SEINE: SEgment-based Indexing for NEural information retrieval

Sibo Dong
sd1242@georgetown.edu
InfoSense, Dept. of Computer Science
Georgetown University, USA

Justin Goldstein
jjg130@georgetown.edu
InfoSense, Dept. of Computer Science
Georgetown University, USA

Grace Hui Yang
grace.yang@georgetown.edu
InfoSense, Dept. of Computer Science
Georgetown University, USA

ABSTRACT

Many early neural Information Retrieval (NeurIR) methods are re-rankers that rely on a traditional first-stage retriever due to expensive query time computations. Recently, representation-based retrievers have gained much attention, which learn query representation and document representation separately, making it possible to pre-compute document representations offline and reduce the workload at query time. Both dense and sparse representation-based retrievers have been explored. However, these methods focus on finding the representation that best represents a text (aka metric learning) and the actual retrieval function that is responsible for similarity matching between query and document is kept at a minimum by using dot product.One drawback is that unlike traditional term-level inverted index, the index formed by these embeddings cannot be easily re-used by another retrieval method. Another drawback is that keeping the interaction at minimum hurts retrieval effectiveness. On the contrary, interaction-based retrievers are known for their better retrieval effectiveness.

In this paper, we propose a novel SEgment-based Neural Indexing method, SEINE, which provides a general indexing framework that can flexibly support a variety of interaction-based neural retrieval methods. We emphasize on a careful decomposition of common components in existing neural retrieval methods and propose to use segment-level inverted index to store the atomic query-document interaction values. Experiments on LETOR MQ2007 and MQ2008 datasets show that our indexing method can accelerate multiple neural retrieval methods up to 28-times faster without sacrificing much effectiveness.

ACM Reference Format:
Sibo Dong, Justin Goldstein, and Grace Hui Yang. 2022. SEINE: SEgment-based Indexing for NEural information retrieval. In *Proceedings of The ACM SIGIR 2022 Workshop on Reaching Efficiency in Neural Information Retrieval (ReNeuIR) (SIGIR ReNeuIR'22).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Neural Information Retrieval (NeurIR) methods use deep neural networks to rank text documents in response to user queries [33] and achieve state-of-the-art performance for ad hoc retrieval [2, 9, 12, 13, 15, 17, 18, 24, 26, 32, 34, 45, 46, 54, 59]. However, most NeurIR methods are re-ranking methods, which must rely on a first-stage conventional retriever such as BM25 [43] to obtain a manageable set of relevant document candidates. They re-rank this sent because for NeurIR methods to rank the entire corpus is computationally prohibitive.

To resolve the efficiency issue, representation-based NeurIR approaches have been explored for they can pre-process document collections offline and offload much of the computation burden. Representation-based methods encode a query or a document into a single fixed-size vector. Each query and each document has its own embedding vector, trained in a way that the embeddings can be used in dot product to measure query-document similarity. This setup allows pre-computation of document representations of the entire collection at offline time. Compared to a conventional retrieval system that consists of an indexing phase and a retrieval phase [44], representation-based approaches' learning and storing the document embeddings can be thought of the "indexing" phase; and calculating and sorting the dot products is the "retrieval" phase. It can be shown in the following pipeline (Here $E(q)$ and $E(d_i)$ are embeddings for query q and document d_i, respectively):

$$\begin{tikzcd}
\text{Learn } E(d_i) \arrow[r] & \text{Store } E(d_i) \text{ in index}, \forall i \arrow[dr] & & \\
& & \forall i, E(q)\cdot E(d_i) \arrow[r] & \text{Top-k results by MIPS, ANN.} \\
\text{Learn } E(q) \arrow[r] & E(q) \arrow[ur] & &
\end{tikzcd} \tag{1}$$

The embedding vectors can be either dense or sparse. Dense vectors are usually short, with nearly all non-zero entries. Sparse vectors, on the contrary, can be long and with many zero-valued entries. Dense, representation-based retrievers (e.g., DPR[23], SBERT [42], Condenser [10], ICT [25], RocketQA [41], ANCE [53], RepBERT[57], and ColBERT [24]) have gained much attention recently. They study pre-training or fine-tuning (mostly fine-tuned from the BERT embeddings [6]) methods to obtain dense low-dimensional encodings for queries and documents. Unlike results obtained from the sparse bag-of-words (BoWs) representations, top-K results obtained from these dense representations cannot be efficiently found without any approximation. Instead, they must be assisted with efficient search algorithms for approximate nearest neighbor search (ANN) [20, 53] or maximum inner-product search (MIPS) [3]. Popular efficient search algorithms leverage ideas such as hashing (e.g. LSH [47]), clustering [29], product quantization (e.g. ScaNN [14], FAISS [20]) and dimension reduction (e.g. PCA [21], t-SNE [49]), equipped with carefully chosen data structures.

