DIFFERENTIABLE SEARCH INDEXING *

Alessio Borgi, Eugenio Bugli, Damiano Imola
1952442, 1934824, 2109063
Sapienza Università di Roma
Rome
{borgi.1952442, bugli.1934824, imola.2109063}@studenti.uniroma1.it

ABSTRACT

In this project report we present a novel Differentiable Search Indexing (DSI) approach. Unlike traditional contrastive learning-based dual encoders, this architecture maps a query to its relevant document identifier (docid) during inference, enabling efficient retrieval with standard model inference and optional beam search for ranking. We explore advanced indexing methods, including semantic and syntactic data augmentation techniques like stopword removal, Num2Text transformation, and POS-MLM augmentation, to enhance dataset quality. Our model innovations feature dynamic pruning for efficient training, parameter-efficient fine-tuning via LoRA, and memory optimization through QLoRA with 4-bit quantization.

***K*eywords** DSI · POS-MLM · QLoRA · Dynamic Pruning · LoRA

1 Introduction

Trainable Information Retrieval (IR) systems are characterized by two phases:

- **Indexing:** Indexing of a corpus, which means to associate the content of each document with its corresponding docid.
- **Retrieval:** Learn how to retrieve efficiently from the index, which means to

Instead of using Contrastive Learning based Dual Encoders, the paper proposes an architecture which directly map a query **q** to a relevant docid **j**. This architecture, called DSI, it's implemented with a pre-trained Transformer and all the information of the corpus are encoded within the parameters of the language model. When we are doing inference, the give to the trained model a text query as input and we expect to obtain a docid as output. If we are interested in a ranked list of relevant documents we can also use Beam Search. Our DSI system uses standard model inference to map from encodings to docids, instead of learning internal representations that optimize a search procedure. DSI can be extended in different ways:

- **Document representation:** there are several ways to represent documents (e.g. full text, bag-of-words representations, ...)
- **Docid representation:** (e.g. unique tokens, structured semantic docids, text strings, ...)

1.1 Indexing Methods

Given a sequence of document tokens, the model is trained to predict the docids. There can be used different strategies:

- **Inputs2Target**:
- **Targets2Inputs**:
- **Bidirectional**:

* *Citation*: **Authors. Title. Pages.... DOI:000000/11111.**

2 Dataset

In our work, we have used the first version of the MS Marco Dataset (reference bib), which is composed by 100k real Bing questions and human generated answers. The dataset is already partitioned into Training (82326 samples), Validation (10047 samples) and Test (9650 samples). Each partition is organized as a dictionary where the most important keys are the following:

- **answers**: the answer related to the query based on the text informations.
- **passages**: contains another dictionary where we can find the complete corpus of the text and each passage that composes it.
- **query**: it is the question asked.

Since we are interested in the ranking of the documents, we have used the SimpleSearcher from pyserini [1], in order to obtain the most relevant 1000 documents. The pre-processing applied to the dataset is explained in the following subsections.

2.1 Tokenization

Starting from the dataset, we have computed the maximum length of the inputs of the encoder (1797) and decoder (4), which will be useful during the tokenization process. We have used the pretrained tokenizer from the small version of the T5 model (reference). Our tokenized dataset is composed only by the following parts:

- **Query**: tokenized version of the original query
- **Query and Corpus**: tokenized input text, which is composed by the concatenation of the query and the corpus
- **Document IDs**: tokenized version of the identificator of each document
- **Ranked Document IDs**: tokenized version of the ranked list of the first 1000 document ids

2.2 DSI Multi-Generation

In this subsection we have described the procedure to generate semantically structured document ids, which are characterized by the associations between queries and standard document ids. In other words, the docid should be able to capture some informations related to the semantics of the associated document. To do this, we have followed the algorithm provide by (reference ...)

