Master Thesis Data Science

Radboud University Nijmegen

Applying Learned Sparse Retrieval

Author:
Tom Rust
s1040068

First supervisor/assessor:
Prof. Dr. Ir. Djoerd Hiemstra

Second assessor:
Prof. Dr. Ir. Arjen de Vries

June 25, 2024

Abstract

Machine learning algorithms are achieving better results each day and are gaining popularity. The top-performing models are usually deep learning models. These models can absorb vast amounts of training data, improving prediction results. Unfortunately, these models consume a large amount of energy, which is something that not everyone is aware of. In information retrieval, large language models are used to provide extra context to queries and documents. Since information retrieval systems typically have large datasets, a suitable deep learning model must be chosen to find a balance between accuracy and energy usage. Learned sparse retrieval models are an example of these deep learning models. These models work by expanding all documents to create the optimal document representation that allows this document to be found correctly. This step is done before creating the inverted index, allowing for conventional ranking methods such as BM25.

With this research, we compare different learned sparse retrieval models in terms of accuracy, speed, size and energy usage. We also compare them with a full-text index. We see that on MS Marco, the learned sparse retrievers outperform the full-text index on all popular evaluation benchmarks. However, the learned sparse retrievers can consume up to 100 times more energy whilst creating the index, which then has a higher query latency, and it uses more disk space. For WT10g we see that the full-text index gives us the highest accuracies whilst also being more energy efficient, using less disk space and having a lower query latency.

We conclude that learned sparse retrieval has the potential to improve accuracy on certain datasets, but a trade-off is necessary between the improved accuracy and the cost of increased storage, latency, and energy consumption.

Contents

Chapter 1

Introduction

1.1 Motivation

The rise of neural networks and deep learning has had a lot of impact on machine learning tasks. However, it was only until the introduction of BERT [14] that deep learning was properly introduced in the field of information retrieval.

We zoom in on two important steps in the information retrieval pipeline:

1. First stage retrieval: Efficiently capture the top N most relevant documents for a query.

2. Re-ranking: Using expensive but accurate methods, try to re-rank these top N documents.

For this thesis, we are interested to try to use the knowledge from the re-rankers and store this in an inverted index. This is called learned sparse retrieval [42] and some models (such as Splade [15] and DeepCT [11]) have achieved near state-of-the-art scores on the MS Marco dataset, whilst being much more efficient [39]. However, these are still large language models, so they still consume a large amount of energy. We also investigate the increased cost in energy usage, disk usage and latency, such that a balance can be found between accuracy and cost.

1.2 Research

Learned sparse retrieval seems like a promising method to use for the Open Web Search[1] index [21]. Learned sparse retrieval requires large amounts of computing power to create the inverted index, but once the index is created, ranking can be done using conventional methods such as BM25. Because of this focus on efficiency, it is much more suitable to run on consumer grade

[1] `https://openwebsearch.eu/`

hardware. It is also possible to use the learned sparse retrieval model to expand the queries, however this would mean that specialized hardware would be required for each single query. Hence, we do not use query expanding.

Unfortunately, Open Web Search does not have proper training data. So we can not fine-tune our network on the Open Web Search dataset. An alternative is to use a zero shot approach [56], where we use a model that is fine-tuned on MS Marco. The idea is that it then generalizes to a different dataset. Our research question is:

- **RQ1:** Can we achieve better quality retrieval by using a learned sparse retrieval method compared to using a standard inverted index?
- **RQ2:** How much energy is needed to compute the indexes in **RQ1**?
- **RQ3:** How large are the indexes generated in **RQ1**?
- **RQ4:** What is the query latency when querying on indexes in **RQ1**?

For **RQ1**, we plan to do the experiments on the WT10g dataset [2], as this should be a good representation of the web [53]. This includes a small test set of 100 query / document pairs. We compare different inverted indexes with a BM25 ranker in terms of ranking and in terms of energy usage. For evaluating the quality of ranking we use MRR, NDCG, MAP, recall, and precision. To compare energy usage for **RQ2**, we use CodeCarbon [9] to measure the energy usage during all experiments.

1.3 Content

This thesis is focused on applying learned sparse retrieval on real datasets. Our goal is to aid in making the decision to use a learned sparse retrieval method when creating an information retrieval system. In chapter 2 we explain the background concepts required to understand learned sparse retrieval. This includes an overview of the information retrieval pipeline (section 2.2), some information on commonly used datasets (section 2.3) and an introduction to learned sparse retrieval (section 2.5). Next to that, we explain the concept of zero-shot retrieval in section 2.6 and provide some common background on the environmental aspect of information retrieval in section 2.9.

In chapter 3 we explain the setup of our experiments, in which we compare different learned sparse retrieval models with BM25 baselines on the MS Marco and WT10g datasets. We compare the results of these experiments in chapter 4.

In the conclusions (chapter 5) we answer our research questions. Some possible improvements for learned sparse retrievers on datasets with long documents are given in chapter 6.

Chapter 2

Background

In this chapter, we explain the information retrieval pipeline in section 2.2. Some background information on the MS Marco and WT10g dataset is given in section 2.3. In section 2.5 we explain how learned sparse retrieval can help solve the vocabulary mismatch problem (as explained in section 2.1). After that, we give some background on the environmental impact of information retrieval in section 2.9.

2.1 Vocabulary Mismatch Problem

One of the classic problems in NLP is the lexical ambiguity [26]. Languages are ambiguous, things or concepts can be described by different words. This means that two different words can have the same meaning, when a query is done on a word (School) and a synonym of that word (University) is in a document, we might also want that document to be retrieved. This is called the vocabulary mismatch problem. Unfortunately, this is unlikely to happen with traditional ranking methods such as BM25.

2.2 Information Retrieval Steps

In this section, we describe some steps that are present in information retrieval systems. See Figure 2.1 for a visual representation.

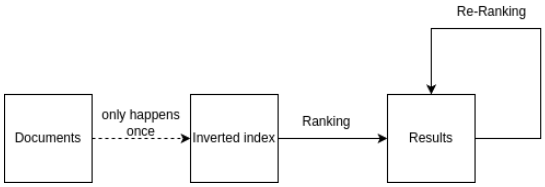

Figure 2.1: Information retrieval pipeline

2.2.1 Documents

The first step in an information retrieval system is getting all documents. We need to capture all available documents that we want to query on. Typically, web crawlers do this, some readily available examples are MS Marco [43] and WT10g [2]. Next to that, an important part of this step is to get the documents in a common format. See section 2.3 for more information on these datasets.

2.2.2 Inverted Index

Once we have successfully captured and stored our documents. We want to create an inverted index. This is a large index that basically for each word captures in which documents it occurs and how often [10]. This inverted index can be stored on disk, so it only has to be created once. An index that is created directly from the documents is also called a full-text index [36].

2.2.3 Ranking

In the ranking step, we get a query and we want to return the top N matching documents. For this, usually a relatively simple BM25 [50] algorithm is used. This is an efficient algorithm that can find documents that have words matching the words in a query. BM25 takes the length of the document, average document length and relative word frequency in consideration when ranking.

2.2.4 Re-ranking

The first ranking steps narrowed down the number of possible candidates. However, in this step, expensive re-rankers are used to find the best matching documents among the N candidates. Typically, this is done by a BERT powered reranker [44]. These rerankers can provide additional meaning and context to the documents, and we can overcome the vocabulary mismatch problem. For example, by linking entities in the text with the abstract from Wikipedia page [18, 48] belonging to this entity. The rerankers only consider the documents that were selected in the ranking step (section 2.2.3). Unfortunately, these algorithms are very costly and require specialized hardware (such as graphic processing units) to be executed. This has to be done for every query, so not only does it consume a large amount of electricity, it also increases query latency.

