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

This project presents a novel approach to enhancing the Differentiable Search Index (DSI), a neural inverted index framework, by introducing three data augmentation techniques: (1) converting numerical values to words (Num2Word), (2) removing stopwords, and (3) leveraging a Part of Speech Masked Language Modeling (POS-MLM) strategy. These augmentations aim to improve the robustness and effectiveness of the DSI model in diverse retrieval scenarios. Additionally, we propose and evaluate four advanced variants of the DSI model: DSI+LoRA, DSI+QLoRA, DSI+AdaLoRA, and DSI+ConvoLoRA, which integrate parameter-efficient fine-tuning methods to optimize performance and resource utilization.

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

1 Introduction

Traditional information retrieval (IR) approaches, which often rely on static indexing and matching techniques, struggle to scale in dynamic, large and complex data environments. Neural Information Retrieval (NIR) methods, that leverages deep learning, have emerged allwing models to learn semantic representations and retrieve information with huge results. Among these, the Differentiable Search Index (DSI) stands out as a neural inverted index that maps document identifiers to semantic embeddings, offering an alternative to traditional IR paradigms.

The original DSI model faces challenges in adapting to diverse data distributions and resource constraints (DAVVEROOOOOO??????). To address these limitations, we enhance the DSI framework through both data augmentation and model optimization techniques. First, we introduce three novel data augmentation strategies tailored to enrich the input data: (1) Num2Word, which transforms numerical values into their word equivalents to standardize representation and improve semantic understanding; (2) stopwords removal, which eliminates non-informative words to focus on meaningful content; and (3) POS-MLM, a masked language modeling approach guided by part-of-speech tags to reinforce syntactic and semantic learning.

In parallel, we develop and evaluate four advanced variants of the DSI model to optimize its performance further. These include:

- DSI+LoRA: Incorporating Low-Rank Adaptation (LoRA) to fine-tune model weights with minimal computational overhead.
- DSI+QLoRA: Extending LoRA with quantization techniques for enhanced efficiency.
- DSI+AdaLoRA: Adapting LoRA dynamically based on task-specific requirements for improved flexibility.
- DSI+ConvoLoRA: Introducing convolutional layers within the LoRA framework to capture local patterns and context.

These enhancements will significantly improve retrieval accuracy and computational efficiency across various benchmarks. By combining innovative data augmentation techniques with parameter-efficient fine-tuning methods, our work

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

contributes a robust and adaptable solution for modern information retrieval challenges. This Project not only advances the capabilities of neural inverted indices but also gives insights into designing scalable and resource-aware IR systems.

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" $\rightarrow$ "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.

3.1.4 AdaLoRA

In addition to the original LoRA approach and its quantized version, we explored *Adaptive LoRA* (AdaLoRA) as introduced by [4]. The fundamental idea behind AdaLoRA is to dynamically adjust the effective rank of the LoRA adapters throughout training. By starting with a higher initial rank (`init_r`), the model first gains sufficient capacity to represent the task. After a specified warm-up period, AdaLoRA gradually prunes the less important dimensions in the low-rank matrices, converging to a smaller final rank (`target_r`). This rank-reduction process is governed by parameters such as:

- `tinit` and `tfinal`: defining the interval (in steps) during which the rank adaptation is active;
- `deltaT`: controlling how frequently to update (prune or reallocate) the ranks;
- `lora_alpha` and `lora_dropout`: same scaling and dropout parameters as standard LoRA.

The benefit of this method is that we can allocate sufficient capacity at the start of training, and let the model self-regularize to a much smaller final rank, thereby reducing the number of trainable parameters and mitigating overfitting. This is especially useful when dealing with large-scale models or when GPU memory is limited as in our case.

3.2 ConvLoRA (LoCon)

Another extension to LoRA that we investigated is *ConvLoRA* [5], sometimes referred to as LoCon. This method adds a lightweight convolutional operation to the LoRA adaptation, originally developed and thought for image-generation models (e.g., Stable Diffusion). In the NLP setting, we introduce a small depthwise convolution layer that operates

across the sequence dimension, followed by a low-rank linear mapping. Finally, a scaling factor (`alpha`) and a residual connection are applied, analogous to standard LoRA.

The motivation behind ConvLoRA is to allow the adapter to capture local token-level patterns more effectively than purely linear low-rank updates. The depthwise convolution can emphasize local features, potentially offering a richer representation. The rank of the linear mapping (`conv_lora_rank`) and the kernel size (`conv_kernel_size`) are hyperparameters that can be tuned. The idea is to freeze all the original transformer parameters and only training the convolutional and low-rank projection weights, preserving most of the benefits of parameter-efficient fine-tuning while enriching the local feature modeling.