Sparse, representation-based retrievers have also been explored to further improve indexing and retrieval efficiency by forcing more zero entries in the embeddings. Existing methods to enforcing sparsity include using gating to select entries (e.g., SparTerm

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGIR ReNeuIR'22, July 15, 2022, Madrid
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1]), assigning zeros to non-query terms (e.g., EPIC [28]), and regularizing the vectors by minimizing flop operations (e.g., SPLADE [8] and FLOPS [39]), etc. Although these methods make the embedding vectors sparser, they do not change the basics of the dual-encoder setup (illustrated in Eq. 1). Most of them still use the over-simplified dot product as their retrieval function (e.g., SparTerm [1], EPIC [28], and SPLADE [8]). It is unclear how the benefit gained from sparse representations can be leveraged to employ more efficient data structures such as the inverted index.

We argue that representation-based methods, including both dense and sparse ones, have a few drawbacks for document retrieval. First, unlike traditional term-level inverted index, the index formed by a representation-based approach's embeddings cannot be easily re-used by another retrieval method. Representation-based methods focus so much on finding the representation that best represents a piece of text, aka learning the metric space as in metric learning. Every index that is learned by a representation-based method is unique to itself, which makes it hard to be re-usable by others. When one works on a representation-based method, each time she must process the document collection and rebuild the entire index. It contradicts a common expectation for an index – that it should be general and re-usable by down-stream tasks such as various retrieval methods.

Second, in a representation-based method, the actual retrieval function responsible for similarity matching between query and document is kept to the bare minimum. Most of them use a dot product. ColBERT [24] used a slightly more sophisticated maximum cosine similarity function, but it is still over-simplified for expressing the interaction between query and document. In fact, researchers have discovered that separating query and document in the indexing phrase and keeping their interactions at minimum hurts retrieval effectiveness. Wang et al. pointed out that it is necessary for dense, representation-based retrievers to interpolate with sparse, interaction-based retrievers such as BM25 [50] to gain better performance. Luan et al. proved that embedding size imposes an upper bound on effectiveness that a dense retriever can achieve and on text length that it can handle [27]. It explains why few successes for these approaches have been seen on first-stage, long document retrieval.

On the other hand, interaction-based retrievers have been known for their superior retrieval effectiveness. This can be witnessed by the long-time record set by BM25, a sparse, interaction-based method in the pre-neural era, and by the top performance set by MonoBERT [34], an all-in-all, dense interaction-based method marked by its extensive interactions among query terms and document terms. Many early NeurIR methods are also sparse, interaction-based methods [7, 13, 36–38, 43, 48, 52]. When receiving a query from the user, they generate a query-document interaction matrix in real-time and feed it into neural networks to predict a relevance score. Early interaction-based neural models do not have an index and must construct a large interaction matrix at query time, which is the main obstacle that prohibits them from succeeding in applying to first-stage, full-length documents.

A few pieces of work have been proposed to support indexing for interaction-based neural retrievers. Most of them take advantage of an inverted index for its fast lookup functions. They share a pipeline illustrated as the following:

$$\begin{tikzcd}
\text{Process } d_i \arrow[r] & \text{Store}\langle v, d_i\rangle \text{ in index}, \forall i \arrow[dr] & & \\
& & \text{Lookup } q \text{ in } v \arrow[r] & \text{Top-k results by } s(q, d_i), \\
\text{Process } q \arrow[r] & q \arrow[ur] & &
\end{tikzcd} \tag{2}$$

where v is the vocabulary obtained from the document collection, $\langle v, d_i\rangle$ is the interaction between v and document d_i, and $s(q, d_i)$ is a retrieval function that measures the similarity between q and d_i.

For instance, DeepCT [4] substituted term frequency with context-aware term weights aggregated from BERT embeddings [6], and stored them in an inverted index. Their followup work, HDCT [5], succeeded in full-length document retrieval. TILDE [61] stored conditional term probabilities in an inverted index to support deep query-likelihood calculations in its retrieval function. SPARTA [58] stored dot products between vocabulary terms to document terms in an inverted index.

While it is encouraging to see these methods build indices for sparse, interaction-based retrievers, we find their indices are tailored to the specific neural retrieval function that they use in their retrieval phase. It is suboptimal if these indices cannot be general enough to be re-used by other neural retrievers. For instance, neural retrievers developed prior to BERT, including KRNM [43], HiNT [7], and DeepTileBars [48], are effective on document retrieval but have no index to support them. SNRM can be made to support them, as we demonstrated in our experiments, however, the matchings are done with latent terms, not actual lexical terms and the effectiveness degrades much.