2.2.5 Evaluation

After retrieving a set of documents, there are some evaluation metrics that we can use to determine the retrieval quality. For the queries that we want

to evaluate, we have an ordered list of the documents that should be found when doing a query. This is also known as qrels. As mentioned in section 1.2, we use the following methods for evaluation: MRR (Mean Reciprocal Rank) [33], NDCG (Normalized Discounted cumulative gain) [24], MAP (Mean Average Precision) [8], recall and precision.

2.3 Datasets

In this section, some background information is given on the datasets that are used in this paper. We explain their origin, and we give the size of the datasets and common evaluation methods.

2.3.1 MS Marco

MS Marco was released in 2016, it has two variants: document and passage. The document dataset contains 3.2 million documents, and it has 367 thousand training queries, with each having one matching document. The MS Marco passage dataset contains passages of the documents in the document dataset. This results in a larger dataset with 8.8 million documents (or passages), to train on these passages there are 503 thousand queries with most having only one matching passage. The MS Marco passage dataset has an average passage length of 56 words[1]. The average number of distinct words is 42. MS Marco contains web document and Bing query relevance pairs that have been released by Microsoft. For our experiments, the MS Marco passage dataset is used, evaluation is done using 6980 queries that have labelled query relevance pairs. The official evaluation measure is the mean reciprocal rank for the first 10 results. However, it is common to also use the recall at 1000 and the normalized discounted cumulative gain at the first 10.

2.3.2 WT10g

WT10g, also known as webtrack is, like the name implies, a 10 GB dataset. It contains 1.69 million documents that are crawled from the internet. WT10g has an average document length of 562 words, the average number of distinct words is 259. Both these values are higher than MS Marco. WT10g is constructed in such a way that it should represent the internet, special care is taken to select documents that have relations with other documents and to support a wide spread of queries [3]. WT10g was released in 2000, so it is not the most recent dataset. WT10g is evaluated using the precision at the first 30 and the mean average precision. WT10g has a very small development set that can be used for evaluating. This contains 400 queries

[1] `https://github.com/microsoft/MSMARCO-Passage-Ranking/blob/master/stats.txt`

and was made by information retrieval researchers, of these 400 only 100 queries are used for scoring[2].

2.4 Sparse vs Dense

In the information retrieval pipeline that was explained in section 2.2.2, the documents are sparsely encoded. This means that each document is encoded as a very large vector, where each element in the vector represent a word. The value for each element represents the number of times this word appears in our document. Next to sparse retrieval, there also exists dense retrieval [25]. With dense retrieval, we have a smaller but very dense vector from which we cannot easily grasp information. The main advantage of sparse retrieval is that we can create a conventional (inverted) index. Because of this, we can use simple methods such as BM25 [50]. With this we can retrieve documents quickly and we can inspect biases. Another difference is that storing a dense index may require 70 times as much storage compared to a (non-learned) sparse index [7].

2.5 Learned Sparse Retrieval

With learned sparse retrieval, we try to learn information captured by neural re-rankers (see section 2.2.4) and store this in the inverted index. This idea was first introduced in 2018 by Zamani et al [60]. This means that the expensive re-ranking part is less important, and it may even be omitted completely. It does mean that building an inverted index becomes a GPU task, which means it becomes more expensive and it takes much longer. However, this is a one time operation, so we only need access to GPU hardware for this task. Another advantage is that the machine learning model is no longer limited by the documents that are selected in the initial ranking stage (see section 2.2.3).

There are different implementations of learned sparse retrieval models, some of them are explained in section 2.7. What all models have in common is that they have an encoder that encodes a document into a vector. This encoder can have some extensions, such as a Multi-Layer Perceptron [51] or a distillation [23] layer which expands the terms in a document, for example using an external model such as DocT5Query [45]. Another possible extension is the use of a BERT model (also being referred to as a masked language model) to alter the term weights based on the BERT output. The second thing that they have in common is that they all have a method to keep the output vector sparse. This basically limits the number of different words that can occur in a generated document. Finally, since we are talking

[2]These can be downloaded from `https://github.com/castorini/anserini-tools/tree/master/topics-and-qrels/topics.adhoc.451-550.txt`

about learned models, we need a certain set of training data. Some models use labels generated by humans, others use labels that are created by other rankers (such as a BM25 model with a neural re-ranker). In that case, we are basically transferring knowledge from the neural re-ranker onto our learned sparse model. This is also known as transfer learning [46].

2.5.1 Machine Learning

When we regard learned sparse retrieval models from a machine learning perspective, we can say that a learned sparse retriever predicts the importance of a term in a document. This is done for each term in the vocabulary, so it is not limited to terms in the document. To optimize this prediction, models use a loss function that typically is a combination of the ranking loss [6] and the regularization loss. The ranking loss measures how many of the selected documents (using the learned sparse retriever) are relevant for a query, and the regularization loss captures the "sparsity" of the inverted index. Since this regularization loss is not always perfect, when making predictions, the models also have a maximum number of words that are in the new document representation. The words with the highest scores are chosen. For this the letter k is used, a typical value is 400. So this means that the new representation of a document has at most 400 different words.

It has been shown that the amount and quality of "hard negatives" can significantly boost the performance of representation learning models [1, 30]. Learned sparse retrieval is a form of representation learning, since we are trying to represent our documents in such a way that our rankers can make better predictions [5].

2.5.2 Latency

One downside of using learned sparse retrieval models is that they tend to increase the size of the created inverted indexes. This can result in an index that is more than 10 times the size of a non-generated index for MS Marco [42]. This larger inverted index also means that query latency can be 40 times larger (see Table 4.5). This issue is pointed out by Lassance et al. [28] They propose to use a Two-Step version of Splade to decrease the query latency.

2.5.3 Large documents

With this research, we apply learned sparse retrieval models to a large set of documents. The WT10g dataset that is used has an average document length of 562 words (see section 2.3.2). This is larger than the typical passages that are used for evaluating information retrieval models, MS Marco for example has an average length of 56 words. Nguyen et al. propose

several methods for learned sparse retrieval methods to handle larger documents [41]. Their approach is to split the large document up into multiple smaller ones, make the calculations on the smaller documents, and then merge the results back together into a new representation of the original document. They claim that the max-score aggregation method is the most robust. They also explain a sequential dependence model [40] approach, based on this max-score method that can outperform the naive max-score method.

2.6 Zero-shot Retrieval

Machine learning models require large amounts of training data to make accurate predictions. However, in the field of information retrieval, this is often quite difficult to obtain [12, 55]. Because of this, most neural information retrieval models train on the MS Marco dataset, since this dataset has a very large training set (see section 2.3.1). So the actual dataset that the model will retrieve data from may be very different from MS Marco, so the model should be able to generalize as best as possible. This technique of using a model on data that it has not seen before (for example in a training set) is called zero-shot retrieval. Strictly speaking, BM25 is also a zero-shot retrieval method, since it does not have a training phase. Research has been done to investigate the performance of learned sparse retrieval in a zero-shot scenario by Thakur et al [56]. This has shown that from the learned sparse retrieval models, Spladev2 performed the best on the BEIR benchmark.

2.7 Learned Sparse Retrieval Models

In this section, we explain Splade and Sparta, both are learned sparse retrieval model families.

2.7.1 Sparta

Sparta (Sparse Transformer Matching) was introduced in 2020 [61], it does not do query expansion, and it offers the possibility for knowledge distillation. Sparta tries to predict the word scores that rank the correct document as high as possible, this is similar to Splade (see section 2.7.2), where they predict the importance of a word in a document. Sparta uses a masked language model for both term scoring and document expansion. As a sparse regularizer, Sparta uses a naive approach to keep the top k words that have the highest score. Typically, for k a value of 400 is used. We train and evaluate a version of Sparta on MS Marco in section 3.1.

