MSc Artificial Intelligence
Master Thesis

Semantic Search & Large Language Modeling at Albert Heijn

by
Mitchell Verhaar
11239069

September 25, 2023

48
08-12-22 / 26-09-23

Supervisor:
Dr. F. Sarvi

Examiner:
Prof. M. Aliannejadi

Second reader:
Dr. F. Sarvi

Universiteit van Amsterdam

Contents

Abstract

This research examines the possibility & feasibility of applying a Large Language Model (LLM) in the search architecture within the grocery shopping domain of Albert Heijn (AH), in order to improve the quality of the company's search engine. Through the creation of a custom data set, based on the data that AH provides, as well as experiments on the tokenizer from the model we fine-tuned the Conditional Masked Language Model (CMLM) Yang et al. (2020) in several different set ups to improve the quality of product search. We analyzed the resulting performance of the LLM through various retrieval metrics like Precision and Recall. With these results we concluded that the model under performs when compared to the currently productionized standard from AH. We discovered that the data was too sparse and the gap between query and product length cannot be crossed by the model. Additionally, the model is fine-tuned with less products than the standard from AH which impacts the model negatively. Lastly, we defined a set of specific queries to highlight the differences between the model instances in this research and manually computed the Precision for these queries. These Precision results showed the key differences in performance between the models and highlighted which model is the best fit for product search.

Chapter 1

Introduction

Albert Heijn (AH) is one of the biggest grocery stores in The Netherlands and Belgium. In 2022 the company profited from a remarkable 37% market share in the grocery domain of the Netherlands. This is an increase of 1.3% compared to the year prior[1]. The company, governed by the parent company Ahold Delhaize, offers a vast range of grocery products in hundreds of stores across the country. In addition, AH also offers an online e-commerce retail platform through their app and website. This online e-commerce platform requires a powerful search engine on their back end as customers need to be able to find the products that they are looking for.

The current implementation of the search system has shown some unreliable results when trying to use queries that require more than just the textual information. Examples of queries requiring more information are "chocolade zonder hazelnoot" (translation: "Chocolate without hazelnut"), that still contains hazelnut chocolate in the results, or "zout" (translation: "salt") which returns products for "zoutjes" (translation: "salty snacks") above table salt in the product order. These queries produce incorrect results as the current model is not capable of capturing such *semantic* information from the grocery shopping domain. Other non-grocery shopping companies that deal with e-commerce data, such as Spotify[2] or H&M[3] are researching ways to provide the best search service to their customers as well, through methods such as Large Language Models (LLM). Large Language Models are shown to be effective when applying them to e-commerce (Peeters et al., 2020). Their capability of processing human readable text has been incorporated to successfully perform semantic matching of queries with products. Because of these advancements, we use this research opportunity to validate this phenomenon.

Research.

In order to investigate the aforementioned phenomenon we re-purpose the Conditional Masked Language Model (CMLM) to fine-tune on the data that AH provides. Specifically, the set up includes the encoders of the Conditional Masked Language Model (CMLM) (Yang et al., 2020) and will be able to match queries to products based on semantic similarity amongst the two. By incorporating this architecture the following research question will be answered: "How can a Large Language Model (LLM) be adapted and extended to improve the search domain in the e-commerce of Albert Heijn?". In order to further tackle this question, several sub-questions have been proposed.

[1] `https://nieuws.ah.nl/albert-heijn-sluit-bijzonder-jaar-af-met-370-marktaandeel/`

[2] `https://engineering.atspotify.com/2022/03/introducing-natural-language-search-for-podcast-episodes/`

[3] `https://hyperight.com/fashion2vec-jonas-grunditz-hm-group/`

First, a suitable data set is needed for the LLM to fine-tune on. AH possesses an enormous amount of raw data with several topics, such as product data, click data etc. However, this data is still in a raw form and requires processing to transform into a publicly usable data set. Furthermore, some qualities in Albert Heijn's data, such as sparsity or query length, could be difficult due to the specific ecosystem of e-commerce grocery shopping that AH resides in. This leads to the following sub-question: "What are the key characteristics and potential challenges that describe the data from AH?"

The second sub-question focuses on the model tokenization. Tokenization is the model's process of mapping a given input sentence to specific known tokens by translating the individual words of the sentence. This is often necessary in order to streamline the different inputs that the model will face. However, the current set of tokens may not be fully encapsulating some of the key tokens in the data that AH provides and might need extra domain specific tokens to be added to the collection. Therefore, we aim to answer the question: "How do the current and expanded set of tokens each impact the performance of the model within the search domain of AH?"

Finally, the third sub-question entails the model itself. By modifying the input that encoders of the CMLM receive we can create different instances of the model, which are implemented into AH's pipeline. Therefore, this sub-question focuses on the feasibility of the model output. By measuring the performance of the model with retrieval benchmarks such as Precision or Recall we aim to answer the following question: "How can the use of Large Language Models in product matching improve the performance of AH search engine?"

We hypothesize that the matching of search queries with products can be improved using a Large Language Model. By utilizing the custom data set that is derived from this research and combining that with the different sets of tokens we expect to see an increase in model performance in comparison with the baseline, which is the current model in production that AH provides.

Contributions.

This research aims to provide various contributions to not only the Albert Heijn itself but to the entire *search in e-commerce* domain. First, the data set that the company provides will be cleaned and anonymized to provide a public version of the data. The data contains the products as well as the user queries from the original ranking algorithm output that AH employs. This new data set will also be used to compute a corresponding score for these respective query - product list combinations. The score indicates a product's relevance to the original query but has been modified to ignore positional bias and preserve the anonymity of sensitive information. This public version can and will be used in this paper as well as be made available for other researchers to incorporate into their research[4].

Secondly, we investigate the reproducibility of the CMLM model in a real-world experimental setup. We incorporate real data into our pipeline and report our results and insights. By doing so we aim to provide a valuable analysis on how Large Language Models can be used in a real environment and what the challenges of setting up such a pipeline can be. Additionally, the model is made available on Github to encourage further research in this field[5]. Lastly, we dive into the model's tokenizer and investigate whether the tokenization process can be improved by incorporating domain specific knowledge.

[4] `https://github.com/Mitchell-V/Thesis`

[5] `https://github.com/Mitchell-V/Thesis`

Chapter 2

Related work

This paper relies on previous work that has been conducted in both the e-commerce domain as well as the domain of Natural Language Processing. The papers listed in this section provide inspiration and useful models/tools that serve as a foundation on which this research is built.

The BERT language model (Devlin et al., 2018) is a language model built on the underlying Transformer architecture Vaswani et al. (2017). BERT, or Bidirectional Encoder Representations from Transformers, pre-trains deep bidirectional representations from the input text by jointly conditioning on both left and right directions of the context in all layers. This results in representations that are dependent on the entire context and not just the preceding context. One of the key defining features of BERT is the simplicity of adapting BERT to a new task or setting. Fine tuning the pre-trained version of BERT can easily be done by only adding 1 output layer to the model and then using that expanded version of BERT to train on the data set of the problem definition. This allows BERT to be applicable to a huge range of tasks or problems. BERT has already been applied in various LLM set ups successfully and is used as inspiration for this research project's set up.

An example of a successful application of BERT is the work of VILCEK et al. (2018). the author demonstrates a query to product matching system. The research uses the DeBERTa model from Microsoft (He et al., 2023) to augment the semantic distance between products based on their similarity measurements in the output of the model. By doing so, the model creates clusters of locally similar products that could be analyzed to find matching categories. The main similarity measure used in the described research is the cosine similarity measure. This research is relevant to our work, as we base the model's distance metric off the same cosine similarity metric as our distance function.

The Conditional Masked Language Model, taken from Yang et al. (2020), is a Large Language Model used to effectively learn sentence representations on large scale unlabeled corpora. The model integrates sentence representation learning into Masked Language Model training by conditioning on the encoded vectors of adjacent sentences. This allows the model to use a bidirectional context when computing the masked tokens in the target sentence.

The model itself consists of two parts. The first part uses a Transformer encoder to convert a given input sentence into a sentence vector v using pooling techniques. Afterwards, the sentence vector v is projected through a projection operation P to obtain projections of the sentence vector v_N. These projections are then fed to the second stage of the model. This part of the CMLM masks the target sentence and, by feeding the context derived from the sentence projections v_N together with the masked sentence into another Transformer encoder, computes

the correct words to fill into the masked positions of the target sentence. This set up allows the model to condition on the context of previous sentences.

Siamese network setup is a specific implementation of Neural Networks that share their weights. By sharing their weights they can learn similar representations of input sequences and is the model capable of producing similar output embeddings of each output by the individual networks. As shown in this work, the author employed two Multi-layer perceptrons in a Siamese setup with weight sharing to compute similarity between the outputs through the cosine similarity metric. These networks were capable of producing similar output for two different input samples (Chicco, 2021). Our work uses this research as inspiration to base the structure of the current model on, as our model encodes two different input sources and produces corresponding embeddings as well.

To further support our work with Siamese set ups, we draw inspiration from the work of Xie et al. (2022). In this paper, the authors perform embedding-based grocery search as this domain and its' accompanying models have seen a surge in interest. Furthermore, the researchers at Instacart developed a two tower Siamese based architecture using the *MiniLM-L3-v2* (Reimers and Gurevych, 2019) model in a pre-trained version as a baseline. The authors also implement negative in-batch sampling through various techniques such as sampling in-batch or self-adversarial negatives. The authors incorporated both the model and the evaluation into a bigger pipeline consisting of both offline and online architectures to improve the performance of the search engine at Instacart. The proposed model outperforms several other implementations across multiple retrieval benchmarks such as nDCG and Recall. The authors also visualized the embedding space, which can be seen in Figure 2.1. This research serves as inspiration into how embeddings could be visualized as well as how negative sampling can be carried out in several ways.

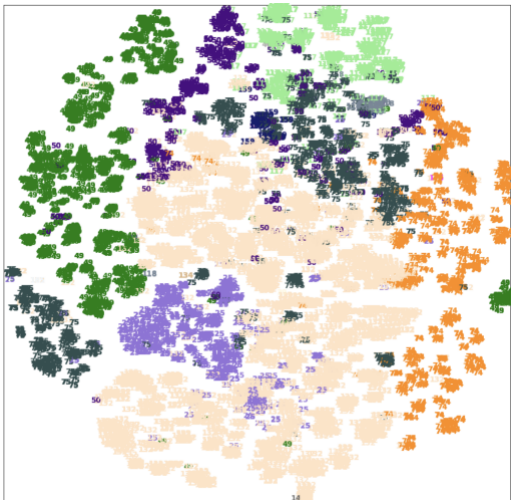

Figure 2.1: Embedding Visualization. Colored by different product categories. They sampled products from food and the light orange color represents the sub-category pantry (Xie et al., 2022).

Generative Pre-trained Transformer 3 (GPT-3) has been a wildly popular LLM that is used in a whole range of research applications (Brown et al., 2020). One of such applications is product retrieval using GPT-3. This paper has introduced a working pipeline that utilizes the search query input from a user and feeds it through GPT-3 (Kim et al., 2022). The output of this model is averaged to obtain the product categories, after which the product to category

mapping table selects candidate products from the corresponding categories. Lastly, these lists of products are fed through two separate BERT models, instantiated with Multi-layer Perceptrons (Minsky and Papert, 1988), which are trained on the queries and products, respectively. The proposed model outperforms other product retrieval models across multiple data sets and retrieval metrics, such as the author's own platform data. The authors also conclude that the cold start problem with new products is effectively handled by the addition of the GPT-3 model (Kim et al., 2022). We use this research as an example of how Large Language Models can successfully be applied with real data. Additionally, this paper serves interesting insights into how new products could be handled for the CMLM as well.