In this paper, we propose a novel **SEgment-based Neural Indexing method**, **SEINE**, which provides a general indexing framework that can flexibly support a variety of neural retrieval methods. We focus on facilitating interaction-based retrieval methods for their higher effectiveness. During query time, a retriever can only look up pre-computed interaction function values for corresponding query terms from the index, quickly merging them into a query-document interaction matrix and sending it to neural networks to predict a relevance score. Moreover, we adopt a flexible, segment-based design in our index to support query-document interaction at different granularities. For instance, the query-document interaction can be done at the document-level (e.g. BM25 [43] and TILDE [61]), term-level (e.g., KRNM [43]), or topical segment-level (e.g., HiNT [7] and DeepTileBars [48]). We believe as long as we can decompose the retrieval methods and identify the interaction units used between the query and the document, we should be able to build an index that is general, modularized, and reusable. Our segment-level inverted index stores common components, which we call atomic interaction values, including term frequency, BERT embedding, conditional probabilities, etc. We also leverage Spark programming to accelerate the entire indexing process. Experiments on LETOR MQ2007 and MQ2008 datasets show that our indexing method can accelerate multiple neural retrieval methods up to 28-times faster without sacrificing much effectiveness.

Note that the most effective neural retrievers at the moment when the paper is being written are dense, interaction-based methods, such as MonoBERT [34] and monoT5 [35], whose all-in-all interactions within its transformer blocks cannot be easily decomposed. We leave creating indices for this type of retrievers as future work and focus on sparse, interaction-based methods.

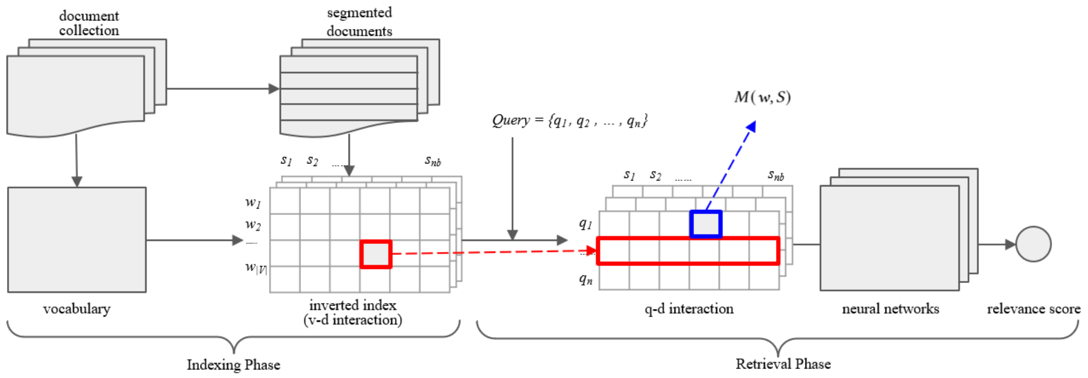

Figure 1: SEINE: SEgment-based Indexing for NEural information retrieval.

Our work makes the following contributions:

- We propose a general indexing framework for constructing reusable indices to support sparse, interaction-based neural retrievers;
- We design our index to enable multiple sparse, interaction-based NeurIR methods that previously have no index and suffer from high latency; Our work rejuvenates them for first-stage, full-length document retrieval;
- We also demonstrate how to make use of the Spark programming to accelerate the entire indexing process.

2 METHOD

Our focus is on decomposing existing interaction-based neural retrievers by two measures: 1) recognizing that they perform query-document interactions at different granularities and providing a flexible segment-based approach to suit them all; 2) identifying a list of atomic interaction function values that can be pre-computed and stored in an inverted list to support common neural retrievers.

Figure 1 illustrates SEINE's framework. It supports end-to-end ad hoc document retrieval and has two phases, an indexing phase and an retrieval phase. The **indexing phase** is query-independent and can be done offline. During indexing, we (1) process the entire corpus and obtain a vocabulary v, (2) segment each document in the collection, and then (3) create an inverted index by interacting each vocabulary term w_i with each document d_j on the segment level. We call this interaction *v-d interaction* meaning each vocabulary word interacting with each document in the collection. The inverted index is used to store a list of interaction values for each word-segment pair. The number of segments per document is standardized for all documents. (4) We also adopt Spark[1] and make use of its parallel programming to accelerate the indexing process. At the **retrieval phase**, when a query q comes in, each query word is looked up from the v-d interaction matrix stored in the index and the matched terms' rows are extracted and stacked into another interaction matrix called *q-d interaction.* We feed the q-d interaction matrix into a neural retrieval function to obtain the final relevance scores $s(q, d_j), \forall j$. Since the number of query terms are small, calculations at the retrieval phase is sparse and quick to finish.

[1]https://spark.apache.org/.

2.1 Pre-Processing a Document Collection

To begin, we process the entire corpus $C = \{d_1, d_2, ..., d_{|C|}\}$ and obtain a vocabulary $v = \{w_1, w_2, ..., w_{|v|}\}$. The vocabulary is the set of unique terms appearing in the corpus. Standard text pre-processing steps are taken to attain the vocabulary. First, we tokenize all documents in the collection C into a sequence of tokens using the Wordpiece tokenizer [51]. Then, we remove the most frequent 10% and the least frequent 10% terms from the vocabulary based on their collection-level frequency. It is to exclude misspellings, rare words, programming scripts, punctuation, special symbols, stopwords, etc. While we take this pass on the entire document collection, we keep track of the inverse document frequency of each term ($idf(w_i), w_i \in v$) and store them: $idf(w_i) = \log \frac{|C|}{|\{j|w_i \in d_j\}|+1}$.

