**

Appendix B Supporting tables **71**

Appendix C Example Queries **75**
C.1 30 Examples of Human Annotated Queries 75
C.2 30 Examples of Synthetic Queries before Path Summarization 77
C.3 30 Examples of Synthetic Queries after Path Summarization 79

Appendix D Operation CHARM Raw Data **81**

Chapter 1
Introduction

1.1 Background

In the past few years substantial advancements have been made in the field of Large Language Models (LLMs). LLMs excel in their ability to interpret and generate human-like text, and are often trained on large amounts of public data, such as books, web pages, and other internet sources. This diverse training data typically yields a model with broad knowledge, capable of producing impressive results in multiple different fields [1][2].

However, even the most recent and state-of-the-art Large Language Models do not come without drawbacks and flawed instances of generated answers. One commonly occurring problem is LLMs' tendency to produce hallucinations, which is generated text not consistent with real-world facts [3].

Further, for domain-specific usage of LLMs, for example in a professional medical or technical setting, the data suitable for training is often private, and might be consistently updated with new information. In such professional settings, it is of course of utmost importance that hallucinations are avoided: the generated answers must be up to date, and it is crucial that the source of the information can be obtained. If the model does not have access to information needed to answer a certain query, this should be clear in the LLM output rather than the output being hallucinated and incorrect.

Retrieval-Augmented Generation, RAG, is a technique used in natural language processing first introduced in 2020 meant to tackle the challenges discussed above [4]. RAG combine, as the name suggests, retrieval methods with a generative LLM component. Without the use of RAG technology, the LMM is stuck at a specific time determined by the time stamp on the data used for training. With the introduction of RAG, the retrieval part can bring the model into the present.

The idea is that the model has access to a corpus, a large information base, which is updated continuously and thus kept up to date. Rather than retraining and fine-tuning the LLM as new data becomes available, which of course given the computational cost is unsustainable

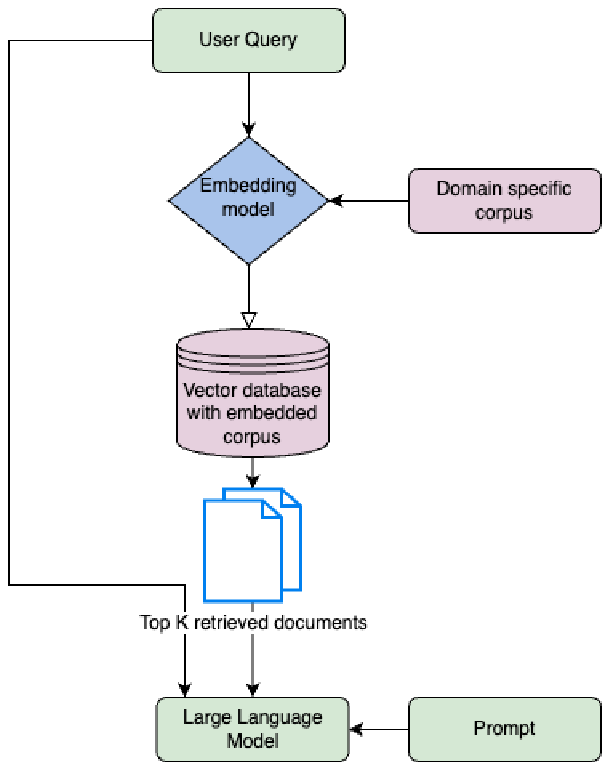


Figure 1.1: The RAG architecture. A vector database, containing corpus embeddings, is used for semantic search. A user query is embedded, and sent to the vector database from which relevant documents are retrieved. The user query, along with the retrieved documents and a system prompt is then used as input to a generative large language model.

if data updates are common occurrences, the retrieval part of the system is leveraged.

The corpus data, consisting of documents or similar textual content, serves as input to an embedding model. This model transforms the data into dense vector representations, which are subsequently stored in a vector database. Similarly, the query issued by the user is embedded, whereupon a search in the vector database is performed to find the vector representations that most closely resemble the users intent. The closest context matches are then retrieved, and together with the user query passed as input to the generative part of the RAG. The end result thus becomes an LLM generated answer complemented by the domain-specific information present in the corpus, effectively reducing the risk of hallucinations and enabling the model to use present information.

Retrieval-augmented generation is a cost-effective and low-risk technique suitable for improving the performance of generative AI models. Further, it is easy to implement and can be constructed as a composite architecture of pre-trained embedding and LLM models. The retrieval sub-system used is crucial for the final overall performance of a RAG system, as this determines the relevancy of the documents used in the text generation step.

For domain specific tasks, optimizing the retrieval performance introduces additional challenges, as the corpus might contain text very dissimilar to the data the embedding model has been pre-trained on. The performance of the embedding model then drops drastically, as the model can not grasp semantic relationships in for example a technical manual.

Further, domain specific corpora are in the majority of cases unlabeled, meaning there are no queries matched to relevant documents. A common approach to tackling this is synthetic data generation, which then also becomes an integral part of improving retrieval systems [5].

Domain adaptation of existing pre-trained dense retrieval systems presents an interesting research objective, where fine-tuning of an embedding model using modern training techniques, Hard Negative Mining, as well as data generation and processing is central for obtaining state-of-the-art performance.

1.2 Research Objective

This thesis aims to investigate and compare a range of different approaches to domain adaptation of dense retrieval systems, with the objective of improving and optimizing retrieval performance. This includes optimizing the training data, the system architecture and the embedding model fine-tuning methods for retrieval performance. Evaluations of the performance were conducted both subjectively in the case of query quality, and quantitatively using standard retrieval system metrics such as recall, normalized discounted cumulative gain, and mean reciprocal rank on both synthetic and human annotated test data.

We limit this research by investigating dense retrieval adaptation for only one domain. Technical car repair manuals provided by Operation CHARM [6] have been used as data resources for implementation, training and evaluation.

1.2.1 Research Questions

Based on the research objective, the following research questions were formulated:

RQ1: Given a large, unlabeled, domain-specific set of documents, what is the most effective set of techniques to achieve good retrieval performance?

RQ2: Is synthetic training data a viable option for fine-tuning an embedding model on a technical domain?

RQ3: Can a small fine-tuned embedding model outperform a much larger "off-the-shelf" embedding model?

1.3 Scientific contribution

This thesis is a contribution to the Natural Language Processing community's ongoing work on how to optimize retrieval performance in highly domain-specific fields, with RAG-systems being an important application. We propose a pipeline that utilizes a Large Language Model for data generation, data processing, and Hard Negative Mining. Further, our proposed system includes a two step fine-tuning process of the embedding model, training first on positive examples and then incorporating contrastive learning.

1.4 Related work

Kexin Wang et al. [7] present a similar approach to ours. In comparison to our approach where we utilize gpt-3.5-turbo to generate queries, the authors instead use the pre-trained T5 encoder-decoder for this task. Similar to our approach the most similar passages are retrieved, using an existing dense retrieval model, which will serve as negative passages. An existing Cross-Encoder is then used to score each (query, passage)-pair and train a dense retrieval model on these generated, pseudo-labeled queries using Margin MSE Loss, similar to our approach. We use an LLM in addition to this to evaluate the relevance of the retrieved passages serving as negatives before scoring them.

The work by Lee Xiong et al. [8] also relates to the work conducted during this thesis. The authors introduce a method to select hard negatives for enhancing dense text retrieval. Similar to our work, the authors propose doing this by employing Approximate Nearest Neighbors (ANN) to efficiently find hard negatives. Embeddings are computed for all text samples, and ANN algorithms are used to identify negatives that are closest to the anchor (query) sample in the embedding space. In comparison to our work, the authors also propose Approximate nearest neighbor Negative Contrastive Learning (ANCE), which updates the ANN index. This means the index is periodically updated in the background without interrupting the training process, ensuring that it remains reasonably up-to-date with the current state of the model.

The paper 'How to Train Your DRAGON: Diverse Augmentation Towards Generalizable Dense Retrieval' [9] presents an approach to curating training data for dense retrieval under the framework of Data Augmentation. The authors mean that synthetic query generation and pseudo-labeling relevancy with a cross encoder are sub-optimal methods for improving dense retrieval tasks, and instead propose using multiple small retrieval models rather than one strong Cross-Encoder for label relevancy supervision. In the paper, a base embedding model is used to retrieve a list of candidate passages for each query. Several different embedding models are then utilized to retrieve additional relevant passages for the same query. The passages retrieved by all models are then combined and the ones that meet a certain relevance threshold are labeled as positives. The process is then performed iteratively to refine the positive labels, with the aim to avoid false negatives and improve contrastive learning. Further, training data queries are generated in a divers manner, generating both synthetic questions and using cropped sentences from the passages. This is a more cost-efficient pipeline compared to our work, as we use a proprietary LLM for label supervision. In addition, the authors use a much larger number of training queries for fine-tuning, 28 000 000 compared to our largest training dataset consisting of 15 000 queries. The findings presented by the authors are in line with our observations; improving performance on tasks the model has been trained for, using for example knowledge distillation for relevancy labeling, often comes at the expense of Zero-Shot learning effectiveness. An increase in training data diversity instead presents as an efficient method for improving retrieval in Zero-Shot learning settings.

Hansi Zeng et al. present in the paper 'Curriculum Learning for Dense Retrieval Distillation' [10] an approach to improve knowledge distillation. The authors build on the framework of using a Cross-Encoder re-ranker by implementing a curriculum in which the difficulty level of the training data produced by the teacher model is increased as the student embedding model learns. Similar to our work, dense embeddings and a Cross-Encoder model are used for optimizing knowledge intensive retrieval.

1.5 Outline

This thesis is structured into six main chapters, where chapter 1 provides the background and motivations for the conducted research, as well as a section on related work, the research objective and the research questions. In chapter 2, the theory and concepts behind the models, techniques and methods used are described. The methods used in the thesis pipeline are presented and explained in chapter 3, with an overview provided in figure 3.1. Chapters 4, 5, 6 include the results obtained, a discussion of the findings, suggestions for future work, and the project conclusions involving circling back to the research questions. In appendices B, C, and D additional results and further details regarding the data are presented.

Chapter 2

Domain Adaptation of Dense Retrieval Systems

The retrieval part is central when developing a RAG-system, especially when intended for domain specific use cases. At the heart of the retrieval system lies the embedding model, which generates semantic vector representations of the corpus. There are numerous pre-trained embedding models available for use, both open-source and proprietary. These models can be fine-tuned on domain-specific data, with the intent of optimizing their performance in the same domain. To successfully perform this domain adaptation it is crucial to use high quality training data, suitable hyperparameters for training and a comprehensive evaluation method. Below follows a chapter which provides an introduction to the theory behind these concepts.

2.1 Embedding models

Embedding models refer to a category of machine learning models designed to transform complex data into numerical values interpretable for a computer. These models are essential in modern machine learning, and are integral for performing a variety of different tasks such as text generation, semantic search, and computer vision.

The transformation is performed by taking high dimensional input, such as images, human written text etc., and via the model converting this data into lower dimensional *dense* vectors, where the components of the vector can be thought of as features of the input. These dense vectors are called *embeddings*, see figures 2.1 and 2.2. Dense retrieval is essentially a retrieval system where data, such as text, is represented as dense vectors. With modern model architectures such as transformers, and particularly sentence transformers for natural language processing tasks, it is possible to capture the semantic meaning, textual relationships and context within the sentence embeddings (dense vectors). This allows for semantic processing of text, where synonyms ideally are encoded similarly. With this, we are no longer

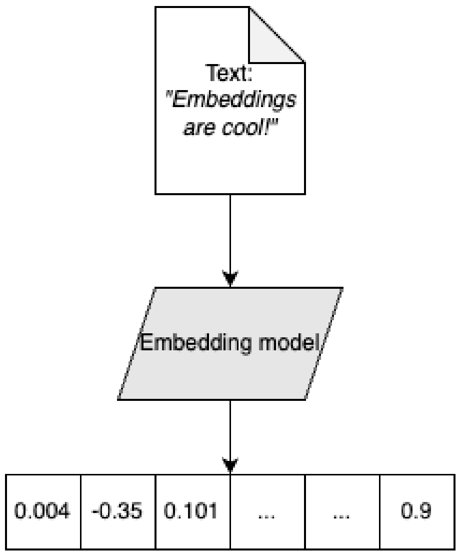


Figure 2.1: Embedding model input and output.

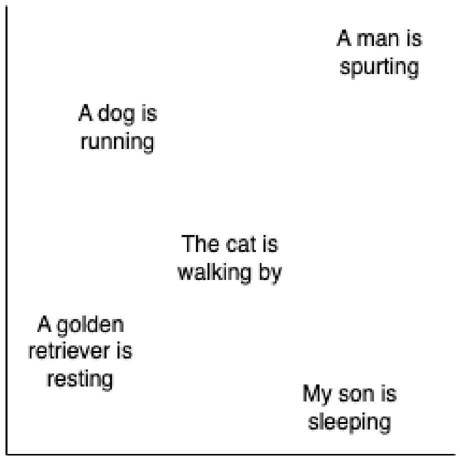


Figure 2.2: 2D representation of an embedding vector space.

limited to lexicon search (sparse retrieval), and can expect the model to interpret text in a human-like manner [5][11]. These types of embedding models for natural language processing were used for implementing a retrieval system during this thesis. Section 3.3.1 provides detailed information on the specific models selected.

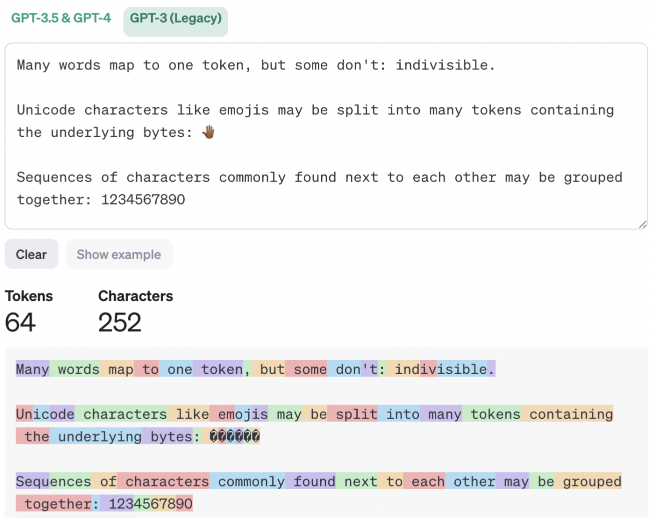
Embedding models used for natural language processing perform tokenization before generating the embeddings. Tokenization is the process of breaking down input sequences, such as a text passage, into smaller units called tokens, often words or parts of words, for which an embedding can be generated. Depending on the embedding model, different tokenizers are used, and the same sentence can be tokenized in several ways. Identical sentences could thus be transformed into different numbers of embeddings by different tokenizers. An example of this is shown in figures 2.3 and 2.4, where the tokenizer used in GPT 3 divides the text in more tokens than the tokenizer for GPT 3.5 and GPT 4 [1]. Section 3.2.2 describes how this concept was taken into account when processing data.

2.1.1 The transformer

The transformer architecture was first introduced in 2017 [12], and has since essentially shifted the machine learning world from Recurrent Neural Networks (RNNs) to self-attention feed forward networks when dealing with sequential data.

Transformers not only boost the performance of Natural Language Processing (NLP) tasks, but are also more computationally feasible, as they can process the input data in parallel, as opposed to recurrent networks. One of the issues with RNNs is that they become quite inefficient when elements, such as words or tokens, are far apart in the input sequence, as the recurrent connections result in a computational chain where earlier inputs are subdued when the chain grows.

Transformers overcome this challenge by utilizing a multi-headed self-attention mech-



```
Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many tokens containing
the underlying bytes: 🤚🏾

Sequences of characters commonly found next to each other may be grouped
together: 1234567890
```

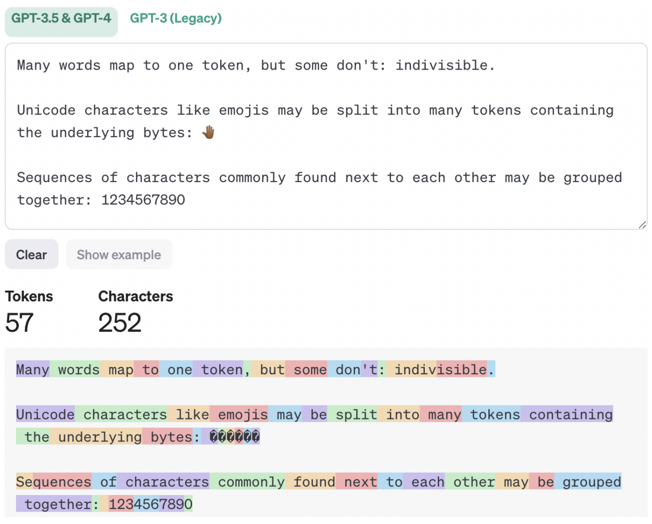
Tokens 64 Characters 252

```
Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many tokens containing
the underlying bytes: ������

Sequences of characters commonly found next to each other may be grouped
together: 1234567890
```

Figure 2.3: Example tokenization with the tokenizer used in GPT 3.



```
Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many tokens containing
the underlying bytes: 🤚🏾

Sequences of characters commonly found next to each other may be grouped
together: 1234567890
```

Tokens 57 Characters 252

```
Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many tokens containing
the underlying bytes: ������

Sequences of characters commonly found next to each other may be grouped
together: 1234567890
```

Figure 2.4: Example tokenization with the tokenizer used in GPT 3.5 and GPT 4.

anism (figure 2.5), where each word or token in the sequence is independently processed together with all other words or tokens, whilst avoiding the feed-back connections.

Before the attention mechanism is used, an encoder generates numerical vector embeddings with fixed-sized dimensions for each token in the input text. It is these vector embeddings that are used to compute the attention scores, which are a measure of how much each word's interpretation should be impacted by other words in the input. The attention scores are calculated via a number of mathematical operations, where similar embeddings generate a higher score.

The original transformer model is built on an encoder-decoder architecture, where the foremost substructure is used to generate numerical vectors representative of the input, with deep contextual understanding. The decoder instead mirrors the encoder, and can, based on the encoded output it receives, generate new text. An example usage of such a system is translation, where a passage in English can be used as input to the transformer which then outputs the same passage in another language.

For retrieval tasks, the generative part of the model is usually not of interest. Instead, the semantically rich vector embeddings can be used directly and put into a vector space, where semantic search can be performed based on metrics such as cosine similarity.

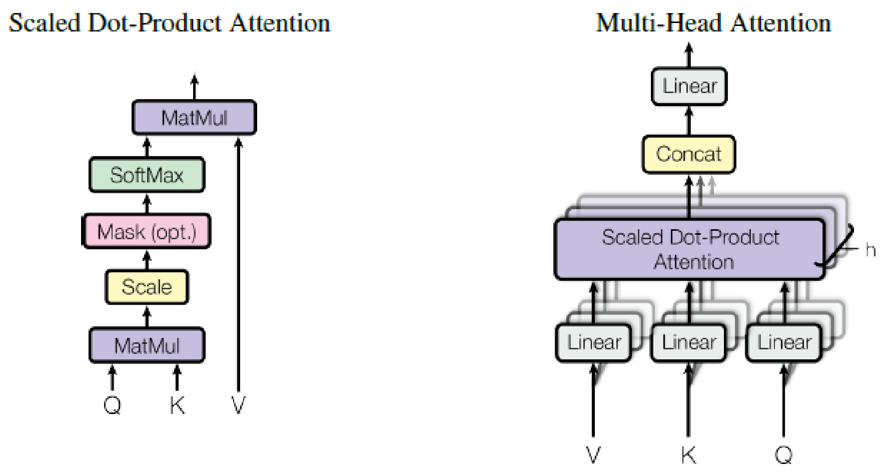


Figure 2.5: (right) Scaled Dot-Product Attention. (left) Several parallel-running attention layers creating a Multi-head Attention mechanism.

