EXPERIMENTS ABOUT DIFFERENTIAL SEARCH INDEXING USING TRANSFORMER MODELS

Ludovico Comito, Giulio Fedeli, Lorenzo Cirone
Sapienza University of Rome
{comito.1837155, fedeli.1873677, cirone.1930811}@studenti.uniroma1.it

ABSTRACT

In the modern landscape of the approaches to Information Retrieval, one of the most interesting solutions was published in [1], where a single encoder-decoder architecture was trained in a multitask fashion to perform both indexing and retrieval of documents at the same time. The core idea is to map string queries directly to relevant docids, simplifying the retrieval process. With this project, we explore solutions based on the same DSI concept, but exploring new architectures and new training approaches to further deepen our understanding of the original paper and trying to clarify and uncover its strengths and weaknesses.

1 Introduction

The original DSI model [1] is based on an encoder-decoder architecture, specifically a pre-trained T5 model [2]. In the DSI approach, the model is trained to perform both indexing and retrieval at the same time in a multi-task fashion. We chose as our baseline the Okapi BM25 scores and as the base model to work with the T5, as specified in the aforementioned paper. From there we proceeded to experiment with the model's architecture. More specifically we implemented the use of a model belonging to a recently presented class of models (Lamini [3]) pretrained using knowledge distillation. Finally, we repeated the experiment again but with a custom encoder-decoder architecture. Furthermore we tried to tinker with various pre-processing strategies, first leveraging the multi-task approach as suggested by the referenced paper and, after that, following our own intuition applying data augmentation through query generation taking inspiration from [4].

2 Dataset

In [1] the dataset used is NQ in its NQ10K, NQ100K and NQ320K variants. Our work on the contrary will be focused on the MS Marco Document ranking dataset. All the original files to download can be found inside **Microsoft's github repo** under the "document ranking dataset" paragraph. The msmarco-docs.tsv file **(download here)** is a 22GB file containing all the dataset's documents and comprising the columns [docid, url, title, body].

Each split (train, validation and test) is essentially made of two files: one tsv containing queries and their corresponding query id (qid) and one top 100 file that maps each query id to the top 100 ranked docids.

As the original dataset contains millions of documents, we decided to extract 8k train queries, 2k validation queries and 2k test queries. In orther to further reduce the number of documents to load, we reduce the rankings to the first 10 relevant documents instead of the first 100. The final datasets will look like this:

- A Training Dataset that contains all the documents to be indexed plus the training queries (to train the model in a multitask fashion), consisting in almost 80k files (i.e. the 10 files to be ranked for each 8k query).
- Validation and Test documents are composed of almost 20k files each (i.e. 10 files for 2k validation and 2k test queries).

On top of this we decided to extend the training dataset by concatenating also the 8k queries at the end of the training set in order to tap into the aforementioned multitask strategy.

Since the training corpus is very large (22gb), we will use the **dask** library that allows to load the corpus as a dataframe and split it in chunks that can fit into memory. Finally it is worth mentioning that in section 4 we will show and justify any further preprocessing used in order to better tune the performance of our experiments.

3 Model Architecture

Baseline Approach In order to evaluate the performances of our model against a non-trivial baseline, we decided to utilize Okapi BM-25. BM-25 is a traditional method utilized for information retrieval. This method takes into account term frequency, document length and inverse document frequency to determine the relevance of a document to a query.

Model Choice The chosen model to perform our experiments is the T5 encoder-decoder architecture. The first experiment is run on the base T5 model in order to have a viable comparison between the methodology used in the original paper and ours. The first iteration of our methodology is to implement the Lamini-Flan-T5 model. Its main features are the use of both the Flan T5 variant and the Lamini method. Flan T5 utilizes the same encoder-decoder architecture as T5, but has been trained on a larger instruction finetuning dataset.
On the other hand the Lamini method [3] is based on knowledge distillation, which means training a smaller model (called student) leveraging the knowledge of a bigger model (the teacher). In this case the teacher is gpt-turbo-3.5 and the student is Flan T5. The teacher is used to produce a synthetic output that the student will learn on. The versions of Flan T5 and Lamini used in our work are specified in [3].
The second iteration of our methodology is to perform the training on a custom encoder-decoder architecture. The choice for the encoder fell on bert-based-uncased model, this is because it is one of the most commonly recognised starting points of any LLM centered research worth its salt. The choice for the decoder fell instead on gpt-2 since it is a decoder only model that has shown incredibly promising results.
In the following section 4 we will go in depth on how we tested the various architectures and how we devised the strategy for the experiments that we conducted, including the additional preprocessing and the test conducted.