2.7.2 Splade

Splade was introduced in 2021 [17]. Since the release, some changes have been made to improve on the original model, such as Splade v2 [15]. Splade uses the BERT masked language model to alter the term weights. What sets Splade apart from other learned sparse retrieval models is that Splade uses the FLOPS regularizer [47] to control the sparsity. A FLOPS regularizer minimizes the number of floating point operations when making a representation, this leads to sparser representations.

To overcome the vocabulary mismatch problem (see section 2.1), document expansion is added to the Splade pipeline to add more context to documents. DistilSplade-max has been shown to perform significantly better than the default Splade model. In this model, the hard negatives used for training are mined using a distillation technique. We experiment with a Splade model in section 3.6.

2.8 Environmental Impact

Both deep learning and environmental awareness are two popular topics with a lot of interest from the scientific community. Because of this, it is not a surprise that research has been done to determine the environmental effects of deep learning models. Desislavov et al. has shown that the inference costs of a deep learning model can be up to 10 times the cost of training the model [13].

Next to that, the AI sector is also focused on creating sustainable AI [57]. Their main goals are to create awareness for the environmental effects of AI models, but also for social economic effects [4]. Part of this includes a framework to measure environmental impact of any experiment [20]. In such a way, researchers are immediately confronted with the consequences of their experiment choices.

2.9 Environmental Impact of Learned Sparse Retrieval

To minimize carbon emissions, Scells et al. proposes to reduce, reuse and recycle as much as possible [52]. The nature of learned sparse retrieval leans itself perfectly for this. Since the resource-intensive GPU calculations are only required for computing the index, whilst this index can be queried in a much more efficient way. Secondly, the datasets that are used to fine-tune the models are all based on MS Marco and these are available online[3]. This means that reusing these datasets is relatively easy.

[3]For example, the msmarco_multiple_negative is available on huggingface `https://huggingface.co/datasets/sentence-transformers/msmarco-hard-negatives`

2.9.1 Electricity Consumption

As explained in section 2.5, one of the advantages that learned sparse retrieval provides us is that the resource-intensive GPU computations are done when creating the inverted index. This also makes it easier to compute the electricity usage, since these tasks are run on our own hardware. Researchers, such as Strubell et al. [54] compute the carbon emissions from their energy measurements and compare tasks based on the amount of carbon emissions. However, since the carbon emissions are region dependent [27] and thus should not be a concern when comparing implementations, we decided to focus on electricity usage. From electricity usage, it is always possible to compute the carbon emissions using tools such as Electricity Maps[4].

2.9.2 Terminology

As explained in section 2.9.1, we focus on electricity usage. We use the terms energy and electricity interchangeably (even though this is not strictly accurate, since electricity is a form of energy). We make a distinction between power (P) and energy (E) [34]. Energy is the amount of work done, for which we typically use the unit kilowatt-hour or kWh. When we are comparing different appliances, power may be used. This is the amount of energy that is used per time unit, this most likely is in watt (W) or Kilowatt (kW). See equation 2.1 for the relation between energy (E in kWh), power (P in kW) and time (t in hours)

$$E = P \times t \tag{2.1}$$

2.9.3 Measuring Tools

Nowadays, most processors support power measuring. Tools query the power usage of the processors at certain intervals and combine this to compute the total energy usage (using the equation 2.1). It is important to note that these tools will always make an estimate. Furthermore, different processing units may be optimized for different tasks, or even for running tasks at very large scales, so measuring electricity may never be perfect.

For this research, we use the CodeCarbon tool [9][5]. This tool can measure the CPU, GPU, and RAM power usages. To measure GPU energy usage, it uses the NVIDIA Management Library. This means that it can very accurately measure power usage from the graphics cards. To measure the power usage of the processor, the Intel RAPL interface is used. This can also quite accurately measure the power usage of the processor. The RAPL interface seems to measure the total power usage of a processor, so when a

[4] `https://www.electricitymaps.com/`
[5] `https://github.com/mlco2/codecarbon`

multiple-core processor is used, it measures the power of all cores. Finally, to measure the RAM power usage, it is assumed that every 8 GB of RAM uses 3 watts (so 0.375 watt per GB of RAM). Other things like storage or internet usage are not taken into account.

Chapter 3

Experiments

Ultimately, we want to create an improved inverted index using a learned sparse retrieval model. However, this index may become much larger. This means that performing queries takes more time, but we also require a larger amount of storage to store the index. Another aspect is that creating such an index takes longer, and it consumes more resources.

In these experiments, we investigate these variables, such that potential users of a learned sparse retrieval model can make an informed decision on which learned sparse retriever to use. Our experiments are done using both the MS Marco dataset and the WT10g dataset (see section 2.3). In the experiments, we measure the time it takes and estimate the energy usage using CodeCarbon (see 2.9.3). When evaluating the rankers, we use the suggested evaluation methods for both MS Marco and WT10g. This means that we evaluate on MRR@10, R@1000, nDCG@10, P@30 and MAP (see section 2.2.5). When doing this, we can also evaluate the zero-shot performance of our trained models on WT10g (see section 2.6).

Some changes to the original learned sparse retrieval framework[1] were made to support the WT10g dataset, for example. The new code can be found on GitHub[2].

For the experiments, we use the Radboud csedu cluster[3], this has up to 80 GB of memory and 5 RTX 2080 Ti graphic cards with an Intel Xeon Intel(R) Xeon(R) Silver 4214. We use a different server for CPU tasks (such as indexing and querying), this server does not have graphics cards, and it uses an Intel Xeon E5-2670 CPU with up to 32 GB of memory. To use the Radboud cluster, a repository with examples is available on GitLab[4].

[1] `https://github.com/thongnt99/learned-sparse-retrieval`
[2] `https://github.com/tomrusteze/learned-sparse-retrieval`
[3] `https://wiki.icis-intra.cs.ru.nl/Cluster`
[4] `https://gitlab.science.ru.nl/trust/learned-sparse-retrieval-scripts`

3.1 Using Sparta on MS Marco Passage

To become familiar with creating an inverted index using a learned sparse retrieval method, we follow the experiments using the unified framework from Nguyen et al. [42][5] We do the experiments using Sparta (see section 2.7.1) on the MS Marco passage dataset. Running this experiment consists of 5 steps:

1. Fine-tuning the Sparta model
2. Running inference on the Sparta model
3. Creating an inverted index using these score
4. Run queries on the created index
5. Evaluate the query results

Some of these steps will have great similarities with the basic information retrieval steps (see section 2.2), some differences include that we already have a collection of documents, and we do not perform re-ranking. Next to that, we are also interested in evaluating the results.

In the following sections, we explain the steps that we have taken to get results. Note that we used the MS Marco passage dataset, this means that we do not have the entire documents. When using the WT10g dataset, we are talking about documents. So it may also be interesting to run this experiment on the MS Marco document dataset. However, This is more computationally intensive. Note that we chose Sparta for this experiment because it can give us a good learned sparse retrieval baseline since Sparta can be seen as the naive approach to learned sparse retrieval.

3.1.1 Fine-tuning the Sparta Model

In this step, we get the Sparta model and fine-tune it using the query relevance evaluations from the MS Marco dataset. More specifically, we use the sparta_msmarco_distil configuration from the learned-sparse-retrieval framework from Nguyen et al. [42] This is heavily based on the original Sparta experiments, but with some slight changes in training data and hyperparameters. Running this step using 2 graphic cards and a per-device batch size of 4 took around 8 hours.

In an ideal scenario, it would be easier to re-use a pretrained model. However, not all models used in these experiments have a pretrained model that is ready to use. So it is necessary to do some fine-tuning, furthermore, the learned-sparse-retrieval framework has no proper support for pretrained models.

[5] `https://github.com/thongnt99/learned-sparse-retrieval`

3.1.2 Inference

In this step, we use the just-trained model to predict the words and their scores for each document. So this is the step where we compute the actual sparse vectors. The idea is that these words represent the document in the best way possible. In this way, we prepare our documents for the inverted index step (see section 2.2.2). Inference takes the most amount of time and computing power. We use the ir_datasets [38][6] framework to easily obtain the dataset.