Facebook AI Similarity Search (FAISS) is a toolkit developed by Meta AI to assist with exploring semantic spaces and the vectors that lie in such spaces (Johnson et al., 2019). The toolkit allows the user to search and cluster for dense vector representations in a shared space to find the closest lying vectors that represent the given input query.[1] The toolkit builds a representation of this vector space in RAM memory by initially receiving the full list of vectors that lie in this semantic space (i.e. products from a product catalogue encoded as embeddings). Afterwards, the user can encode their input query using the same encoder model to map it to the embedding structure that the space represents. Finally, FAISS will search the space for vectors in close proximity through computing the Euclidean Distance between the vector embeddings (Johnson et al., 2019). We incorporated the FAISS toolkit in this research as a vital component in determining the model's performance as it computes the semantic similarity between the query and products from the model output.

[1] https://github.com/facebookresearch/faiss

Chapter 3

Methodology

In this section an overview of the methods, that are used during this research project, is given. The main LLM, as well as the data set that is derived from AH's sources, is discussed. The custom data set is described in-depth, accompanied by its' purpose and challenges. The data description is followed by an extensive outline of the different types of models, configurations, testing set ups and experiments.

3.1 Data

Albert Heijn possesses a vast collection of data. Part of this data is of great importance to this research project as well. For instance, consumer data consists of numerous sections, such as account information as well as purchase history and search queries of products. Other data could consist of product information provided by third-party suppliers. This product data could contain fields such as name, price, quantity etc. In this project, the main focus will rest on the click data collected from users on the search pages as well as the main product data derived from the available products that Albert Heijn offers. Using this data that AH collected allows us to produce a data set containing queries and their produced products according to their respective relevancy score. Having such a data set could be a valuable asset in training LLMs to match queries to products in product search.

3.1.1 User Interaction Data

The customers of Albert Heijn generate an enormous amount of search data when searching for products on the website or in the application. Whenever a customer starts typing a search query into the search engine of AH the engine will pass the (incomplete) input onto the current search algorithm which will try to auto complete the query and return relevant results. Each further augmentation of the search query by the user will refine the search algorithm's output. All of these individual search attempts are logged in the click data as *search-as-you-type* queries that the user ends up generating. When the user has entered their desired search query and initialize the final search, the currently used model returns the list of products that it deems relevant to the user. When these products are shown, the user can continue to execute the following logged actions: View the product's detail page, add-to-cart which will add the product to the consumer's current basket, or remove-from-cart which will remove the product from the consumer's current basket. All of these user interactions are logged in the user interaction data and are available for product relevancy analysis.

Important to note is that the user interactions are captured on the search models that AH has currently running or is A/B testing. This means that the data is influenced by the models that were employed during that data collection period. The data from both the production models as well as the A/B tests is obtained through *Google Analytics*[1]. Google Analytics stores each event that is fired during the individual searches. This includes all the auto complete events that the user generates during their search. All these events are kept in the data as processing the final query only is infeasible when each edit to the query gets processed by the auto-complete system.

Next to the search query, the click data contains numerous fields describing interactions on the output of the search algorithm as well. These fields contain the list of products returned, whether the user clicked on any of the given products and much more. For the purpose of this research project, not all data fields were taken into consideration as these fields were irrelevant to the project. In Table 3.1 the selected fields from the data set are shown.

Column	**Type**	**Example**
Search query	String	"wasmiddel"
Product ID	Integer	000001
Product's current position	Integer	04
User Interaction	String	"clicked"
Search ID	Integer	123456

Table 3.1: Overview of the fields selected from the data.

Furthermore, the last 2 months of data that the production and A/B tested models produced is taken into consideration. The main reason for this is that it is a long enough time period to obtain sufficient user data to make meaningful aggregations. Additionally the current model powering the search engine behind AH's web and app search only uses the last 2 months of data as well. Exceeding the 2 months of data could influence the results of the comparison between the current standard and this research model. In order to highlight the structure of the data set that was extracted from AH an example entry is shown in Table 3.2.

Column	**Value**
Search query	melk
Product ID	123123
Score of product adds	43210
Score of product removes	01234
Score of highest scored product in current search	50000
Score of the given product w.r.t. highest product	0.9

Table 3.2: Example mock data set entry.

Once the appropriate fields were extracted the pre-processing step continued by ensuring that the data is anonymous. Albert Heijn has requested that any publishable data should not reveal sensitive information. Sensitive information could include price history, explicit product rankings, user information and more. In order to publish the data some pre-processing was needed. First off, the data was filtered to drop any queries that did not occur frequently enough in order to provide relevant data. The limit to drop irrelevant searches was set at a 1000

[1]https://analytics.google.com/analytics/web/provision/#/provision

searches. The reason as to why 1000 was selected can be seen in Figure 3.1. In this figure the majority of queries have more than 1000 searches and therefore it was deemed as a solid cutoff point. Additionally, queries below 1000 searches are assumed to consist mostly of meaningless content.

Figure 3.1: Search frequency per distinct query.

After the cutoff point was determined the propensity score is computed on the remaining searches. The main reason as to why the propensity score is introduced is that the propensity score eliminates positional bias. Positional bias can occur if products are returned in one specific position in the ranking more often than others. By training a matching model on such a bias it will learn to rank products according to their preferable position. This will induce positional bias in the outcomes of the matching model as well. The propensity score eliminates the positional bias through the use of the following computations:

1. Determine how often a given product is added to a basket by a user.
2. Determine how often a product is removed from a basket by a user.
3. Subtract the number of removes from the number of adds (granted that the product has been added at least once before removal).
4. Remove products that are added to the cart less than five times (as these products are often irrelevant/noisy for the data).
5. Aggregate the result by product position to obtain the average position across all searches.
6. Normalize the aggregation by the aggregated total (corrected) add-to-carts of position one for each respective query.

After the propensity scores have been computed we weighed all the (corrected) add-to-cart ratios of the products in each query with the aforementioned propensity scores and aggregate this result to form query-product pairs with this newly defined score. Then, this score is normalized over the maximum score within the respective search query. By doing so, the query-product pairs will have an associated score that is anonymized yet still reflects the relational scores to the best product. Next to the anonymity of the scores this set up also allows

for products to be compared across the given queries. The raw scores were not able to be compared due to wide range that they can be distributed across.

Having computed these relevancy scores allows us to analyze the distribution. Figure 3.2 shows the complete distribution of average propensity scores per query and associated pairs of products across all data splits. As visible in the graph, the distribution tends to spike around $0.3 \sim 0.4$ which is where the biggest concentration of relevancy scores lies. We computed the total average for each of these data splits. This can be seen in Figure 3.2 on the right-side image; all data splits seem to average at around 0.35 as indicated by the highlighted color lines in the graph. Therefore, these distributions of relevancy scores can have a significant impact on the training of the model as it is more difficult to discern between the individual products that the model returns if the original data the model trains on is packed more densely.

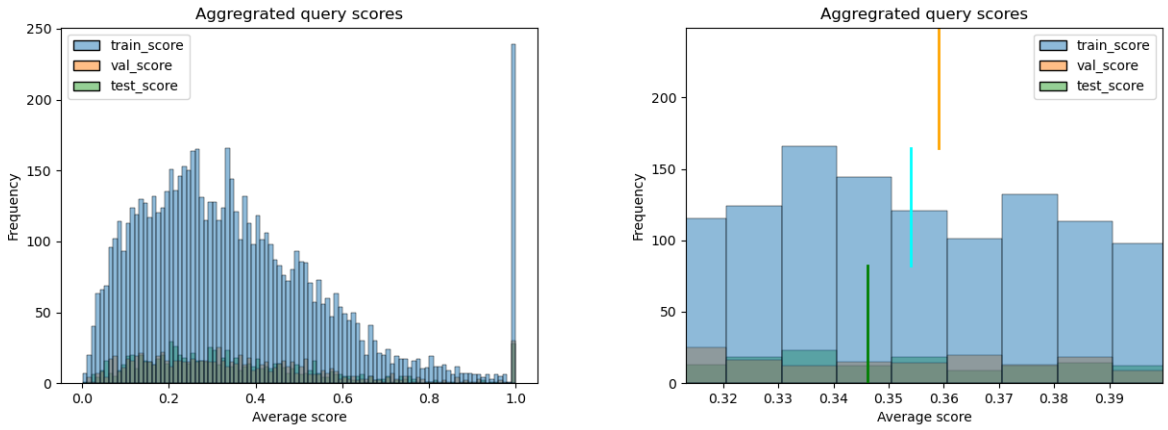

Figure 3.2: Average propensity score per query products pair. The 3 colored vertical lines represent the total average for the respective train, validation and test data splits.

3.1.2 Product Data

Each product supplied by third-party suppliers as well as Albert Heijn itself is logged in the data. These products are associated with various data fields such as product name, brand, properties and much more. Much of this information is deemed relevant to the entire search domain of AH. By employing data such as the taxonomy structure of the product the search engine can refine its' returned ranking to the user's desire. Additionally, much of the product data is extremely relevant for this search project as well, since this product data is used during fine-tuning, evaluation and inference of the LLM. However, numerous fields in the product data that are irrelevant to this project still exist in the data and therefore have to be removed in the pre-processing stage.

To start off, all fields except for the fields present in Table 3.3 are dropped to ensure only relevant data is used in the data set. Next up, the properties field is processed to only include useful information. Normally the properties field contains several topics such as allergy information, the target group of customers this product caters to, whether the product was on sale during that search and more. The properties had to be cleaned and processed to represent only relevant properties in the newly forged data set. Relevant properties are properties that are deemed useful for the model to train on during the training phase. This includes all of the fields mentioned in Table 3.3 as well as specific properties such as allergy information, Nutri-score

(an indication of a product's nutritional value that AH uses) etc. By including such properties users will be able to search for products that specifically contain these definitions. Afterwards, both the product and click data were compared and query - product pairs that only existed in a single direction were removed in order to prevent both the edge cases where products are never searched for as well as searches that turn up with unknown products.

Field	Description
ID	Product ID used internally to represent a product.
Title	Name of the product.
Brand	Brand/company that manufactured the product.
Highlights	Contains a textual description of the product to display on the webpage of the product.
Properties	Contains specific properties of the product.
Taxonomy	Contains the taxonomy structure that AH employs to represent the product.

Table 3.3: Product data column descriptions.

Both tables were converted to CSV format for publishing and use throughout this research project. The processed click data contains 221.122 (non-unique) rows of searches executed in the search engine of AH. The processed product data contains 20.515 products that AH has offered through the past two recent months. Despite the large collection of click & product data that already exists in the processed data set, one large drawback still skews the data. Only the iOS app data is taken into account. The Android app data was unavailable during the formation of the data set and, due to time constraints, will not be taken into account for this research project. Adding the Android app search data to the data set and by extent to the model could have a significant impact on the results as Android users represent a large portion of the user base that AH maintains.

In addition to the product data described above we included more data about the taxonomy of the products. This data was extracted from a third party source outside of AH and included a more in-depth definition of the different product categories than the pre-existing taxonomy from the product data. By incorporating this *MIAC* taxonomy data into the products a more precise definition of product categories can be defined.

3.2 Model

In order to explore the possibility of adapting an LLM to the product matching & search domain at AH a suitable model had to be selected for use during this research. We provide the incentive for using the CMLM and the Siamese network set up below.