4 Metodology

4.1 Experiments' Layout and Process

The starting point of our investigation is the BM25 baseline which we proceeded to implement in our code. Since this is not a ML model there is no training to be conducted, its significance is given by the fact that it represents the foundational ranking scheme of most enterprise search applications.
The first model we performed training on was the base T5.
We then proceeded to experiment with Lamini-Flan T5 with the following variations:

- Batch size 32, learning rate 0.0005, token length 32
- Batch size 64, learning rate 0.0005, token length 32
- Batch size 32, learning rate 0.0001, token length 32

All of these were performed on the dataset with a basic preprocessing procedure. After this we decided to perform a specific technique of data augmentation through the use of query generation with a pretrained generative model. The so generated queries were then appended at the beginning of the corresponding document. This is better explained in the following section about data preprocessing.
The basic parameters of the run were almost identical to the base run of the Lamini-Flan-T5 model:

- Batch size 32, learning rate 0.0005, token length 64

Lastly we implemented our custom encoder-decoder architecture with bert-base-uncased as encoder and gpt-2 as decoder. The main hyperparameters remained the same as the base Lamini-Flan-T5 for comparability.
Our workflow was mainly driven by the desire to explore how big of an impact can have simple yet precisely aimed improvements on the choice of a model and the goodness of the dataset when it comes to work with LLMs.

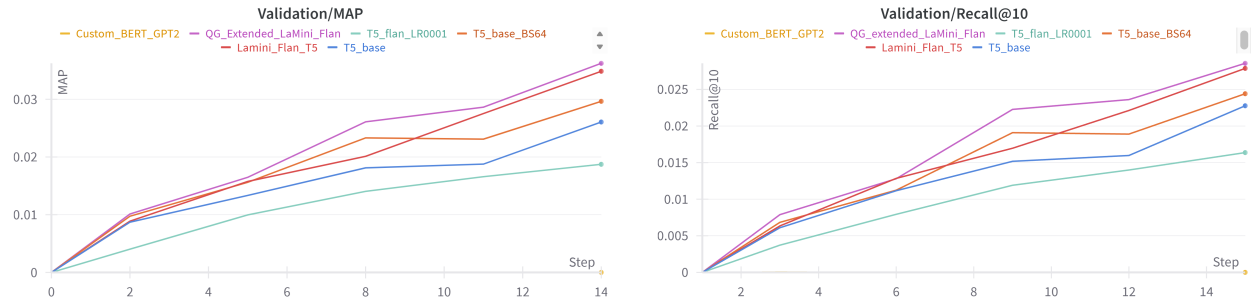
4.2 Preprocessing

As briefly mentioned in 2 we sample from the original MS Marco dataset 8k queries from the train split, 2k queries from the validation split and 2k queries from the test split for our train, validation and test datasets. After sampling, we make sure to clean the datasets by removing null values or duplicate rows and cut the documents to the first 50 words

for memory efficiency purposes (also notice that at most 64 tokens for each document are considered at training time). Moreover, we merge the title and body of each document, as we believe that the title carries significant information (this technique was also tested in [1]). The final step of our preprocessing pipeline is to re-assign the document ids of each document with newly generated semantic ids.
The clustering method re-implements the concept of Semantically Structured Identifiers. As a first step, we created the embeddings for our documents utilizing a distilled version of the Roberta model present in **HuggingFace's Sentence Transformers library** . This model maps sentences to a 256 dimensional embedding space that represents the semantic of the given text.
We chose this model over BERT as it produces lower dimensional representations that are more manageable memory-wise, especially within the Google Colab free tier.
Once the embeddings for all the documents are obtained, we perform hierarchical clustering utilizing clusters obtained by k-means recursively until we obtain clusters containing at most 100 documents. Documents within the same cluster will be assigned with the same prefix. Following this practice allows us to obtain identifiers that actually represent the semantic of the document, and place similar documents close within their ID representation.
Finally, in order to create the dataset for our query generation approach, we utilize a **doc2query generation model** to create queries for each document within the training dataset. Once all the queries are produced, we add each query at the beginning of each corresponding document and finally concatenate the train queries.
This creates a new document format where the query acts like a concise summary or starting point for the document's content. By prepending the queries, we essentially provide the training model with additional context about the documents. This enriches the training data and helps the model in its retrieval task.