Running this step took around 5 hours using 5 graphic cards with a batch size of 64 on MS Marco passage, however, this is very likely to scale based on the input dataset. Both the length of the individual documents and the number of documents may influence this. The output of this experiment is a large file (around 100 GB) with text and terms for each document. A small extract of this can be found in appendix A.1. Note that the words sometimes contain hashtags (#). This is because the documents are using the BERT word pieces vocabulary.

3.1.3 Index

In this step, we create an inverted index using Anserini [31][7]. More specifically, we use a modified version[8] that supports our input with pairs of words and scores. Our input is pretokenized (our document is already split up into different words), and it may contain hashtags because of the BERT notation.

3.1.4 Query

In this step, we run queries on the created Anserini index [58, 59]. We use the MS Marco passage small development set[9]. Running these queries (6980) took roughly 1 hour. Our index is tokenized by BERT, so before querying, we also tokenize the query using a BERT tokenizer. We do not perform query expansion, since this is not something that we are interested in for this research.

3.1.5 Evaluation

Here, we evaluate the results that we got on the queries in step 4 (see section 3.1.2) with the relevance results. We do this using the ir-measures framework [37][10]. We evaluate our queries on recall at the first 1000 results,

[6] https://ir-datasets.com/index.html
[7] https://github.com/castorini/anserini
[8] https://github.com/thongnt99/anserini-lsr
[9] https://ir-datasets.com/msmarco-passage.html#msmarco-passage/dev
[10] https://ir-measur.es/en/latest/

mean reciprocal rank@10, precision@30, average precision and normalized discounted cumulative gain@10.

See Table 4.1 for the results, we see that they are very similar to the results from Nguyen et al. (MRR@10 = 35.3, R@1000 = 96.8). These small differences can be explained by different training environments and testing setups.

3.2 Measuring Electricity

Since getting insight into the energy usage of learned sparse retrieval models is an important part of this thesis, we measure the electricity usage in this and all following experiments. As explained in section 2.9.3, there are tools that we can deploy to measure electricity usage with varying levels of accuracy. To determine the energy usage of building a learned sparse retrieval index, we run an experiment very similar to the one explained in section 3.1. One notable difference is that we now use a different configuration, which represents the experiment that was used in the original Sparta paper [61], this experiment is called sparta_original. Nevertheless, this experiment has the same structure as section 3.1.

3.2.1 Fine-tuning

This Sparta model is fine-tuned on an MS Marco model with additional negatives. This experiment was run with 5 graphic cards, 32 processors and 80 GB memory. See appendix A.2 for the final transmitted EmissionsData output. With this data, we can create Table 3.1 to show the energy usage for this step. As a sanity check, we can compare the estimated GPU energy

GPU (kWh)	**CPU (kWh)**	**RAM (kWh)**	**Total (kWh)**	**Time**
13.63	0.78	0.87	15.28	18.4 hours

Table 3.1: Energy usage according to CodeCarbon for fine-tuning Sparta

usage with a simple calculation. Since we have 5 RTX 2080 Ti's that have used 13.63 kWh in 18.4 hours. This means that one RTX 2080 Ti uses 148 watts on average (see equation 3.1). This is within the official 260 watts power rating[11]. It also makes sense that the graphics card is not running at its peak power for the entire time.

$$13.63/18.4/5 = 0.148 \ \texttt{kW} = 148 \ \texttt{W} \tag{3.1}$$

[11] https://www.nvidia.com/nl-nl/geforce/graphics-cards/compare/?section=compare-20

3.2.2 Inference

Running inference on the MS Marco dataset (8.8 million documents) to create the sparse vectors was faster than fine-tuning. We used the same 5 RTX 2080 Ti setup to do these computations. See Table 4.1 for the energy usage.

When we now compute the average power for a single graphics card, we see that with only 48 watts, it is considerably less than in section 3.2.1, where it was 148 watts.

$$1.19/4.99/5 = 0.048 \ \texttt{kW} = 48 \ \texttt{W} \tag{3.2}$$

3.2.3 Index and Querying

The index and query steps do not require a graphics card. Next to that, they also take a lot less time to compute. So because of this, they naturally use a lot less energy. See Table 4.1 for the energy usages for the index step. We index the entire MS Marco passage dataset (8.8 million documents) and use the queries from the small development set (6980 in total).

3.2.4 Evaluation

See Table 4.1 for the scores, these scores are a bit lower than the improved Sparta model used in section 3.1.

3.3 Evaluating MS Marco with BM25

In order to evaluate the performance of a generated inverted index using a learned sparse retrieval method. It may be useful to compare it with a full-text index. In both cases, we use BM25 to query the inverted index. In this experiment we build an index using the MS Marco passage dataset, we then perform queries on this created index using BM25. For the BM25 parameters k_1 and b we use the values 0.82 and 0.68 respectively. These parameters are tuned for MS Marco passage dataset [32]. We then evaluate the query results in the same way that we do in section 3.1.5.

See Table 4.1 for the scores and Table 4.3 for the energy measurements, we see that the scores are a bit lower than the Sparta models, however, the energy consumption is also much lower. We also present the query latency and size of the index in Table 4.5.

As a sanity check, we compare these scores with a BM25 baseline on MS Marco [35]. We see that both the recall and MRR are within 0.05 of the scores in the paper (MRR@10 = 0.1840, R@1000 = 0.8526). A small difference that is most likely caused by slightly different values for k_1 and b.

3.4 BM25 on WT10g

In section 3.3 we query on a full-text index using BM25. This gives us a baseline for comparing the index size, time/energy to create the index, query latency and evaluation scores. As explained in section 2.3.2, WT10g is evaluated on the mean average and precision for the first 30 results.

In this experiment, we create the index, query it and then evaluate the scores. The three statistics that we are mainly interested in are the evaluation scores, energy usage and the index size. See Table 4.4 for the energy measurements of running BM25 on WT10g.

Since WT10g is considerably smaller than MS Marco, it is much faster. We only have 400 test queries, which are processed at a rate of 117 queries per second. The scores are visible in Table 4.2. We see that our mean average precision of 0.1951 is very close to results in literature [19] (MAP = 0.1842).

3.5 Using Sparta on WT10g

In section 3.1 we trained a Sparta model to recreate the MS Marco index. Next to that, in section 3.4 we experimented with the WT10g dataset. In this experiment, we combine these two experiments by creating an index for WT10g using Sparta. Since the passages in WT10g are larger than in the MS Marco dataset, we do multiple experiments with values of 400 and 1600 for k (see section 2.5.1). We can also set the parameter for the maximum input length (m). As a default, this is set at 256. We do some experiments with a maximum input length of 512. Setting this higher is not possible, since this is a limitation of the BERT model.

We again use the sparta_msmarco_distill model. See Table 4.2 for the evaluation scores and Table 4.4 for the energy usage. We see that our scores differ a bit from using BM25 on WT10g. An important thing to note is that Sparta has been fine-tuned on MS Marco, which contains much smaller documents. So it may not be able to capture information from larger documents as good as smaller passages.

When we directly compare the different scores for different values of k, we see that a higher value yields slightly better results. However, the indexing time (and energy consumption) has increased quite a lot. Furthermore, the index generated with $k = 1600$ is also much larger (see Table 4.6).

Comparing different values for m, we see that the index size is increased massively. This is most likely because, with a higher value of m, the model can grasp more information.

3.6 Splade

One of the most popular learned sparse retrieval models is Splade, since Splade has proven to be very effective, and it uses the most advanced techniques (See section 2.7.2 for more information). In this experiment, we use a variant of the Splade model to create an index on the MS Marco dataset and the WT10g dataset. It is also possible to use a pre-trained model of Splade[12], we do this in section 3.7. We use the splade_asm_msmarco_distil_flops_0.1_0.08 configuration to do our experiments. See Table 4.1 for the evaluation scores on MS Marco and Table 4.3 for the energy usage.