For most neural retrievers that we support, we assume that all incoming query terms can be found in the vocabulary, so that at query time the *q-d interaction* matrix can be built by looking up in the pre-computed *v-d interaction* matrix, instead of calculating the *q-d interactions* on the fly. We are aware that this assumption may not be valid and out-of-vocabulary terms can certainly be in a query. In some neural indexers that we compare with (e.g. SNRM [56]), however, a latent semantic representation is used as the indexing unit, which can be seen as a way to alleviate the vocabulary mismatch problem.

2.2 Segmenting Documents to Support Various Interaction Granularities

Existing interaction-based neural retrievers perform query-document interaction at different granularities. Some are performed at the document-level (e.g. SNRM [56] and BM25 [43]), which interacts a query q with an entire document d; some are at the term-level (e.g., KRNM [43]), meaning interacting a single query term and a single document term; and others are at segment-level that represent topics in a document (e.g., HiNT [7] and DeepTileBars [48]). We propose a flexible segment-based indexing approach to support query-document interaction at different granularities. Our chosen interaction unit is the segment, whose size can be adjusted to include term- and document-level interactions.

For a document d, we split it into non-overlapping segments following the TextTiling algorithm [16]. The segmentation is done based on the similarity between fixed-sized, neighbouring text windows. If any two neighboring windows show high similarity, we merge them into a bigger segment. If they are dissimilar, they are put into different segments. The resulting segments roughly reflect topical boundaries in a document. Unsurprisingly, the number of segments y in one document can be quite different from that in another. To standardize y for different documents, we (1) pad empty segments if $y <= n_b$, a pre-defined, adjustable dimension parameter, or (2) squeeze all remaining text into the final segment if $y > n_b$. Eventually, all documents contain an equal number of n_b segments: $d = \{S_1, S_2, ..., S_{n_b}\}$. When $n_b = 1$, it is equivalent to interacting at the document-level; when $n_b = |v|$, it is equivalent to interacting at the term-level. Note that driven by a completely different motivation, [27] also spoke about the advantage of using multiple, fixed length vectors, instead of a single vector, to represent a document. Here we introduce segment-level indexing for its flexibility to cover query-document interactions at all levels of granularities, when n_b is set to different values.

2.3 Storing Atomic Interactions in Inverted Index

In this paper, we propose to decompose components in existing neural retrievers that are related to query-document interactions into atomic interaction functions and store their values into the index. These atomic interactions include term frequency, operations over BERT embeddings, kernel functions, conditional probabilities, etc.

For each term $w \in v$ and an interaction unit S, where S is a text segment (which can be adjusted to represent a document or a term), we pre-calculate and store the following atomic interaction function values:

- **Term frequency:** $tf(w, S)$, the number of occurrences of term w in S. It can be stored to support traditional retrieval methods such as BM25 [43] and neural retrievers such as DeepTileBars [48].
- **Indicative inverted document frequency:** $idf(w) \times \mathbb{I}_S(w)$, where $idf(w)$ is the inverse document frequency of term w and $\mathbb{I}_S(w)$ is indicates whether w is in S. This function can be used to support traditional retrieval methods such as BM25 [43] and the neural retrievers such as HiNT [7] and DeepTileBars [48].
- **Dot product:** $\sum_{t \in S} E(w) \cdot E(t)$, where $E(.)$ is an embedding output from a pre-trained neural encoder, such as word2vec [31] or BERT [6]. The dot product measures similarity between the two embeddings. In theory, $E(.)$ can be any dense or sparse representations for a text sequence. Therefore, this interaction function can be used to store interaction between the dense representations of a word and a segment. It is thus can be used to support MatchPyramid [36] and dense retrievers such as COIL [11].
- **Cosine similarity:** $\sum_{t \in S} \frac{E(w) \cdot E(t)}{|E(w)| \cdot |E(t)|}$, similarly to the dot product and used as another similarity function. This function supports neural retrievers such as KNRM [52] and HiNT [7].
- **Gaussian kernel:** $\max_{t \in S} exp(-(E(w) - E(t))^2)$, proposed by [52] to measure the distance between two terms within a semantic neighborhood. It can be used to find the most similar synonym to a word in S. It supports KRNM [52] and DeepTileBars [48].
- **Linear aggregation on BERT Embeddings:** $a \cdot E_w(S) + b$, where $E_w(S)$ is a vocabulary term w's BERT embedding in text S. It linearly combines BERT embeddings using learned weights a, b and can be thought of an aggregated contextual term weight for w in S. It supports DeepCT [4] and can be used in combination with traditional retrievers such as BM25.
- **Max operation on BERT Embeddings:** $\max_{t \in S} f_S(E(t)) \cdot E(w)$, where f_S is the logarithm of the softplus over BERT embeddings. $E(\cdot)$ is a BERT embedding. This function selects the most similar term in a piece of text to a vocabulary term (w) and records their similarity. It can be used to support EPIC [28] and ColBERT [24].
- **Multi-layer Perceptron on BERT Embeddings:** $MLP(E_w(S))$, where $MLP(\cdot)$ is multilayer perceptron with activations over BERT embeddings. It can be used to support retrievers such as DeepImpact [30].
- **Log conditional probability:** $\log P_\theta(w|S)$, where $P_\theta(w|S)$ is the conditional probability for term w in text S and can be obtained by using a language modeling head on the [CLS] token of $E_w(S)$, which is a vocabulary term w's BERT embedding in text S. It can be used to support deep query likelihood models such as TILDE [61].