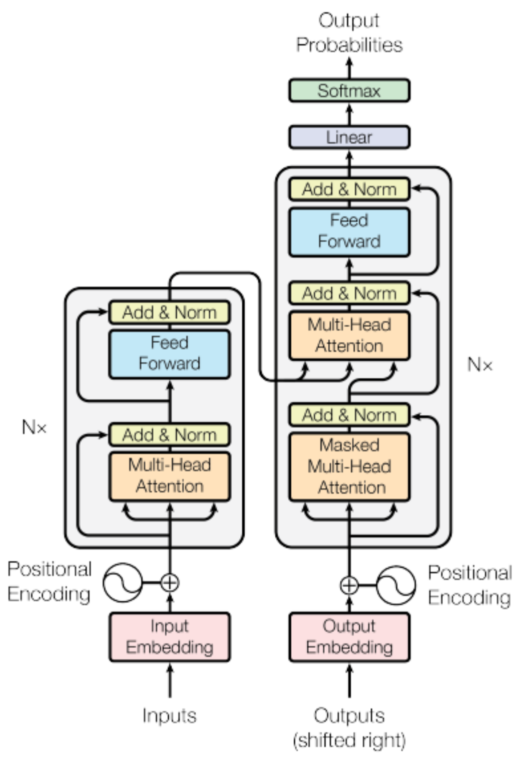


Figure 2.6: The original Transformer architecture.

2.1.2 Bidirectional Encoder Representations from Transformers (BERT)

A model which extends and optimizes the transformer architecture for embedding generation is BERT, short for Bidirectional Encoder Representations from Transformers. As the goal with BERT is to create or generate a language model, and not actually output text, it only uses the encoder component of the transformer architecture. BERT is very similar to the encoder layer of the original transformers, where the self-attention mechanism is utilized allowing for bidirectional, or rather non-directional, input reading instead of sequential processing. Differences between BERT and the encoder part of the original transformer includes some hyperparameter changes, an increased number of self-attention heads as well as a masking mechanism [13].

2.1.3 Sentence Transformers

BERT excels in generating contextual embeddings for words that capture semantics such as synonyms, and produces impressive results in tasks where word granularity is important. However, for document retrieval tasks, it is preferable to work with sentence embeddings, where a single vector embedding is able to represent the semantic and contextual content of a complete word sequence rather than just one token or word. This approach simplifies the retrieval process by allowing comparisons between entire text passages and queries rather than individual words, making semantic search feasible.

Sentence transformers [14] is a Python framework built on BERT for producing state-of-the-art sentence and text embeddings. The models presented in section 3.3.1 are sentence transformers, and are used within this framework.

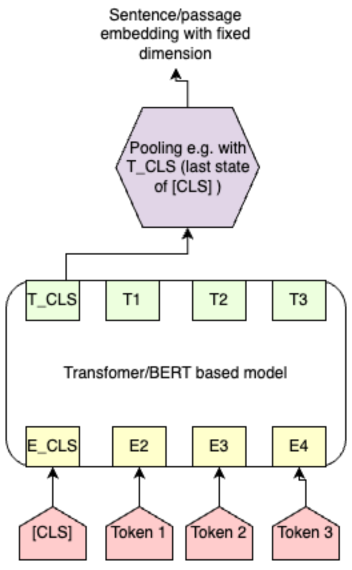

Sentence/passage embedding with fixed dimension
Pooling e.g. with T_CLS (last state of [CLS])
T_CLS
T1
T2
T3
Transfomer/BERT based model
E_CLS
E2
E3
E4
[CLS]
Token 1
Token 2
Token 3

Figure 2.7: The pooling mechanism in Sentence Transformer models enables capturing the semantic meaning of an entire text passage in a single embedding.

The sentence embeddings are created based on the word embeddings generated by a

BERT-model. This is achieved through pooling of the word embeddings. The pooling strategy used varies between different embedding models, and common techniques include max or mean pooling. Max pooling is performed by taking the maximum value of each dimension of the word embeddings and using these values for the sentence embedding, while mean pooling involves taking the element-wise arithmetic mean of the word-level embeddings. Another pooling approach, utilized in the models presented in section 3.3.1, is to use the last hidden state of the "[cls]" token, which is a token added at the beginning of every input sequence in models like BERT. As the model processes the input, the token accumulates and the last state captures a contextual representation of the entire text sequence, making it a suitable embedding for the input passage [15].

2.1.4 Bi-Encoders

A Bi-Encoder is a specific type of embedding model structure designed to compare the similarity between two text sequences. It is essentially an architecture consisting of two identical embedding models, for example two Sentence Transformers, where each model takes one text sequence as input. The model then outputs the respective text embeddings, whereupon the cosine similarity for the two embeddings is calculated and passed as the final output [15]. The cosine similarity for two vectors **A** and **B** is given by:

$$\text{cosine similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||\ ||\mathbf{B}||} \tag{2.1}$$

2.1.5 Cross-Encoders

Cross-Encoders are a class of models supported by the Sentence Transformers library [14]. Although Cross-Encoders are not embedding models in the sense that they do not output any embeddings, their architecture contains many of the components present in embedding models. In contrast to a Sentence Transformers based embedding model, Cross-Encoder takes as input two text sequences simultaneously which are then processed together. Instead of producing embeddings of the texts, the Cross-Encoder returns a similarity score between 0 and 1, where a high score indicates high text similarity between the two text sequences.

The model architecture makes use of Bidirectional Encoder Representations from Transformers (BERT), which processes both sequences together and then outputs encoded data which is passed as input to a classifier. The classifier then outputs a similarity score.

Cross-Encoders achieve better performance than Bi-Encoders in finding semantic similarity in text pairs, but using this for retrieval is not scalable. As no embeddings are produced, a similarity search would have to be performed by sending each search query into the model together with each passage in the database. This would mean that for every single query there would be as many calls to the Cross-Encoder as there are documents in the database [15].

Instead, Cross-Encoders are more commonly used for re-ranking, and they are often referred to as re-rankers in these contexts. An example usage would be to retrieve the top 100 documents in a database for a query using an embedding model. The retrieved documents are then much less in number than the documents in the database, and it is feasible to calculate a similarity score between the query and all retrieved documents. As the Cross-Encoder provides more accurate similarity scores, the documents can then be re-ranked in relevancy based

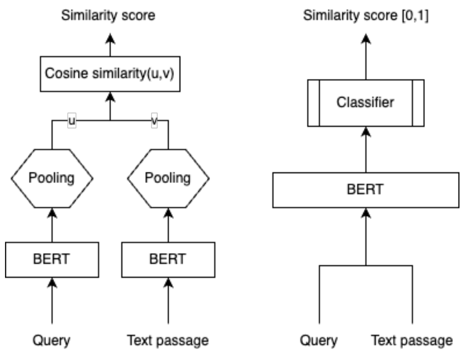


Figure 2.8: The Bi-Encoder compared to the Cross-Encoder architecture.

on the knowledge of the Cross-Encoder, which would in theory then give a better relevancy ranking [15].

Another common use case for Cross-Encoders is **knowledge distillation**. The Cross-Encoder is then used as a teacher model for a student embedding model. The less accurate embedding model learns from the Cross-Encoder by using Cross-Encoder outputs as pseudo-labels for passage relevancy in the training data [7]. This topic is discussed more in depth in the fine-tuning section 2.3. Further, the employment of knowledge distillation during this project is detailed in section 3.3.5.

2.2 Data Processing and Generation

2.2.1 Parsers and Chunking

Embedding models are limited in how many tokens (words or parts of words) they can process per call. This number varies greatly between different embedding models, where some models only take a maximum of 512 tokens while others can process more than 30 000 tokens per request [11].

When working with a large corpus consisting of documents of different lengths, it is crucial to ensure that no passage sent to the embedding model exceeds the specified maximal number of tokens, as this will result in the later part of the document not being included in the generated embedding. To achieve this on a large scale, text parsers performing so called chunking can be used.

Parsers take the input documents and return smaller passages, with the text length adjusted to fit within the context window for the embedding model. There are a plethora of different chunking approaches and parsers to choose from. Selecting the right parser class for a specific task can be tricky, and is heavily context based. Below follow short descriptions of different approaches to chunking.

A token splitter is a basic type of parser, which strictly divides the text based on the token count. From this follow predictable chunk sizes, as all passages except the last one from a document will be precisely as long as the specified number of tokens. However, this also means that sentences or even words will be cut short, and divided into different passages [16]. This is often not ideal, as context is lost.

Sentence splitters try to overcome this problem by prioritizing keeping sentences together above generating passages with a precise token length. These types of parsers are commonly used, as they allow for more reasonable chunking without increasing either expenses or computational cost [17].

A recently proposed chunking technique is semantic text splitting [18]. The idea behind this method is to integrate an LLM in the chunking step. The LLM evaluates documents and identifies main themes in the text, whereupon the text splitting is performed based on what semantically makes sense. To preserve as much semantic meaning as possible, each passage is composed of the largest semantic unit that fits within the specified maximum token length [19].

Both a sentence splitter and a semantic text splitter were used "off the shelf" when performing text chunking, as will be described in section 3.2.2.

2.2.2 Synthetic Query Generation

When dealing with high volumes of unlabeled domain-specific data, fine-tuning of an embedding model and evaluating the performance are not straightforward tasks. In a RAG-system setting, the retriever should return documents that are relevant for the user query sent to the system. When teaching the embedding model to generate embeddings that position queries close to text passages answering them in the vector database, it is natural to use training data formatted as $\boldsymbol{Query}$ $(\boldsymbol{Q})$, $\boldsymbol{RelevantPassage}$ $(\boldsymbol{P+})$, where the relevant passage serves as the label for the query. Doing this manually, i.e. to have domain experts annotating each passage with a relevant query, is both extremely expensive and time-consuming for large datasets. It is, in most cases, simply not feasible.

Synthetic query-answer labeling is a technique, used during this project to generate training, validation, and test data as described in section 3.2.3, that can transform an unlabeled corpus into useful training data with the format $(\boldsymbol{Q}, \boldsymbol{P+})$ by utilizing an LLM. Each text passage in the corpus is sent to the LLM together with a tailored prompt, and a query that can be answered by the passage is generated. The prompt, as well as other parameters such as temperature have a large impact on the quality of the resulting data. The temperature determines, in the context of generative LLMs, the randomness of the generated output. A lower temperature value (closer to 0) results in more deterministic output text, as the standard deviation of the probability distribution for the next generated token is proportional to the selected temperature. Experimenting with different prompts and hyperparameters is thus a large part of the work required to successfully employ synthetic query-answer labeling, where a good synthetic query generation approach needs to result in the use of synonyms and produce questions that contain words or phrases not occurring in the original passage [20][21].

Both open-source models and proprietary LLMs such as GPT are viable options for this task, and can be determined based on data complexity, requirements and cost limits. Generated synthetic query labels can also be used for generating validation and test data, although with some caution as human annotated test datasets reflect reality better and are preferable. Overall, the use of synthetic query-answer labeling allows for scalability and robustness in domain adaptation of retrieval systems. The integration of this technique makes RAG a viable solution in domain-specific areas where manual annotation would not have been an option, leading to improvements in content relevance, quality, and model performance across specialized data landscapes [22][7].

2.2.3 Prompt Engineering

Prompt Engineering is an important field in natural language processing. It is in short the method used for efficiently communicating with language models, instructing them in their tasks. Basic techniques used in prompt engineering include role-based prompting, giving task context, providing examples of what the output should look like, asking the model for a "chain-of-thought", and writing clear instructions that do not leave room for misunderstandings. Prompting often becomes an iterative process, where slight changes to the prompt are made gradually in order to reach the requirements set for the generated outputs [23]. An example prompt for synthetic query generation is given in section 3.2.3. Prompt engineering was utilized for multiple different tasks during this project, described further in sections 3.2.1, 3.2.3, 3.3.4, and 3.4.

2.2.4 Contrastive Learning and Hard Negative Mining

Conventionally, the training data used to train or fine-tune a model consists of examples, such as queries, and respective labels, such as passages. This teaches the model what is correct: it learns to embed $(Q, P+)$ such that they are close to each other in a vector space.

Contrastive learning is a training approach where the model is not only taught which passage is positive for a query, but also which passages are negative, i.e. not relevant for the query. With a supervised contrastive training approach, training data not only has to be labeled with relevant passages, but also with irrelevant ones. The training data should then have the following triplet format: $(Q, P+, P-)$. From this, the model learns to push Q and $P+$ closer together in the vector space, whilst Q and $P-$ are forced further apart [24]. In table 2.1 an example of such a triplet is presented.

Query	What is the capital of France?
Positive Passage	Paris is the capital of France.
Negative Passage	The Eiffel-tower is a Parisian tourist attraction.

Table 2.1: Triplet training example for Contrastive Loss.

The model does not need to be trained on all examples that are negative for a query. Most of the negative passages are already far away enough from the query and are easy for the model to determine as non-relevant, and it is more efficient to focus on the *hard* negative examples. Hard negatives are defined as negative examples that are closer given a selected distance to the query than the positive example.

$$Dist(Q, P-) < Dist(Q, P+) \tag{2.2}$$

By including hard negative examples in our training data, see section 3.3.6, the model decreases its loss not only by bringing $(Q, P+)$ closer together, but also by increasing the distance between $(Q, P-)$ so that the positive passage lies closer to the query than the negative passage.

Contrastive learning on negative examples that are not closer to the query than the positive example, but that are still not far enough away given their irrelevancy, can also be useful for improving performance. Such training samples, which were used during this project as will be explained in section 3.3.4, are referred to as semi-hard negatives, and they are defined in the following way:

$$Dist(Q, P-) < Dist(Q, P+) + margin \tag{2.3}$$

where the $margin$ is a parameter for certain loss functions that still penalize negatives with a distance to the query not larger than that of the positive plus the margin [25][7].

Hard Negative Mining is the process of finding the hard or semi-hard negative examples in order to utilize them in the training data. There are different approaches to this, and it becomes a complex task when working with an unlabeled corpus. The approaches investigated during this projects were novel and adjusted to the specific task, and will be explained in detail in section 3.3.4.

2.3 Fine-tuning an open-source Embedding Model

Many open-source embedding models have been trained on billions or even hundreds of billions of tokens. This training process is referred to as pre-training, and is often performed on one or multiple large diverse training datasets. The training data is usually unlabeled, resulting in this stage being unsupervised or self-supervised. During pre-training, the model learns general structures, features and patterns of text [26].

In order to domain-adapt such pre-trained open-source embedding models, an additional training stage can be performed: fine-tuning. Fine-tuning is ideally performed on a smaller, labeled, dataset within the domain of the intended use, which is in accordance with our methods presented in sections 3.3.2 and 3.3.6. This process adapts the model to a new specific task, and improves the performance in that domain [27]. During fine-tuning, a separate validation dataset representative of the training data should be used to optimize hyperparameters. Additional data not seen or used during training should be saved for testing, to avoid data leakage and get an accurate evaluation. This approach has been taken, and will be further detailed in sections 3.1.2 and 3.6.

2.3.1 Loss Functions for Embedding Model Fine-tuning

The Sentence Transformers library provides a large selection of loss functions for fine-tuning of embedding models [28]. Which loss functions are suitable is heavily dependent on the task and the data. Below, three loss functions appropriate for domain adaptation, used according to what will be described in sections 3.3.2, 3.3.3, 3.3.5, and 3.3.6, are presented.

Multiple Negatives Ranking Loss is the default loss function for fine-tuning with the Sentence Transformers library. This loss function expects as input (Q, $P+$)-pairs, with no labels. During training, all passages in the batch that are not labeled as the positive example are used as negative examples for the queries, i.e. if P_i is the positive example for Q_i, all P_j where j≠i are considered negative. From this, we get that the batch size impacts the number of training samples, as more negative training examples are used for each query if there are more samples in the batch. Multiple Negatives Ranking Loss often yield better performance when the batch size is increased [29].

The mathematical representation of this loss function is as follows:
Let s(Q_i, P_i) denote the cosine similarity between query and passage embeddings. The scaling of the similarity score is denoted as:

$$s'(Q_i, P_j) = \text{scale} \cdot s(Q_i, P_j) \tag{2.4}$$

For each query Q_i , the softmax over all similarities is computed as:

$$P(P_i \mid Q_i) = \frac{\exp(s'(Q_i, P_i))}{\sum_{j=1}^{n} \exp(s'(Q_i, P_j))} \tag{2.5}$$

The loss for each query-positive-passage pair (Q_i, P_i) is the negative log-likelihood:

$$\text{Loss}(Q_i, P_i) = -\log(P(P_i \mid Q_i)) \tag{2.6}$$

Summing over all pairs in the batch, the total loss is:

$$\text{Total Loss} = \frac{1}{n} \sum_{i=1}^{n} -\log\left(\frac{\exp(s'(Q_i, P_i))}{\sum_{j=1}^{n} \exp(s'(Q_i, P_j))}\right) \tag{2.7}$$

Online Contrastive Loss requires training data on the format $(Q, P+, label = 1)$ or $(Q, P-, label = 0)$. This loss function calculates the distance between each provided $(query, passage)$ sample, and calculates the loss only on the hard negatives (negatives that are close) and hard positives (positives far apart). The default distance metric used for calculations is the cosine distance, but this can be altered to a different metric by passing it as a parameter to the loss function. This loss function also has a threshold parameter, where negative samples are allowed at distances larger than the threshold from the query. The threshold value selected can highly impact the success of the fine-tuning [30]. The Online Contrastive Loss is given by:

$$\text{Online Contrastive Loss} = \frac{1}{n} \sum_{i=1}^{n} \left[y_i \cdot d_i^2 + (1 - y_i) \cdot \max(0, m - d_i)^2 \right], \tag{2.8}$$

where y is the label (0 or 1), m is the predefined threshold, and d is the distance between the query embedding and the passage embedding.

Margin Mean Square Error loss is another loss function used for contrastive learning. It expects triplets as input, $(Q, P+, P-)$ labeled with a margin float value. The margin is commonly calculated using a Cross-Encoder as a teacher model, where the margin is the difference between the Cross-Encoder similarity score for $(Q, P+)$ and $(Q, P-)$. The loss is then calculated as the mean square error between the margin and the embedding cosine similarity difference between the positive and negative example. This is a form of knowledge distillation where the idea is that the Cross-Encoder is better at determining passage relevancy for a query, and that the embedding model should learn to generate embeddings in line with the Cross-Encoders similarity judgment. If the Cross-Encoder similarity score is close to the embedding model's similarity scores, then that sample incurs a small loss [31]. Mathematically, this can be represented as follows:

$$\begin{aligned} \text{MarginMSELoss} = \text{MSE}\,(&|\text{embed\_sim(Q, P+)} - \text{embed\_sim(Q, P-)}|\,, \\ &|\text{crossencoder\_sim(Q, P+)} - \text{crossencoder\_sim(Q, P-)}|) \end{aligned} \tag{2.9}$$

2.3.2 Hyperparameters

Hyperparameters include all non-trainable parameters in the fine-tuning stage. Optimizing these is an important part of training, and the optimization is typically performed based on the validation loss. To streamline this process, there are AI developer platforms, such as Weights and Biases [32], that provide tools for tracking and visualizing machine learning experiments. With such tools, sweeps can be performed [33]. Sweeps are a form of grid search, where all specified combinations of hyperparameters are tested and the best performing model can be chosen.