We see that for MS Marco this Splade model performs slightly worse than the baselines in the paper, most likely this is because of a slightly different configuration that is used for both experiments. The difference is about 0.07 for both MRR@10 and R@1000 (MRR@10 = 0.3790, R@1000 = 0.9800). Furthermore, we do not use query expansion, which may also influence the final scores.

We have also tried the splade_msmarco_multiple_negative configuration. This results in lower evaluation scores, but also a much smaller index compared to the splade_asm_msmarco_distil_flops_0.1_0.08 configuration.

3.7 Pretrained Splade

As explained in section 3.1.1, we needed to fine-tune the models for them to work properly. This is one of the limitations of the framework that was used up until this experiment. So to solve this, we use the Splade framework [16][13]. This framework has support for all pretrained Splade models that are published by the authors[14]. We experiment with using the splade−v3−doc and splade−cocondenser−ensembledistil configurations. The Splade framework was altered slightly to support energy monitoring, hugging face authentication and the WT10g dataset. These are trivial modifications, for which no code is provided. The scores for MS Marco can be found in Table 4.1. We see that the differences between the splade−cocondenser−ensembledistil configuration and the model that is fine-tuned by us (flops configuration) is minimal. However, we also see that the recently released splade−v3−doc model [29], which is suited for scenarios without query expansion, outperforms all other models on MS Marco. Furthermore, the scores on WT10g can be found in Table 4.2, the models were run on WT10g with a value of 512 for m (see section 3.5). We see that splade−v3−doc again outperforms the Splade models. However, it can not beat some Sparta models or the BM25 benchmark.

[12] `https://huggingface.co/naver/splade_v2_distil`

[13] `https://github.com/naver/splade`

[14] Currently 20 models are published (June 4th 2024). They can be found at `https://huggingface.co/naver?search_models=splade`

3.8 Long Documents

As explained in section 3.5, learned sparse retrievers can only process 512 words of a document. Furthermore, in our case, they are pretrained on MS Marco, which has an average word length of 56. So we use a naive approach to split WT10g into passages.

1. We split all documents into passages with a length of up to n words.
2. All passages are expanded by splade−v3−doc.
3. We aggregate passages from the same document back into a single document.

See section 2.5.3 for more information.

After some tests, we decided to use a value of 400 for n. Values under 200 take over twice as long to compute and also give us lower scores. Because of the increased amount of passages, the runtime and energy consumption for the inference steps is quite a bit higher compared to using splade−v3−doc (see Table 4.4). Furthermore, we use the mean scores to aggregate the passages, since this seems to perform the best in our scenario. In Table 4.2 we see the scores of this experiment. We see that the original splade−v3−doc outperforms the altered model for long documents.

We re-used a naive implementation to split and aggregate documents, based on the implementation by Nguyen et al.[41].

Chapter 4

Results

In this chapter, we compare all experiments on different aspects. First, we look at the scores for both MS Marco passage and WT10g in section 4.1. Then we look at the energy usage for each experiment in section 4.2. In section 4.3 we compare the different models based on the index size and query latency. Finally, we consider all these different aspects together in section 4.4.

4.1 Evaluation Scores

In this section, we evaluate the performance of our trained models. See Table 4.1 for the scores on MS Marco and Table 4.2 for the scores on WT10g. Note that most of the learned sparse retrievers are not pretrained, but fine-tuned in our experiments. This means that their weights may differ slightly from models used in other papers, also influencing the final scores.

When we compare the scores on MS Marco, we see that $\text{Splade}_{\text{v3-doc}}$ (see experiment 3.7) outperforms the other models, most by quite a margin. This is not a big surprise, since the $\text{Splade}_{\text{v3-doc}}$ model is the latest version of Splade. Which is the best-performing model family according to Nguyen et al. [42] Furthermore, it is optimized for use without query expansion.

On the other hand, when we take WT10g as the dataset, we see that BM25 gives us the highest scores. A possible explanation is that our custom web parser is not as good as the parser from Anserini. Another explanation could be that learned sparse retrievers are unable to handle larger documents. In experiment 3.5, we have seen that the BERT limit of 512 also holds for the input for learned sparse retrievers. So documents with more than 512 words may lose some information when they are being used in a learned sparse retriever. To overcome this limitation, we tried to split the larger documents into smaller passages in experiment 3.8. This did not result in better scores. However, the Sparse regularizer from Splade can not work properly in this scenario, since it can not consider multiple passages.

Furthermore, it could be possible that the learned sparse retrievers can not improve on BM25 when they are used in a zero-shot scenario. This may happen because they do not generalize well enough. Moreover, it could be that the smaller passages in the training data from MS Marco are too different from the larger documents in WT10g. Additionally, WT10g is 16 years older than MS Marco, which may mean a difference in document content. Both WT10g and MS Marco contain web data, however, WT10g contains actual web pages (with HTML tags that have to be stripped) whereas MS Marco contains passages of text.

Model	**MRR@10**	**R@1000**	**nDCG@10**	**P@30**	**MAP**
BM25	0.1881	0.8614	0.2357	0.0187	0.1965
$\text{Sparta}_{\text{original}}$	0.2900	0.8992	0.3466	0.0229	0.2983
$\text{Sparta}_{\text{distil}}$	0.3511	0.9720	0.4162	0.0272	0.3574
$\text{Splade}_{\text{flops}}$	0.2981	0.9303	0.3558	0.0239	0.3056
$\text{Splade}_{\text{multiple-negative}}$	0.2641	0.9116	0.3146	0.0221	0.2717
$\text{Splade}_{\text{cocondenser}}$	0.2926	0.9330	0.3522	0.0240	0.3015
$\text{Splade}_{\text{v3-doc}}$	**0.3620**	**0.9778**	**0.4355**	**0.0278**	**0.3775**

Table 4.1: Evaluation scores for different models on MS Marco

Model	**MRR@10**	**R@1000**	**nDCG@10**	**P@30**	**MAP**
BM25	**0.5686**	**0.6889**	**0.3325**	**0.2167**	**0.1951**
$\text{Sparta}_{\text{original}}$	0.3798	0.2938	0.1857	0.1103	0.0675
$\text{Sparta}_{\text{distil}_{k\,=\,400,\,m\,=\,256}}$	0.5164	0.4163	0.2752	0.1603	0.1163
$\text{Sparta}_{\text{distil}_{k\,=\,400,\,m\,=\,256}}$	0.5332	0.4349	0.2924	0.1727	0.1262
$\text{Sparta}_{\text{distil}_{k\,=\,1600,\,m\,=\,512}}$	0.5643	0.5118	0.3165	0.1857	0.1462
$\text{Splade}_{\text{flops}}$	0.4141	0.3841	0.2414	0.1463	0.0985
$\text{Splade}_{\text{multiple-negative}}$	0.2920	0.3550	0.1676	0.1060	0.0788
$\text{Splade}_{\text{cocondenser}}$	0.3905	0.4530	0.2125	0.1450	0.0996
$\text{Splade}_{\text{v3-doc}}$	0.4716	0.4926	0.2747	0.1683	0.1259
$\text{Splade}_{\text{v3-doc (long documents)}}$	0.4576	0.3989	0.2214	0.1293	0.0897

Table 4.2: Evaluation scores for different models on WT10g

4.2 Energy Usages

Here we compare our different experiments based on energy usage. We do not consider the energy usage of fine-tuning a model, since models already have pre-trained models which should be used. In Table 4.3 we see that BM25 consumes much less energy, this is something that was to be expected since we do not have the expensive inference step. We do see that creating an index on MS Marco with any learned sparse retriever takes roughly the same amount of energy. This is approximately 90 times as much compared to creating a full-text inverted index. It is also interesting to see that the pretrained models consume more energy, whilst taking less time. This could be because of differences between the Splade framework and the learned sparse retriever framework.