4.3 Training

All of our models were trained for 5 epochs utilizing the Pytorch Lightning framework within the limits of the Google Colab free tier. At training time, the goal of our models is to perform both indexing and retrieval utilizing documents' semantic ids as their labels, and the semantic id of the corresponding document (with the highest ranking) for the queries. For all the experiments we utilized AdamW as an optimizer, using its default values for epsilon and betas.
For all of our models, we utilize CrossEntropy as a loss function, where the tokens of the model's predictions are compared with the tokens of the corresponding label. The goal of our training is to make the model learn the relationship between queries, documents and semantic ids within its parameters.
During both validation and test steps, the model generates ten different outputs using beam search as a generation strategy. Since semantic ids are always numeric values, we restrict the vocabulary of our model to tokens that correspond to integer values. Finally, the decoded predictions are evaluated against the labels to compute the scores for Mean Average Precision (4) and Recall@10 (5). We perform checkpointing of the model based on the Mean Average Precision score, performed during the validation epoch.
While in the training step we produce a single label for a query, in the validation and test steps labels will correspond with the ranked list of ten docids for each query.
During training, we conducted hyperparameters tuning on batch size and learning rate. Specifically, we experimented with a batch size of 32 and 64 (the maximum we could get within Colab limits) and learning rates of 0.0001 and 0.0005, which are typical values used for T5 finetuning.
For all the steps, we feed to the encoder the first 32 tokens of the input, except for the query generation case, where we increment the number of tokens to 64 in order to fit both the generated queries and the beginning of the document.



(a) Mean Average Precision (MAP)

(b) Recall@10

Figure 1: Validation Performance Comparison in terms of MAP and Recall@10

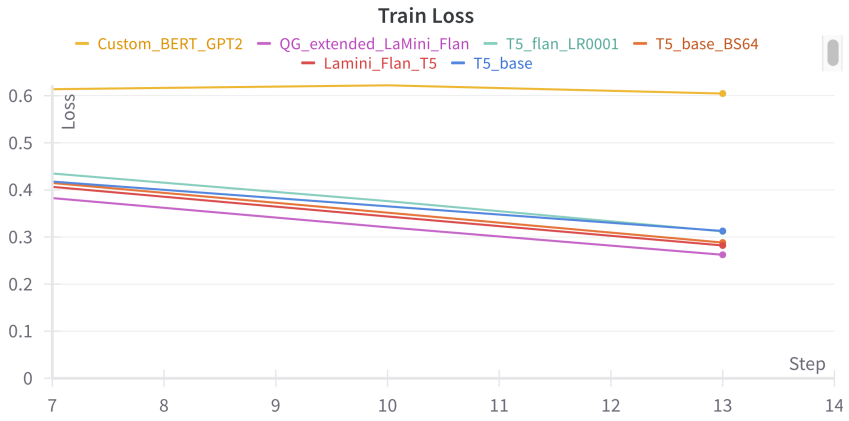


Figure 2: Train Loss

4.4 Testing and Metrics

For each experiment the parameters of the best achieving epoch according to validation MAP and validation Recall@10 were saved in a .ckpt file and at the end of training we executed the tests based on them.
Mean Average Precision (MAP) evaluates the ability of a model to rank relevant items and it's used to evaluate the effectiveness of search systems, like search engines or database queries. It involves calculating the Average Precision (AP), which is the average of the precision values calculated at the points in the ranking where each relevant document is retrieved. Precision itself is a measure of relevancy and is calculated as the number of relevant documents retrieved divided by the total number of documents retrieved. A higher AP signifies a better model at ranking relevant items high in the results. MAP is the average of APs calculated across different categories or thresholds in the task.
The formulas for Precision@K, AP@K and MAP are:

$$Precision@K = \frac{|\ RELEVANT\ RETRIEVED\ DOCUMENTS\ |}{|\ TOTAL\ RETRIEVED\ DOCUMENTS\ |} \tag{1}$$

$$AP@K = \frac{1}{r}\sum_{k=1}^{K} Precison@K \cdot rel(k) \tag{2}$$

where

$$rel(k) = \begin{cases} 1 & \text{if item at } k^{\text{th}} \text{ rank is relevant} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$MAP = \frac{1}{n}\sum_{k=1}^{k=n} AP@k \tag{4}$$

Recall@10 measures the proportion of relevant items the model retrieves within the top 10 results.
Its formula is:

$$Recall@K = \frac{|\ RELEVANT\ DOCUMENTS\ AMONG\ TOP\ K\ |}{|\ \ DOCUMENTS\ IN\ TOP\ K\ |} \tag{5}$$

These metrics are of course tracked also in testing and are the main comparison measures for our experiments.