The CMLM model is vital to this research project for two main reasons. First, the model is quite capable of deriving sentence representations from a given input string. This is useful, as the model will need to train on short sentences (queries) as well as large product data (long pieces of text). By being able to condition on sentence level the model will be able to deal with the product data more efficiently. The second reason as to why the CMLM fits this research project is because the model itself is multilingual. AH's data set contains a large variety of products and queries that users have entered into their search engine, but nearly all this data is in Dutch. Most Large Language Models are trained upon English corpora of texts. The

CMLM supports Dutch as one of its' trained languages and therefore, completely eliminates this issue. Furthermore, the CMLM has shown good performance when it comes to dealing with multilingual problem definitions, which includes Dutch as well (Yang et al., 2020).

Siamese networks are of great importance in this research project. By employing the CMLM encoder twice in a Siamese setting similar output embeddings can be learned for the multiple inputs that the model has. These output embeddings will be derived in an equal fashion, with similar weights employing similar metrics such as cosine similarity, on the output of both encoders. This can determine the similarity between the inputs, which on its' turn could determine a possible matching between a search query and a product in Albert Heijn's data set.

3.3 Setup

The model is built up from two main components, both of which are made up of the encoder blocks from the CMLM model. These encoder blocks are set up in a Siamese setting, where the weights between the two encoders are shared. This way the encoders are able to learn similar representations as described in the previous section. One of these encoders is fed a tokenized input query from the data set, whilst the other encoder is fed the tokenized available product data associated with the same query. Following this input, each encoder will produce its' own (but similar) embedding of the respective query and products.

Consequently, each embedding will be pooled using average pooling[2] to retrieve the sentence embedding for the respective output. Next up the respective sentence embeddings are *projected* ($P(\dots)$) into N spaces ($\boldsymbol{v}_p = P(\boldsymbol{v}) \in \mathcal{R}^{d*N}$). The reason for this is that the model learns to associate similar sentence embeddings and their related products this way. Several different options for projecting the sentence representations exist, such as a three layered Multi Layer Perceptron or by taking the average sentence embedding out of the available projections ($\boldsymbol{v}_s = \frac{1}{N}\Sigma_i \boldsymbol{v}_p^{(i)}$). Recent research has already shown that both are of comparable performance in the state-of-the-art research comparison (Yang et al., 2020), so for simplicity the latter (average sentence embedding) is adapted into our work.

After the sentence embeddings have been produced from each respective encoder component the similarity between these embeddings will be computed to derive the vector space that these embeddings share. Because of the nature of Siamese networks the embeddings lie in a similar vector space. By employing the cosine Similarity the *semantic distance* between the embeddings can be derived. This will reveal a measure of similarity between the query and the products respectively. Products that resemble the query more will have less distance between them and the original query. Conversely, products that do not match the original query well will have a larger semantic distance between them.

FAISS, the semantic search toolkit discussed before, is used during the inference phase of the set up. The model output consists of the encoded embedding of the input, which is fed to the FAISS as input for the semantic search functionality. Initially, FAISS requires all of the product embeddings in order to generate the index to search through. Once the model output embeddings for all the products have been processed by FAISS, the algorithm is used to search the semantic space for product representations that have a high cosine similarity score w.r.t. the query embedding from the model output. FAISS constructs a list of products with their

[2]Other pooling methods, such as max pooling, or the output of marked sentence tokens could also be used.

associated cosine similarity scores. This final combination of query and associated products is taken for further evaluation. Johnson et al. (2019)

The next step in the research is the analysis of the model performance. For the analysis, we have taken several Information Retrieval metrics, such as *Precision* and *Recall*, and Ranking metrics, such as Mean Reciprocal Rank (*MRR*) and Normalized Discounted Cumulative Gain (*nDCG*), into account to determine the model's performance. The Precision & Recall highlight the model's capability of retrieving relevant products from the entire product catalogue. The *MRR* determines the average position in the order of products. The *nDCG* computes the ranking against the ideal ranking of the products to determine the performance of the model. All of these metrics are useful as they can provide an accurate assessment of the model's performance w.r.t. product retrieval. In addition, the set up allows for easy manual inspection in the data and the comparison between the output from AH's productionized model and the Large Language Model can be visualized. This would aid in the exploration of the model performance.

The fine tuning pipeline is implemented in Python 3.10. The CMLM is taken from the *HuggingFace* library [3]. This model is then used as a baseline to perform the fine tuning process on. The fine tuning process relies on the sentence-transformer library, which is provided by *HuggingFace* as well. The custom created data set featured in this thesis project has been established as the fine tuning data for the pipeline.

However, in order to be used for this purpose, the data had to be processed even further. For each query - product pair in the data an appropriate threshold for the earlier defined *Relevancy score* had to be determined. This threshold defines which products serve as a positive reinforcement to the model and which products serve as a negative reinforcement to the model. Using such a threshold is important as this allows the model to increase performance as the fine tuning process proceeds. A threshold of 0.2 w.r.t. the similarity score of the products in the LLM is taken, after inspection of the preliminary results which showed a decline around this similarity score value. After the threshold value is determined we incorporate this threshold into the classification of relevant products. If the product's relevancy score is below the threshold it is classified as a negative sample, and vice versa. This allows the algorithm to sample negative samples within the same batch for simplicity.

The evaluation phase in the process relies on a custom implementation of the *InformationRetrievalEvaluator* in the *sentence-transformer* package from *HuggingFace*. This custom implementation extends the evaluator to support the data set that is featured in this project as well as allows the evaluation to focus on specific metrics such as the *MRR* and the *nDCG* scores. These metrics have been selected as they are featured in the online setting that Albert Heijn provides. Mirroring this setup allows for more accurate comparisons to be drawn between the existing and proposed models.

The fine-tuning process itself consists of several different hyperparameter setups to find the best model version. These hyperparameter configurations are shown in Table 3.4, with the final hyperparameter configuration underlined. Furthermore, the model uses a similar set up as Yang et al. (2020) in terms of optimizer and step size. Note that the main hyperparameter configuration from the original model training can differ from the set up used here. We are using a pre-trained version of the CMLM and fine-tune that model on the data that we have acquired during this research. Therefore, some alternative set up choices are taken.

[3] https://huggingface.co/sentence-transformers/

Hyperparameter	Value
Learning rate	$\underline{2e^{-05}}$
Epochs	$\underline{20}$ | 40
Threshold	$\underline{0.2}$ | 0.3 | 0.4
Batch size	$\underline{32}$ | 16 | 8 | 4
Optimizer	$\underline{\text{LAMB}}$ | LAMB
Step size	1.000.000
Number of projections	15

Table 3.4: Hyperparameter Configurations; note that the optimizer, step size and number of projections are taken from the original source paper of Yang et al. (2020).

In order to determine the model's performance several other configurations were taken into account. First off, peer research from other researchers within the team at AH revealed that the *product highlights*, which was extracted from the product data, could lead to sub-optimal performance when utilized for training an LLM. We hypothesize that the product highlights share a vast amount of similarities amongst one another with several words appearing in the highlights quite frequently, which could confuse the model during the fine tuning. To this end, we fine-tuned different Large Language Models. By doing so, we are able to analyze the impact of using this part of the product information from the newly proposed data set and potentially remove it in order not to harm the model performance.

Secondly, the CMLM model contains a built-in tokenizer that keeps track of sets of tokens for each of the 100+ languages the model has been trained on. However, manual inspection revealed that the token set for Dutch, the primary language within the AH data, did not match well with most of the Dutch language that the data used. Examples of this are specific product names such as "allesbinder" or "maizena", product types like "dropveter" or "pepernoten". These tokens often did not appear in the original token set of the tokenizer, which caused the model to process these words incorrectly. In order to solve this problem a new set of tokens, derived from the product data, is incorporated into the tokenizer for use during the fine-tuning process. We generated the new token set by drawing the tokens from a combination of the product brand and title and feeding them to the tokenizer such that it could train with the added tokens. Several model variants, such as one trained with the tokens and without the product highlights or one model that has both, have been fine-tuned to analyze the impact of this token set change. An example of the added tokens can be seen in Table 4.1.

Thirdly, whilst the CMLM showed promising results in related research, we incorporated a different model type as well. This new model is a fully trained LLM on the Dutch language only.[4] By incorporating this model an analysis can be made on how a multilingual model with a multilingual token set performs w.r.t. a monolingual model with a monolingual token set. Both models undergo the same pipeline, with the same fine-tuning process and inference/evaluation steps. However, the Dutch model consists of a different architecture and uses different pre-training methods to be trained with. These differences could be part of the reason as to why this model performs differently w.r.t. the instances from the CMLM.

For the evaluation we set up different instances of the CMLM model. These variations and their descriptions can be seen in Table 3.5. All of these instances have been fine-tuned using the same data and pipeline. Furthermore, all of these instances are evaluated similarly and with the same metrics to guarantee that proper comparisons can be made.

[4] `https://huggingface.co/textgain/tags-allnli-GroNLP-bert-base-dutch-cased`

Type	Full name	Used fields	tokenization	Type of model
A	Highlights, no tokens	Product brand, product title, MIAC taxonomy	Default	CMLM
B	No highlights, no tokens	Product Brand, product title, MIAC taxonomy	Default	CMLM
C	No highlights, extra tokens	Product brand, product title, MIAC taxonomy	Extra domain-specific tokens	CMLM
D	Highlights, extra tokens	Product brand, product title, product highlights, MIAC taxonomy	Extra domain-specific tokens	CMLM
E	Highlights, extra tokens, double epochs	Product brand, product title, product highlights, MIAC taxonomy	Extra domain-specific tokens	CMLM
F	Highlights, extra tokens, Dutch model	Product brand, product title, product highlights, MIAC taxonomy	Extra domain-specific tokens	Dutch model

Table 3.5: Different instances of the Conditional Masked Language Model.

3.4 Evaluation

The offline evaluation of the model will be done through several retrieval metrics as seen in Table 3.6. Computing these metrics at the specified input sizes ensures a solid measure of performance for the models and highlights how the models perform in settings where larger amounts of products need to be returned. In addition to these metrics, we compare the results from our models with the results from the model in production at AH.

Metric	Input size
Precision	10 | 25 | 50 | 100
Recall	10 | 25 | 50 | 100
Mean Reciprocal Rank (MRR)	-
Normalized Discounted Cumulative Gain (nDCG)	-

Table 3.6: Overview of the metrics/benchmarks used to evaluate the model performances.

The current model in production is a variant of ElasticSearch (ES)[5]. ES is, amongst others, a powerful search engine based on the BM25 retrieval method. BM25 consists of metrics such as Term Frequency (TF) and Inverse Document Frequency (IDF) to compute document-level statistics used during search. (Robertson and Zaragoza, 2009) This productionized model outputs a total of 180 products that it matches the search query against. These products are then fed to a *Learning To Rank* (LTR) algorithm which reranks the order of the products based on several matching features, such as add-to-cart ratios of products or their profit margins. The LTR algorithm uses the *XGBOOST* LTR model (Chen and Guestrin, 2016), which is a decision tree boosting system to re-arrange the order of given items. All of this data is available from the incoming data stream within AH. The output of LTR is then displayed on the website as search results for the customer.

[5] `https://www.elastic.co/`

In this research setting, we have modified the original model to exclude the LTR part. This way the results from the CMLM can be directly matched with the results from ES. Incorporating the LTR model into both set ups was not feasible as the LLM side did not contain the same amount of features used for matching as ElasticSearch's side contains. Using LTR for both would eventually cause the models to not have an equal ranking method. Therefore, we dropped LTR from both set ups.

Chapter 4

Results

In this section we highlight the main results from the research project. Each research sub-question is thoroughly examined with the associated results in the dedicated section. Furthermore, we highlight some key observations and provide some rationale as to how or why this phenomenon occurred.