The energy usage for creating indexes on WT10g can be found in Table 4.4, we again see that BM25 is by far the most energy-efficient option, and also the quickest. There is quite a difference between the energy usage for different learned sparse retrievers. In general, we see that the models with larger values for k and m have a higher power consumption. We also see that the experiment with long documents (see section 3.8) consumes more resources, this makes sense since it has to expand a lot more documents.

It should be noted that due to cluster limitations, the number of available graphic units that were used for each experiment was not always the same. This can result in different energy usage.

Model	**Inference (kWh)**	**Index (kWh)**	**Total (kWh)**	**Total time (h)**
BM25	**0.000**	**0.021**	**0.021**	**0.3**
$\text{Sparta}_{\text{original}}$	1.640	0.215	1.855	6.5
$\text{Sparta}_{\text{distil}}$	1.637	0.215	1.852	6.8
$\text{Splade}_{\text{flops}}$	1.799	0.078	1.877	5.2
$\text{Splade}_{\text{multiple-negative}}$	1.490	0.031	1.521	4.5
$\text{Splade}_{\text{cocondenser}}$	2.330	0.038	2.368	4.3
$\text{Splade}_{\text{v3-doc}}$	2.310	0.063	2.373	4.8

Table 4.3: Energy usage for different models when creating an index for MS Marco

Model	**Inference (kWh)**	**Index (kWh)**	**Total (kWh)**	**Total time (h)**
BM25	**0.000**	**0.010**	**0.010**	**0.1**
$\text{Sparta}_{\text{original}}$	0.627	0.082	0.709	2.7
$\text{Sparta}_{\text{distil}_{k\,=\,400,\,m\,=\,256}}$	0.589	0.044	0.633	2.2
$\text{Sparta}_{\text{distil}_{k\,=\,1600,\,m\,=\,256}}$	0.690	0.173	0.863	3.9
$\text{Sparta}_{\text{distil}_{k\,=\,1600,\,m\,=\,512}}$	1.168	0.157	1.325	5.2
$\text{Splade}_{\text{flops}}$	0.685	0.031	0.716	2.2
$\text{Splade}_{\text{multiple-negative}}$	1.013	0.012	1.025	3.1
$\text{Splade}_{\text{cocondenser}}$	1.468	0.020	1.488	2.7
$\text{Splade}_{\text{v3-doc}}$	1.510	0.024	1.534	2.9
$\text{Splade}_{\text{v3-doc (long documents)}}$	4.265	0.012	4.277	8.9

Table 4.4: Energy usage for different models when creating an index for WT10g

4.3 Index Size and Query Latency

As a final aspect, we compare the created indexes in terms of size and query latency. Measuring the index size is done by looking at the disk usage for all index files created by Anserini. To get the query latency, we get the number of queries per second that are processed by Anserini. With this, we calculate the query latency. For the results, see Table 4.5 for the index sizes and latency on MS Marco and Table 4.6 for the index sizes and latency on WT10g.

We see that the full-text indexes are typically much smaller than the indexes generated by the learned sparse retrievers. However, this hugely depends on the learned sparse retriever and dataset that is used. Indexes from Splade seem to be much smaller than those generated from Sparta. Most likely this is because the flops regularizer (see section 2.7.2) from Splade works better than the naive approach from Sparta (see section 2.7.1). We see that this trend is visible for both MS Marco and WT10g, with a single exception for the $\text{Splade}_{\text{multiple-negative model}}$ on WT10g, here we see that the index is smaller than the full-text index. Most likely this is because the Splade model can not capture all information from the original document (due to the limit) and the produced document becomes smaller than the original document. This phenomenon is not visible on the MS Marco dataset. Perhaps because the average number of distinct words for this dataset is 42, whilst this is 259 for WT10g (see section 2.3). It is however interesting to see that the inference step for the $\text{Splade}_{\text{multiple-negative}}$ model consumed much more energy than the inference step for the $\text{Splade}_{\text{flops}}$ model. This

may mean that more computations were done to create a smaller document representation.

It should be noted that the query latency can differ quite a lot on different hardware. Even though all the experiments are done on the same cluster, the only thing that we should look at is the relative difference between experiments. When we inspect the query latency for the different models on the MS Marco dataset, we see that queries on indexes generated by Sparta models roughly take 40 times as long as queries using a full-text index. When we compare the full-text indexes with indexes generated by Splade models, we see that the indexes generated by a Splade model can be up to 10 times as long (depending on which Splade model is used). Comparing the different latencies on the WT10g dataset, we see that the full-text index has the lowest query latency. When we compare this with the Splade and Sparta generated indexes, we see that query latency for Splade generated indexes is about 2–3 times as long and for the Sparta generated indexes it is about 6 times as long. All queries on an index created by a learned sparse retriever use a BERT tokenizer. So this increases their latency by a bit. Furthermore, the indexes generated by learned sparse retrievers are larger, so because of this, queries are more likely to take longer.

Model	**Index size (GB)**	**Query latency (ms)**
BM25	**0.6**	**12**
$\text{Sparta}_{\text{original}}$	9.1	588
$\text{Sparta}_{\text{distil}}$	9.3	500
$\text{Splade}_{\text{flops}}$	3.5	167
$\text{Splade}_{\text{multiple-negative}}$	1.5	50
$\text{Splade}_{\text{cocondenser}}$	3.1	63
$\text{Splade}_{\text{v3-doc}}$	2.5	216

Table 4.5: MS Marco index sizes and query latency for different models

Model	**Index size (GB)**	**Query latency (ms)**
BM25	0.7	**16**
$\text{Sparta}_{\text{original}}$	3.0	111
$\text{Sparta}_{\text{distil}_{k\,=\,400,\,m\,=\,256}}$	1.8	91
$\text{Sparta}_{\text{distil}_{k\,=\,1600,\,m\,=\,256}}$	6.3	83
$\text{Sparta}_{\text{distil}_{k\,=\,1600,\,m\,=\,512}}$	9.3	105
$\text{Splade}_{\text{flops}}$	0.9	56
$\text{Splade}_{\text{multiple-negative}}$	**0.5**	36
$\text{Splade}_{\text{cocondenser}}$	0.7	53
$\text{Splade}_{\text{v3-doc}}$	1.0	91
$\text{Splade}_{\text{v3-doc (long documents)}}$	1.7	43

Table 4.6: WT10g index sizes and query latency for different models

4.4 Comparison

When we combine the different aspects of the results (evaluation scores, energy usage, disk usage and query latency), we see that for MS Marco, a Splade model can provide quite a significant gain in evaluation scores whilst sacrificing some disk usage and query latency. However, the biggest difference is that it can take more than 100 times as much energy to compute. Whether this performance boost is worth it, depends on the scenario.

Looking at WT10g, we see that the BM25 retriever outperforms the learned sparse retrievers on all evaluation benchmarks, whilst also being more energy efficient and having a better query latency. Hence, using a learned sparse retriever for this dataset would not make sense. As mentioned in section 4.1, learned sparse retrievers seem to perform worse on larger datasets, which may explain this difference.

Chapter 5

Conclusions

In conclusion, we have successfully created a testing environment to measure the following aspects of generating inverted indexes (with or without using learned sparse retrieval models): Quality of ranking, electricity used when creating the index, query latency and index size. This setup was used to test different learned sparse retrieval models of the Splade and Sparta families on both the MS Marco and WT10g datasets.

To answer **RQ4**, query latencies from queries on the full-text index are lower than those from queries run on an index generated by a learned sparse retriever. In a best-case scenario, the latency of a query on a learned sparse retriever index is around 2–3 times as long. For **RQ3**, we inspect the size of the generated index. We see that the generated indexes from learned sparse retrievers tend to consume a bit more disk space, however, this depends on the number of words that the original document has.