Algorithm 1 Generating Semantically Structured Identifiers

Require: Document embeddings $X_{1:N}$, where $X_i \in \mathbb{R}^d$ generated by a small 8-layer BERT model with $c = 100$
Ensure: Corresponding docid strings $J_{1:N}$

function GENERATE_SEMANTIC_IDS($X_{1:N}$)
  $C_{1:10} \leftarrow$ CLUSTER($X_{1:N}$, $k = 10$) # k-means clustering
  $J \leftarrow$ empty list
  for $i \leftarrow 0$ to 9 **do**
    $J_{\text{current}} \leftarrow [i] \times |C_{i+1}|$
    if $|C_{i+1}| > c$ **then** # recursion if there are more than c documents
      $J_{\text{rest}} \leftarrow$ GENERATE_SEMANTIC_IDS(C_{i+1})
    else
      $J_{\text{rest}} \leftarrow [0, \ldots, |C_{i+1}| - 1]$ # assign arbitrary number from 0 to $c - 1$
    $J_{\text{cluster}} \leftarrow$ ELEMENTWISE_STR_CONCAT(J_{current}, J_{rest})
    $J \leftarrow J$.APPEND_ELEMENTS(J_{cluster}) # Append all elements of J_{cluster} to J
  $J \leftarrow$ REORDER_TO_ORIGINAL(J, $X_{1:N}$, $C_{1:10}$)
  return J

After this procedure we have followed (reference understanding ...) to create the pseudo-queries, which are new queries generated by a model that takes in input the corpus of the documents.

$$
\begin{cases}
\mathcal{U} = \mathcal{O} \bigcup \mathcal{P} \\
\mathcal{O} = \bigcup_i \mathcal{O}_i = \bigcup_i \{d_i^1, d_i^2, \cdots, d_i^m\} \\
\mathcal{P} = \bigcup_i \mathcal{P}_i = \bigcup_i \{pq_i^1, pq_i^2, \cdots, pq_i^m\}
\end{cases}
\tag{1}
$$

2.3 Data Augmentation

In this section, we present a first part of the introduced novel contributions that collectively enhance the dataset—through both semantic, stopwords and num2text augmentation as well as a more advanced POS-MLM augmentation. We decide to apply these techniques to the whole dataset corpus, by having Semantic, Stopwords and Num2Text Augmentation to be applied to all the corpuses, while having POS-MLM Augmentation to be applied to only 10% of the corpuses' phrases.

2.3.1 Semantic Augmentation: Stopwords and Num2Text

Traditional DSI models commonly rely on a raw (or minimally processed) corpus, which often incorporates stopwords and punctuation as well as numbers. These low-value tokens can consume model capacity without contributing actually to semantic. In addition, purely numerical tokens are often not well captured in semantic embedding spaces. Furthermore, frequent words and repeated terms may overwhelm the more meaningful tokens, leading to reduced retrieval effectiveness and, at the same time, make the model process useless words, thereby extending training.

More in detail, the strategy we follow is:

- **Stopword Removal:** By integrating established stopword and punctuation lists (using `nltk`) and frequency-based heuristics, we remove non-informative words (e.g., "the," "and," "or"). Each token is compared against a set of known stopwords and punctuation. If the token is in this set, it is removed.
- **Num2Text Augmentation:** Numeric values (*e.g.*, "42") are treated as discrete tokens and may lose contextual meaning if not converted into text. In practice, we transform (using the `inflect` library) numeric tokens into their textual representations (*e.g.*, "42" → "forty two") so that the model can better capture semantic relationships involving numbers.

2.3.2 POS-MLM Augmentation

DSI models under examination often struggle to bridge the gap between document structure and actual query intent. Techniques like Part-of-Speech (POS) tagging have proven effective for syntactic parsing [2], while Masked Language Modeling (MLM) has become a cornerstone of learning contextualized representations [3]. Typically, these approaches are applied independently. Here we combine them in a novel manner.

POS-MLM Augmentation synthesizes syntactic insights from POS tagging with the context-driven capabilities of MLM, consisting in:

- **POS Tagging:** Using spaCy (a lexical and syntactic parser), we assign part-of-speech labels (NOUN, VERB, ADJ, etc.) to each token. POS tags are used to identify tokens that have key syntactic roles, enabling a structured roadmap for masking.
- **Selective Token Masking:** Tokens holding key syntactic roles (here, verbs) are replaced with a placeholder token (e.g., `[MASK]`), so that the LM must focus on reconstructing them.
- **Predictive Training (MLM):** The masked sequences are processed by a pretrained Transformer model (e.g., `bert-base-uncased`), which infers the masked tokens. By leveraging the broader context, MLM reconstructs syntactic structures and promotes a deeper understanding of semantic relationships.