4.1 Data Set

With the creation of the custom data set from the data sources of AH we discovered the characteristics and challenges that such a data set possesses. Through the relevancy scores, visible in Figure 3.2 we discovered that the majority of the queries returned products that had lower relevancy scores. A similar phenomenon is visible in Figure 4.1. This is an example graph of the scores of model instance A w.r.t. the score extracted from ES. Both of the score types are packed more densely around 0.3 which corresponds with the computed average. However, a key difference that is visible in the figure is the density of the CMLM model's output, which is considerably more dense than the average output of ES across the queries. This seems to indicate that the model is having considerable difficulty processing the queries and responding with a similar product list, which in turn could be appropriated to the average ES score of 0.3.

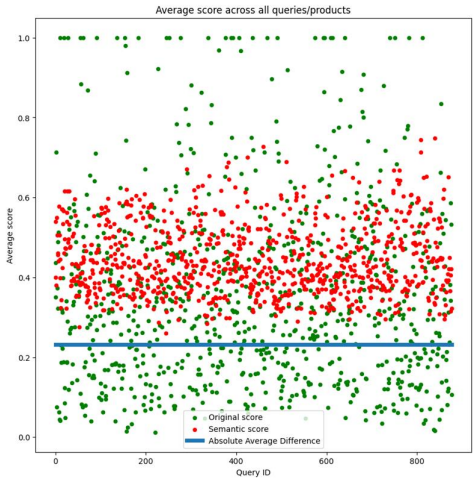

Figure 4.1: Average difference between ES scores & model A scores. The scores from model A are packed more densely together than ElasticSearch's scores.

By exploring the key qualities that the public data set possesses we identified several challenges in the data, such as the sparsity, size and individual entry lengths. The data set itself contains information from the specific grocery shopping domain and several entries were missing or not filled in correctly. The size of the data set is limited as well, roughly 20.000 products and about 250.000 queries. Lastly, there's a huge difference between the size of the individual queries and the associated products (in terms of sentence length & individual characters). The average length for queries is roughly 2 words per query whilst the average length for the product full text definitions is around 5 up to 20 words on average, depending what aspects of the product data are used. Some set ups have the product highlights included which account for a substantial uptick in word count as these highlights are reasonably extensive. Another issue that the data set faces is the limited scope of the data itself. The data lies in the grocery shopping domain of AH, which only has a limited set of products (roughly -35.000). An LLM might need more data in order to successfully fine-tune on this data set.

4.2 Tokenization

The grocery shopping domain that AH's data entails contains various domain-specific tokens and knowledge. This could harm the model's performance as it needs to process tokens that it has never seen before. For this reason we expanded the token set from the model's tokenizer to include tokens extracted from the domain specific data set created within this research project. An overview of some example queries is seen in Table 4.2. These queries have been selected as they provide a clear overview of the differences in tokenization between the model instances. Note that we also include the Dutch model (model F from Table 3.5) to show how a different model with a different tokenizer handles these domain specific tokens as well. We have ommitted the *[CLS]* & *[SEP]* tags from the tokenizer output to preserve clarity.

Query	**CMLM's default token set**	**CMLM's expanded token set**	**Dutch model's default token set**	**Dutch model's expanded token set**
Groene appels	Groene appels	Groene appels	Groene appels	Groene appel s
melk 1.5L	melk 1 . 5l	melk 1 . 5l	melk 1 . 5/l	m el k 1 . 5/l
chocola zonder hazelnoten	choc/ola zonder haz/elno/ten	chocola zonder hazelnoten	chocola zonder ha/zel/noten	choco la zon der hazelnoten
zout	zout	zou/t	zout	zout
maizena	maize/na	maizena	ma/i/zen/a	maizena
appelsientje	appels/ient/je	appelsientje	appels/ientje	appelsientje
winegun	wine/gun	wine/gun	win/e/gun	wine gu/n
kaas voor pasta	kaas voor pasta	kaas voor pasta	kaas voor pasta	k aa s v oo r pasta

Table 4.1: Tokenization differences between the model instances across the selected query set.

As can be seen from the table the tokenization process can have impact on the correct processing of domain specific tokens. Several unique tokens that the model's tokenizer had not seen before are now processed correctly. However, this does not guarantee an increase in performance necessarily. Additionally, model F (the Dutch model) seems to perform worse than model D with the new token set. Regardless, in order to check if the models did benefit from this tokenization change, we have fine-tuned the different model instances with tokenization changes (as shown in Table 3.5). Specifically, the results for model D and model F are shown in Figure 4.2.

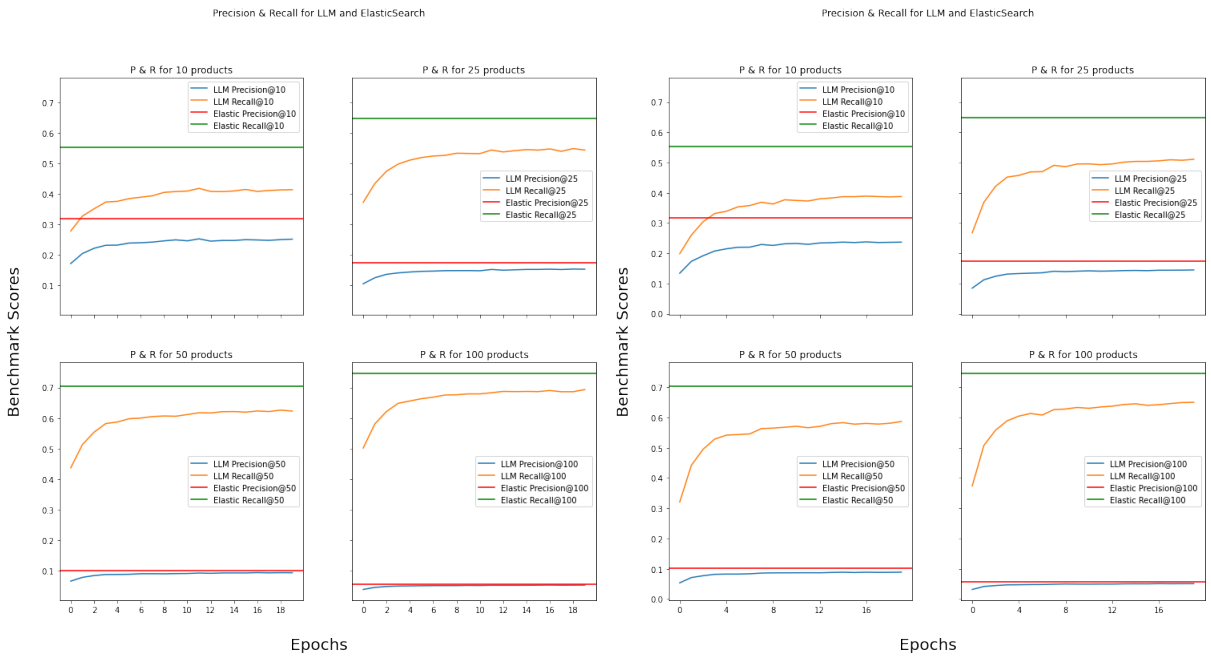

Figure 4.2: Model D (left) & F (right) Precision & Recall scores. Model F seems to perform slightly worse than model D for these benchmarks.

From these results we can establish that the Dutch model (model F) is performing slightly worse than model D. This seems to suggest that the Dutch model, though seemingly improving the tokenization process, does not handle the additional tokens in the same way that model D does. However, both models are nevertheless performing worse than ElasticSearch, which indicates that the models are still not capturing the same products as AH's baseline captures. From these results we can state that the added tokenization process, despite being impactful, does not deliver the desired performance improvement. Although, one important aspect to take into consideration is that the Dutch model (model F) is built with a different architecture and tokenization method. Therefore, the diminished performance could be attributed towards these differences in design.

4.3 Model Performance

The results of the individual models from Table 3.5 are described in this section. For each model that was taken for further inspection we have computed the *nDCG*, *MRR*, *Precision* and *Recall* at the end of each epoch and stored those results. In addition, we compute the average difference between the model outputs for each model across the queries. This average difference shows the relationship between the productionized AH and the LLM model. On top

of the average difference we compute the direct comparison between the score from the original model and the similarity score from the LLM. This way we can analyze the high level performance differences. Lastly, we use the fine-tuned model to generate results for some unseen queries (not present in the training or evaluation data) and recompute the Precision manually to determine the major differences in output between the original model and the LLM. For clarity reasons, only the best performing model is inspected more thoroughly. The remaining graphs & other results can be found in the supplementary material and in Appendix 6.1.

The results of Model A, as described in Table 3.5, are visualized in Figures 4.3, 4.4 and 4.5. An overview of the different metrics can be seen in the graphs that are depicted in this section. Model A shows the best performance w.r.t. Precision, Recall, MRR and the nDCG scores when compared against the other model variants. The graphs of the metrics show how the model performed across the epochs that it was fine-tuned on. For comparison, we have included the performance of the ES model that is currently in production within AH so that the difference is clearly visible.

For the Precision & Recall scores, as seen in Figure 4.3, model A does not seem to outperform the ES baseline for lower product list sizes. As the list size increases the performance of model A seems to increase as well, up to the point where it matches and even exceeds ES at 100 products in the model output. We also computed the Precision & Recall scores for the lower model output sizes and the visualisations of the graphs can be seen in Appendix 6.1. However, all of the model instances seem to suffer from the low output size and have a rather large performance gap between them and the ES model.

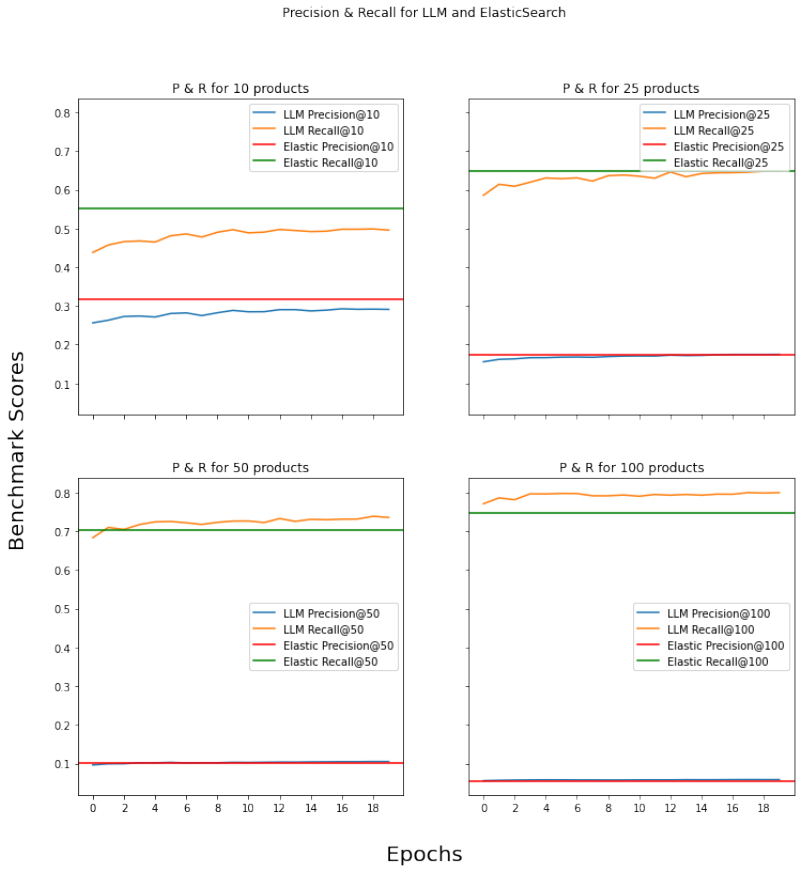

Figure 4.3: Precision & Recall of Model A across different product list sizes.