Below follows an overview of important hyperparameters when fine-tuning an embedding model, that were optimized using sweeps as will be further detailed in section 3.6, with a short explanation of their meaning.

Number of Epochs is a hyperparameter that refers to the number of complete passes through the entire training dataset during the training process. In each epoch, all training samples are used to update the weights in order to minimize the loss function. Increasing the number of epochs allows the model to learn more thoroughly from the data, with the risk of overfitting to the training data. Setting the number of epochs to a lower number may instead lead to underfitting, where the complexity of the data isn't adequately captured. Finding the optimal number of epochs usually involves minimizing the validation loss.

Batch Size refers to the number of training samples in each batch. The weights are updated by minimizing the loss function computed over all samples in one batch. A larger batch size hence means fewer weight updates per epoch, but each update incorporates more samples. The batch size can significantly impact training efficiency and performance, especially for certain loss functions such as Multiple Negatives Ranking loss. Larger batches can stabilize the training process and improve convergence, but they also require more memory and computational resources.

Evaluation Steps determines a fixed interval at which the model's performance is evaluated on the validation data. For example, if evaluation steps are set to 50, the model is evaluated after every 50:th batch is processed. Evaluating the model periodically helps monitor the training process and detect overfitting.

Learning Rate is a crucial hyperparameter that controls the amount by which the model's weights are updated during training. It determines how big of a step is taken in the negative gradient's direction. Setting the learning rate to a large value allows the model to converge faster, with the risk of overshooting the optimal solution. A lower learning rate allows for more fine adjustments, but may result in slow convergence or the model getting stuck in a local minima. When setting the learning rate the optimization algorithm used should be taken into account, as vastly different learning rates are suitable depending on the optimizer.

Distance Function is the method used to calculate the distance between embeddings when training an embedding model. Common distance functions include the cosine distance (1-cosine similarity) and the euclidean distance.

Loss Function determines what should be minimized during training. Some common examples for embedding model loss functions are discussed in section 2.3.1.

Optimization Algorithm is the algorithm used for the training optimization process. These algorithms include how the model weights are adjusted to minimize the loss. A commonly used optimizer for training and fine-tuning embedding models is *AdamW* [34], a variant of the Adam optimizer which incorporates weight decay corrections.

2.3.3 Zero-shot Learning (ZSL)

Zero-shot learning (ZSL) is a machine learning scenario in which an AI model is trained to recognize and categorize objects or concepts without having seen any examples of those categories or concepts beforehand. ZSL facilitates the transfer of knowledge from related domains [35]. This is a beneficial method when evaluating AI models on domain specific tasks. By training the model only on samples and classes that are unseen in the test data, the model's performance within the domain can be accurately evaluated. ZSL evaluation was conducted for a number of test datasets, as will be described in section 3.7.2.

2.4 Vector Database

When dealing with large data volumes, such as a text corpus, numerous embeddings representing all the data are generated. Finding the most similar embedding, i.e. the relevant document, to a query then means comparing the embedding distance between the query and all corpus embeddings. One way to streamline this process, used in the system implemented during this project as will be described in section 3.5, is to use a vector database. Vector databases are databases specialized for storing and efficiently retrieving dense, high dimensional, vector data. Together with the embedding model, the vector database serves as the final, ready-to-use, retrieval system, visualized in figure 2.9.

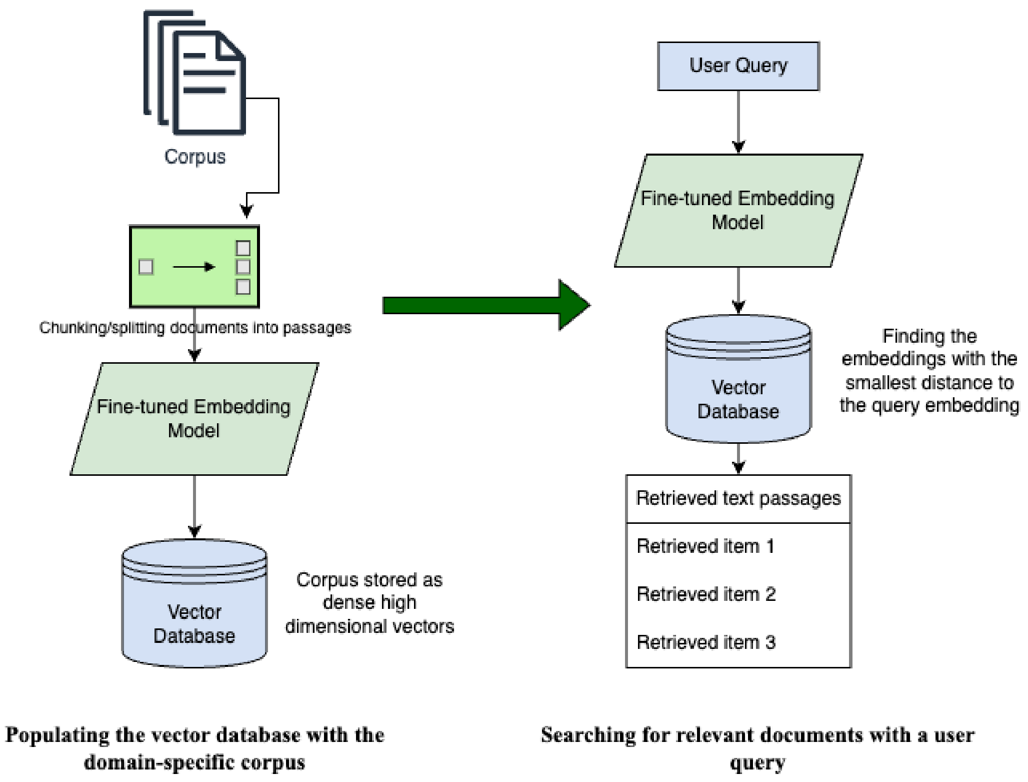

Corpus
Chunking/splitting documents into passages
Fine-tuned Embedding Model
Vector Database
Corpus stored as dense high dimensional vectors
Populating the vector database with the domain-specific corpus
User Query
Fine-tuned Embedding Model
Vector Database
Finding the embeddings with the smallest distance to the query embedding
Retrieved text passages
Retrieved item 1
Retrieved item 2
Retrieved item 3
Searching for relevant documents with a user query

Figure 2.9: (left) Storing corpus embeddings in the vector database. (right) A search query is sent to the system and relevant documents are retrieved from the vector database.

There are a lot of vector database alternatives available for use. An example of such is the open-source library Faiss [36], developed primarily by the fundamental AI research team of Meta. Vector Database libraries usually contain algorithms that can efficiently perform similarity search among vectors of any size.

Similarity search is performed by the vector database building a data structure from a given set of vectors with dimension d. When a new vector, for example an embedded query, of the same dimension is added, the following operation is performed:

$$j = \arg\min_i ||x - x_i||, \tag{2.10}$$

where $||x - x_i||$ is the distance (often the Euclidian or Cosine distance) between query embedding x and corpus embeddings x_i.

This is essentially a K-nearest neighbor search, where the number of nearest neighbors returned, K, is a parameter that can be adjusted according to wants and needs. Motivations behind the parameter choice for the implemented system will be presented in section 3.5.

2.5 Evaluating Retrieval System Performance

When designing an information retrieval system, it is critical to be able to evaluate it with insightful metrics. Realistic results are achieved when evaluation is performed on data not used in training or tuning of the model, and separate test data should be kept for this purpose. An important parameter when evaluating performance is the number of documents retrieved per search query. Depending on the intended use and the required precision, we can choose to retrieve only the document deemed most relevant by our system, or the 10, 100 etc. most relevant ones. It is of course a harder task for the system if the number of retrieved document is decreased, which is reflected in the evaluation scores.
In information retrieval, the system should return relevant documents to a search query, where the order of the returned documents are based on relevancy. The evaluation scores should thus be higher if a relevant document is retrieved as the first compared to the tenth one.
Retrieval evaluation can be categorised into two subgroups; order-aware and order-unaware metrics. Order-aware metrics return different scores based on the ranking of the documents, whereas order-unaware metrics only measure whether a relevant document is retrieved or not [27].

2.5.1 Metrics

This section covers some of the most common metrics used when evaluating information retrieval, including both order-aware and order-unaware examples. The metrics presented below were used for evaluation of retrieval performance of our implemented system, with some additional adjustments which will be further explained in section 3.7.1.

Mean Reciprocal Rank (MRR) is an order-aware metric, calculated as:

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\text{rank}_i} \tag{2.11}$$

where $|Q|$ is the total number of queries, and rank_i is the rank position of the first relevant result for the i-th query.

Normalized Discounted Cumulative Gain (NDCG) is another order-aware metric calculated as:

$$\text{NDCG} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{\text{DCG}_i}{\text{IDCG}_i} \tag{2.12}$$

where $|Q|$ is the total number of queries, DCG_i is the **Discounted Cumulative Gain (DCG)** for the i-th query, and IDCG_i is the **Ideal Discounted Cumulative Gain (IDCG)** for the i-th query.

DCG is calculated as:

$$\text{DCG}_i = \sum_{j=1}^{p} \frac{2^{\text{rel}_{ij}} - 1}{\log_2(j+1)} \tag{2.13}$$

where p is the number of results, and rel_{ij} is the relevance score of the j-th result for the i-th query.

IDCG is calculated as:

$$\text{IDCG}_i = \sum_{j=1}^{p} \frac{2^{\text{rel}^*_{ij}} - 1}{\log_2(j+1)} \tag{2.14}$$

where rel^*_{ij} is the relevance score of the j-th result in the ideal ordering for the i-th query.

Hit-rate is an order-unaware metric that measures the percentage of queries for which a relevant document was retrieved. The hit-rate is calculated as follows:

$$\text{Hit Rate} = \frac{|R \cap H|}{|R|}, \tag{2.15}$$

where R is the total number of relevant items and H is the total number of items retrieved.

Another standard metric used for evaluating information retrieval systems is Recall. Recall is calculated by dividing the number of true positives with the number of true positives and false negatives.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{2.16}$$

2.5.2 Bootstrapping

A dataset, although consisting of many samples, is usually only a subset of the entire data population: the dataset is in itself a sample. Even when evaluating on a test set where data leakage has been avoided, the measured evaluation metrics are not statistically certain, as the actual performance is only approximated from the data population sub-sample.

Bootstrapping, as shown in figure 2.10, is a simple and straight-forward method that allows for estimation of the sampling distribution. A bootstrap sample is generated from a dataset of size N by randomly selecting M samples with replacements from that dataset. The bootstrap sample can hence contain duplicate samples, while other samples are "out-of-bag", i.e. not present. This process is repeated many times, resulting in multiple new datasets containing different samples. Evaluation and metrics calculation is then performed on all generated datasets, and a histogram of the bootstrap means can be created. The histogram

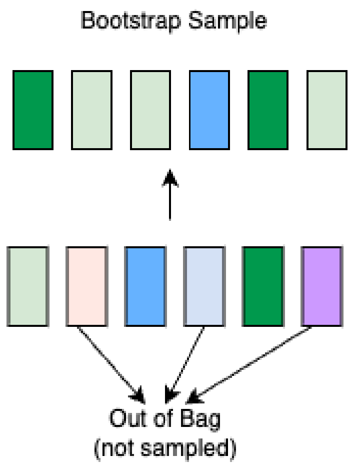


Figure 2.10: A Bootstrap sample is generated by randomly sampling with replacement.

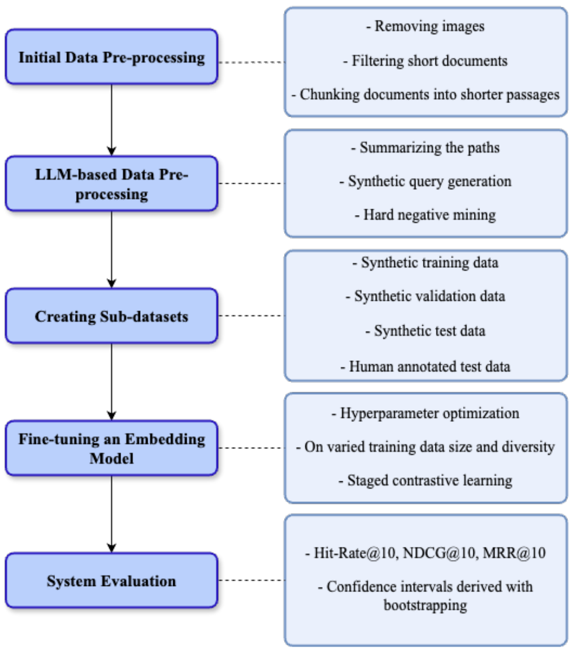
provides an estimate of the sample distribution, and can be used to derive confidence intervals of the measured results. For example, the 95% confidence interval can be obtained by finding the 2.5:th and 97.5:th percentiles of the bootstrap distribution. By doing this, an estimate of the uncertainty around the results is provided which helps determine statistical significance [37].

Bootstrapping for confidence interval generation was used when evaluating our retrieval system, where further details of the implemented approach will be described in section 3.7.1.

Chapter 3

Method

This chapter details the methods and techniques employed during this thesis. The chapter is structured based on the final pipeline used for domain adaptation of the retrieval system. However, it should be noted that the work was iterative, and involved refining the individual methods over time. Figure 3.1 displays an overview of the main steps and methods utilized for creating the final dense retrieval system.



Initial Data Pre-processing
- Removing images
- Filtering short documents
- Chunking documents into shorter passages
LLM-based Data Pre-processing
- Summarizing the paths
- Synthetic query generation
- Hard negative mining
Creating Sub-datasets
- Synthetic training data
- Synthetic validation data
- Synthetic test data
- Human annotated test data
Fine-tuning an Embedding Model
- Hyperparameter optimization
- On varied training data size and diversity
- Staged contrastive learning
System Evaluation
- Hit-Rate@10, NDCG@10, MRR@10
- Confidence intervals derived with bootstrapping

Figure 3.1: Overview of methods used in the project pipeline to successfully adapt a dense retrieval system to car repair and diagnosis manuals from the CHARM corpus.

3.1 Selecting the Domain-specific Dataset

In order to explore techniques for domain adaptation of a retrieval system, a domain-specific dataset used for implementations and experiments had to be defined. Requirements for the dataset included highly technical or concept heavy unlabeled text, and enough text for training, validation and testing. The embedding model was also investigated to ensure that it had not been pre-trained on the selected dataset or similar data. The dataset, *CHARM*, suggested by Modulai AB, fulfilled these requirements and was thus selected.

3.1.1 CHARM - Car repair manuals

Operation CHARM (Collection of High-quality Auto Repair Manuals) [6] provides a highly technical, domain-specific dataset consisting of Manuals for Repair and Diagnosis of various car models from different car brands. The data is structured in a hierarchical way, with the root being a list of car brands, branching to car models for a specific brand, and then car components for that model such as the airbag etc. with repair and diagnosis text and images. Appendix D displays example documents from the CHARM corpus.

3.1.2 The Sub-datasets

In typical machine learning fashion, training, validation, and test data are needed for fine-tuning an embedding model for domain adaptation. CHARM provides diagnosis and repair manuals for over 50 different car brands, and for a number of car models from different years for each brand. Six car models from six different car brands were selected randomly for the construction of training and validation data; Saab 2010 9-3 FWD, Hyundai 2008 Accent, Chrysler 2013 200, Volvo 2006 C70, Fiat 2012 500, and Lexus 2011 CT 200h Hybrid. These car models will henceforth be referred to by the car brand name alone, or by the car brand and the model year. The validation data is for all datasets an accurate representation of the matching training data, as the validation data was gathered by random sampling from the same dataset as the training data. Samples included in validation were naturally not included in training. Based on the aim to investigate the impact of training data size and diversity (the number of different car brands included in a training dataset), six different training datasets and the corresponding validation datasets, with a varying number of training examples and diversity were constructed. In table 3.1 a summary of these datasets is presented, where the training and validation size refers to the number of $(query, passage)$-pairs:

Test data needs to remain completely unseen during training to avoid data-leakage, as stated in section 2.5. This could have been achieved by randomly selecting and removing samples from the datasets listed above, creating test datasets with the same origin as training and validation data. However, a more interesting research objective is to investigate if the fine-tuning increases performance across the whole domain, i.e. if the model trained on the car models above is able to generalize across car manuals from other models or even brands. For this purpose, two models from new car brands; Ford 1992 Aerostar and Jaguar 2002 S-Type, as well as a different Saab model (Saab 2004 9-5 Aero), were selected to construct the synthetic test datasets. These datasets allow for testing and comparison of generalization abilities over different domain widths: within the same car brand and for "never-seen-before"

No. of Car models	Car Brands	Training Size	Validation Size
1	Saab (2010)	2500	1000
2	Saab (2010), Hyundai (2008)	2500	1000
3	Saab (2010), Hyundai (2008), Chrysler (2013)	2500	1000
3	Saab (2010), Hyundai (2008), Chrysler (2013)	5000	1000
3	Saab (2010), Hyundai (2008), Chrysler (2013)	7500	1000
6	Saab (2010), Hyundai (2008), Chrysler (2013), Volvo (2006), Fiat (2012), Lexus (2011)	15000	3500

Table 3.1: Datasets used during training and validation.

car brands. For each test-set, between 1300 and 1600 queries were generated. In order to create a more difficult task for the model without having to go through the time-consuming process of generating more queries, the vector database was populated with additional documents, without a matching query. Given this, the model had to distinguish between a larger set of documents, and the probability of randomly retrieving the correct document is significantly reduced, allowing for a more fair assessment of the model's abilities. For each test-set, all documents in the vector database originated from the same car model as the one used for the synthetic query generation. This is motivated by the fact that not all queries mention the specific car model they are generated from. For example, a query generated from a Ford airbag manual could be about airbags in general, making it impossible for the model to, based on semantics, know if it should retrieve airbag related documents from Ford, Saab, or Jaguar. Further, this approach is realistic: when searching for information about their car, the car owner knows which specific car models manual to look in. The synthetic test-sets that were generated are presented in table 3.2.

Car model	Number of (Q, P)-pairs	Number of Passages in Database
Saab (2004)	1333	6063
Ford (1992)	1592	7602
Jaguar (2002)	1332	4692

Table 3.2: Synthetic Test Datasets used for Evaluation.

Evaluating on synthetic data is an efficient method for getting insight into the model's ability to generalize and its performance, especially if the synthetic queries have been prompted in a way making them realistic. Despite this, nothing can with certainty be said about how the model would perform on real, human queries. In order to test this, two human annotated test datasets were created. First, queries were generated using a prompt crafted based on the requirements that they should be human-like, realistic and general. In the prompt, example questions found on car repair forums were used. With these, still synthetic but more human-like queries as a basis, human annotation was performed. We manually went through all queries; changing the wording, making them shorter, more diverse, and more general. Data from two different car models (two models also used for the synthetic test datasets), Saab 2004 9-5 Aero and Jaguar 2002 S-Type, was humanly annotated, creating two small test datasets with 100 queries in each. Thirty human annotated queries from the Jaguar dataset is presented in Appendix C.1. Additional documents with no matching queries were added to the vector database in these cases as well. As human annotation was performed on both a car brand unseen during training (ZSL) and a car model from a brand used in training, the model's ability to generalize across varying domain widths could be investigated. In table 3.3 follows a summary of the resulting human annotated test datasets.