Model	Batch Size	Learning Rate	Mean Average Precision	Recall@10
BM-25 baseline			0.003%	0.002%
T5 Base	32	0.0005	2.53%	1.92%
Lamini Flan T5	32	0.0005	3.31%	2.62%
Lamini Flan T5	32	0.0001	1.81%	1.44%
Lamini Flan T5	64	0.0005	**3.34%**	**2.62%**
Lamini Flan T5 QG	32	0.0005	3.31%	2.61%
BERT-GPT2 custom E-D	32	0.0005	0.0019%	0.0016%

Table 1: Performance Comparison

5 Results and Conclusions

5.1 Results

As expected, the BM-25 baseline scored very poor results, reaching only 0.003% MAP and 0.002% for Recall@10. Our hypothesis is that the model is just able to capture statistical correlations between words, but not the full semantics of the documents and their relationship with their semantic ids.
The T5-base model scored 2.53% in MAP and 1.92% in Recall@10, and was outperformed by the Lamini Flan T5 in both metrics while utilizing the same hyperparameters (batch size 32 and learning rate 0.0005). As the Lamini Flan T5 model demonstrated better performances, we decided to conduct hyperparameter tuning on it, by experimenting with different batch sizes and learning rates. The best result was achieved with batch size 64 and learning rate 0.0005, achieving 3.34% MAP and 2.62% Recall@10.
Our hypothesis is that thanks to its extensive pre-training and the Lamini knowledge distillation techniques, the Lamini Flan T5 model has higher capabilities at dealing with multi-task learning tasks. Although performing the overall highest validation MAP and Recall@10 during training, the model trained on the Query Generation augmented dataset scored the same performances as the Lamini T5 model.
Finally, the worst performing model was the BERT-GPT2 custom encoder-decoder, which only scored 0.0019% MAP and 0.0016% Recall@10. In our opinion, this is due to the incompatibility between the representation of both encoder and decoder and could improve by incrementing the number of training epochs. In **Table1** you can find all the results of the experiment that we ran.

5.2 Conclusions

Based on what we've learned from our trials we can conclude that the most effective way to improve the task at hand, given the current training capabilities at our disposal, is to work with jointly pretrained encoder-decoder architectures and pretraining techniques. The most interesting development of our work would surely begin with heavier training strategies like working with more epochs and more powerful computational units. In light of this, exploiting more powerful machines would allow to experiment with larger models in terms of parameters, as one of the core concepts of the DSI approach is to leverage the transformer memory to index and retrieve documents. Finally, one crucial improvement could be the use of a better embedding model and clustering algorithm to produce better semantic ids. Since beam search generation tends to produce outputs that are very similar one to another, having closer semantic representations will very likely lead to a significant improvement in performances.

References

[1] Yi Tay , Vinh Q. Tran , Mostafa Dehghani , Jianmo Ni , Dara Bahri , Harsh Mehta , Zhen Qin , Kai Hui , Zhe Zhao , Jai Gupta , Tal Schuster , William W. Cohen , Donald Metzler. **"Transformer Memory as a Differentiable Search Index"**, 2022.

[2] Colin Raffel , Noam Shazeer , Adam Roberts , Katherine Lee , Sharan Narang , Michael Matena , Yanqi Zhou , Wei Li , Peter J. Liu. **Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer**, 2023.

[3] Minghao Wu , Abdul Waheed , Chiyu Zhang , Muhammad Abdul-Mageed and , Alham Fikri Aji. **Lamini-LM: A Diverse Herd of Distilled Models from Large-Scale Instructions**, 2024.

[4] Yubao Tang , Ruqing Zhang , Jiafeng Guo , Jiangui Chen , Zuowei Zhu , Shuaiqiang Wang , Dawei Yin , Xueqi Cheng. **"Semantic-Enhanced Differentiable Search Index Inspired by Learning Strategies"**, 2023.