The performances of the other models introduced in Table 3.5 differ from the performance of model A. The key takeaway in the performance difference is that the models with the extra highlights, tokenization, or even more epochs (and any combination of these different aspects) all saw a drop in the Precision and Recall scores, with the larger drop occurring in the models that introduced the expanded token set into the fine tuning process. Additionally, the CMLM models did not reach similar performance as ES when the product list lengths increased. However, apart from the Dutch model and the models with the expanded tokens which suffered an even bigger performance drop, all the other models only had minimal differences in performance amongst them.

These results indicate that model A was the best performing model in terms of the retrieval metrics measured. Although, in terms of ranking metrics such as the MRR or the nDCG score, no notable performance difference was noted amongst any of the model instances that we used during the evaluation. They all have a similar sub-par performance when compared to ES and the pipeline behind it. An example of the ranking benchmarks can be seen in Figure 4.4. This figure contains the MRR & nDCG scores for model A and clearly shows that the performance of model A w.r.t. these benchmarks is worse than ElasticSearch.

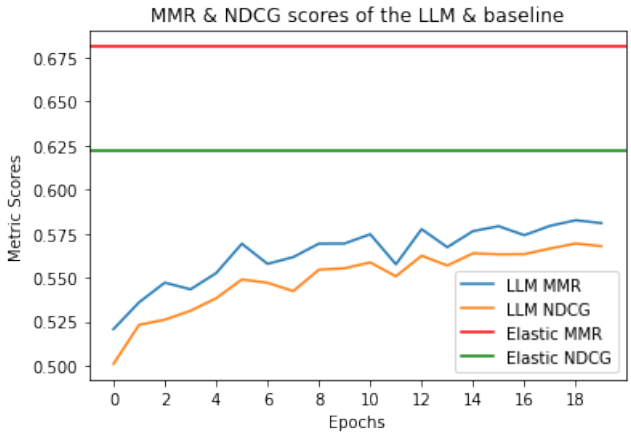

Figure 4.4: Model A's Mean Reciprocal Rank & normalized Discounted Cumulative Gain benchmark scores. As visible, ElasticSearch outperforms the model on both these benchmarks.

The average scores across all the queries and their respective product lists can be seen in Figure 4.5. The distribution of average scores seems to peak at around 0.4 for model A and 0.3 for ES. One key difference that this graph highlights between the models is that the ES model contains numerous scores that are capped at 1.0, which is due to the original model only returning a single product for these queries with the filtering that is applied in the pipeline (e.g. through the use of the LTR model).

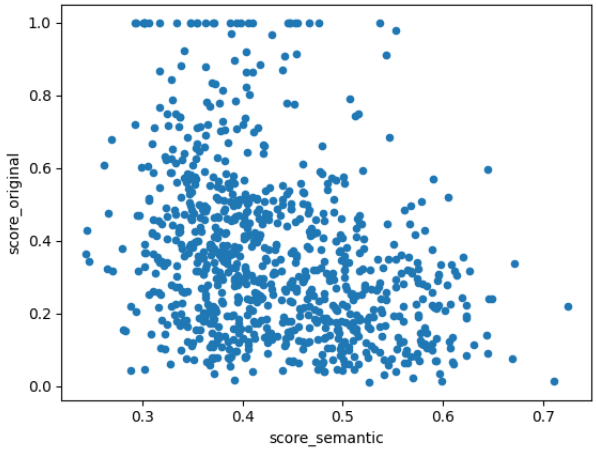

Figure 4.5: Average score of ElasticSearch & model A across the queries. Each point marks the average scores of all products for the given model by representing the score on each axis.

We have also computed the average difference between the output of ES and model A as this could help determine the appropriate thresholding values on which we can fine-tune the model with. The average differences between the queries from both models can be seen in Figure 4.1. The horizontal line indicates the total average across all the differences between the model outputs and shows that the mean difference between the models boils down to roughly 0.2 in score difference.

Another interesting observation from the differences between the average scores for each query is the density of the output. The ES original scores are all sparsely packed together, varying from 1.0 to near 0.0 in score. However, the output from model A seems to be packed considerably more dense together in comparison. We have observed similar behavior when comparing the other model instances as well. This indicates that model A is not able to make the same clear distinctions in the product listings as ES does. This observation is supported even further by the total mean difference between the models, which is 0.2. This substantial gap indicates that the individual query results from each model differ substantially from one another.

4.4 Qualitative Analysis

A manual analysis is also performed during the evaluation of the different model instances. In order to analyze the performance of the models manually we take the ES output as the baseline to which we can compare the results. Consequently, we run different queries through each of the model instances and compute lists of products that the model returns as output to the given query. Some of these queries have been classified as *"difficult queries"* for ES as well, since they initially produced few relevant results. By incorporating these difficult queries into the results of the CMLM instances we can examine how the model can provide meaningful products. An overview of the queries and their classified difficulty can be seen in Table 4.2.

Query	Difficulty
Appelsientje	Easy
Chocolade zonder hazelnoten	Hard
Groene appels	Easy
Maizena	Hard
Melk 1.5L	Easy
Kaas voor pasta	Easy
Winegun	Hard
Zout	Easy

Table 4.2: Queries classified by difficulty for the models.

The results of these individual queries from the models can be seen in both tables below. The CMLM instances have a fixed output size of 108 products, whereas the output size of ES differs based on the scores of the products and the difficulty of the query. For each query, we have computed both the top-20 relevant products from all the model instances as well as the total amount of relevant products. The top-20 results can be seen in Table 4.3 whereas the total product list results can be seen in Table 4.4. For clarity we computed the Precision for ES, as the total product size varies per query.

Query	**Model A**	**Model B**	**Model C**	**Model D**	**Model E**	**Model F**	**ElasticSearch**
Appelsientje	19	19	0	0	0	0	20
Chocolade zonder hazelnoot	0	1	0	0	0	0	0
Groene appels	1	3	2	2	2	0	1
Maizena	2	2	0	0	0	0	2
Melk 1.5l	12	13	18	17	16	0	0
Kaas voor pasta	6	3	3	5	3	10	2
Winegun	0	0	0	0	0	0	4
Zout	19	17	19	18	19	0	20

Table 4.3: The top-20 relevant product count for the queries defined in Table 4.2 across all the model instances.

Query	**Model A**	**Model B**	**Model C**	**Model D**	**Model E**	**model F**	**ElasticSearch | % relevant**
Appelsientje	31	31	0	0	0	0	25 | 100%
Chocolade zonder hazelnoot	5	6	1	3	0	0	4 | 8%
Groene appels	2	3	3	3	3	0	1 | 2.5%
Maizena	2	2	0	0	0	0	2 | 66.67%
Melk 1.5l	65	61	63	62	61	0	0 | 0%
Kaas voor pasta	16	15	9	16	13	11	4 | 100%
Winegun	1	1	0	0	0	0	7 | 14%
Zout	30	27	29	31	28	0	29 | 100%

Table 4.4: The total relevant product count for the queries defined in Table 4.2 across all the model instances.

From both Table 4.3 and 4.4 we can observe that model A and model B are the best on average. When taking the full product list into account these models have the most amount of relevant products when compared to the other models (excluding SemanticSearch). However, in the top 20 results model A and B are outperformed by the other instances for some queries. Nevertheless, these observations seem to reveal that model A and model B are the best performing models on average, which seems to align with the benchmark results discussed earlier.

One key observation with the qualitative analysis is that ElasticSearch does not seem to outperform the model ensemble for these queries. As mentioned in Table 4.2, some of these queries are classified as difficult queries for ES and the results clearly show this. For some queries ES is outperformed by the other models, which is in line with the benchmark results for some of the models. Specifically, model A and model B are outperforming ES on most of the queries. However, ES is far superior in terms of Precision, as the baseline from AH is more than capable of limiting its' model output with only relevant products. This is demonstrated by the queries "Appelsientje", "Kaas voor pasta" and "Zout" which all have a 100% Precision. The other models in the ensemble do not reach such scores at all, with the highest Precision being 60% from model A for the query "Melk 1.5l."

Chapter 5

Conclusion

This research project explored the possibility and feasibility of adapting an LLM to the search architecture within the e-commerce of major grocery brand AH. By analyzing the data, model performance and separate tokenization we can conclude that applying an LLM type of model within such a search domain did not lead to improved semantic matching. We will outline the hypothesized reasons below, starting with how the data set played a major role in realizing these results.

The data set is one of the key contributions in this project and, through publishing, could form a valuable asset for the research community as a Dutch grocery shopping data set based on real data. However, as discussed earlier the data set does have some challenges such as data sparsity, data set size and length differences between several components of the data.

These three aspects cause difficulties when fine-tuning Large Language Models as they play a significant role in the final performance of the model. Future research could look into improving this data set in terms of diversity, size and average sentence length. Moreover, the query set could be enhanced through query extension based on the associated product list, for example by taking the most common product brand/title from the resulting product list. This could have a major impact on the data set quality as well.

Furthermore, we also investigated a possible area of improvement w.r.t. the performance of the individual models. We established that the tokenizer, although trained on multilingual corpora, experienced difficulty when tokenizing certain words in the data. We also identified the lack of specific domain knowledge (of the grocery domain) to play a significant role in this issue. In an attempt to fix this problem the token set was extended to include the tokens from our custom data set and a new set of models were fine-tuned with this updated token set. However, the results indicated that this change did not have the desired effect of improving the performance nor did it positively influence the qualitative analysis. Future research could look at augmenting the token set even further, adding more specific grocery domain knowledge from other Dutch grocery shopping domain data sets. Also, the token set could be augmented pre-training instead of during the fine tuning phase as another useful change. This would allow the model to train on the new token set for far longer and more effectively than just on the fine tuning stage. That could potentially be a solution to the token issue.

Lastly, another key contribution of the research is the feasibility study on the LLM application. Moreover, by implementing the Conditional Masked Language Model in this new setting we explored and measured the impact that such an LLM model could have. We can conclude that the different variations of the model each displayed substantially different results and did

not align completely. However, model A and model B from Table 3.5 seem to be the best performing models in the ensemble regardless, even surpassing ElasticSearch on some specific (difficult) queries. These findings suggest that Large Language Models could play a significant role in semantic search in the grocery shopping domain, especially when further research could look into improving the models even further.

One key observation in the results is that ElasticSearch achieves higher Precision with the qualitative analysis. The main reason as to why this occurs is that ES limits the model output based on relevancy of the products far better than any of the CMLM can. This is one of the major factors in the performance gap between the ensemble and the baseline from AH. Future research could look into improving the thresholding of positive and negative products by the model in order to massively reduce the respective output size. This would have a positive influence on the model performance, as the number of irrelevant products would greatly diminish.

The filtering applied by ES on the product output carries an additional downside with it. Unfortunately, the custom data set is based on the results extracted from ES as well, with the data stream being received directly from the customer interaction on the model output. Consequently, the custom data set crafted from this data stream already contains specific filtering as per the model's perspective. This results in numerous relationships between the queries and the products to be missing from the data, as ES deemed these products unfit for the query even though they could be classified as *false negatives* (products that should have been added) or *false positives* (irrelevant products that have been added) by the LLM. Additionally, the LLM is less effective in learning the relationships that exist within the data as a large portion of these are missing. Future research could look into enriching the data with results that are not extracted from ES, perhaps through a manual survey. This could help improve the quality of the data and thus, the resulting fine tuning based on this data set.