Some of these atomic interaction functions are proven to be essential for ad-hoc retrieval and others are widely used in interaction-based neural retrievers. Note that the list of functions is not exhaustive and one can certainly expand the list or choose a subset of functions in their index. One condition must be satisfied to identify those atomic functions – that the vocabulary entries, in our case the terms, must be independent of each other. In all-in-all interaction-based methods, such as MonoBERT, however, terms interact within and across a query and document and the interactions in the transformer blocks cannot be easily decomposed. We thus do not support them.

We define a vocab-segment interaction column vector $M(w_i, S_k)$ for the i^{th} vocabulary word w_i and the k^{th} segment (across all documents), and keep each of the above atomic interaction function values in it. These term-segment interactions $M(w_i, S_k)$ are then form the v-d interaction matrix:

$$M_{v,d} = concat\{\ M(v, S_1),\ M(v, S_2),\ \dots,\ M(v, S_k),\ \dots,\ M(v, S_{n_b})\ \} \quad (3)$$

where $M(v, S_k)$ is obtained by combining for all terms $w_i \in v, \forall i$ in the k^{th} segment. Eventually, we generate an interaction matrix of dimension $|v| \times n_b \times n_f$ which stores the atomic interactions for every vocab term-document pair, where $|v|$ is the vocabulary size, n_b is the number of segments in a document, and n_f is the number of atomic interaction functions.

At retrieval time, we obtain the q-d interaction matrix for an incoming query by looking up query terms in the vocabulary and stacking the rows in the index that stores their pre-computed interaction scores:

$$M_{q,d} = stack_{w_i \in q \cap v}\{\ M_{w_1,d},\ M_{w_2,d},\ \dots,\ M_{w_i,d}\ \dots\}. \quad (4)$$

Algorithm 1 Spark pseudo-code for indexing.

```
1: Initialize Spark environment and configuration
2: Import functions segmentation, interaction
3: Vocab  ← $RDD\{w_1, w_2, ..., w_{|V|}\}$                ▷ create RDD
4: Corpus ← $RDD\{d_1, d_2, ..., d_{|C|}\}$                ▷ create RDD
5: Segmts ← Corpus.$map$ (segmentation)        ▷ document segmentation
6: Cart   ← Vocab.$cartesian$(Segmts)
7: Index  ← Cart.$map$ (interaction)           ▷ calculate $M$ as in § 2.3
8: Index  ← Index.$filter$ ($tf > \sigma_{index}$)
9: Index  ← Index.$reshape$                    ▷ v-S to v-d
10: Index.$saveAsPickleFile$ ()
```

Our index can support neural interactions at different granularities, only by varying the segment size. To support neural retriever with document-level interactions, we let the segment size equal the document length so that there is one segment and it is d. The v-d interaction matrix becomes $M_{v,d} = M(v,d)$. To support neural retrieval methods with term-level interactions, we treat each term as a segment, i.e. the segment size is one term. The v-d interaction matrix becomes: $M_{w_i,d} = concat\{\ M(w_i,t_1),\ M(w_2,t_2),\ \dots,\ M(w_i,t_{n_d})\ \}$, and n_d is the document length.

2.4 Accelerating with Spark

Large corpora can contain tens of millions of documents and a vocabulary of tens of thousands of words. Given a dozen to three dozens of segments in a document, there can be trillions of query-segment interactions when building the index. In our implementation, we leverage Spark [55] to accelerate the indexing process. Spark uses a special data structure called the resilient distributed dataset (RDD), which can hold a large distributed collection of objects and has a built-in parallel programming infrastructure. Each RDD automatically splits into multiple data partitions and can be computed on different computer nodes [22]. Spark has two types of programming functions, transformations and actions, and uses a lazy evaluation mechanism. Transformations are not real computations but prototyping functions that wait for computation paths optimization and data parallelization are done by the underlying Spark infrastructure. Actions do the actual computations, but only when it is absolutely necessary to compute.