Car model	Number of (Q, P)-pairs	Number of Passages in Database
Saab (2004)	100	1696
Jaguar (2002)	100	4970

Table 3.3: Human Annotated Test Datasets used for Evaluation.

3.2 Data Processing

The data was scraped from the CHARM documents into json-files, where the links leading to the corpus were marked as *titles*, followed by the text from that page. Images were naturally excluded, leading to some documents with text mainly referring to these images becoming irrelevant. To minimize this issue, a lower character limit of 200 was set, where documents containing less text were removed. This approach does not guarantee a 100% efficient filtering, but was assessed as reducing the problem significantly based on manual inspection of the corpus.

A choice was made to keep all the titles, i.e. the page-paths as part of the corpus documents. This decision was based on the paths giving context to the text, as early experiments indicated that the task for the embedding model would be unrealistic when excluding the title. However, as the titles were organized as search-paths, often containing specific diagnosis error codes and not semantically organized text, a concern about lexicon rather than semantic search being utilized by the model was raised. Upon further inspection, it was found that queries generated based on documents including these titles often contained the specific error codes mentioned, confirming the concerns about lexicon search. When conducting early experiments on this data, the performance on the synthetically generated queries was found to be very high. This further increased concerns as human-written queries would in the majority of cases not be able to rely on lexicon search, as it is reasonable to assume they are more general in nature.

Grounded on these observations, an additional, LLM-based data-processing method was added to the pipeline, with the purpose of generating more human-like queries for the training data and creating a harder task for the model, enabling fine-tuning based on semantics rather than specific words or error codes. This method, *Path Summarizing* is discussed in more depth in the following section 3.2.1.

3.2.1 Summarizing the Paths

As previously motivated in section 3.2, leaving the titles in their original state resulted in the synthetically generated queries being unrealistically detailed, creating too easy a task for the model where the performance increase would not be transferred to real, human-written queries. Removing the titles completely would on the other hand create a task that was too hard, as the text passages would lack essential context. Finding a golden middle way was thus necessary when moving forward, where the paths were kept - but on a modified form.

Changing the paths manually was not an option, given the size of the dataset. Instead, a Large Language Model was used to streamline the process. OpenAI's model GPT-3.5-turbo [38] was selected, motivated by the fact that this model do not require any access to powerful GPUs, as the model does not run locally as opposed to open-source alternatives. The cost is despite the proprietary model very low; with an input-cost of \$0.5 per 1M tokens and an output cost of \$1.5 per 1M tokens, which put the total cost for this task at less than a couple of US dollars.

Using the OpenAI API, code was implemented where the title for each document was sent to GPT-3.5-turbo together with a prompt describing the task for the model. The aim of this process was to get the model to generate short summaries of the titles, leaving out specific diagnosis error codes and changing the text structure from paths to running text with comprehensible sentences.

The first tests were performed using a simple summarization prompt, simply asking the model to provide concise summaries of the title. Crafting a good prompt is an iterative process, and repeated tweaks to the prompt were performed while manually inspecting the results produced. Techniques such as role-based prompting and providing concrete examples of what the end result should look like were utilized in the final prompt.

When a prompt that produced results satisfying our requirements had been crafted, the titles for all documents in our datasets were processed in this manner. New documents where the old titles had been replaced with the generated summarized titles were created and used moving forward.

3.2.2 Document Chunking

The documents were divided into chunks of text, *passages*. This is a crucial step in order to avoid information loss, see section 2.2.1, as embedding models have a limit in how many tokens they can take per call.

Two different chunking approaches, Sentence Splitting [17] and Semantic Splitting [19], were tested. For this, two classes provided by LlamaIndex were utilized. LlamaIndex is a data framework specifically designed for building applications based on LLMs.

The Semantic Splitter takes an LLM as a parameter, and OpenAI's gpt-3.5-turbo[38] was selected based on the same criteria as for path summarizing. Both splitters also take *chunk size* as a parameter, corresponding to the maximum number of tokens allowed in one chunk. Since tokenization is performed differently depending on the tokenizer used, it was critical to match the tokenizer used when splitting to the tokenizer later used in the embedding model.

The two chunking approaches were tested on a set of documents. Based on visual inspection of the resulting text passages and preliminary experiments assessing the impact on performance, the Sentence Splitter proved to yield better results and was thus used in subsequent stages.

An issue arose due to the chunking, as many of the resulting passages lost their title, leading to context loss. This occurred when the documents contained more text than specified by the maximum chunk size. As all documents began with a summarized path (title), no passages created from that document except the first one contained the title. This was solved by implementing code for extracting the title from a document prior to chunking, and then adding it back at the beginning of all passages created from that document. This also meant that the chunk size had to be adjusted in order to not exceed the maximum number of tokens allowed by the embedding model when the title was added to the passages.

3.2.3 Synthetic Query Generation

The CHARM dataset is an unlabeled corpus, meaning the documents, which later are divided into passages, do not include any matching queries. In order to perform supervised fine-tuning of a retrieval system, labeled data is needed.

Synthetic query-answer labeling is a feasible choice for transforming the corpus into data that can be used for training, as well as evaluation. For each text passage, OpenAI's gpt-3.5-turbo model [38] was used to generate synthetic queries. One query was generated for each passage, with the answer to the query being present in the originating passage. The parameter choice of only generating one query per passage was motivated by the nature of the data: many of the passages are short and it would not make sense to expect two or more reasonable, diverse, queries from such a short text.

Successfully prompting the model is an essential part of generating high quality queries useful for fine-tuning. The end goal was to generate queries that felt human-like, were diverse in sentence structure, concise, and not extremely specific (such as referencing the manual in the query).

As a starting point, a pre-defined default prompt from a LlamaIndex class was used. The default prompt is shown below, giving some context to how prompts can be structured.

```
DEFAULT_QA_GENERATE_PROMPT_TMPL = """\
Context information is below.

---------------------
{context_str}
---------------------

Given the context information and no prior knowledge,
generate only questions based on the below query.

You are a Teacher/ Professor. Your task is to setup
1 question for an upcoming
quiz/examination. The questions should be diverse in nature
across the document. Restrict the questions to the
context information provided."
"""
```

The resulting queries generated with the default prompt did not meet our requirements; they were not human-like in nature, very detailed, and long. Prompting is, as previously stated in section 2.2.3, an iterative process, and a "trial-and-error" process of tweaking the prompt was thus performed. Progress was made, where important improvements included changing the role from Teacher/Professor to an Automotive Technician, formulating the task differently, and specifically stating that the generated queries should be short and concise.

Summarizing the titles before performing synthetic query generation also contributed to more realistic queries. In appendix C.2, thirty example queries generated from documents with non-summarized paths and a sub-optimal prompt are presented. These can be compared to the queries in appendix C.3, which were generated with a tweaked prompt from documents with summarized paths.

3.3 The Embedding Model

3.3.1 Selecting an Embedding-model

When selecting an embedding model to fine-tune and use in the implemented retrieval system, the important factors to consider do not, given our research aim, include finding the highest performing or most state-of-the-art model available. Instead, the main focus was on finding a sufficiently good model with a relatively low computational cost, as GPU resources were limited. This approach also proves useful for situations where computational resources are limited and sufficient performance needs to be reached by utilizing a smaller, but well fine-tuned model.

The choice of excluding proprietary models such as those provided by OpenAI was made. OpenAI does not provide fine-tuning possibilities for their best performing embedding models, and with the additional costs aspect taken into account the option of fine-tuning an older OpenAI embedding model was excluded as well.

Hugging Face [39], which is a machine learning community and platform, provides a leader-board [40] with embedding models for natural language processing tasks ranked based on massive text embedding benchmarks. From this leader-board the model "bge-small-en-v1.5" [41] developed by Beijing Academy of Artificial Intelligence (BAAI) was selected. This model is built on the sentence transformer architecture previously described in section 2.1. The model is small compared to most other text embedding models on the leader-board, but still performs well across a variety of tasks. Further, BAAI also provides much larger embedding models, with "bge-large-en-v1.5" [42], being the best performing one suitable for comparisons with the smaller version. The model "bge-large-en-v1.5" was thus selected as a benchmark, leading to the following research question: *Can the smaller fine-tuned model outperform the larger, not fine-tuned model?*

In table 3.4 follows a summary of the size, input token limit and number of embedding dimensions for both models.

Model Name	Model size	Max Tokens	Memory Usage	Dimensions
bge-small-en-v1.5	33 Million Parameters	512	0.12 GB	384
bge-large-en-v1.5	335 Million Parameters	512	1.25 GB	1024

Table 3.4: bge-small-en-v1.5 (used for fine-tuning) is around 10 times smaller than bge-large-en-v1.5 (used as a benchmark).

3.3.2 Fine-tuning the Embedding Model

"Bge-small-en-v1.5" was fine-tuned with a Sentence Transformers provided Multiple Negatives Ranking Loss function [29] on the six curated and synthetically labeled training datasets presented in section 3.1.2. The training data thus consisted of $(Q, P+)$-pairs, and varied in number of training samples and training sample diversity across the datasets. Multiple Negatives Ranking Loss was used, as this is a common and suitable choice of loss function for data on this format when training embedding models in dense retrieval settings.

3.3.3 Staged fine-tuning

Fine-tuning of the embedding model was performed in two stages: after the initial fine-tuning with Multiple Negatives Ranking Loss, a second training step utilizing a loss function for contrastive learning was used.

Experiments where the initial fine-tuning was excluded and the model only was trained with contrastive loss were conducted, but yielded worse accuracy. Given this, contrastive fine-tuning was mainly performed on initially fine-tuned embedding models.

Two different contrastive loss functions from the sentence transformers library were used; Margin MSE loss [31] and Online Contrastive loss [30]. As these loss functions requires the training data on a different format than for initial fine-tuning, additional data processing steps, including Hard Negative Mining and margin calculations, were necessary. The following sections 3.3.4 and 3.3.5 describe these steps.

3.3.4 Mining Hard Negatives

Contrastive loss is built on the premises that labeled hard negative passages are present in the training data. This is often not the case, and we then need to perform Hard Negative Mining, i.e. finding and labeling the hard negatives.

Determining which passages are hard negatives for the queries can be a challenge. Manual labeling is not only extremely time-consuming for large datasets, but also requires expertise in the domain. Two automated alternative approaches were instead tested.

For both approaches, the non fine-tuned embedding model "bge-small-en-v1.5" was used to retrieve the 10 most relevant passages according to the model's embedding similarity scores, for each query in the training dataset. The hope was to find a hard or semi-hard negative example among these 10 retrieved passages. Using a "bge-small-en-v1.5" model for this is a natural choice, as we want to find negative passages that are hard for the model intended for fine-tuning. Deciding whether to use the base "bge-small-en-v1.5" model or an initially fine-tuned version was more challenging. As synthetic query-answer labeling is performed in such a manner that each passage only is labeled as a positive for one query, there is a risk of false negative passages. Minimizing the occurrence of passages not labeled as positives, but still being relevant for the query, being retrieved and labeled as hard negatives and then explicitly used contrastively during training, was a priority. Given this, the non fine-tuned model was used, as this model theoretically should be worse at retrieving relevant passages.

The first, most straight forward approach tested, was to after the retrieval step loop through the retrieved passages in reversed order. As there is only one labeled positive passage for each query, at least 9 of the retrieved passages for each query were not labeled positives.

In the cases where the model failed to retrieve the positive example, all 10 retrieved passages were not labeled positives. Starting from the bottom of the retrieved list, the first non-positive labeled passage was explicitly labeled as a hard negative. Looping in reversed order was another choice made for the purpose of avoiding labeling passages that are actually positive as negative. This is based on the assumption that the probability of a false negative being retrieved as number 10 is lower than it being retrieved as number 1.

The second, more advanced approach tested, was utilizing an LLM. OpenAI's gpt-3.5-turbo [38] was once again selected, as a replacement for a domain expert. For each query, all 10 passages retrieved were sent to the LLM together with a prompt. The LLM was instructed to answer with 'True' if the passage contained information that could answer the query and 'False' if it did not contain information that could answer the query. The 10 retrieved documents for each query were sorted from most relevant to least relevant. The first passage which the LLM assessed as 'False' in this list was added as a hard negative passage to that query. The usage of an LLM was implemented with the aim to avoid labeling false negatives (passages not labeled as positives but still relevant to the query) as hard negatives.

Two datasets, one for each of the above described approaches, both containing 7500 triplets on the format $(Q, P+, hardnegativepassage(P-))$ were created from the training dataset with 7500 $(Q, P+)$-pairs sampled from three different car brands. All staged fine-tuning experiments were performed on only these training datasets, due to the computational power and costs associated with mining hard negatives.

3.3.5 Knowledge Distillation and Margin Calculations

One of the loss functions used for contrastive learning, Margin MSE Loss [31], requires the triplets in the training data to be labeled with a margin. Labeling with a margin, calculated with a teacher model such as a larger embedding model or a Cross-Encoder, is a form of knowledge distillation, see section 2.3.1, where the assessment of the teacher model is used as the ground truth. There are of course no guarantees that the teacher model's judgements are correct, and the process is thus referred to as *pseudo-labeling*.

We tested a few different models for margin calculation, but as preliminary experiments indicated that model choice had a limited impact on performance, "Bge-reranker-base" [43], a Cross-Encoder model from BAAI within the same class as our embedding models, was selected.

The margin was calculated by sending query and positive or negative passage simultaneously as input to the Cross-Encoder. The cross encoder similarity score for $(Query, Passage)$ was then received as output. We denote the similarity score as $p - score$ for $(Q, P+)$-pairs, and as $n - score$ for the $(Q, P-)$-pairs. The margin was then calculated as follows for all triplets in the training dataset: $margin = p\_score - n\_score$. From this, yet another training dataset was created, consisting of the triplets labeled with the calculated margins.

3.3.6 Fine-tuning on hard negatives

For the second fine-tuning stage, in which contrastive learning, see section 2.2.4, was performed, two different loss functions were tested.

Margin MSE Loss, see equation 2.9, was used for training on the $(triplet, margin - labeled)$ dataset. Selecting this loss function was motivated by the margin not labeling passages as strictly relevant or non-relevant, but instead labeling passages with a relevancy score between 0 and 1 [7]. This is an advantage given the issue with possible false negatives in the dataset.

Online Contrastive Loss, see equation 2.8 was used interchangeably with Margin MSE Loss. Fine-tuning with this loss function required the training data passages to be labeled as strictly positive or negative, and such a dataset was constructed using the previously mined hard negatives. Online Contrastive loss was selected over Contrastive Loss [44], as it trains only on negatives that are closer than the specified margin, and positives which are further apart, often yielding better results [30].

3.4 Final Prompts

3.4.1 Path Summarization Prompt

The final prompt used for path summarization was established as follows:

```
SUMMARIZATION_PROMPT_TEMPLATE: str = (
    "You are an Automotive Technician with expertise in giving
       concise and general summaries. As an example, 'A L L
       Diagnostic Trouble Codes ( DTC )/Testing and Inspection/B Code
        Charts/B1368/Airbag Control/Monitor Scantool Data' should be
       summarized as 'Diagnostic Trouble Codes for Testing and
       Inspection of Airbag Control with Monitor Scantool Data'.
       Avoid including specific code charts and troubleshooting codes
       . Do not cite the original text. Restrict the summary to the
       context information provided."
)
```

The example provided in the prompt closely resembles the results obtained after summarization. Below follows an example of a path before summarization:

- **"Lighting and Horns/Door Switch/Description and Operation/Switch, Door, Front, Right (54FR)/"**

And after summarization:

- **"Description and Operation of Door Switch for Front, Right, Lighting and Horns."**

In a majority of the cases the model succeeded in removing trouble codes and strange sentence structures, while the context was kept to facilitate the search.

3.4.2 Synthetic Query-Answer Labeling Prompt

The final prompt used for generating synthetic queries was:

```
PROMPT_TEMPLATE = """\
Context information is below.
```

```
---------------------
{context_str}
---------------------

Given the context information and not prior knowledge.
generate only questions based on the below query.

You are a Automotive Technician helping customers with their cars.
   Your task is to setup \
1 domain specific questions that could be asked by one of your
   customers. The questions should be diverse in nature \
across the document. The questions should be kept short and general.
   Restrict the questions to the \
context information provided."
"""
```

This prompt resulted in more human like queries compared to what was generated with the default prompt, facilitating domain adaptation. However, a majority of the queries still begins with "How". In Appendix C.3 thirty examples of the synthetic queries generated with the above prompt are presented. The queries are generated from a dataset that combines data from Saab, Hyundai and Chrystler.

3.4.3 Prompt for LLM-based Hard Negative Mining

The final prompt for evaluating if passages were positives or negatives when performing Hard Negative Mining was established as:

```
HNM_PROMPT: str = (
    "You are an Automotive Technician evaluating document relevance
       for technical automotive queries. For the query, determine if
       the relevant information mentioned are provided in the CHARM
       document. Respond ONLY with 'True' if the document contains
       the necessary information to answer the query. If the document
        does not contain the specific information requested in the
       query, respond ONLY with 'False'. You are under no
       circumstances allowed to answer anything other than 'True' or
       'False'."
)
```

To streamline the process of selecting negative passages the model was instructed to answer only with True or False. This resulted in output that could easily be evaluated while keeping the output token count low to minimize costs.

3.5 Vector database

FAISS (Facebook AI Similarity Search) [36] was used for implementing a vector database in our retrieval system. FAISS offers efficient algorithms for similarity search, scalable indexing, and various clustering and quantization techniques enabling high performance and low memory usage, making it an appropriate choice for RAG applications. Further, FAISS is highly customizable and optimized for both CPU and GPU usage, making it versatile for different deployment environments. Additionally, there are complete Wrappers for Python, and it is thus easy to implement and set up.

Similarity search is performed by FAISS building a data structure in RAM from a given set of vectors with dimension d. When a new vector, for example an embedded query, of the same dimension is sent as input FAISS performs similarity search based on the Euclidean distance. Our implementation allowed for all queries in a test set to be given to FAISS in one file, after which the 10 most relevant passages (the closest embeddings) for each query could be retrieved from the vector database.

Retrieving the top 10 passages, and not for example only the closest or the 100 closest passages, was a parameter selection made based on both performance and system effectiveness. With an increased number of retrieved passages, the hit rate was of course much higher and even the base model (non fine-tuned model) often succeeded, making comparisons between different models more difficult. Further, a lower number of retrieved passages would not allow for as comprehensible assessment of the order-aware metrics, as only the retrieved passages are taken into account when evaluating the ranking. Retrieving 10 passages per query, corresponding to about 0.2% of the samples in the vector database, means significantly reducing the text a user would have to go through, creating an effective system while still not giving the embedding model an extremely difficult task.

3.6 Experimental Set-up

The experimental set-ups are presented in table 3.5. The tabulated "bge-small-en-v1.5" embedding models, fine-tuned with different approaches, were used in the implemented retrieval system and evaluated on all synthetic and human annotated test datasets. Training size refers to the number of $(Q, P+)$-pairs or $(Q, P+, P-)$-triplets in the training data, and diversity refers to the number of different car models included in the training data. For all cases were staged fine-tuning was performed, the number of training-sample pairs used for the initial fine-tuning is equal to the number of triplets used for contrastive fine-tuning.

Grid search was performed for hyperparameter optimization. This was done by running *Sweeps* [33] with Weights and Biases [32]. Hyperparameters, outputs and metrics were logged with Weights and Biases during the sweeps, and hyperparameter values were selected according to what yielded the highest accuracy on the validation datasets.