Even more research could be carried out for the data. First off, one could look into the data set and enrich it with the aforementioned options to improve the quality of the data. Other sources of the data could also be consulted to combine this information into a larger, more diverse data set. For example, different large grocery brands could cooperate and combine their grocery domain data to benefit research on a larger scale. Even within the data domain of just AH itself numerous improvements can be made. By processing more data and equipping the data set with more properties sourced from the underlying vast collection of user interactions the data set could already contain more semantic information. This could already have a major positive influence on the model's performance. Due to time and complexity constraints this research suggestion remains open.

Another suggestion for future research could be to explore more diverse models within this grocery shopping domain. Applying an LLM to this domain to strengthen semantic search remains to be an option with substantial potential. However, in due time new models could surface within the research community that, when employed within the domain of AH, might lead to an even better performance than what has been displayed within this research project. As we have shown in the related work of Kim et al. (2022), another LLM such as GPT-3 has already proven itself to be a useful asset within the task of product retrieval. A more concrete suggestion for future research would be to implement or fine-tune such a model like GPT-3 to see if a performance boost can be obtained.

Lastly, in the current research project there was no opportunity to explore the model's performance in an online setting. The offline performance of the models simply was not up to the standard that AH deemed fit for a possible A/B test with live consumer data. Additionally, due to technical constraints the model could not be improved further to meet this threshold of performance. This is why we recommend to perform an A/B test with an improved version of the models in this research. Having the model in an online setting and be available to potentially millions of customers could lead to deeper insights into the strengths and weaknesses of employing an LLM in such a search architecture.

To conclude, the main focus of this research project was to establish the impact of applying an LLM to the search architecture within the grocery shopping domain of AH. By building a custom data set and enriching the token set with domain-specific tokens from the custom data we have trained an array of models to assess their performance. We summarized that, based on the findings from these models and their variations, the current suite of models did not fully perform up to standard when matched against the baseline from AH, even though some models did surpass the baseline in the qualitative analysis. We explored the different reasons as to why this performance issue might occur and provided suggestions for further research to tackle these issues and improve the performance of the models in a substantial way. By having conducted this research, we hope that AH can thrive within the semantic search division and find a potential set up where an LLM can flourish, based on the outcome of this research. We would like to thank AH for this research opportunity and guidance during this project.

Bibliography

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Chen, T. and Guestrin, C. (2016). XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

Chicco, D. (2021). Siamese neural networks: An overview. *Artificial neural networks*, pages 73–94.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

He, P., Gao, J., and Chen, W. (2023). DeBERTav3: Improving deBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations*.

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

Kim, S. Y., Park, H., Shin, K., and Kim, K.-M. (2022). Ask me what you need: Product retrieval using knowledge from gpt-3. *arXiv preprint arXiv:2207.02516*.

Minsky, M. L. and Papert, S. A. (1988). Perceptrons: expanded edition.

Peeters, R., Bizer, C., and Glavaš, G. (2020). Intermediate training of bert for product matching. *small*, 745(722):2–112.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

VILCEK, A., MOTTAGHINEJAD, S., SHI, S., GUPTE, K., PASUMARTY, S., PANG, L., and MEHROTRA, P. (2018). Transformer-based deep siamese network for at-scale product matching and one-shot hierarchy classification.

Xie, Y., Na, T., Xiao, X., Manchanda, S., Rao, Y., Xu, Z., Shu, G., Vasiete, E., Tenneti, T., and Wang, H. (2022). An embedding-based grocery search model at instacart. *arXiv preprint arXiv:2209.05555*.

Yang, Ziyi, Y., Cer, D., Law, J., and Darve, E. (2020). Universal sentence representation learning with conditional masked language model. *arXiv preprint arXiv:2012.14388*.

Acronyms

AH Albert Heijn. i, 1, 2, 6, 7, 9–14, 16–19, 23–26

CMLM Conditional Masked Language Model. i, 1, 2, 5, 10–17, 20–22, 24, 25

ES ElasticSearch. 14–16, 18–23, 25, 31–39

LLM Large Language Model. i, 1–6, 9, 10, 12, 13, 15, 17–19, 24–26, 30

Chapter 6

Appendix

6.1 Model Results

In this section a clear overview of the remaining model results are shown. Each graph belongs to a different model instance as defined in Table 3.5. For each of the models, we display the graph for the MRR, nDCG, Precision, and Recall benchmarks. Furthermore, the individual score distributions of the LLM's output w.r.t. the original relevancy score of the remaining models is shown in these figures as well. Lastly, we show the differences in average query score for both the LLM and the original relevancy score for each model in the ensemble.

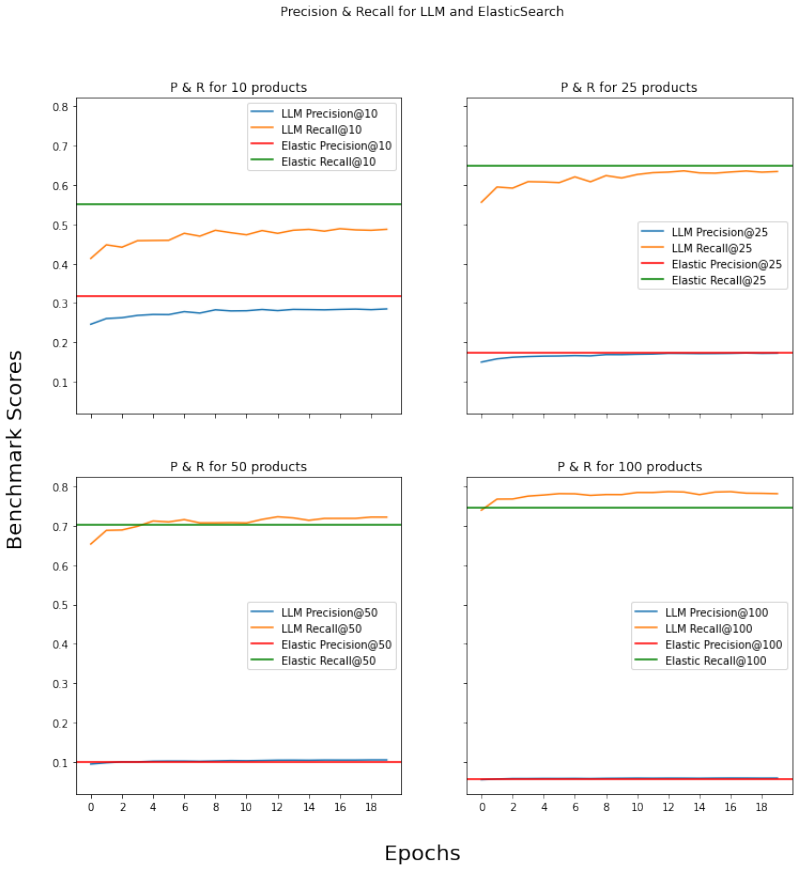

Figure 6.1: Model B's Precision & Recall scores.

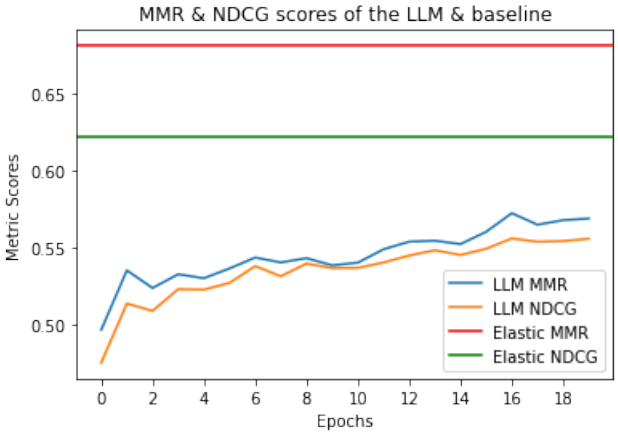

Figure 6.2: Model B's Mean Reciprocal Rank & normalized Discounted Cumulative Gain benchmark scores. As visible, ElasticSearch outperforms the model on both these benchmarks.

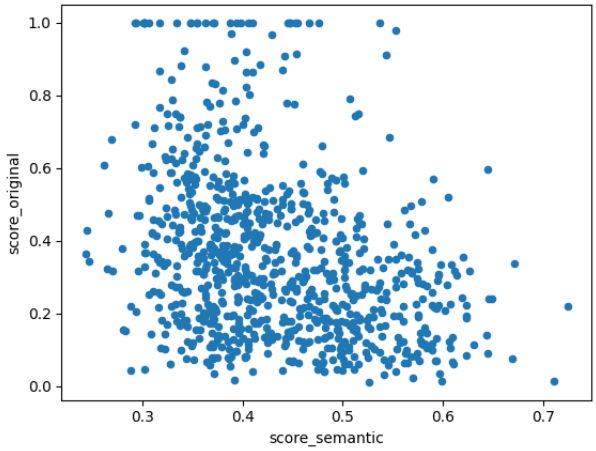

Figure 6.3: Average score of ElasticSearch & model B across the queries. Each point marks the average scores of all products for the given model by representing the score on each axis.

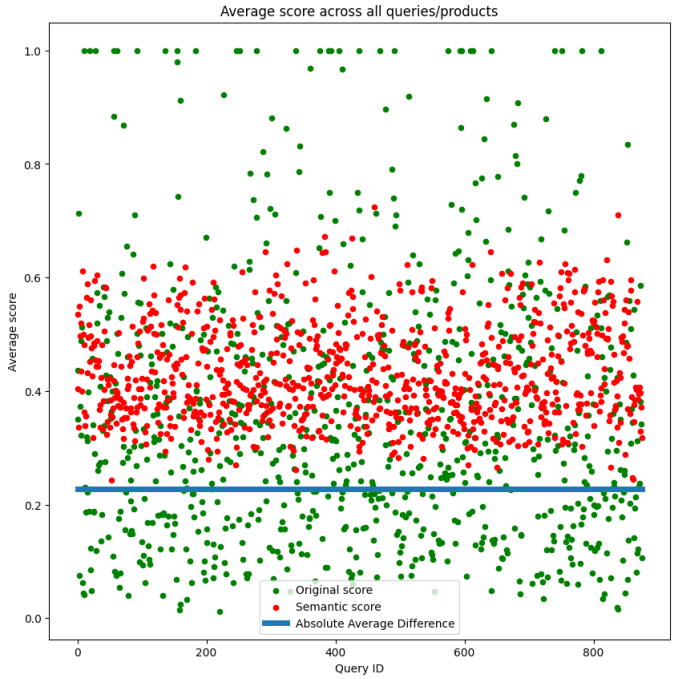

Figure 6.4: Average difference between ElasticSearch scores & model B scores. The scores from model B are packed more densely together than ElasticSearch's scores.