3.2.1 Comparison of Techniques

In summary, we have that Table 1 summarizes the main differences among the proposed methods for memory-efficient fine-tuning and optimization. We highlight key ideas, how each method impacts model parameters and memory usage, and any special advantages or considerations.

Technique	Key Idea	Memory/Params Impact	Advantages / Notes
LoRA	Inserts low-rank adapters in the attention layers	Very few additional trainable parameters	Parameter-efficient fine-tuning; simpler training, fast convergence.
QLoRA	Quantizes base model to 4-bit and LoRA	Smaller memory footprint than LoRA	Maintains higher-precision adapters for fine-tuning while cutting base-model GPU usage.
AdaLoRA	Dynamically adjusts the rank of the LoRA adapters	Freezes base and variable-rank adapters	Allows higher rank initially for better capacity, then prunes rank to mitigate overfitting.
ConvLoRA (LoCon)	Depthwise convolution layer before low-rank projection	Small convolution and low-rank adapter	Better local token-level pattern modeling than LoRA

Table 1: **LoRA Family Comparison** these techniques are used for efficient T5-based DSI fine-tuning.

4 Training and Results

The fine-tuning experiments were conducted using PyTorch Lightning, in which we employed automatic checkpointing (every 4 batches), and mixed precision computation. For all experiments, we set the LoRA rank to 8 and used a scaling factor (α) of 32. A dropout rate of 0.1 was applied to prevent overfitting. The model was trained using mixed 16-bit precision (FP16) to improve memory efficiency. We used the AdamW optimizer with a learning rate of 5×10^{-5}, and the training was limited to a maximum of 5 epochs for obvious limitation in power. To prevent overfitting, early stopping was enabled (although not used here), terminating training after 3 consecutive epochs without validation loss improvement. To evaluate the efficiency of different fine-tuning techniques, we measured the number of trainable parameters, the total parameter count (including frozen parameters), the training time per 1,000 samples, and the validation time per 1,000 samples.

Technique	#Trainable Params	#Total Params	TrainTime (1K)	ValTime (1K)	ModelSize
LoRA	-M	-M	-	-	-
QLoRA	-M	-M	-	-	-
AdaLoRA	-M	-M	-	-	-
ConvLoRA (LoCon)	-M	-M	-	-	-

Table 2: **MS-MARCO100K Fine-Tuning Techniques Comparison**: A comparison of training/validation times over 1000 samples, total parameter counts, and number of trainable parameters across different fine-tuning techniques.

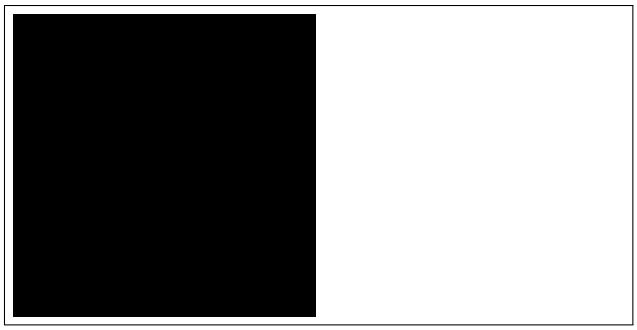

Figure 1: Sample figure caption.

4.1 Figures

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

5 Conclusion

In this work, we explored various Parameter-Efficient Fine-Tuning (PEFT) techniques for adapting a T5-based Dense Sparse Indexing (DSI) model to the MS-MARCO100K dataset. Our experiments evaluated LoRA, QLoRA, AdaLoRA, and ConvLoRA, assessing their efficiency in terms of trainable parameters, computational cost, and retrieval performance.

Future research will focus on extending these approaches to larger-scale retrieval datasets, exploiting additional peft methodologies further optimizing model efficiency for real-world applications.

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.

[4] Zequan Liu, Jiawen Lyn, Wei Zhu, Xing Tian, and Yvette Graham. Alora: Allocating low-rank adaptation for fine-tuning large language models, 2024.

[5] Zihan Zhong, Zhiqiang Tang, Tong He, Haoyang Fang, and Chun Yuan. Convolution meets lora: Parameter efficient finetuning for segment anything model, 2024.

[2] Sample of the first footnote.