Training size	Diversity	Initial fine-tuning	Contrastive fine-tuning
0	0	no	no
2500	1	yes	no
2500	2	yes	no
2500	3	yes	no
5000	3	yes	no
7500	3	yes	no
15000	5	yes	no
7500	3	yes	Margin MSE Loss on non-LLM mined hard negatives
7500	3	yes	Margin MSE Loss on LLM mined hard negatives
7500	3	yes	Online Contrastive Loss on non-LLM mined hard negatives
7500	3	yes	Online Contrastive Loss on LLM mined hard negatives

Table 3.5: List of considered settings, training datasets, and loss functions for fine-tuning the "bge-small-en-v1.5" embedding model. The first row corresponds to the non fine-tuned base model.

The final optimal hyperparameters for Multiple Negatives Ranking Loss are listed in table 3.6

Hyperparameters	Value
Number of Epochs	2
Batch size	32
Learning Rate	$2 \cdot 10^{-5}$
Evaluation Steps	50
Optimizer	AdamW
Distance/Similarity Function	Cosine Similarity

Table 3.6: Hyperparameters used when fine-tuning with Multiple Negatives Ranking Loss.

A further increased batch size would likely yield even better performance, but due to memory limits the sweeps could not be performed on batch sizes larger than 32.

The optimized hyperparameters for Online Contrasive Loss and Margin MSE Loss are presented in table 3.7.

3.7 Evaluation

3.7.1 Evaluation Measures

Three metrics explained in section 2.5.1 were used for evaluation; Mean Reciprocal Rank (MRR), Normalized Discounted Cumulative Gain (NDCG), and hit-rate. These are all standard metrics for evaluating information retrieval systems, and give a comprehensive insight

Hyperparameters	Margin MSE Loss	Online Contrastive Loss
Number of Epochs	5	5
Batch size	32	16
Learning Rate	$2 \cdot 10^{-5}$	$2 \cdot 10^{-5}$
Evaluation Steps	50	50
Optimizer	AdamW	AdamW
Distance/Similarity Function	Pair-wise Dot Score	Cosine Distance
Threshold	-	0.7

Table 3.7: Hyperparameter values used for contrastive learning.

into the model performance. The hit-rate was calculated based on if the relevant passage was retrieved in the top 10 for its matching query: if yes, it was counted as a hit. For NDCG@10 and MRR@10, the scores were calculated using only the top 10 retrieved passages. As a result, a score of zero is obtained for a query regardless of if the relevant passage is ranked in 11:th or 1000:th place. This approach was intentional, as the aim was to reward the model only if it managed to place the relevant passage among the 10 retrieved ones. The measures obtained for each test dataset are the average of the scores given for all queries in that test dataset.

Recall is another standard metric used for evaluation information retrieval system. However, since there is only one positive passage per query in the test data, recall becomes equivalent to hit-rate.

Bootstrapping on the test data was performed for all evaluations in order to generate confidence intervals used to determine the statistical certainty of our findings. This was carried out by randomly constructing 100 bootstrapped samples from each test set for each evaluation, and then calculating the performance metrics for all bootstrapped samples in a test set. All bootstrapped samples were created to have the same size as the original test set, corresponding to the case $M = N$ in section 2.5.2.

3.7.2 Zero-Shot Learning (ZSL)

In order to investigate how well the fine-tuned embedding models were adapted to the domain, Zero-Shot Learning (ZSL), see section 2.3.3, was performed. As stated in section 3.1.2, two of the synthetic and one of the human annotated test datasets consisted of car brands unseen during training; the Jaguar datasets and the Ford dataset. By evaluating the performance on these datasets, the ability of the fine-tuned embedding models to perform Zero-Shot learning could be assessed.

3.7.3 Standard Testing

Section 3.1.2 also mentions that one synthetic and one human-annotated test dataset included a car brand seen during training, specifically the Saab 2004 test datasets. Note that all data in the test datasets were unseen during training, as the training data included documents from a different Saab model (Saab 2010). The goal of including such test datasets was to investigate how performance differs when evaluating on classes seen during training compared to classes unseen during training but within the same domain.

3.7.4 Benchmarking Against bge-large-en-v1.5

The performance of "bge-large-en-v1.5", the 10 times larger embedding model stated in section 3.3.1 as a selected benchmark, was calculated for all five test-sets. The model was taken "off-the-shelf", and not fine-tuned. The same evaluation metrics and strategies as mentioned above were used.

Chapter 4
Results

4.1 Retrieval Performance

In this section follow the results obtained when evaluating and comparing the retrieval performance for various embedding models fine-tuned with different approaches. Important to note is the fact that the results obtained for the different test datasets are not numerically comparable, as each test dataset presents different levels of task difficulty. The number of text passages present in the vector database varies between the test datasets, and further, some of the datasets evaluate zero shot learning while others do not. Comparisons should instead be made between the performance of different embedding models within one test dataset, and on the percentage increase seen after fine-tuning.

4.1.1 Synthetic Test Datasets

Tables B.1, B.2, and B.3 display a summary of the performance metric values obtained for differently fine-tuned embedding models across the three synthetic test datasets. The metrics are calculated as the average of all queries in a test dataset.

The embedding model that optimized performance across the synthetic data was fine-tuned in stages, first with Multiple Negatives Ranking Loss on 7500 $(Q, P+)$-pairs and then with Online Contrastive Loss on 7500 LLM mined $(Q, P+, P-)$-triplets.

Table 4.1 features the percentage increases in Hit-rate@10, MRR@10, and NDCG@10 between the "bge-small-en-v1.5" and the best performing fine-tuned model for the three synthetic test datasets. In table 4.2, the percentage increases from "bge-large-en-v1.5" to the same optimized model are shown. The percentage increases are calculated as:

$$\text{Percentage Increase} = \left(\frac{\text{New Value} - \text{Old Value}}{\text{Old Value}}\right) \times 100\%$$

	Hit-Rate@10	MRR@10	NDCG@10
Ford			
Small base model score	0.605	0.325	0.392
Optimized model score	0.708	0.393	0.468
Percentage Increase	16.5%	20.9%	19.4%
Jaguar			
Small base model score 0.680	0.400	0.467	
Optimized model score	0.763	0.461	0.531
Percentage Increase	12.2%	15.3%	13.7%
Saab 2004			
Small base model score	0.686	0.408	0.474
Optimized model score	0.809	0.520	0.589
Percentage Increase	17.9%	27.4%	24.3%

Table 4.1: Percentage increases and scores from the base model (bge-small) to the optimized fine-tuned model for the three synthetic test datasets.

	Hit-Rate@10	MRR@10	NDCG@10
Ford			
Large base model score	0.647	0.325	0.392
Optimized model score	0.708	0.393	0.468
Percentage Increase	9.4%	14.6%	12.5%
Jaguar			
Large base model score	0.686	0.409	0.475
Optimized model score	0.763	0.461	0.531
Percentage Increase	11.2%	12.7%	11.8%
Saab 2004			
Large base model score	0.695	0.422	0.487
Optimized model score	0.809	0.520	0.589
Percentage Increase	16.4%	23.2%	20.9%

Table 4.2: Percentage increases and scores from the large base model (bge-large) to the optimized fine-tuned model for the three synthetic test datasets.

For all three synthetic test datasets we found that retrieval performance with "bge-small-en-v1.5" was increased across MRR@10, NDCG@10 and Hit-rate@10 after fine-tuning.

Figures 4.1, 4.2, and 4.3 display results from the same experiments as those listed in tables B.1, B.2, and B.3, with the addition of bootstrap generated 95% confidence intervals. The width of the confidence intervals indicate the level of uncertainty surrounding the obtained results.

The 95% confidence intervals for the base model and the best performing fine-tuned models do not overlap, indicating statistical significance at the 5% significance level. Furthermore, the best performing fine tuned models also outperform "bge-large-en-v1.5", achieving statistical significance at the same 5% level. Increasing the training data diversity and the number

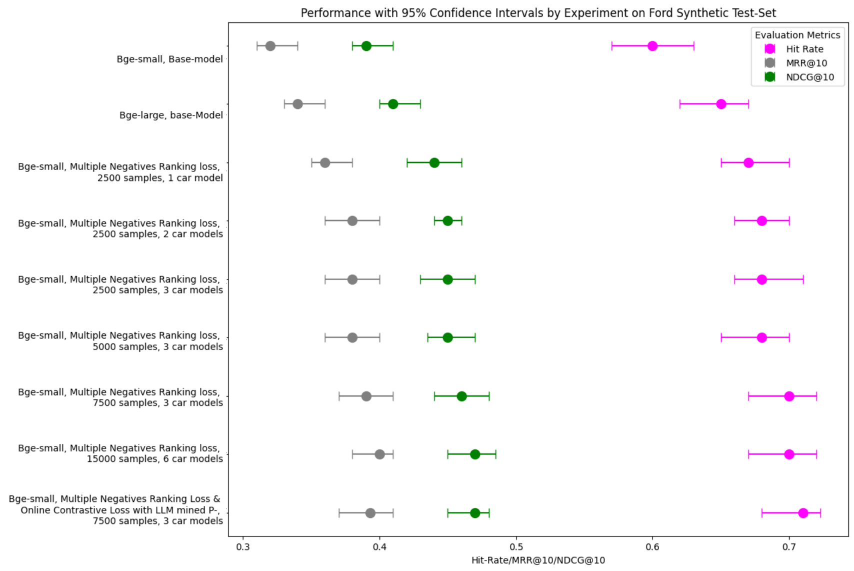


Figure 4.1: Retrieval Performance with 95% confidence intervals on Ford synthetic Test-set for base embedding models, models fine-tuned on varying data size and diversity, and a staged fine-tuned model. This is a zero shot learning setting.

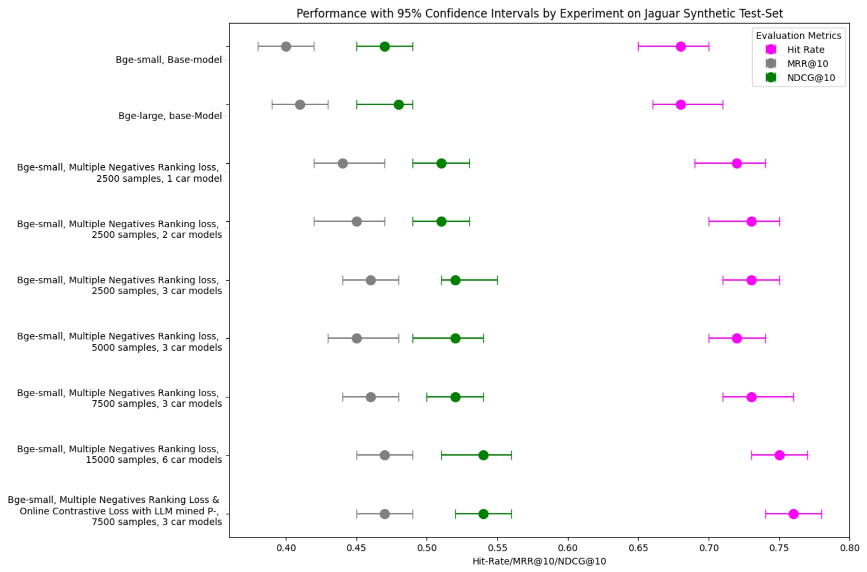


Figure 4.2: Retrieval Performance with 95% confidence intervals on Jaguar synthetic Test-set for base embedding models, models fine-tuned on varying data size and diversity, and a staged fine-tuned model. This is a zero shot learning setting.

of training samples slightly improves the bootstrapped median performance, when looking at the overall pattern. However, the differences in performance obtained for the differently fine-tuned embedding models are not statistically certain. Additionally, while the median performance of "bge-large-en-v1.5" is higher than for "bge-small-en-v1.5", the confidence intervals overlap indicating statistical uncertainty. The model fine-tuned in stages, displayed

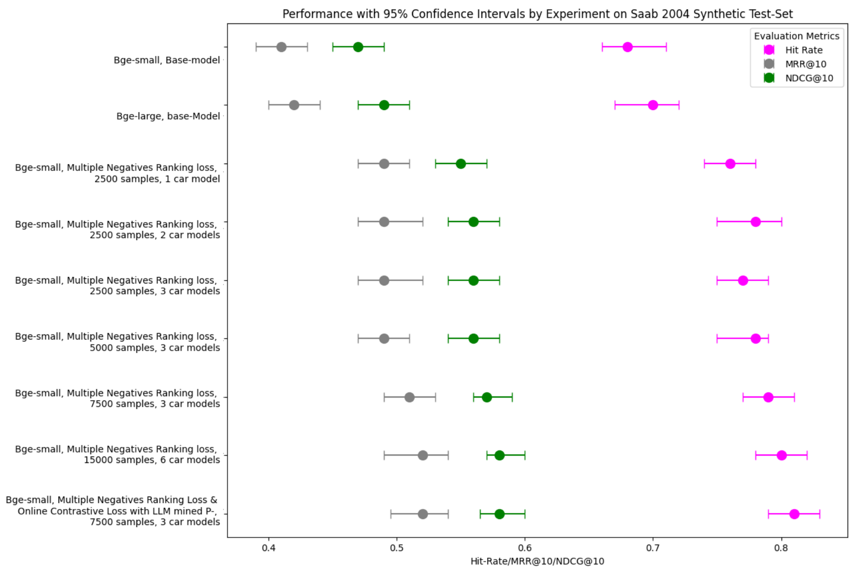


Figure 4.3: Retrieval Performance with 95% confidence intervals on Saab 2004 synthetic Test-set for base embedding models, models fine-tuned on varying data size and diversity, and a staged fine-tuned model.

at the bottom of figures 4.1, 4.2, and 4.3, gives the highest median hit rate for all three test datasets, but does not improve MRR@10 or NDCG@10. The confidence intervals of all the fine-tuned embedding models, including the optimized, overlap for all test datasets, making performance comparisons between different fine-tuning approaches statistically uncertain. The most dramatic increase in performance was seen for the Saab 2004 test dataset, see figure 4.3, where the confidence intervals for none of the fine-tuned embedding models overlap with the base model's. This test dataset was, as opposed to the other two, not a Zero-Shot learning setting, as Saab data (Saab 2010) was included in all training datasets.

4.1.2 Human Annotated Test Datasets

Tables B.4, B.2, and B.5 display a summary of the performance metric values obtained for 9 differently fine-tuned embedding models across the two human annotated test datasets. The metrics are calculated as the average of all queries in a test dataset.

Table 4.3 features the percentage increases in Hit-rate@10, MRR@10, and NDCG@10 between the base model and the best performing fine-tuned model for the two human annotated test datasets. In table 4.4, the percentage increases from "bge-large-en-v1.5" to the optimized model are shown. The best performing model differs between test datasets; the model fine-tuned in stages on 7500 training samples yields optimal performance for Saab 2004 while the highest accuracy for Jaguar is reached when fine-tuning once on 15000 training examples. In tables 4.3 and 4.4, comparisons are made between the two base models and the model yielding optimal performance **for that test dataset**.

Figures 4.4, and 4.5 display the results for the human annotated test datasets with the addition of bootstrap generated 95% confidence intervals.

The confidence intervals are very wide, a natural consequence of the human annotated

	Hit-Rate@10	MRR@10	NDCG@10
Jaguar			
Small base model score	0.490	0.270	0.322
Optimized model score	0.610	0.357	0.417
Percentage Increase	24.5%	32.2%	29.5%
Saab 2004			
Small base model score	0.660	0.466	0.512
Optimized model score	0.890	0.639	0.700
Percentage Increase	34.8%	37.1%	36.7%

Table 4.3: Percentage increases from the base model (bge-small) to the optimized fine-tuned model (differently fine-tuned for Saab 2004 than for Jaguar) for human annotated Jaguar and human annotated Saab 2004 test datasets.

	Hit-Rate@10	MRR@10	NDCG@10
Jaguar			
Large base model score	0.500	0.276	0.330
Optimized model score	0.610	0.357	0.417
Percentage Increase	22.0%	29.0%	26.4%
Saab 2004			
Large base model score	0.720	0.490	0.544
Optimized model score	0.890	0.639	0.700
Percentage Increase	23.6%	30.4%	28.7%

Table 4.4: Percentage increases from the large base model (bge-large) to the optimized fine-tuned model (differently fine-tuned for Saab 2004 than for Jaguar) for human annotated Jaguar and human annotated Saab 2004 test datasets.

test datasets only consisting of 100 queries each, making the performance highly dependent on individual samples. For the Zero-Shot learning scenario with the human annotated Jaguar dataset, all 95% confidence intervals overlap, and we can not conclude a statistical significant improvement in performance. Looking at the medians, we do see an increase in performance with training data size and diversity. However, the model fine-tuned in stages, which yields the highest accuracy for all other test datasets, is not optimal for the human annotated Jaguar data. It performs worse than four of the six embedding models only fine-tuned in one step, indicating that contrastive learning decreases performance.

For the human annotated Saab 2004 data, results indicate that fine-tuning the embedding model does enhance performance at the 5% significance level. For all evaluation metrics, most of the fine-tuned embedding models' confidence intervals do not overlap with the base model's. The model fine-tuned in stages reach statistical significant improvements from the large base model for hit-rate@10 and NDCG@10, but not for MRR@10, as the confidence intervals overlap.

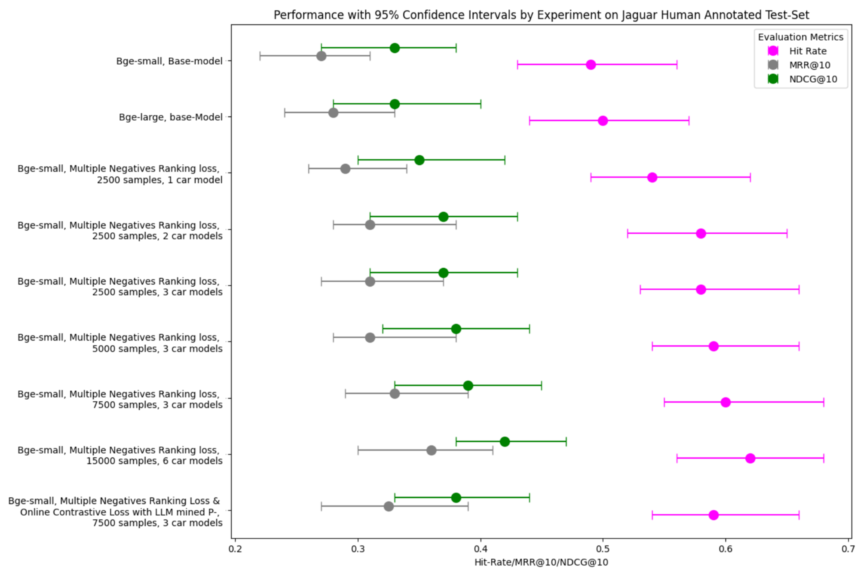


Figure 4.4: Retrieval Performance with 95% confidence intervals on the Human Annotated Jaguar Test-set for base embedding models, models fine-tuned on varying data size and diversity, and a staged fine-tuned model. This is a zero shot learning setting.