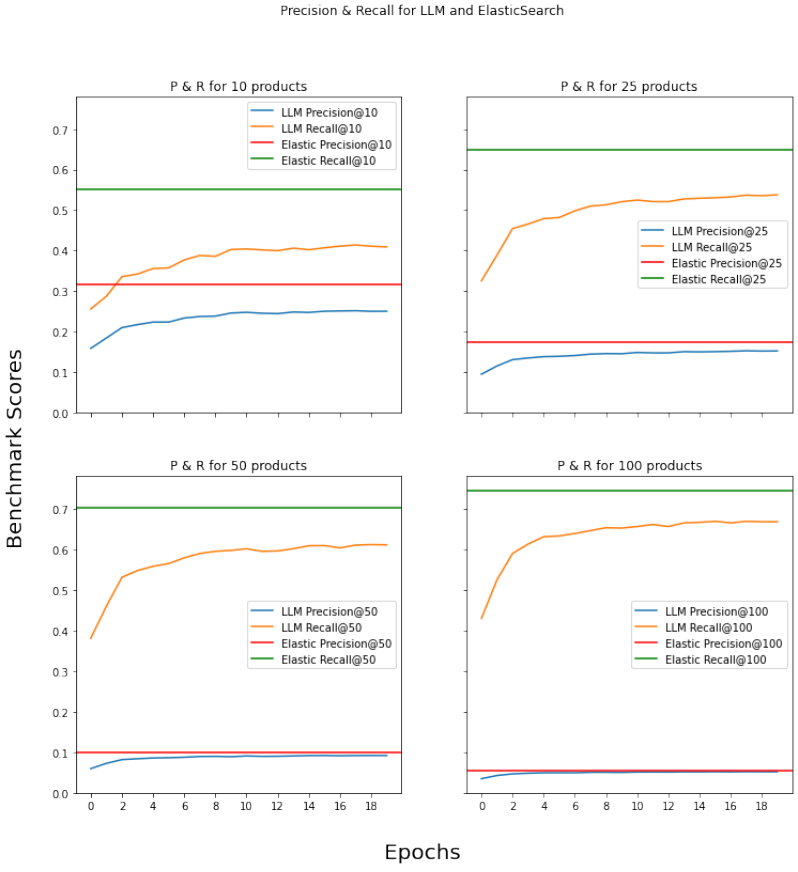

Figure 6.5: Model C's Precision & Recall scores.

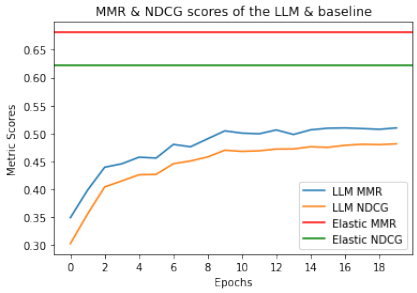

Figure 6.6: Model C's Mean Reciprocal Rank & normalized Discounted Cumulative Gain benchmark scores. As visible, ElasticSearch outperforms the model on both these benchmarks.

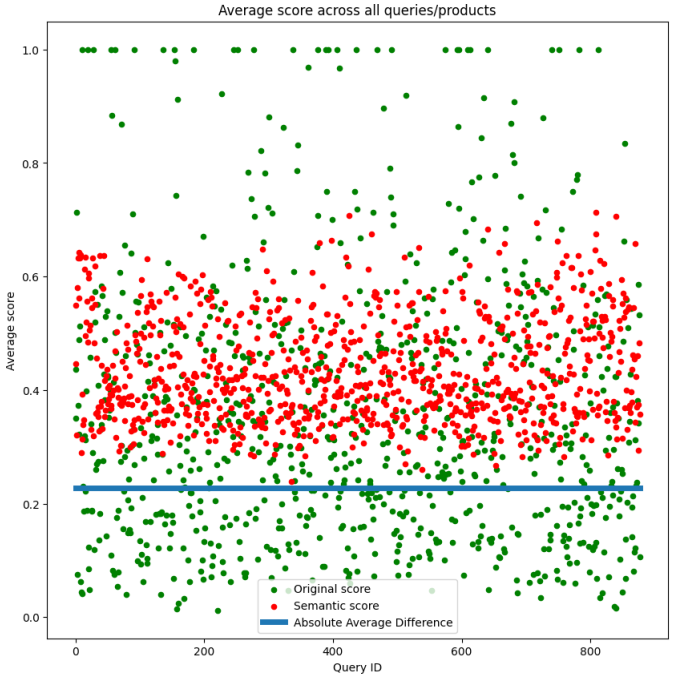

Figure 6.7: Average difference between ElasticSearch scores & model C scores. The scores from model C are packed more densely together than ElasticSearch's scores.

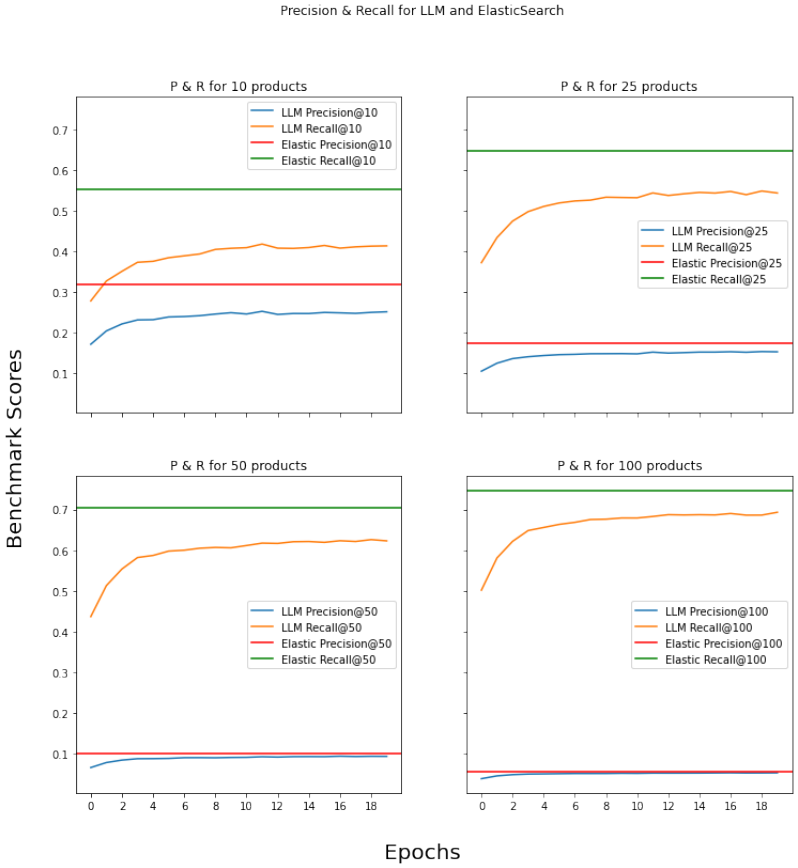

Figure 6.8: Model D's Precision & Recall scores.

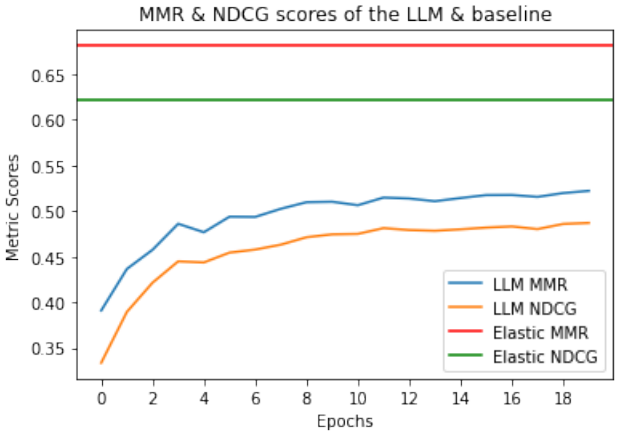

Figure 6.9: Model D's Mean Reciprocal Rank & normalized Discounted Cumulative Gain benchmark scores. As visible, ElasticSearch outperforms the model on both these benchmarks.

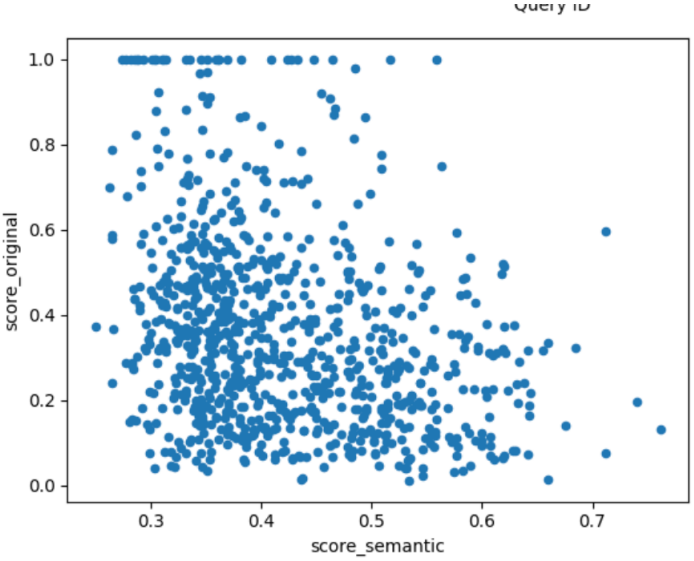

Figure 6.10: Average score of ElasticSearch & model D across the queries. Each point marks the average scores of all products for the given model by representing the score on each axis.

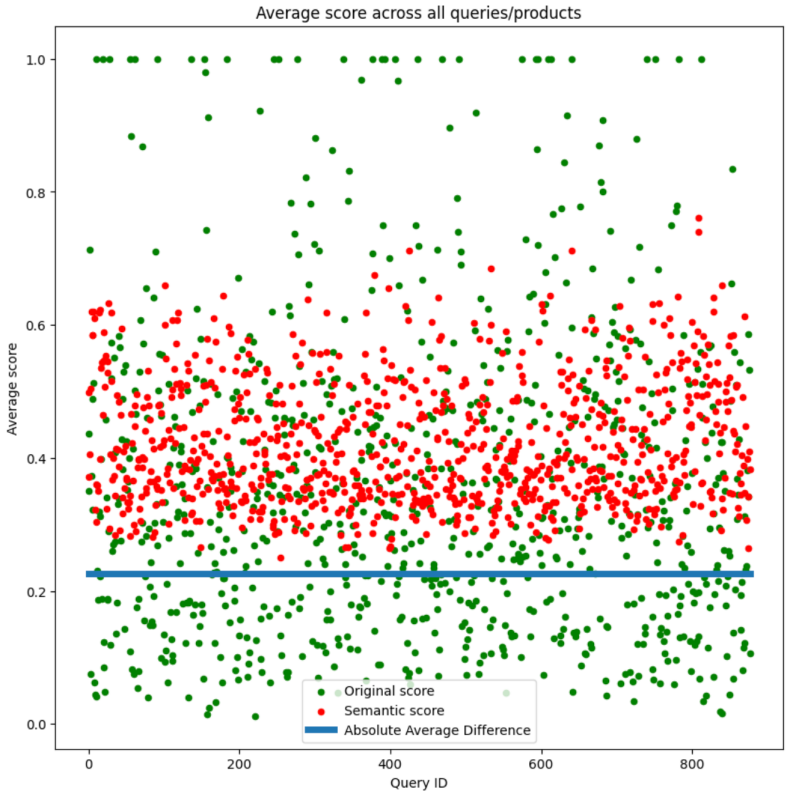

Figure 6.11: Average difference between ElasticSearch scores & model D scores. The scores from model D are packed more densely together than ElasticSearch's scores.