We employ Spark to accelerate the indexing process. After obtaining the vocabulary v and segmenting all documents into segments, we perform v-d interaction over all the terms in v and all segmented documents in corpus C: (1) We create RDDs for both vocabulary term list and document list. (2) We use a transformation operation cartesian$(\cdot,\cdot)$ to compute a Cartesian product between two RDDs, for example we have a vocabulary $V = \{apple, banana\}$ and a collection $C = \{d1, d2\}$, the Cartesian function returns a list of term-document pairs: $\{(apple, d1), (apple, d2), (banana, d1), (banana, d2)\}$. (3) We use another transformation operation map$(\cdot)$ to calculate interaction matrix by applying function interaction$(\cdot)$ to v-d pairs, where interaction$(\cdot)$ is defined in Section 2.3. (4) In order to balance the memory storage with information loss, we use a transformation operation filter to control the index sparsity by the threshold σ_{index}. For example, if $\sigma_{index} = 0$, only the documents containing the corresponding term are stored in the index. With $\sigma_{index} > 0$ can further improve the sparsity and efficiency of the index, but it may lead to information loss and effectiveness drop. (5) We combine and reshape the word-segment interaction into a v-d interaction matrix. (6) We then use an action operation to write the results into disk. Algorithm 1 outlines our Spark implementation.

3 EXPERIMENTS

We experiment on LETOR 4.0, a widely used benchmark dataset for document retrieval. It uses the Gov2 web page collection (~2M pages) and two query sets from TREC Million Query (MQ) 2007 and 2008 Tracks [40]. MQ2007 contains about 1,700 queries and 65,323 annotated documents; MQ2008 800 queries and 15,211 annotated documents. We use the official effectiveness metrics in LETOR. They include Precision (P), Normalized Discounted Cumulative Gain (nDCG) [19] at various positions, and Mean Average Precision (MAP) [60]. For efficiency measures, we report the wall clock time in milliseconds for processing a query-document pair during training and during testing, respectively. The training time calculates the average time used for each sample $(q, d_1, d_2, label)$ per epoch. We compare the time spent by a retriever when there is no index supporting it versus there is an index. For experimental runs with No Index, the training time includes the time spent on generating the interaction matrix; for experimental runs with an an index, it contains the time to lookup from the index. The test time calculates the average time spent on predicting a score for each query-document pair (q, d). For all runs, we adopt a five-fold cross-validation and report its results.

3.1 Baseline Runs

We organize the experiments by separating a retriever's indexing method from its retrieval method and report the effectiveness and efficiency of the combinations. All runs in our experiments perform first-stage document retrieval, not re-ranking nor passage retrieval. Note that there are many recent neural retrievers being proposed, we select a few representative ones for their advanced retrieval functions, especially for those early neural methods that had no index nor dense representation to remedy the efficiency issue. Previously, they were limited to act as a re-ranker on passage retrieval tasks. In this work, one of our main purpose is to rejuvenate them to apply to first-stage, full-length document retrieval. We therefore select the following *retrieval methods* for our experimentation:

• **Dot Product** In [56], they calculated the relevance score by $s(q,d) = \sum_{|q_i|>0} q_i d_i$, where q_i are the non-zero elements of q and d_i is the document with non-zero elements in the i-th index. For our runs with SEINE index, we enable dot product and score the relevance by summing over all query terms.

• **BM25[43]:** We compare BM25 results using an inverted index with conventional bag-of-words term weights vs. using the inverted index with one of SEINE's interaction values turned on. The interaction is the BERT-based term weight proposed by DeepCT [4].

• **KNRM [52]:** an interaction-based neural retrieval method that performs term-level interaction over term embeddings, then uses kernel-pooling to extract multi-level soft matching features to learn the relevance score.

• **HiNT [7]:** a hierarchical neural retrieval method that generates segment-level interaction matrices as the network's input, then

Table 1: Retrieval Effectiveness and Efficiency on MQ2007 and MQ2008. * denotes statistically significant degradation on effectiveness and † for statistically significant improvement on efficiency compared to corresponding retrieval method with "No Index".

(a) **MQ2007.**

		Effectiveness					Efficiency			
Indexing	**Retrieval**	P@5	P@10	MAP	nDCG@5	nDCG@10	Training (ms)		Test (ms)	
No Index	Dot Product	0.328	0.344	0.418	0.282	0.332				
	KNRM	0.355	0.382	0.461	0.379	0.412	42.63		16.70	
	HiNT$\diamond$	0.461	0.418	0.505	0.463	0.490	1139.34		1009.11	
	DeepTileBars	0.429	0.408	0.474	0.398	0.434	163.91		73.68	
InvIdx‡	BM25$\diamond$	0.388	0.366	0.456	0.384	0.414				
SNRM	Dot Product	0.288*	0.307*	0.368*	0.254*	0.302*				
	KNRM	0.322*	0.347*	0.417*	0.337*	0.404	35.56	1.2×	13.17	1.3×
	HiNT	0.401*	0.358*	0.423*	0.379*	0.402*	958.03	1.2×	860.76	1.1×
	DeepTileBars	0.281*	0.304*	0.359*	0.238*	0.290*	131.43	1.2×	56.97	1.3×
SEINE	Dot Product	0.328	0.344	0.418	0.282	0.332				
	BM25 w/ DeepCT weight	0.315	0.327	0.397	0.266	0.314				
	KNRM	0.342	0.372	0.447	0.374	0.401	11.67†	3.7×	1.22†	13.7×
	HiNT	0.453	0.409	0.492	0.452	0.483	834.64	1.4×	706.42	1.4×
	DeepTileBars	0.412	0.404	0.468	0.391	0.427	22.62†	7.4×	2.67†	28.1×