To inspect the energy usage of learned sparse retrievers as asked in **RQ2**. We see that creating an index using a learned sparse retriever consumes roughly 100 times more energy than creating an index directly on the documents, whilst also taking longer and requiring special hardware.

When evaluating the scores on MS Marco, we see that our best performing learned sparse retriever, $\text{Splade}_{\text{v3-doc}}$, has higher scores on all used benchmarks compared to a full-text index. On the other hand, this is not the case for the evaluation of the WT10g dataset. Here, the full-text index has the highest scores on all benchmarks. It seems that the documents from WT10g and MS Marco are too different for the learned sparse retrieval models to work in a zero-shot scenario. To conclude **RQ1**: for the MS Marco dataset, it may be beneficial to use a learned sparse retriever to improve the quality of retrieval, even though this means a larger query latency, more disk usage and higher energy consumption. The evaluation scores are much higher compared to using a full-text index. Nevertheless, when indexing the WT10g dataset. It would be better to not use a learned sparse retriever. Since the full-text index scores higher on all aforementioned benchmarks.

Chapter 6

Future Work

In our experiments, we used a learned sparse retriever that is fine-tuned on the MS Marco passage dataset, this has an average word length of 56 words. Furthermore, the maximum number of words that can be processed by a learned sparse retriever per document is 512. This is a limitation when learned sparse retrievers are used to generate the words for documents larger than 512 words. In this chapter, some options are given to minimize this problem.

6.1 Fine-tuning on MS Marco Document

An initial approach would be to fine-tune the learned sparse retriever using the MS Marco document dataset. This contains large documents, with on average, 1131 words per document [49]. However, some internal structures of the learned sparse retriever, such as the document encoder, may also limit the number of words per document that the model accepts.

6.2 Sequential Dependence Models

In section 3.8 we experiment with splitting a document into multiple passages, after which we merge them back into a single document using a naive method. Alternatively, we could use sequential dependence models [40]. These seem to outperform naive methods when merging passages into documents [41]. This paper comes with a GitHub repository with examples[1].

6.3 Significance testing

We have not done any significant testing on the evaluation. With this, we could determine if the difference in evaluation scores is due to randomness

[1] `https://github.com/thongnt99/lsr-long`

or because one model is better than the other. This would be especially interesting on the WT10g dataset since this has a much smaller evaluation set. To compute this, we would try to dispute the null hypothesis H_0. This hypothesis would say that there is no difference between two retrieval methods A and B. If we can show based on data that H_0 does not hold, then we can say that either A consistently outperforms B or B consistently outperforms A [22].

6.4 Fine-tuning

As explained in section 2.5, we explain that the models are sometimes fine-tuned using mined hard negatives. This would mean that the training data can be generated by a different model. So if we could find a powerful reranker that scores well on the WT10g dataset, we could use this to generate the training data to fine-tune a learned sparse retriever. This technique of transfer learning [46] could be more effective than our current zero-shot approach.

Bibliography

[1] Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Dipendra Misra. Investigating the role of negatives in contrastive representation learning, 2021.

[2] Peter Bailey, Nick Craswell, and David Hawking. Engineering a multi-purpose test collection for web retrieval experiments. *Information Processing & Management*, 39(6):853–871, 2003.

[3] Peter Bailey, Nick Craswell, and David Hawking. Engineering a multi-purpose test collection for web retrieval experiments. *Information Processing & Management*, 39(6):853–871, 2003.

[4] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA, 2021. Association for Computing Machinery.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[6] Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. Ranking measures and loss functions in learning to rank. *Advances in Neural Information Processing Systems*, 22, 2009.

[7] Xilun Chen, Kushal Lakhotia, Barlas Oğuz, Anchit Gupta, Patrick Lewis, Stan Peshterliev, Yashar Mehdad, Sonal Gupta, and Wen tau Yih. Salient phrase aware dense retrieval: Can a dense retriever imitate a sparse one?, 2022.

[8] Gordon V. Cormack and Thomas R. Lynam. Statistical precision of information retrieval evaluation. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, page 533–540, New York, NY, USA, 2006. Association for Computing Machinery.

[9] Benoit Courty, Victor Schmidt, Sasha Luccioni, Goyal-Kamal, MarionCoutarel, Boris Feld, Jérémy Lecourt, LiamConnell, Amine Saboni, Inimaz, supatomic, Mathilde Léval, Luis Blanche, Alexis Cruveiller, ouminasara, Franklin Zhao, Aditya Joshi, Alexis Bogroff, Hugues de Lavoreille, Niko Laskaris, Edoardo Abati, Douglas Blank, Ziyao Wang, Armin Catovic, Marc Alencon, Christian Bauer, Lucas-Otavio, JPW, and MinervaBooks. mlco2/codecarbon: v2.4.1, May 2024.

[10] D. Cutting and J. Pedersen. Optimization for dynamic inverted index maintenance. In *Proceedings of the 13th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '90, page 405–411, New York, NY, USA, 1989. Association for Computing Machinery.

[11] Zhuyun Dai and Jamie Callan. Context-aware term weighting for first stage passage retrieval. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, SIGIR '20, pages 1533–1536, New York, NY, USA, 2020. Association for Computing Machinery.

[12] Hamed Damirchi, Cristian Rodríguez-Opazo, Ehsan Abbasnejad, Damien Teney, Javen Qinfeng Shi, Stephen Gould, and Anton van den Hengel. Zero-shot retrieval: Augmenting pre-trained models with search engines, 2023.

[13] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems*, 38:100857, 2023.

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

[15] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. Splade v2: Sparse lexical and expansion model for information retrieval, 2021.

[16] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. From distillation to hard negative sampling: Making sparse neural ir models more effective, 2022.

[17] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. Splade: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page

2288–2292, New York, NY, USA, 2021. Association for Computing Machinery.

[18] Emma J. Gerritse, Faegheh Hasibi, and Arjen P. de Vries. Entity-aware transformers for entity search. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 1455–1465, New York, NY, USA, 2022. Association for Computing Machinery.

[19] Parantapa Goswami, Massih-Reza Amini, and Eric Gaussier. Language-independent query representation for ir model parameter estimation on unlabeled collections. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, pages 121–130, 09 2015.

[20] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning, 2022.

[21] Gijs Hendriksen, Michael Dinzinger, Sheikh Mastura Farzana, Noor Afshan Fathima, Sebastian Schmidt Maik Fröbe, Saber Zerhoudi, Michael Granitzer, Matthias Hagen, Djoerd Hiemstra, Martin Potthast, and Benno Stein. Impact and development of an open web index for open web search. *Journal of the Association for Information Science and Technology*, 1-9(n/a), 2023.

[22] Djoerd Hiemstra. *Using Language Models for Information Retrieval.* Phd thesis - research ut, graduation ut, University of Twente, Jan 2001.

[23] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. Improving efficient neural ranking models with cross-architecture knowledge distillation, 2021.

[24] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, oct 2002.

[25] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering, 2020.

[26] Robert Krovetz and W. Bruce Croft. Lexical ambiguity and information retrieval. *ACM Trans. Inf. Syst.*, 10(2):115–141, apr 1992.

[27] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning, 2019.

[28] Carlos Lassance, Hervé Dejean, Stéphane Clinchant, and Nicola Tonellotto. Two-step splade: Simple, efficient and effective approximation

of splade. In Nazli Goharian, Nicola Tonellotto, Yulan He, Aldo Lipani, Graham McDonald, Craig Macdonald, and Iadh Ounis, editors, *Advances in Information Retrieval*, pages 349–363, Cham, 2024. Springer Nature Switzerland.

[29] Carlos Lassance, Hervé Déjean, Thibault Formal, and Stéphane Clinchant. Splade-v3: New baselines for splade, 2024.