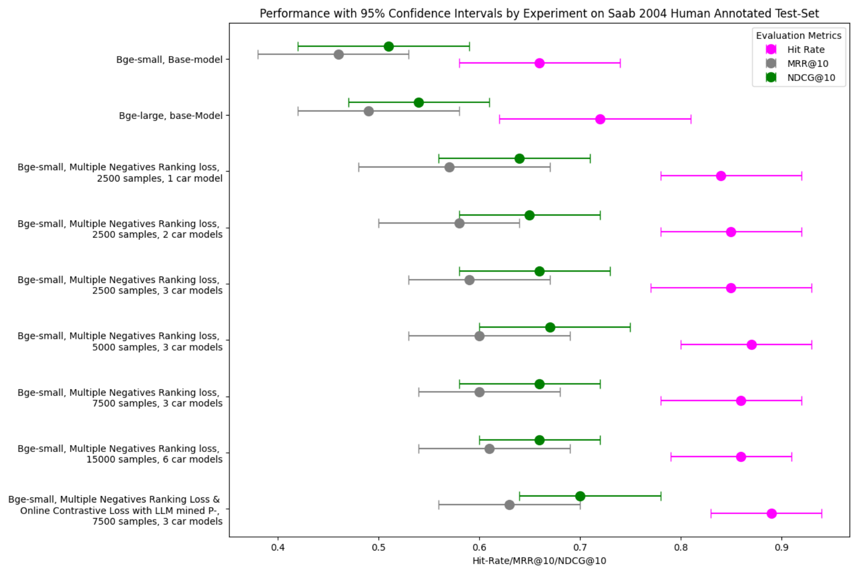


Figure 4.5: Retrieval Performance with 95% confidence intervals on the Human Annotated Saab 2004 Test-set for base embedding models, models fine-tuned on varying data size and diversity, and a staged fine-tuned model.

4.2 Contrastive Learning and Hard Negative Mining

Different approaches for staged fine-tuning and hard negative mining were tested. The sections above include the results for an embedding model fine-tuned first with Multiple Negatives Ranking loss (MNR loss) and then with Online Contrastive loss (OC loss) on LLM mined hard negatives. Figures 4.6, 4.7, 4.8, 4.9, 4.10 and 4.11 display the performance of other contrastive learning approaches on the Jaguar synthetic test dataset. Results were consistent across most test datasets.

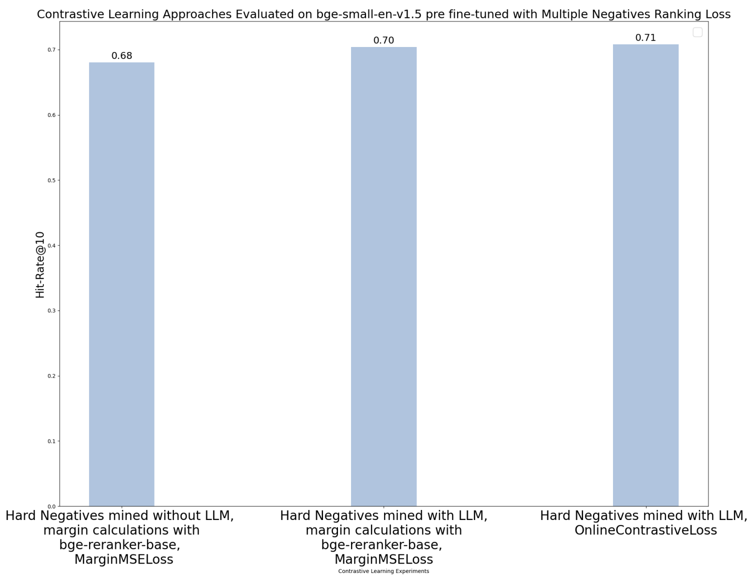


Figure 4.6: Hit-Rate@10 for embedding models trained on hard negatives mined without an LLM versus on LLM mined hard negatives. LLM-based hard negative mining (middle, right) sligthly improves results compared to mining without LLM (left).

Using LLM mined hard negatives seems preferred compared to selecting the hard negatives based on their retrieval ranking, as this slightly enhances performance, see figures 4.6, 4.7, and 4.8. The hit rate is increased by two percentile points when using Margin MSE loss on the LLM mined hard negatives, compared to on the non-LLM mined hard negatives. Online Contrastive Loss on the LLM mined hard negatives yields optimal performance.

As shown in figures 4.9, 4.10, and 4.11, performance is higher across all evaluation metrics for models fine-tuned with Multiple Negatives Ranking loss compared to models trained with solely Margin MSE or Online Contrastive loss. Staged fine-tuning, using both Multiple Negatives Ranking and Margin MSE or Online Contrastive loss improves accuracy slightly. Among these, employing Online Contrastive loss in the second fine-tuning stage produces the most accurate model.

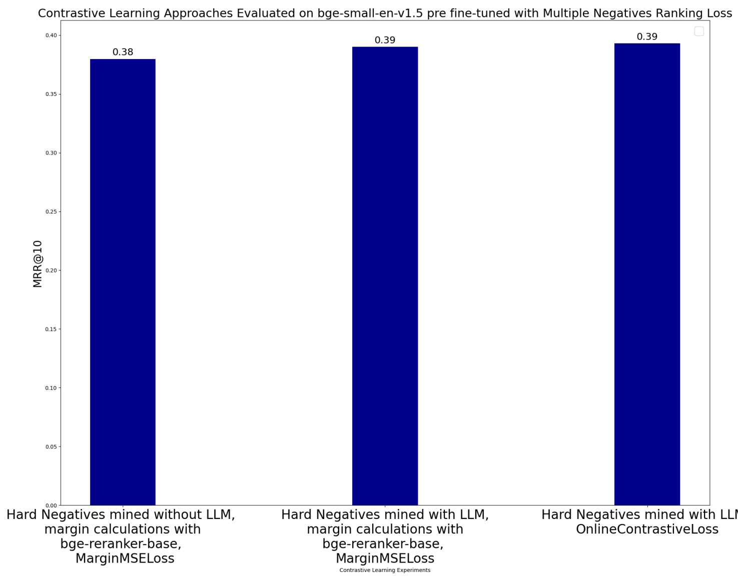


Figure 4.7: MRR@10 for embedding models trained on hard negatives mined without an LLM versus on LLM mined hard negatives. LLM-based hard negative mining (middle, right) sligthly improves results compared to mining without LLM (left).

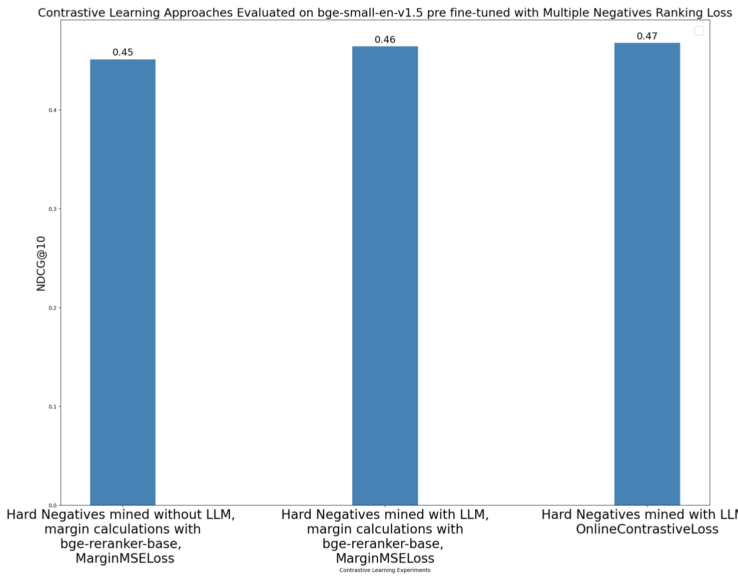


Figure 4.8: NDCG@10 for embedding models trained on hard negatives mined without an LLM versus on LLM mined hard negatives. LLM-based hard negative mining (middle, right) sligthly improves results compared to mining without LLM (left).

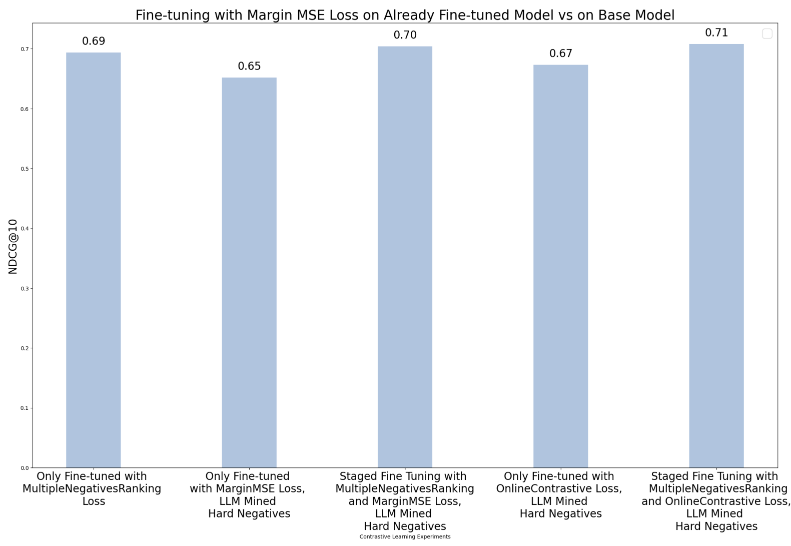


Figure 4.9: Hit-Rate@10 for the following embedding models (from left to right): 1. fine-tuned with MNR loss on 7500 $(Q, P+)$ samples. 2. Finetuned with Margin MSE loss (MM loss) on 7500 LLM mined triplet samples. 3. Staged fine-tuning with MNR loss on 7500 $(Q, P+)$ samples and MM loss on 7500 triplets. 4. Fine-tuned with OC loss on 7500 LLM mined triplets. 5. Staged fine-tuning with MNR loss on 7500 $(Q, P+)$ samples and OC loss on 7500 LLM mined triplets.

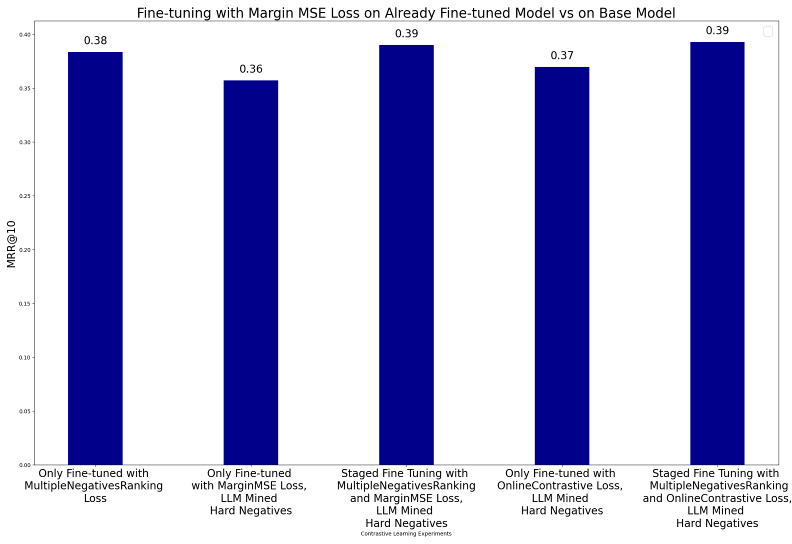


Figure 4.10: MRR@10 for the following embedding models (from left to right): 1. fine-tuned with MNR loss on 7500 $(Q, P+)$ samples. 2. Finetuned with Margin MSE loss (MM loss) on 7500 LLM mined triplet samples. 3. Staged fine-tuning with MNR loss on 7500 $(Q, P+)$ samples and MM loss on 7500 triplets. 4. Fine-tuned with OC loss on 7500 LLM mined triplets. 5. Staged fine-tuning with MNR loss on 7500 $(Q, P+)$ samples and OC loss on 7500 LLM mined triplets.

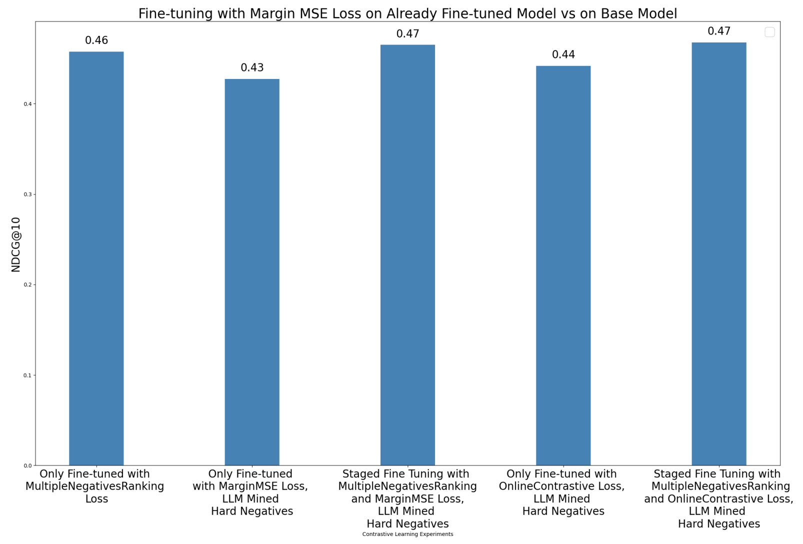


Figure 4.11: NDCG@10 for the following embedding models (from left to right): 1. fine-tuned with MNR loss on 7500 (Q, $P+$) samples. 2. Finetuned with Margin MSE loss (MM loss) on 7500 LLM mined triplet samples. 3. Staged fine-tuning with MNR loss on 7500 (Q, $P+$) samples and MM loss on 7500 triplets. 4. Fine-tuned with OC loss on 7500 LLM mined triplets. 5. Staged fine-tuning with MNR loss on 7500 (Q, $P+$) samples and OC loss on 7500 LLM mined triplets.

Chapter 5

Discussion

5.1 Reflections

When initialising this project, one of the main aims was to create a system that could perform well on human queries, despite being fine-tuned on synthetic training data. As we started to perform preliminary experiments, it became clear that it was difficult to generate human-like training data from the CHARM dataset. A significant portion of the project time was therefore dedicated to data processing. Processing the corpus in order to increase training data quality was not straight forward, and we could not rely solely on standard methods. Using a large language model for data processing, such as for summarizing the paths, proved to be very effective, and substantial improvements in data quality were made. Despite this, we expected the increase in retrieval performance after fine-tuning to be substantially lower for the human annotated test datasets than for the synthetically generated data. However, evaluations indicate the opposite. Although the confidence intervals are much wider for the human annotated datasets, making these findings less certain, this is likely a consequence of the test datasets becoming more sample dependent due to the small number of test samples rather than an indication of true performance. With this reasoning, we argue that larger human annotated test datasets would yield similar performance medians with more narrow confidence intervals. The causes of the human annotated datasets experiencing a more drastic increase in retrieval accuracy post fine-tuning are not established. Speculations about what may have contributed are that the queries are generally shorter and more semantically contextual. The synthetically generated test datasets likely include a portion of unusual queries where the retrieval is not improved by fine-tuning, dragging down the average performance for the synthetic data.

Increasing training data size and diversity did contribute some in increasing performance, but the gains were marginal. It is possible that substantial improvements would be observed if the number of training samples were increased more drastically, for example, by an order of magnitude of 100 or 1000. However, the large improvements seen after fine-tuning on only 2500 training samples, followed by much smaller additional gains for the larger training datasets, indicate a logarithmic relationship between training data size and performance.

The model performance increase on the Zero-Shot learning tasks is surprisingly similar to the increase seen for the Saab 2004 test datasets. This is an important finding, indicating that fine-tuning on a different part of the domain is almost as effective as fine-tuning on the exact same domain region as the test data. Further, this implies that the model can generalize across the whole domain, in this case across a variety of different car models and brands. Staged contrastive learning improves accuracy across all test datasets except the human annotated Jaguar queries. A reasonable explanation for this is the fact that the queries in this test dataset are more dissimilar to the training queries compared to the other test datasets. Not only are the queries crafted by humans instead of an LLM, but this is also a Zero-Shot learning setting. When fine-tuning in two stages, in the second stage with contrastive loss, the embedding model fits more aggressively to the training data, possibly overfitting it to synthetic queries and car brands included in the training dataset.

Only a quite small amount of training data was required in order for "bge-small-en-v1.5" to outperform "bge-large-en-v1.5". This indicates that embedding model size is not the most important factor for achieving good retrieval performance in domain-specific fields. Rather, the success of domain adaptation depends on fine-tuning. It is thus reasonable to conclude that it is preferable to use a small but fine-tuned embedding model over a larger one that has only been pre-trained, even if if the training data available is limited.

Another interesting finding is that peak performance for both the synthetic and human annotated Saab 2004 test datasets is not reached when the model has been fine-tuned on other Saab data exclusively. Instead, we observe an increase in performance when car models from other brands are also included in the training data. This indicates that diverse training data enhances the embedding model's ability to learn the semantic meaning of the corpus.

5.2 Issues and Limitations

5.2.1 The CHARM dataset

The CHARM dataset contains, in its original state, images. These images are excluded when the data is scraped into json-files, resulting in a portion of the documents becoming irrelevant as they refer to a figure. Filtering based on a lower character limit of 200 was performed as an attempt to tackle this issue. While this filtering reduced the occurrence of irrelevant documents, the problem was not completely eliminated, and instances of non-contextual text remained in the corpus. Eliminating this issue with 100% accuracy would require time-consuming manual inspection and filtering.

5.2.2 Choice of Chunking method

The choice of chunking method had a large impact on the result. Since the CHARM corpus in many instances does not have a semantic structure, chunking methods like Semantic Splitting, which uses an LLM to chunk the text, was not the best choice. Instead, a more simple chunking approach yielded superior results. However, for many other datasets, for example corpora of scientific papers, semantic splitting would likely have been the most effective method.

5.2.3 Hard Negative Mining and Margin MSE Loss

When performing fine-tuning with Margin MSE loss, it was observed that the choice of margin calculator did not have a large impact on the performance. However, due to limited memory and GPU access, we could not use the most state-of-the-art embedding models or Cross-Encoders as teacher models. It is possible that the usage of for example an LLM-based embedding model for margin calculations would have yielded better results. It was observed that performing LLM-based mining of hard negatives yielded higher evaluation scores compared to hard negatives mined without the use of an LLM. However, it should be noted that there is a trade-off between slightly increasing performance by performing LLM-based hard negative mining and saving time and resources by skipping this step.

5.3 Future Work and Applications

5.3.1 Re-labeling passages

The synthetic queries were generated from one passage each, which resulted in one positive passage for each query. When retrieving passages it was observed that documents labeled as negatives in some cases were relevant to the query. If more time was given, one approach that would likeley benefit the system and result in more realistic performance metrics would be to, with the facilitation of an LLM, evaluate the top K retrieved passages and re-label false negative passages as positives.

Mixtral is an example of an LLM that was briefly tested on this task and seemed to perform well on re-labeling passages. Mixtral is an open-source model by Mistral AI. The model is a high-quality sparse mixture of experts model (SMoE) with open weights, licensed under Apache 2.0. Mixtral outperforms Llama 2 70B and GPT3.5 on most standard benchmarks [45].

In section 1.4, the paper 'How to Train Your DRAGON: Diverse Augmentation Towards Generalizable Dense Retrieval' [9] is discussed. The authors propose another approach, using a multitude of retrieval models, to re-label passages. Using a method inspired by this for refining training data labels is an alternative approach to how future work could be conducted.

5.3.2 Increased Computational Power

The limited computational power and GPU time constrained the training data size and the choice of embedding model. For future work it would be interesting to test if the performance would increase further by drastically increasing the variety and size of the training data.

An increase of computational power would unlock the possibility to train larger embedding models, which would likely yield higher performance.

5.3.3 Applications

The use of chatbots utilizing Retrieval Augmented Generation (RAG) has become increasingly common in enterprise settings [46], particularly for enhancing customer service and internal support. The most common approach is to use a pre-trained embedding model without performing further fine-tuning on the domain data [47][48][49].