(b) **MQ2008.**

		Effectiveness					Efficiency			
Indexing	**Retrieval**	P@5	P@10	MAP	nDCG@5	nDCG@10	Training (ms)		Test (ms)	
No Index	Dot Product	0.333	0.281	0.462	0.411	0.183				
	KNRM	0.355	0.355	0.472	0.499	0.225	41.27		16.86	
	HiNT$\diamond$	0.367	0.255	0.505	0.501	0.244	1139.34		1009.11	
	DeepTileBars	0.425	0.321	0.567	0.548	0.259	167.20		74.96	
InvIdx‡	BM25$\diamond$	0.337	0.245	0.465	0.461	0.220				
SNRM	Dot Product	0.380†	0.300†	0.513†	0.483†	0.223†				
	KNRM	0.303*	0.229*	0.417*	0.417*	0.199*	36.78	1.1×	12.68	1.3×
	HiNT	0.291*	0.209*	0.401*	0.445*	0.221*	941.09	1.2×	904.16	1.1×
	DeepTileBars	0.356*	0.291*	0.481*	0.434*	0.200*	134.10	1.2×	58.14	1.3×
SEINE	Dot Product	0.333	0.281	0.462	0.411	0.183				
	BM25 w/ DeepCT weight	0.307	0.239	0.448	0.400	0.197				
	KNRM	0.346	0.252	0.462	0.485	0.218	11.58†	3.6×	1.24†	13.6×
	HiNT	0.362	0.250	0.485	0.489	0.236	828.84	1.4×	687.85	1.5×
	DeepTileBars	0.422	0.322	0.569	0.544	0.255	22.62†	7.4×	2.67†	28.1×

$^\diamond$ Results reported in Fan et al. [7]. ‡ Indexing method is not applicable to KNRM, HiNT, and DeepTileBars.

uses a local matching layer and a global decision layer to learn the relevance score.

• **DeepTileBars [48]:** a neural retrieval method that segments documents by topics then scans them by multiple varied-sized Convolutional neural networks (CNNs) to learn the relevance score.

For the *indexing methods*, we experiment on:

• **No Index:** This is the case when a retriever directly processes the document collection and interacts a query to all documents at query time. Most neural retrievers are of this type, including the most effective MonoBERT [34]. In our experiments, we test on the above-mentioned neural retrievers to illustrate.

• **Inverted Index (InvIdx):** the traditional indexing method using the bag-of-words representation. It stores a posting list for each vocabulary term w, consisting all the $(d, tf(w, d))$ pairs for every document d containing w. Note that InvIdx does not store any semantic interactions nor embedding-based interactions, so it cannot support the recent neural retrieval methods.

• **SNRM [56]:** a neural indexing method that learns a sparse latent representation for the interaction between each pair of training query and document, and stores the latent nodes in an inverted index. Note that in SNRM, the latent words are independent of each other, which makes it satisfy the condition that we mentioned in Section 2.3 that vocabulary entries must be independent of each other. We can therefore use the latent nodes as vocabulary entries in the inverted index and manage to apply SNRM to KNRM, HiNT, and DeepTileBars. We first represent the query and document as a sequence of latent words, and then generate the interaction matrix based on the latent words. The average document length is

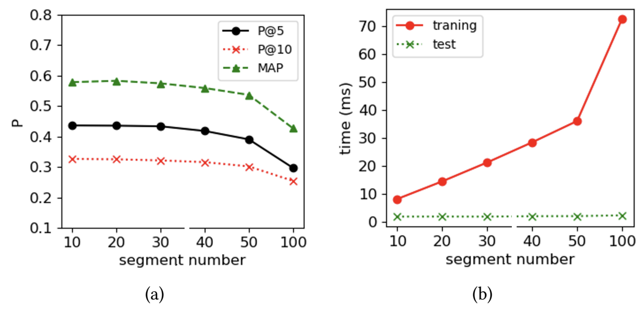

Figure 2: Effectiveness and Efficiency Results with different number of segments. Experimented with DeepTileBars+SEINE on MQ2008.

significantly reduced from 1500 words to 130 latent words on the ClueWeb corpus.

• **SEINE:** The segment-based neural indexing method proposed in this paper. Note that for different retrieval methods, SEINE turns on a different subset of atomic interaction functions. For instance, DeepTileBars are supported by term frequency, indicative inverted document frequency and Gaussian kernel values; KNRM are supported by cosine similarity.