This dual focus enables more precise ranking of documents by their structural and semantic relevance, thus improving *relevance ordering*, i.e., the arrangement of documents based on their contextual and structural pertinence.

3 Model

3.1 Model Architecture Novelties

We adopt T5 as our core architecture, but additionally optimize training and memory usage by applying three key techniques: **Dynamic Pruning**, **LoRA**, and **QLoRA**.

3.1.1 Dynamic Pruning

Large-scale DSI frameworks, especially when operating on extensive datasets like MS-MARCO, can consume considerable memory and runtime. While fixed or static pruning methods reduce computational overhead, they also risk discarding parameters essential to retrieval accuracy.

In response, we propose **Dynamic Pruning**, a type of **unstructured pruning** which adaptively prunes weights based on real-time feature-importance scores. Concretely:

- **Feature Scoring:** At each training step, the model computes importance indicators (e.g., from attention distributions) that highlight which features most strongly impact retrieval.
- **Dynamic Cutoffs:** A context-dependent threshold then prunes low-importance features while preserving the most influential parameters. Specifically, weights with the smallest magnitudes (by L1 norm) are set to zero.
- **Iterative Refinement:** Throughout training, the thresholds and importance metrics are continuously updated, balancing the removal of redundant weights with the retention of crucial ones.

By removing less critical parameters, unstructured pruning substantially lowers memory and computational demands without degrading retrieval performance. In our experiments, Dynamic Pruning—used alongside Mixed Precision and Gradient Accumulation—shortened training time from over five hours to just 45 minutes for each epoch (considering the whole dataset).

3.1.2 LoRA: Parameter-Efficient Fine-Tuning

Fine-tuning large T5 models from scratch can be expensive in both time and GPU memory. To alleviate this, we decided to employ **LoRA** (Low-Rank Adaptation). In a nutshell, we have that LoRA *freezes* the original T5 weights and injects small low-rank adapter modules into the attention layers. During training, only these adapter weights are updated—dramatically cutting the total trainable parameter count and thus reducing memory needs and speeding up convergence.

3.1.3 QLoRA: 4-bit Quantization for T5

Although LoRA alone is parameter-efficient, large T5 models can still require substantial GPU memory. We decide therefore to also provide for an alternative model version (in addition to the basic one and to the LORA version), to further minimize memory consumption, we apply **QLoRA**, which additionally quantizes the T5 weights to a *4-bit* format using *bitsandbytes* library. In this setup, the base T5 is compressed into 4-bit precision, while LoRA's low-rank adapters remain at higher precision for effective fine-tuning. This model, further reduce memory requirements and training overhead without sacrificing retrieval accuracy.

4 Training and Results

gradient accumulation for each 4 batches mixed precision 16 bits

The documentation for `natbib` may be found at

`http://mirrors.ctan.org/macros/latex/contrib/natbib/natnotes.pdf`

Of note is the command `\citet`, which produces citations appropriate for use in inline text. For example,

```
\citet{hasselmo} investigated\dots
```

produces

> Hasselmo, et al. (1995) investigated…

`https://www.ctan.org/pkg/booktabs`

4.1 Figures

See Figure 1. Here is how you add footnotes. [2]

[2]Sample of the first footnote.

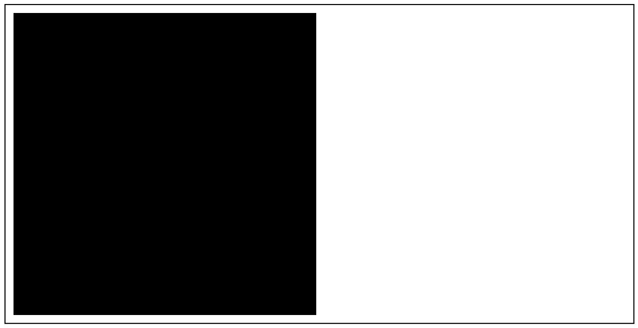

Figure 1: Sample figure caption.

Table 1: Sample table title

Part		
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

4.2 Tables

See awesome Table 1.

5 Conclusion

Your conclusion here

Acknowledgments

This was was supported in part by......

References

[1] 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 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 2356–2362, 2021.

[2] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2003)*, pages 173–180, 2003.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.