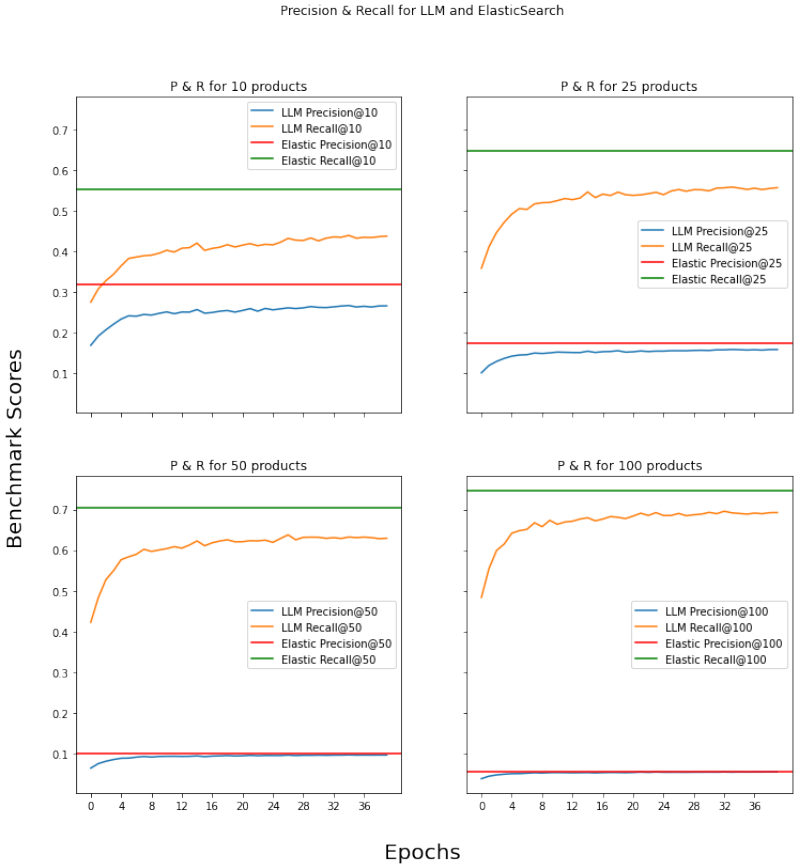

Figure 6.12: Model E's Precision & Recall scores.

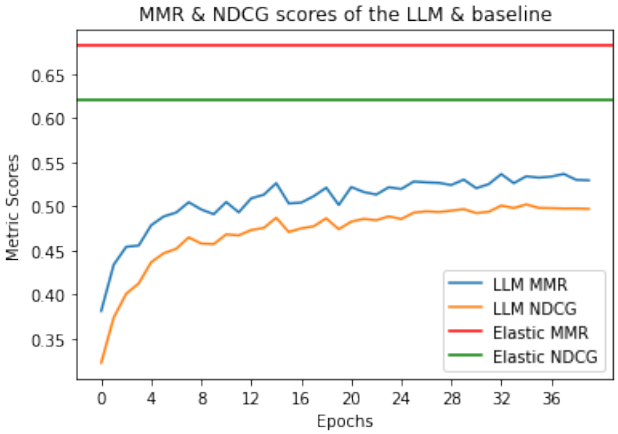

Figure 6.13: Model E's Mean Reciprocal Rank & normalized Discounted Cumulative Gain benchmark scores. As visible, ElasticSearch outperforms the model on both these benchmarks.

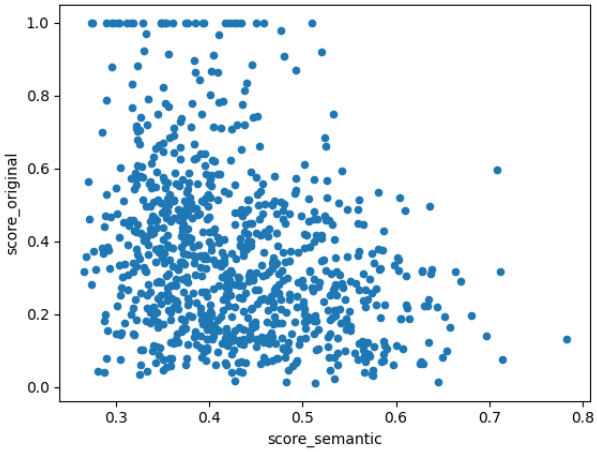

Figure 6.14: Average score of ElasticSearch & model E across the queries. Each point marks the average scores of all products for the given model by representing the score on each axis.

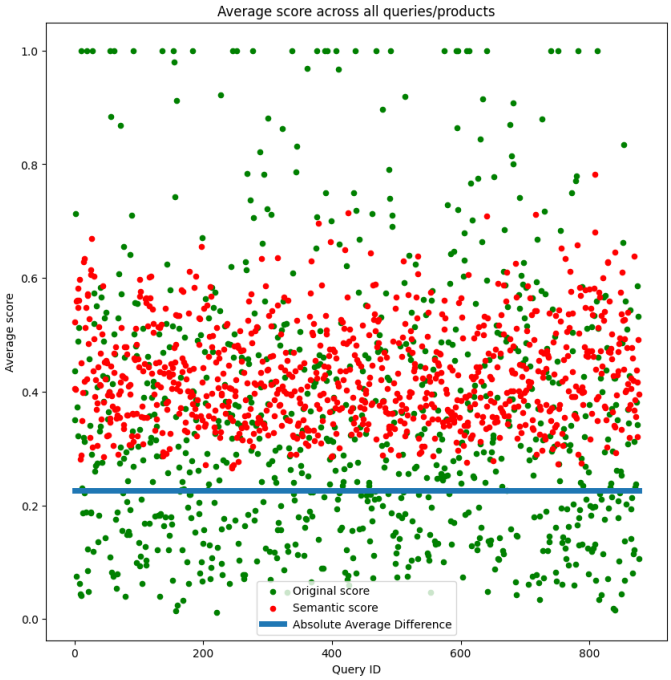

Figure 6.15: Average difference between ElasticSearch scores & model E scores. The scores from model E are packed more densely together than ElasticSearch's scores.

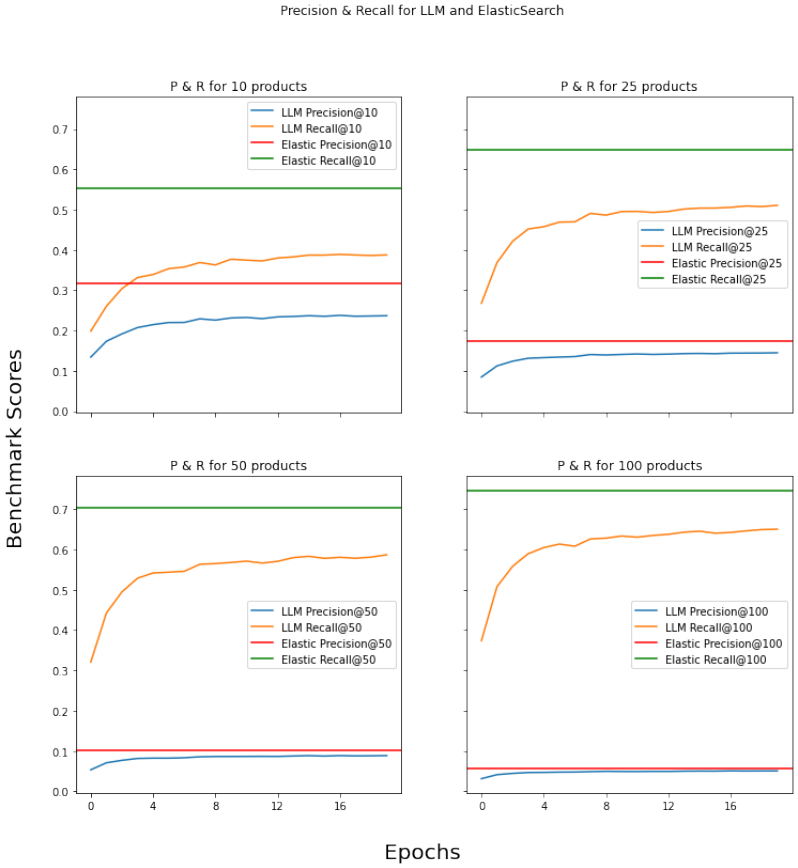

Figure 6.16: Model F's Precision & Recall scores.

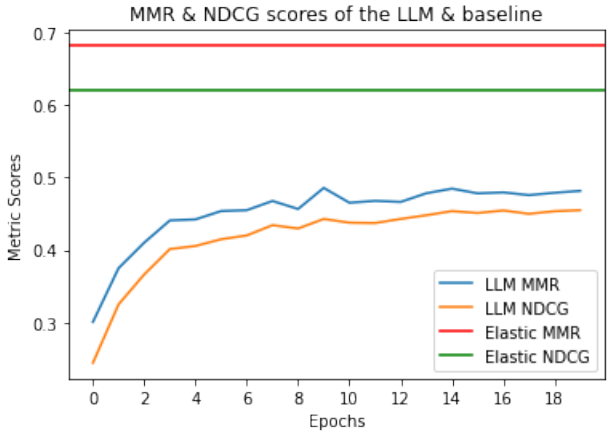

Figure 6.17: Model F's Mean Reciprocal Rank & normalized Discounted Cumulative Gain benchmark scores. As visible, ElasticSearch outperforms the model on both these benchmarks.

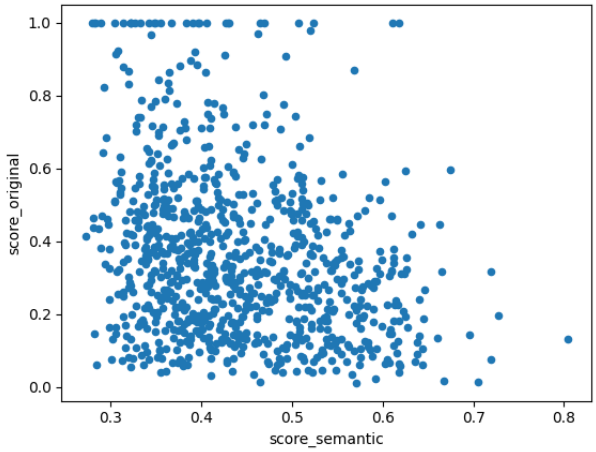

Figure 6.18: Average score of ElasticSearch & model F across the queries. Each point marks the average scores of all products for the given model by representing the score on each axis.

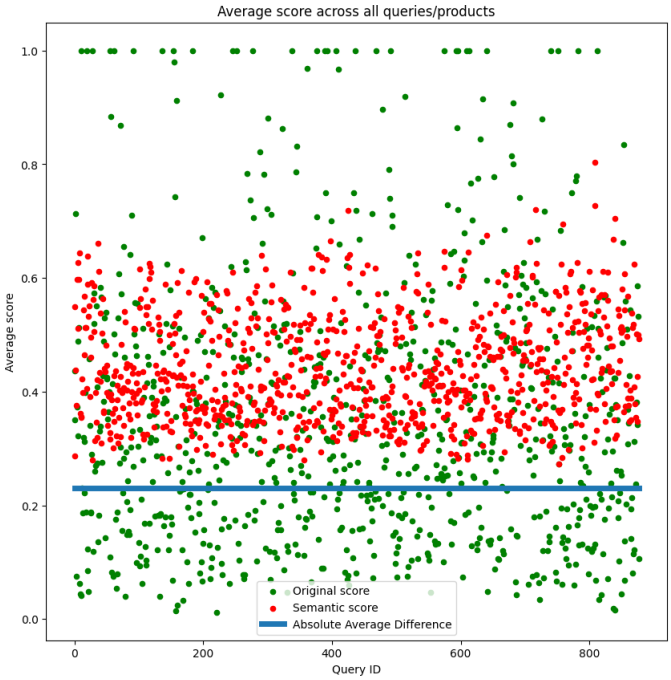

Figure 6.19: Average difference between ElasticSearch scores & model F scores. The scores from model F are packed more densely together than ElasticSearch's scores.

6.2 Supplementary Material

This section contains supplementary material which can be examined to provide more insight into the model results. Specifically, the qualitative analysis is supported more extensively by including the original tables that model returned as output for the queries defined in Table 4.2 across each of the models defined in Table 3.5. This supplementary material can be found in the Github repository associated with this research[1].

[1] `https://github.com/Mitchell-V/Thesis`