[30] Phuc H. Le-Khac, Graham Healy, and Alan F. Smeaton. Contrastive representation learning: A framework and review. *IEEE Access*, 8:193907–193934, 2020.

[31] Jimmy Lin, Matt Crane, Andrew Trotman, Jamie Callan, Ishan Chattopadhyaya, John Foley, Grant Ingersoll, Craig Macdonald, and Sebastiano Vigna. Toward reproducible baselines: The open-source ir reproducibility challenge. In Nicola Ferro, Fabio Crestani, Marie-Francine Moens, Josiane Mothe, Fabrizio Silvestri, Giorgio Maria Di Nunzio, Claudia Hauff, and Gianmaria Silvello, editors, *Advances in Information Retrieval*, pages 408–420, Cham, 2016. Springer International Publishing.

[32] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 2356–2362, New York, NY, USA, 2021. Association for Computing Machinery.

[33] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[34] Kadan Lottick, Silvia Susai, Sorelle A. Friedler, and Jonathan P. Wilson. Energy usage reports: Environmental awareness as part of algorithmic accountability, 2019.

[35] Xueguang Ma, Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. Document expansion baselines and learned sparse lexical representations for ms marco v1 and v2. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 3187–3197, New York, NY, USA, 2022. Association for Computing Machinery.

[36] Y. S. Maarek and F. Z. Smadja. Full text indexing based on lexical relations an application: software libraries. In *Proceedings of the 12th*

Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '89, page 198–206, New York, NY, USA, 1989. Association for Computing Machinery.

[37] Sean MacAvaney, Craig Macdonald, and Iadh Ounis. Streamlining evaluation with ir-measures. In Matthias Hagen, Suzan Verberne, Craig Macdonald, Christin Seifert, Krisztian Balog, Kjetil Nørvåg, and Vinay Setty, editors, *Advances in Information Retrieval*, pages 305–310, Cham, 2022. Springer International Publishing.

[38] Sean MacAvaney, Andrew Yates, Sergey Feldman, Doug Downey, Arman Cohan, and Nazli Goharian. Simplified data wrangling with ir_datasets. In *SIGIR*, 2021.

[39] Antonio Mallia, Joel Mackenzie, Torsten Suel, and Nicola Tonellotto. Faster learned sparse retrieval with guided traversal. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 1901–1905, New York, NY, USA, 2022. Association for Computing Machinery.

[40] Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, page 472–479, New York, NY, USA, 2005. Association for Computing Machinery.

[41] Thong Nguyen, Sean MacAvaney, and Andrew Yates. Adapting learned sparse retrieval for long documents. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 1781–1785, New York, NY, USA, 2023. Association for Computing Machinery.

[42] Thong Nguyen, Sean MacAvaney, and Andrew Yates. A unified framework for learned sparse retrieval, 2023.

[43] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human-generated MAchine reading COmprehension dataset, 2017.

[44] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert, 2020.

[45] Rodrigo Nogueira, Jimmy Lin, and AI Epistemic. From doc2query to doctttttquery. *Online preprint*, 6(2), 2019.

[46] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[47] Biswajit Paria, Chih-Kuan Yeh, Ian E. H. Yen, Ning Xu, Pradeep Ravikumar, and Barnabás Póczos. Minimizing flops to learn efficient sparse representations, 2020.

[48] Nina Poerner, Ulli Waltinger, and Hinrich Schütze. E-BERT: Efficient-yet-effective entity embeddings for BERT. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 803–818, Online, November 2020. Association for Computational Linguistics.

[49] Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models, 2021.

[50] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[51] Dennis W Ruck, Steven K Rogers, and Matthew Kabrisky. Feature selection using a multilayer perceptron. *Journal of neural network computing*, 2(2):40–48, 1990.

[52] Harrisen Scells, Shengyao Zhuang, and Guido Zuccon. Reduce, reuse, recycle: Green information retrieval research. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 2825–2837, New York, NY, USA, 2022. Association for Computing Machinery.

[53] Ian Soboroff. Does wt10g look like the web? In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, page 423–424, New York, NY, USA, 2002. Association for Computing Machinery.

[54] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp, 2019.

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

[56] Nandan Thakur, Kexin Wang, Iryna Gurevych, and Jimmy Lin. Sprint: A unified toolkit for evaluating and demystifying zero-shot neural sparse retrieval, 2023.

[57] Aimee Van Wynsberghe. Sustainable ai: Ai for sustainability and the sustainability of ai. *AI and Ethics*, 1(3):213–218, 2021.

[58] Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Enabling the use of lucene for information retrieval research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1253–1256, New York, NY, USA, 2017. Association for Computing Machinery.

[59] Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Reproducible ranking baselines using lucene. *J. Data and Information Quality*, 10(4), oct 2018.

[60] Hamed Zamani, Mostafa Dehghani, W. Bruce Croft, Erik Learned-Miller, and Jaap Kamps. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 497–506, New York, NY, USA, 2018. Association for Computing Machinery.

[61] Tiancheng Zhao, Xiaopeng Lu, and Kyusong Lee. Sparta: Efficient open-domain question answering via sparse transformer matching retrieval, 2020.

Appendix A

Appendix

A.1 Sample from experiment 3.1.2


```
{
  "id": "9",
  "text": "One of the main reasons Hanford was selected as a
      site for the Manhattan Project's B Reactor was its
      proximity to the Columbia River, the largest river flowing
      into the Pacific Ocean from the North American coast.",
  "vector": {
    "han": 2.686464786529541,
    "##ford": 2.602689743041992,
    "manhattan": 2.571538209915161,
    "largest": 2.4362142086029053,
    "reactor": 2.414797067642212,
    "b": 2.1998252868652344,
    "columbia": 2.174396514892578,
    "project": 2.1703920364379883,
    "biggest": 2.1618852615356445,
    "river": 2.0818800926208496,
    "where": 1.996505618095398,
    "reactors": 1.9707117080688477,
    "nyc": 1.9564570188522339,
    "nuclear": 1.948696494102478,
    "rivers": 1.8943818807601929,
    "ocean": 1.8765983581542969,
    "located": 1.8681899309158325,
    "why": 1.8174678087234497,
    "reasons": 1.8168331384658813,
    "location": 1.729302167892456,
    "pacific": 1.7047481536865234,
    "flows": 1.6737322807312012,
  }
}
```


Example word scores output from the Sparta model. Only the top 22 results are kept (out of a total of 400).

A.2 CodeCarbon output example

```
EmissionsData(
    timestamp='2024-04-23T11:54:05',
    project_name='codecarbon',
    run_id='ea889ba4-e800-49f2-bb47-50f6e27379c6',
    duration=66313.9405298233,
    emissions=5.4121288612118015,
    emissions_rate=8.161374241933051e-05,
    cpu_power=42.5,
    gpu_power=0.0,
    ram_power=47.07496976852417,
    cpu_energy=0.7830239961678791,
    gpu_energy=13.625636186333587,
    ram_energy=0.8665060176565875,
    energy_consumed=15.275166200158056,
    country_name='The Netherlands',
    country_iso_code='NLD',
    region='gelderland',
    cloud_provider='',
    cloud_region='',
    os='Linux-5.19.0-46-generic-x86_64-with-glibc2.35',
    python_version='3.10.6',
    codecarbon_version='2.3.5',
    cpu_count=32,
    cpu_model='Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz',
    gpu_count=5,
    gpu_model='5 x NVIDIA GeForce RTX 2080 Ti',
    longitude=5.8593,
    latitude=51.8421,
    ram_total_size=125.53325271606445,
    tracking_mode='machine',
    on_cloud='N',
    pue=1.0
)
```

Output of CodeCarbon for fine-tuning Sparta. See the CodeCarbon documentation for an explanation of the columns[1].

[1] https://mlco2.github.io/codecarbon/output.html