NEURAL SEARCH INDEXING OPTIMIZATION: INTEGRATING AUGMENTATION AND PEFT FOR EFFICIENT RETRIEVAL

Alessio Borgi, Eugenio Bugli, Damiano Imola
1952442, 1934824, 2109063
Sapienza Università di Roma
{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 · Dynamic Pruning · LoRA · QLoRA · AdaLoRA · ConvoLoRA

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. 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+ALoRA: 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.

The DSI model operates within a **multi-task learning** framework, where **indexing** and **retrieval** are treated as distinct yet complementary tasks. This dual-task approach enhances model robustness, mutually reinforcing the system's overall effectiveness.

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 [**?**], in order to obtain the most relevant 1000 documents. The pre-processing applied to the dataset is explained in the following subsections.

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

2.1 DSI-QG-Merge Dataset Generation

In order to obtain a better dataset for our model we followed the generation of the DSI-QG-Merge dataset as described in Chen et al. [**?**]. The DSI-QG-Merge dataset was generated using a two-phase procedure to improve the completeness of document representations. First, each document in the corpus was divided into segments, and pseudo-queries were generated for these segments using a query generation model (docT5query). This produced a dictionary of text fragments and their corresponding pseudo-queries. Furthermore, to decrease the magnitude of the resulting dataset and ensure high-quality training data, a dense retrieval model was then used to filter these fragments, retaining only those segments that accurately recalled their original document identifiers. The resulting dataset is comprised of: filtered fragments, pseudo-queries, and original training queries, providing a relevant training corpus for the DSI-QG-Merge model.

In particular, for the docids generation, we optef for a semantically structured mechanism: the docid should be able to meaningful some informations related to the semantics of the associated document. To do this, we have followed the algorithm provided by [**?**] and reported in **??**.

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

2.2 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.

2.2.1 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.

2.2.2 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 [**?**], while Masked Language Modeling (MLM) has become a cornerstone of learning contextualized representations [**?**]. 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

We adopt T5 as our core architecture, but additionally optimize training and memory usage by applying 5 key techniques: **Dynamic Pruning**, **LoRA**[**?**], and **QLoRA**[**?**], **AdaLoRA**[**?**], and **ConvLoRA**[**?**].

3.1 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:

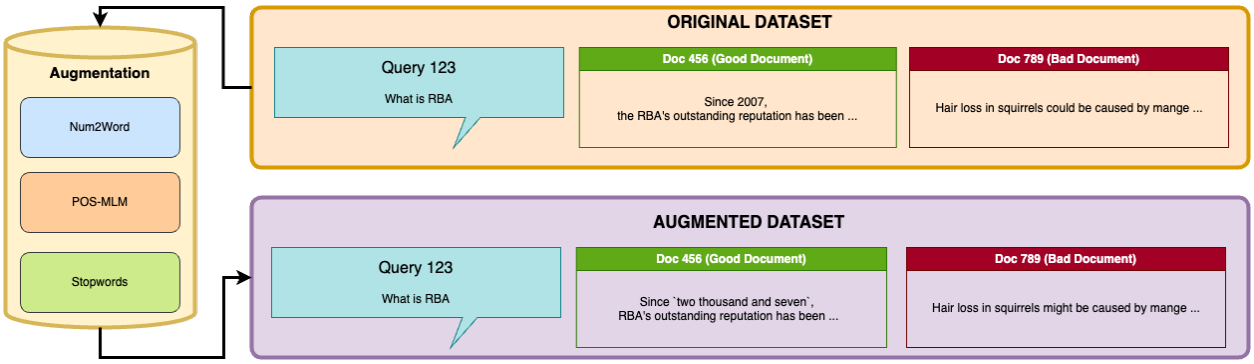

Figure 1: Data Augmentation Pipeline.

- **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).

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.

3.2 Parameter-Efficient Fine-Tuning Techniques

Beyond pruning, we integrate parameter-efficient fine-tuning approaches. The techniques we explored include:

- **LoRA (Low-Rank Adaptation):** 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.
- **QLoRA (4-bit Quantization for T5):** To further minimize memory consumption, we apply **QLoRA**, which quantizes the T5 weights to a *4-bit* format using the *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 setup reduces memory requirements and training overhead without sacrificing retrieval accuracy.

- **AdaLoRA (Adaptive LoRA):** Unlike standard LoRA, which uses a fixed rank, AdaLoRA dynamically adjusts the rank of the LoRA adapters during training. Initially, the model allocates a higher rank (`init_r`) to capture sufficient task-specific information. After a warm-up phase, the method prunes less important dimensions, gradually reducing the rank (`target_r`). This process minimizes trainable parameters while mitigating overfitting, making it ideal for large-scale models with constrained GPU memory.
- **ConvLoRA (LoCon):** A variant of LoRA that incorporates a depthwise convolutional layer before applying the low-rank projection. Originally designed for image-generation models, this extension enhances local feature modeling in text-based tasks by capturing token-level dependencies. The convolutional kernel size (`conv_kernel_size`) and rank (`conv_lora_rank`) are key hyperparameters that control the adaptation behavior.

Table **??** 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.