Our research focuses on data preparation, fine-tuning open-source embedding models and employing hard negative mining to improve the retrieval performance for such systems. This approach ensures that chatbots can provide more accurate and contextually relevant responses, particularly in specialized domains. By enhancing the retrieval component, our methodology can address the gaps present in the majority of current solutions.

Chapter 6
Conclusion

6.1 Answering the Research Questions

Given a large, unlabeled, domain-specific set of documents. What is the most effective set of techniques to achieve good retrieval performance?

We have demonstrated that important considerations when optimizing retrieval performance include knowing your data, and processing it with tailored techniques before generating synthetic training datasets. Facilitating a large language model for title summarization and mining of hard negatives proved effective for the CHARM dataset, but these methods likely needs to be adjusted according to the corpus used. Prompt engineering is another crucial aspect of successfully performing domain adaptation of a retrieval system, and again demonstrates the importance of knowing your data as the prompts are best curated when the data format is taken into account. The training dataset should be diverse and contain an efficiently large number of samples, but more importantly effort should be put into generating human-like queries. Further, using more than one loss function for staged fine-tuning of the embedding model was found to be the method yielding the highest performance.

Is synthetic training data a viable option for fine-tuning an embedding model on a technical domain?

For the second research question, it could be concluded that training the embedding model on synthetically generated questions improved the performance across both synthetic and human annotated test datasets. It was also observed that increasing the number of queries in the training dataset did not have a large impact on the results, which enables boosting of retrieval performance within a domain even if only a limited amount of data is available.

Can a small fine-tuned embedding model outperform a much larger "off-the-shelf" embedding model?

Finally, it was concluded that the smaller embedding model "bge-small-en-v1.5" fine-tuned on synthetic data outperformed the ten times larger base embedding model "bge-large-en-v1.5" at the 5% significance level for four out of five test datasets.

6.2 Final Conclusions

Performing domain adaptation of retrieval systems is an iterative process which requires flexibility and adjustments depending on the domain and the data. We conclude that synthetically labeled training data is a viable option for fine-tuning an embedding model, making domain adaptation feasible across multiple fields where manual labeling is not possible. Further, facilitating large language models in multiple steps of the data processing, such as corpus refinement and hard negative mining, can improve training data quality and enable increases in retrieval performance across human annotated queries.

Circling back to our research objective, we have investigated a range of techniques across the retrieval system implementation pipeline. Selecting which methods to use requires testing on the corpus intended for the domain adaptation. Based on what optimized retrieval performance for the CHARM dataset, we propose a system which utilizes GPT 3.5-turbo for title summarizations and hard negative mining, and performs fine-tuning of the embedding model in a staged manner with Multiple Negatives Ranking and Online Contrastive loss.

Overall, domain adaptation of a retrieval system is a challenging but rewarding task. If done correctly, retrieval performance can be dramatically boosted across technical and domain-specific fields.

References

[1] Humza Naveed et al. “A Comprehensive Overview of Large Language Models”. In: *arXiv preprint arXiv:2307.06435* (2023). URL: `https://ar5iv.labs.arxiv.org/html/2307.06435`.

[2] Microsoft Research AI4Science and Microsoft Azure Quantum. “The Impact of Large Language Models on Scientific Discovery: A Preliminary Study using GPT-4”. In: *arXiv preprint arXiv:2311.07361* (2023). URL: `https://ar5iv.labs.arxiv.org/html/2311.07361`.

[3] Lei Huang et al. “A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions”. In: *arXiv preprint arXiv:2311.05232* (2023). URL: `https://ar5iv.labs.arxiv.org/html/2311.05232`.

[4] Patrick Lewis et al. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. In: *arXiv preprint arXiv:2005.11401* (2020). URL: `https://arxiv.org/abs/2005.11401`.

[5] Patrick Lewis et al. “Improving the Domain Adaptation of Retrieval-Augmented Generation (RAG) for Knowledge-Intensive Tasks”. In: *Transactions of the Association for Computational Linguistics* (2023). URL: `https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00530/114590/Improving-the-Domain-Adaptation-of-Retrieval`.

[6] Operation CHARM. *Charm*. `https://charm.li/`, Accessed: 2024-05-17.

[7] Kexin Wang et al. “GPL: Generative Pseudo Labeling for Unsupervised Domain Adaptation of Dense Retrieval”. In: *arXiv* 2112.07577 (2022). URL: `https://arxiv.org/pdf/2112.07577`.

[8] Lee Xiong et al. “Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval”. In: *arXiv* 2007.00808 (2020). URL: `https://arxiv.org/pdf/2007.00808`.

[9] Sheng-Chieh Lin et al. “How to Train Your DRAGON: Diverse Augmentation Towards Generalizable Dense Retrieval”. In: *arXiv* 2302.07452 (2023). URL: `https://arxiv.org/pdf/2302.07452`.

[10] Hansi Zeng et al. "Curriculum Learning for Dense Retrieval Distillation". In: *arXiv* 2204.13679 (2022). URL: https://arxiv.org/pdf/2204.13679.

[11] Jakub Lála et al. "PaperQA: Retrieval-Augmented Generative Agent for Scientific Research". In: *arXiv preprint arXiv:2312.07559* (2023). URL: https://ar5iv.labs.arxiv.org/html/2312.07559.

[12] Ashish Vaswani et al. "Attention is All You Need". In: *Advances in Neural Information Processing Systems* 30 (2017). URL: https://arxiv.org/abs/1706.03762.

[13] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint arXiv:1810.04805* (2018). URL: https://arxiv.org/abs/1810.04805.

[14] Nils Reimers. *SentenceTransformers Documentation*. https://sbert.net/, Accessed: 2024-05-21.

[15] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *arXiv preprint arXiv:1908.10084* (2019). URL: https://arxiv.org/abs/1908.10084.

[16] LlamaIndex Documentation. *Token Text Splitter Documentation*. https://docs.llamaindex.ai/en/stable/api_reference/node_parsers/token_text_splitter/, Accessed: 2024-05-21.

[17] LlamaIndex. *Sentence splitter*. https://docs.llamaindex.ai/en/stable/api_reference/node_parsers/sentence_splitter/, Accessed: 2024-05-22.

[18] Greg Kamradt. *The 5 Levels Of Text Splitting For Retrieval*. https://www.youtube.com/watch?v=8OJC21T2SL4&t=1933s, Accessed: 2024-05-21. 2024.

[19] LlamaIndex. *Semantic splitter*. https://docs.llamaindex.ai/en/stable/api_reference/node_parsers/semantic_splitter/, Accessed: 2024-05-22.

[20] Yongchao Wu. "Few-shot Question Generation with Prompt-based Learning". In: *DiVA Portal* (2024). URL: https://www.diva-portal.org/smash/get/diva2:1696701/FULLTEXT01.

[21] Rubaba Amyeen. "Prompt-Engineering and Transformer-based Question Generation and Evaluation". In: *arXiv* (2023). URL: https://arxiv.org/pdf/2310.18867.pdf.

[22] Yue et al. "Synthetic Query Generation for Privacy-Preserving Deep Retrieval Systems using Differentially Private Language Models". In: *arXiv* 2305.05973 (2024). URL: https://arxiv.org/pdf/2305.05973.pdf.

[23] OpenAI. *Six Strategies for Getting Better Results from Prompt Engineering*. https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results, Accessed: 2024-05-21.

[24] Prannay Khosla et al. "Supervised Contrastive Learning". In: *NeurIPS* (2020). URL: https://proceedings.neurips.cc/paper/2020/file/d89a66c7c80a29b1bdbab0f2a1a94af8-Paper.pdf.

[25] Panagiotis Antoniadis. *An Introduction to Contrastive Learning*. https://www.baeldung.com/cs/contrastive-learning, Accessed: 2024-05-21.

[26] Hugging Face. "Fine-tune a Pretrained Model". In: *Hugging Face Documentation* (2023). URL: `https://huggingface.co/docs/transformers/training`.

[27] Liang Wang et al. "Improving Text Embeddings with Large Language Models". In: *arXiv* 2401.00368 (2024). URL: `https://arxiv.org/pdf/2401.00368.pdf`.

[28] Reimers, Gurevych. *Losses*. `https://www.sbert.net/docs/package_reference/losses.html`, Accessed: 2024-05-08.

[29] Reimers, Gurevych. *MultipleNegativesRankingLoss*. `https://www.sbert.net/docs/package_reference/losses.html#multiplenegativesrankingloss`, Accessed: 2024-05-22.

[30] Reimers, Gurevych. *OnlineContrastiveLoss*. `https://www.sbert.net/docs/package_reference/losses.html#onlinecontrastiveloss`, Accessed: 2024-05-22.

[31] Reimers, Gurevych. *MarginMSELoss*. `https://www.sbert.net/docs/package_reference/losses.html#marginmseloss`, Accessed: 2024-05-22.

[32] Biewald, Van Pelt and Lewis. *Weights and Biases*. `https://wandb.ai/`, Accessed: 2024-05-08.

[33] Biewald, Van Pelt and Lewis. *Tune Hyperparameters*. `https://docs.wandb.ai/guides/sweeps`, Accessed: 2024-05-08.

[34] PyTorch. *AdamW*. `https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html`, Accessed: 2024-05-21.

[35] IBM. *What is zero-shot learning?* `https://www.ibm.com/topics/zero-shot-learning`, Accessed: 2024-05-20.

[36] Meta AI. *Faiss*. `https://ai.meta.com/tools/faiss/`, Accessed: 2024-05-07.

[37] Penn State University. *4.3 - Introduction to Bootstrapping*. `https://online.stat.psu.edu/stat200/lesson/4/4.3`, Accessed: 2024-05-21.

[38] OpenAI. *GPT-3.5 Turbo*. `https://platform.openai.com/docs/models/gpt-3-5-turbo`, Accessed: 2024-05-07.

[39] Inc. Hugging Face. *Hugging Face*. `https://huggingface.co/`, Accessed: 2024-05-21.

[40] Hugging Face. *mteb/leaderboard*. `https://huggingface.co/spaces/mteb/leaderboard`, Accessed: 2024-05-07.

[41] Beijing Academy of Artificial Intelligence. *BAAI/bge-small-en-v1.5*. `https://huggingface.co/BAAI/bge-small-en-v1.5`, Accessed: 2024-05-08.

[42] Beijing Academy of Artificial Intelligence. *BAAI/bge-large-en-v1.5*. `https://huggingface.co/BAAI/bge-large-en-v1.5`, Accessed: 2024-05-08.

[43] Beijing Academy of Artificial Intelligence. *BAAI/bge-large-en-v1.5*. `https://huggingface.co/BAAI/bge-reranker-base`, Accessed: 2024-05-08.

[44] Reimers, Gurevych. *ContrastiveLoss*. `https://www.sbert.net/docs/package_reference/losses.html#contrastiveloss`, Accessed: 2024-05-22.

[45] Mistral AI. *Mixtral of Experts*. `https://mistral.ai/news/mixtral-of-experts/`, Accessed: 2024-05-20.

[46] Dean Shorak. *RAG: Unlocking Business Innovation with Retrieval-Augmented Generation.* https://medium.com/@deanshorak/rag-unlocking-business/innovation-with-retrieval-augmented-generation-a8e15c3e3667, Accessed: 2024-05-20.

[47] Microsoft. *Implement Retrieval Augmented Generation (RAG) with Azure OpenAI Service.* https://learn.microsoft.com/en-us/training/modules/use-own-data-azure-openai/, Accessed: 2024-05-21.

[48] Storm Reply. *Retrieval Augmented Generation (RAG) on AWS.* https://medium.com/storm-reply/retrieval-augmented-generation-rag-on-aws-68a0738915b2, Accessed: 2024-05-21.

[49] Amazon Web Services. *What is Retrieval-Augmented Generation?* https://aws.amazon.com/what-is/retrieval-augmented-generation/, Accessed: 2024-05-21.

Appendices

Appendix A
Division of Work

The contributions to this thesis were equally shared between the authors, Lycke Fureby and Filippa Hansen. The project development tasks were distributed evenly, ensuring that each author had the opportunity to review and potentially modify the code written by the other. This reciprocal review process helped maintain a high standard of work and facilitated a comprehensive understanding of the entire project by both authors.

The writing responsibilities for the thesis report were similarly divided. Specific sections and chapters were initially drafted by one author and subsequently extended or revised by the other. This collaborative approach not only ensured equal participation but also provided each author with a thorough insight into all aspects of the thesis, promoting a well-rounded and cohesive final document.

Appendix B
Supporting tables

Embedding Model	**Hit Rate @10**	**MRR @10**	**NDCG @10**
Bge-small-en-v1.5, base model	0.605	0.325	0.392
Bge-large-en-v1.5, base model	0.647	0.343	0.416
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 1 car model	0.672	0.367	0.439
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 2 car models	0.677	0.382	0.452
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 3 car models	0.681	0.382	0.453
Bge-small, Multiple Negatives Ranking loss, 5000 training samples with 3 car models	0.679	0.377	0.448
Bge-small, Multiple Negatives Ranking loss, 7500 training samples with 3 car models	0.694	0.384	0.458
Bge-small, Multiple Negatives Ranking loss, 15000 training samples with 6 car models	0.699	0.398	0.470
Bge-small, Multiple Negatives Ranking Loss and Online Contrastive Loss with LLM mined P-, 7500 training samples with 3 car models	0.708	0.393	0.468

Table B.1: Overview of Model Performance Metrics averaged over all queries in the **Ford** Synthetic Test Dataset.

Embedding Model	Hit Rate @10	MRR @10	NDCG @10
Bge-small-en-v1.5, base model	0.680	0.400	0.467
Bge-large-en-v1.5, base model	0.686	0.409	0.475
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 1 car model	0.712	0.444	0.509
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 2 car models	0.727	0.447	0.514
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 3 car models	0.730	0.457	0.523
Bge-small, Multiple Negatives Ranking loss, 5000 training samples with 3 car models	0.718	0.454	0.518
Bge-small, Multiple Negatives Ranking loss, 7500 training samples with 3 car models	0.736	0.459	0.525
Bge-small, Multiple Negatives Ranking loss, 15000 training samples with 6 car models	0.744	0.457	0.528
Bge-small, Multiple Negatives Ranking Loss and Online Contrastive Loss with LLM mined P-, 7500 training samples with 3 car models	0.763	0.461	0.531

Table B.2: Overview of Model Performance Metrics averaged over all queries in the **Jaguar** Synthetic Test Dataset.

Embedding Model	Hit Rate @10	MRR @10	NDCG @10
Bge-small-en-v1.5, base model	0.686	0.408	0.474
Bge-large-en-v1.5, base model	0.695	0.422	0.487
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 1 car model	0.763	0.486	0.553
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 2 car models	0.774	0.494	0.561
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 3 car models	0.773	0.494	0.561
Bge-small, Multiple Negatives Ranking loss, 5000 training samples with 3 car models	0.775	0.495	0.562
Bge-small, Multiple Negatives Ranking loss, 7500 training samples with 3 car models	0.787	0.505	0.573
Bge-small, Multiple Negatives Ranking loss, 15000 training samples with 6 car models	0.794	0.520	0.586
Bge-small, Multiple Negatives Ranking Loss and Online Contrastive Loss with LLM mined P-, 7500 training samples with 3 car models	0.809	0.520	0.589

Table B.3: Overview of Model Performance Metrics averaged over all queries in the **Saab 2004** Synthetic Test Dataset.

Embedding Model	Hit Rate @10	MRR @10	NDCG @10
Bge-small-en-v1.5, base model	0.490	0.270	0.322
Bge-large-en-v1.5, base model	0.500	0.276	0.330
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 1 car model	0.540	0.297	0.352
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 2 car models	0.580	0.309	0.371
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 3 car models	0.580	0.309	0.373
Bge-small, Multiple Negatives Ranking loss, 5000 training samples with 3 car models	0.590	0.316	0.381
Bge-small, Multiple Negatives Ranking loss, 7500 training samples with 3 car models	0.600	0.329	0.393
Bge-small, Multiple Negatives Ranking loss, 15000 training samples with 6 car models	0.610	0.357	0.417
Bge-small, Multiple Negatives Ranking Loss and Online Contrastive Loss with LLM mined P-, 7500 training samples with 3 car models	0.590	0.316	0.381

Table B.4: Overview of Model Performance Metrics averaged over all queries in the human annotated Jaguar Test Dataset.

Embedding Model	Hit Rate @10	MRR @10	NDCG @10
Bge-small-en-v1.5, base model	0.660	0.466	0.512
Bge-large-en-v1.5, base model	0.720	0.490	0.544
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 1 car model	0.840	0.575	0.639
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 2 car models	0.850	0.586	0.651
Bge-small, Multiple Negatives Ranking loss, 2500 training samples with 3 car models	0.850	0.596	0.658
Bge-small, Multiple Negatives Ranking loss, 5000 training samples with 3 car models	0.870	0.603	0.667
Bge-small, Multiple Negatives Ranking loss, 7500 training samples with 3 car models	0.860	0.602	0.664
Bge-small, Multiple Negatives Ranking loss, 15000 training samples with 6 car models	0.860	0.607	0.669
Bge-small, Multiple Negatives Ranking Loss and Online Contrastive Loss with LLM mined P-, 7500 training samples with 3 car models	0.890	0.639	0.700

Table B.5: Overview of Model Performance Metrics averaged over all queries in the human annotated Saab 2004 Test Dataset.

Appendix C
Example Queries

C.1 30 Examples of Human Annotated Queries

1. My front door window makes a loud noise when going up or down, what could be causing this issue?
2. My rear view mirror keeps coming loose, is there a specific fix for this issue?
3. I have performed specific gravity testing on my Jaguar. How do I know if my Jaguar's battery needs to be replaced based on the results?
4. I noticed that my portable cellular phone holder is damaged, could this be causing issues with my phone's connectivity in the car?
5. My Jaguar's remote key fob is not locking the vehicle, but it locks fine with the key. What could be the issue?
6. How can I troubleshoot issues with my Jaguar's ignition system if the keyless entry remote fob is not locking the vehicle properly?
7. My front seat is making a loud noise when adjusting, what could cause this?
8. How often should the air filter element in the air cleaner housing be replaced in my Jaguar?
9. Can you tell me how to identify and locate the control module pins in my Jaguar V8 engine for troubleshooting purposes?
10. I have completed the repair process for the heater control valve in my Jaguar. How can I ensure proper reinstallation of the dual coolant flow valve?

11. I hear a strange noise coming from my steering column when I turn the wheel, what could this be related to?
12. My luggage compartment lid is difficult to open, what could be causing this issue and how can it be resolved?
13. How do I safely remove the engine coolant temperature sensor without risking personal injury from coolant scalding?
14. I've been experiencing a persistent vibration in my steering wheel at high speeds. Does the front suspension forward mounting bush need replacement?
15. How can I ensure that the main bearing cap is properly installed and tightened to specification in the engine cylinder block assembly with crankshaft for my Jaguar?
16. My engine is making a strange noise when I accelerate, how do I know if this is related to a mechanical or electrical issue in the engine, cooling, or exhaust system?
17. Before proceeding with the repair for harsh or erratic shifting in my Jaguar, how can I ensure that the transmission fluid is clean and free of debris?
18. How can I ensure that the battery lead to the rear body ground point fixing bolt is properly checked and tightened?
19. Stalling issues occur after filling up the tank. How can I check if the float level valve in my fuel tank is defective?
20. I want to prevent stalling during engine performance checks. How can I ensure that the connectors for the crank position sensor are properly secured and not damaged?
21. My luggage compartment lid is still difficult to open after following the steps in the Technical Service Bulletin. What could be causing this issue?
22. When driving, I have driveability issues such as stalling. How can I ensure that the connections at the secondary junction box are properly secured?
23. My car stalls immediately after filling up the fuel tank. What could cause this?
24. What should I do if the ABS warning lamp is illuminated in my Jaguar? Is my ABS system disabled?
25. Why is my ABS light constantly on? Is it related to the wheel speed sensors?
26. Is there excessive wear on the camshaft lobes of my Jaguar's engine?
27. How do I ensure proper alignment after replacing the front axle crossmember in my Jaguar?
28. Is the warning lamp for the Parking Assist System inoperative on my Jaguar vehicle?
29. Provide instructions on how to troubleshoot a parking assist system with an inoperative warning lamp on my Jaguar.
30. I hear a strange noise coming from the rear of my Jaguar when driving at high speeds, what could cause this?