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 5 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 ValLoss	#Trainable Params	#Total Params	Time (1K)	GPU Usage	ModelsSize
Base T5 20.75	60.5M	60.5M	8m	10.7GB	242.03MB
LoRA 3.70	294K	60.8M	3m	8.6GB	243.20MB
QLoRA 3.89	294K	45.1M	5m	8.4GB	180.29MB
ALoRA 8.93	1.6M	62.1M	10m	9.1GB	248.52MB
ConvLoRA (LoCon) 19.45	6.1K	60.5M	2m 30s	2.2GB	242.05MB

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.

5 Conclusion

In this work, we enhanced the Differentiable Search Index (DSI) by integrating novel data augmentation techniques and parameter-efficient fine-tuning (PEFT) methods to improve retrieval effectiveness and computational efficiency. Our augmentation strategies, including Num2Word transformation, stopword removal, and POS-MLM augmentation, enriched semantic representations, reducing noise and improving contextual understanding.

In parallel, we leveraged PEFT techniques such as LoRA, QLoRA, AdaLoRA, and ConvLoRA to optimize training and inference, reducing memory consumption while preserving retrieval accuracy.

Future research will focus on extending these optimizations to larger datasets, refining augmentation techniques, and exploring further parameter-efficient methodologies to enhance retrieval capabilities in real-world applications.

A Semantically Structured Docids Generation Algorithm

The algorithm and the visualization of the semanticalli structured docids are depicted respectively in **??** and **??**.

Algorithm 2 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

B DSI Overall Architecture

The overall architecture of the Differential Search Index model is reported in the figure **??**

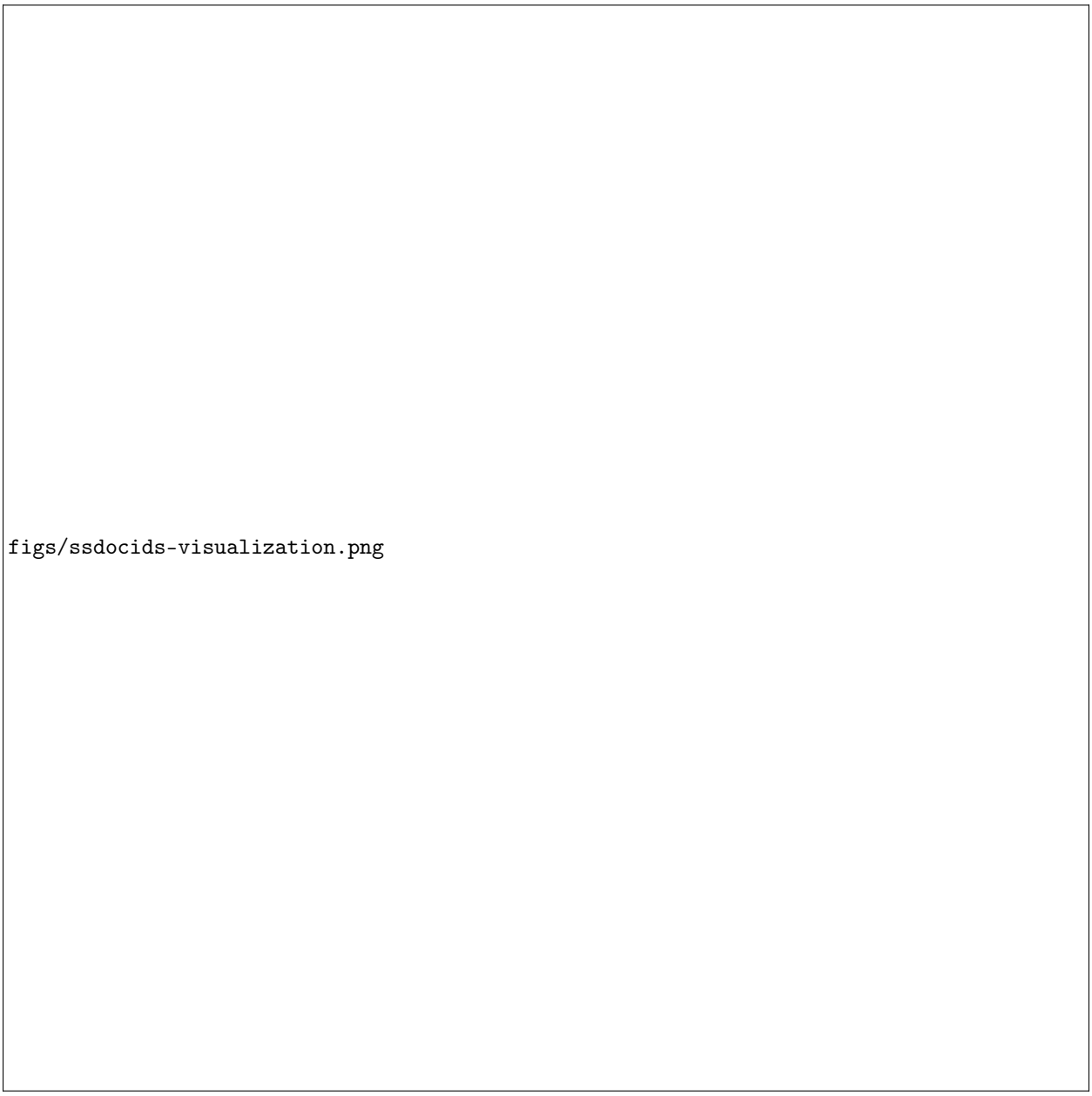

Figure 2: Visualization of the Semantically Structured Docids tree for extracting identifiers from latent space.

Figure 3: Overall Differential Search Index architecture.