C.2 30 Examples of Synthetic Queries before Path Summarization

1. How does understanding the relevant circuits in a schematic diagram help in troubleshooting diagnosis for the Hyundai Accent L4-1.6L?

2. What information can be found on the last page of a schematic diagram for the Hyundai Accent L4-1.6L?

3. In the context of the Hyundai Accent repair and diagnosis manual, what is the purpose of the S̈ymbols Part 1s̈ection? How does it assist in understanding the diagrams and instructions related to the sunroof and moonroof?

4. Explain the significance of the S̈ymbols Part 2s̈ection in the context of the Hyundai Accent repair and diagnosis manual. How does it contribute to the comprehension of the diagrams and instructions pertaining to the roof and associated components?

5. What are the steps involved in replacing the seat back frame in a Hyundai Accent 2008? Include the necessary precautions and service procedures relevant to the airbag.

6. What is the recommended torque specification for the drive plate bolt on a 2008 Hyundai Accent L4-1.6L with an automatic transmission?

7. HHow many kilograms-force meters (kgf-m) are equivalent to the specified torque range for the drive plate bolt on a 2008 Hyundai Accent L4-1.6L?

8. What is the recommended torque range for tightening the flywheel bolt on a 2008 Hyundai Accent L4-1.6L with a manual transmission?

9. Convert the torque range for the flywheel bolt on a 2008 Hyundai Accent L4-1.6L with a manual transmission from Newton meters to foot-pounds.

10. What are the tightening torques for the engine control system and the fuel delivery system in the 2008 Hyundai Accent L4-1.6L?

11. Can you provide the specifications for the fuel delivery and air induction system in the 2008 Hyundai Accent L4-1.6L?

12. How does the idle speed in D-range differ from the idle speed in neutral or P-range for the Hyundai Accent L4-1.6L?

13. What is the purpose of the Positive Crankcase Ventilation (PCV) valve in a Hyundai Accent L4-1.6L engine? Explain its function and how it contributes to the overall performance of the engine.

14. Provide the specific tightening torque value for the Positive Crankcase Ventilation (PCV) valve in a Hyundai Accent L4-1.6L engine. Explain why it is important to adhere to the recommended torque value when installing or replacing the PCV valve.

15. Which two brands of spark plugs are mentioned in the specifications for the 2008 Hyundai Accent L4-1.6L?
16. What is the purpose of removing the cluster facia panel in the Hyundai Accent L4-1.6L 2008 model during the testing and inspection of the vehicle information display?
17. How can an ohmmeter be used to test and inspect the continuity between terminals when operating the switch in the instrument panel of the Hyundai Accent L4-1.6L 2008 model?
18. What is the specific area of the vehicle that is covered under the S̈teerings̈ection of the repair and diagnosis specifications for the 2008 Hyundai Accent L4-1.6L?
19. What is the purpose of tightening torque in the context of mechanical specifications for the Hyundai Accent L4-1.6L? How does it affect the steering system?
20. What type of specifications are included in the Ẅheels and Tiress̈ection of the document?
21. What is the standard compression pressure for the Hyundai Accent L4-1.6L engine at 250-400 rpm?
22. Explain the significance of the D̈iagnostic Aidss̈ection in the document and how it can assist in troubleshooting issues related to the Instrument Panel, Gauges, and Warning Indicators in the Hyundai Accent.
23. What is the acceptable range for the amount of refrigerant (R-134a) in the Hyundai Accent L4-1.6L?
24. Can a Hyundai Accent L4-1.6L use any other type of automatic transmission fluid apart from Hyundai Genuine Diamond ATF SP-III or SK ATF SP-III?
25. In the context of the document, what specific information can be found under the F̈luid Type Specificationss̈ection for the brakes and traction control system of a 2008 Hyundai Accent L4-1.6L?
26. What is the purpose of the S̈ymbolss̈ection in the context information? How does it relate to the repair and diagnosis of the fuel gauge in the Hyundai Accent?
27. What are the steps involved in diagnosing and repairing the MIL ON issue with the Hyundai Accent L4-1.6L related to DTCs P0705, P0707, and P0708?
28. What are the specific diagnostic trouble codes (DTCs) that may trigger the check engine light in some 2007-2009 Hyundai Accent vehicles, and what are the associated issues with each code?
29. How can the Terminal with lead wire kit (TRK011) be used to repair a damaged connector in the instrument panel of the Hyundai Accent L4-1.6L?
30. Explain the significance of the load label FMVSS 110 compliance for the Hyundai 2008 Accent L4-1.6L and how it is addressed in the superceded TSB number 10-GI-001 and the updated TSB number 13-GI-001.

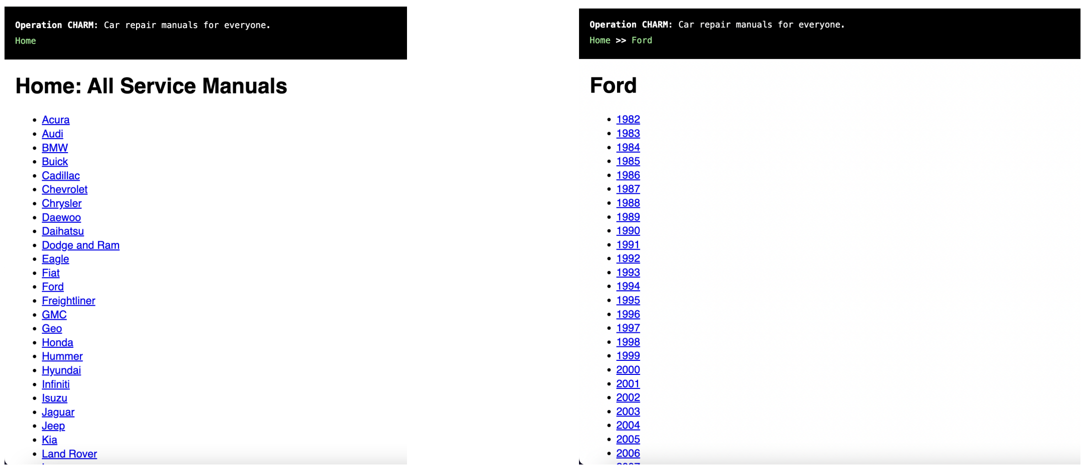
C.3 30 Examples of Synthetic Queries after Path Summarization

1. How can I use a short finder to locate a short circuit in my Hyundai's fuel injector system?
2. How do I remove the sunroof glass on my Hyundai for service and repair?
3. Why is E85 colored red and how does this help distinguish it from pure petrol?
4. How does the RRDM control the rear right-hand window lift in a Saab car?
5. How do I remove the front seat assembly in order to replace the seat belt system?
6. Where are the front impact sensors located in my Hyundai car?
7. How do I reconnect the electrical connectors to the door control module during installation?
8. How can I test for voltage in a circuit without having to separate the connector halves?
9. How can I identify the position number, color code, and cross-sectional area of a cable in my Saab car?
10. Can you explain the symbols used for sensor outputs in the wiring diagrams for lighting and horns in my Saab car?
11. How can I verify if the power door lock/unlock switches are functional on my Chrystler car?
12. How can I test the readings for the Driver's Door Module (702D)?
13. Can the left multi-function switch for lighting and horns be adjusted or repaired, or does the entire unit need to be replaced if there is an issue?
14. How can I determine if the radiator cooling fan motor in my Saab is functioning properly using On Board Diagnostics?
15. How can I tell if my Hyundai vehicle has a High-Line or Low-Line TPMS system?
16. What wire color code should I look for in the wiring diagrams for my Saab's automatic transmission and transaxle system?
17. How can I troubleshoot a remote start inhibitor issue on my Chrysler vehicle?
18. How does the Powertrain Management Computer monitor the efficiency of the catalytic converter in my Chrystler vehicle?
19. How can I verify the accuracy of my car's cooling system complaints before beginning any disassembly or testing?

20. How can I demonstrate the trouble on my car to ensure there are no misunderstandings during fault diagnosis?
21. How can I determine if the function of my car's windows and glass is correct or faulty?
22. How should I handle the control module to prevent damage to internal components during dismantling?
23. What type of detergent or soap should I use to clean my Hyundai's aluminum wheels to avoid surface damage?
24. How can I verify the accuracy of my car's electrical system complaints before beginning any disassembly or testing?
25. How can I determine if the "OUTPUT SPEED SENSOR(PG-B)" signal value is changing according to simulation frequency in my Hyundai car?
26. How can I determine if there is a problem with the O2 Sensor signal circuit in my Chrystler car?
27. How do I clear any DTC's that may have been set in other modules after reprogramming the PCM in my Chrystler vehicle?
28. Can I use synthetic oil in my Saab with a 6-speed AF40 transaxle automatic transmission?
29. How does the control module detect errors in the ignition system and generate diagnostic trouble codes?
30. How do I know if my Hyundai battery needs to be replaced or just charged using the GR8 diagnostic tool?

Appendix D

Operation CHARM Raw Data


Operation CHARM: Car repair manuals for everyone.
Home

Home: All Service Manuals

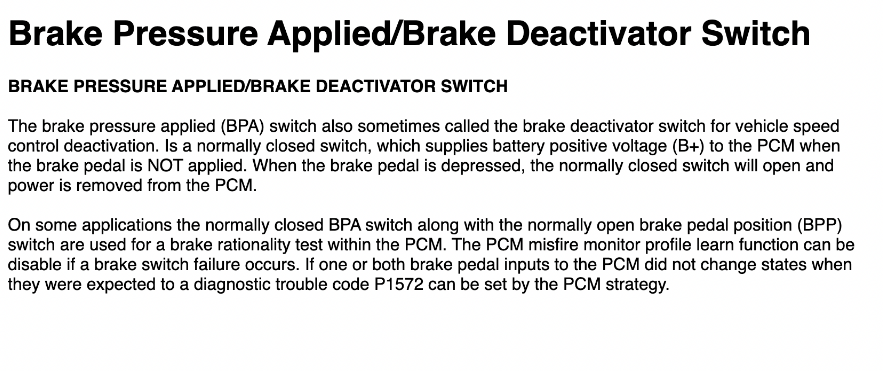
- Acura
- Audi
- BMW
- Buick
- Cadillac
- Chevrolet
- Chrysler
- Daewoo
- Daihatsu
- Dodge and Ram
- Eagle
- Fiat
- Ford
- Freightliner
- GMC
- Geo
- Honda
- Hummer
- Hyundai
- Infiniti
- Isuzu
- Jaguar
- Jeep
- Kia
- Land Rover

Operation CHARM: Car repair manuals for everyone.
Home >> Ford

Ford

- 1982
- 1983
- 1984
- 1985
- 1986
- 1987
- 1988
- 1989
- 1990
- 1991
- 1992
- 1993
- 1994
- 1995
- 1996
- 1997
- 1998
- 1999
- 2000
- 2001
- 2002
- 2003
- 2004
- 2005
- 2006

Figure D.1: Operation CHARM website structure



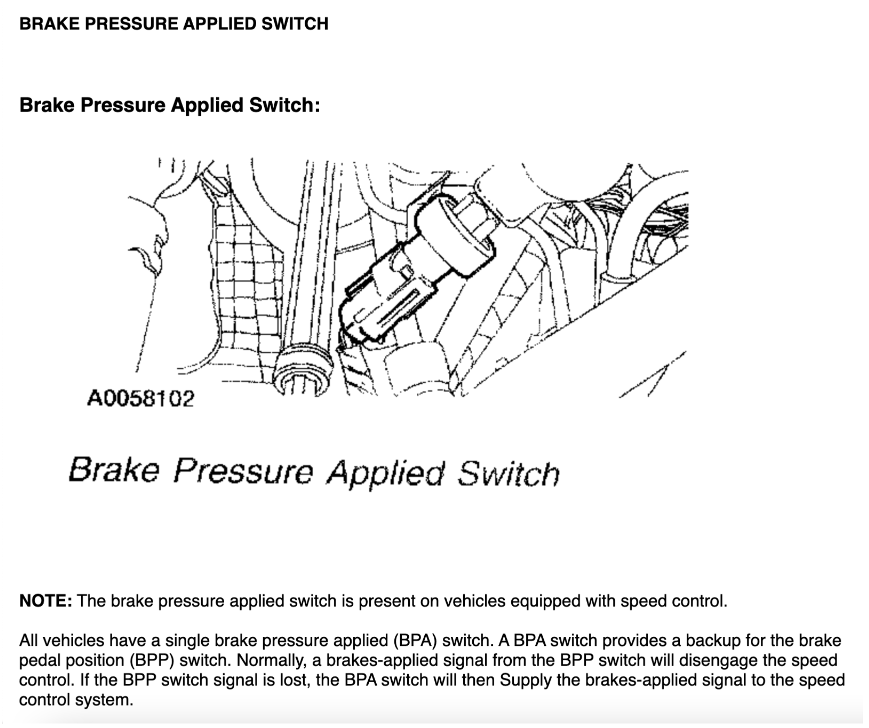
Brake Pressure Applied/Brake Deactivator Switch

BRAKE PRESSURE APPLIED/BRAKE DEACTIVATOR SWITCH

The brake pressure applied (BPA) switch also sometimes called the brake deactivator switch for vehicle speed control deactivation. Is a normally closed switch, which supplies battery positive voltage (B+) to the PCM when the brake pedal is NOT applied. When the brake pedal is depressed, the normally closed switch will open and power is removed from the PCM.

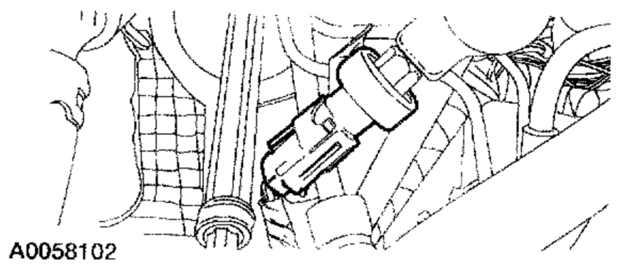
On some applications the normally closed BPA switch along with the normally open brake pedal position (BPP) switch are used for a brake rationality test within the PCM. The PCM misfire monitor profile learn function can be disable if a brake switch failure occurs. If one or both brake pedal inputs to the PCM did not change states when they were expected to a diagnostic trouble code P1572 can be set by the PCM strategy.

Figure D.2: Operation CHARM example document.



BRAKE PRESSURE APPLIED SWITCH

Brake Pressure Applied Switch:



Brake Pressure Applied Switch

NOTE: The brake pressure applied switch is present on vehicles equipped with speed control.

All vehicles have a single brake pressure applied (BPA) switch. A BPA switch provides a backup for the brake pedal position (BPP) switch. Normally, a brakes-applied signal from the BPP switch will disengage the speed control. If the BPP switch signal is lost, the BPA switch will then Supply the brakes-applied signal to the speed control system.

Figure D.3: Operation CHARM example document. Note the sketched figure, which is excluded when the data is scraped.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2024-05-03

EXAMENSARBETE Domain Adaptation of Retrieval Systems from Unlabeled Corpora
STUDENTER Lycke Fureby, Filippa Hansen
HANDLEDARE Patrik Edén (LTH)
EXAMINATOR Elin A. Topp (LTH)

Domänanpassning av söksystem för RAG-applikationer

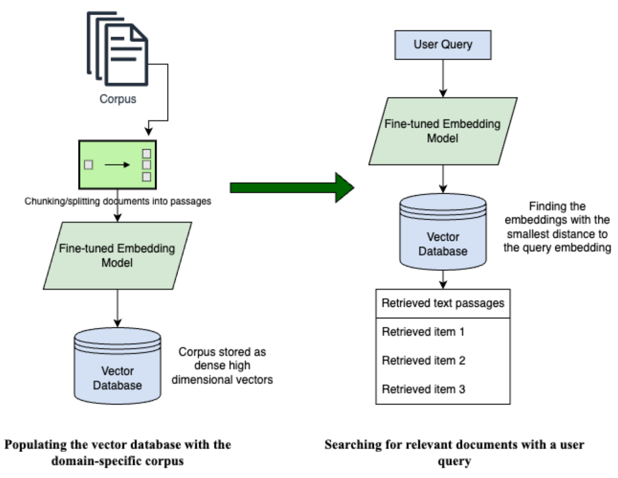
POPULÄRVETENSKAPLIG SAMMANFATTNING **Lycke Fureby, Filippa Hansen**

Under de senaste åren har stora språkmodeller, med ChatGPT i spetsen, slagit igenom på allvar. Nu tar även Retrieval Augmented Generation (RAG) plats i rampljuset. Detta arbete har undersökt vilka tekniker som kan användas för att domänanpassa söksystemen som är en del av RAG.

I takt med att stora språkmodeller blivit allt mer förekommande har även kraven på att säkerställa att de svar som genereras är faktabaserade och aktuella blivit hårdare. Att informationen är spårbar och relevant väger extra tungt när avsedd användning är inom områden som medicin eller juridik. Stora språkmodeller har en tendens att hallucinera, det vill säga producera svar vars innehåll inte stämmer överens med verkligheten. Vidare är det mycket kostsamt att uppdatera med aktuell information, vilket resulterar i att modellernas kunskapsbas är frusna i tiden.

RAG är en metod som kombinerar generativa stora språkmodeller med avancerad hämtning av aktuell och spårbar information. Embedding-modeller ligger till grund för denna typ av system, vilket möjliggör att "mänsklig" förståelse snarare än lexikon-baserad sökning används för att hitta relevanta dokument i en databas. Embedding-modeller producerar imponerande resultat inom många områden, men prestandan blir sämre då texten är mycket domänspecifik.

Vi har i vårt examensarbete undersökt vilka tekniker som kan användas för att öka relevansen hos hämtade dokument från domänspecifika, omärkta dataset. Arbetet innefattar generering av syntetiska frågor med hjälp av en stor språkmodell, vilka sedan har använts för att träna en embedding-modell inom ett domänspecifikt område. Ytterligare metoder innefattar Hard Negative Mining för kontrastiv inlärning, där embedding-modellen tränas omvänt på dokument som är väldigt lika det relevanta dokumentet men som inte svarar på den tillhandahållna frågan.


Corpus
Chunking/splitting documents into passages
Fine-tuned Embedding Model
Vector Database
Corpus stored as dense high dimensional vectors
Populating the vector database with the domain-specific corpus
User Query
Fine-tuned Embedding Model
Vector Database
Finding the embeddings with the smallest distance to the query embedding
Retrieved text passages
Retrieved item 1
Retrieved item 2
Retrieved item 3
Searching for relevant documents with a user query

Systemet utvärderades på manuellt skrivna frågor, där resultaten visar en tydlig förbättring i prestanda efter att träning på syntetisk data genomförts.