We use the following parameter settings in our implementations. To keep a manageable vocabulary size, we use the middle 80% frequent terms in the vocabulary. For LETOR 4.0, the vocabulary is around 40,000 words. We set $\sigma_{index} = 0$, which means documents containing the corresponding term and only them are stored into the index. With $\sigma_{index} > 0$ can further improve the sparsity and efficiency of the index, but it may lead to information loss and effectiveness drop; so we set it to be zero. For all experimental runs, whenever possible, we use the default settings recommended in their original published papers. We employ the pre-trained word2vec model [31] to represent terms for KNRM, HiNT, and DeepTileBars, and BERT embeddings in BERT-base-uncased for DeepCT.

3.2 Main Results

Table 1 reports the effectiveness and efficiency results for a few retrieval methods, such as KNRM, HiNT, and DeepTileBars, combined with different indexing methods on LETOR 4.0 first-stage document retrieval task. Our experiments demonstrated similar results for both MQ2007 and MQ2008. Here No Index is the baseline indexing method, which shows the performances when a retrieval method processes the entire document collection at query time without using any index. InvIdx can only support bag-of-words retrievers and we show BM25 to illustrate. SNRM and SEINE can both be used to support various retrieval methods. Note the original DeepCT system is included as one run under SEINE since we include DeepCT weights in our index.

Both SNRM and SEINE are able to significantly improve retrieval efficiency. SNRM makes the index sparse enough to handle a large corpus by mapping document text to new latent terms. Its sparsity helps reduce the amount of the interaction matrix calculation, thereby speeding up the retrieval phase. For instance, it gets 1.2×, 1.2×, 1.2× faster for training the KNRM, HiNT, DeepTileBars models, and 1.3×, 1.1×, 1.3× for test on the MQ2007 dataset. However, SNRM has a large degradation, ranging from -40% to -9% on the effectiveness metrics, over No Index. This might be caused by SNRM's ineffectiveness in lexical matching, i.e., exact matching. SNRM is good at semantic matching, which aims to address a variety of linguistic phenomena, such as synonymy, paraphrase, and term variation; its index only stores the latent terms. Although it can be applicable to multiple neural retrievers as SEINE does, it is difficult for SNRM to get overlapping terms between query and document. Lexical information is lost in the latent semantic nodes. On the LETOR datset, our results indicate SNRM makes a poor trade-off between efficiency and information loss.

On the contrary, SEINE stores the term-segment interaction in its index and generates a q-d interaction matrix by looking up and stacking rows for actual query terms. With minimal degradation in effectiveness, SEINE gets 3.7×, 1.4×, 7.4× faster for training the KNRM, HiNT, and DeepTileBars models, and 13.7×, 1.4×, 28.1× for test on the MQ2007 dataset. Similar results on the MQ2008 dataset can be observed.

3.3 Impact of Segment Size

We also look into the effects of different segment size in SEINE. We experiment with different number of segments per document using DeepTileBars+SEINE and report the findings on the MQ2008 dataset. Figure 2(a) is about effectiveness. It shows that using 20 segments per document performs the best for precision (in fact for all other effectiveness metrics too). Figure 2(b) is about efficiency. It shows that as the number of segments per document increases, the neural networks take longer time to train while the test time remains more or less the same. We also find that the average segment length for the best choices of segments per document, 20 and 30, are around 270 and 200 words respectively. This length is about the length of a natural paragraph. It suggests that we should select our segment size close to an author's topical breaks, instead of choosing too large a segment size just for the sake of increasing training efficiency. At the same time, since test time does not vary much with segment size, we can select a smaller number of segments to improve query run-time efficiency.

4 CONCLUSION

This paper proposes SEINE, a SEgment-based Indexing method for NEural information retrieval, that moves heavy computations in interaction-based neural retrievers offline and only keeps essential calculations at query time. It provides a general indexing framework and flexibly supports a variety of interaction-based neural retrieval methods. During the indexing phase, we build a vocabulary from the entire corpus and compute and store the vocabulary-segment interactions in the index. We propose to use segment-level inverted index to store the atomic query-document interaction values. Our indexing method can make the query run-time up to 28 times faster without sacrificing their effectiveness on LETOR 4.0 for first-stage, full-length document retrieval.

We propose to store atomic interactions between vocabulary term and segments in an inverted index. These interactions include term frequencies, inverse document frequency, dot products, operations over BERT embeddings, conditional probabilities, etc. Some of them are adopted from a few recent first-stage retrievers, such as DeepImpact [30], EPIC [28], and TILDE [61]. Our experiments did not include them because (1) our main focus is to rejuvenate the index-less re-rankers that achieve high performance, but suffer in terms of efficiency. However, (2) we do identify the atomic interaction function in DeepImpact [30], EPIC [28], and TILDE [61], and include them in our index. We hope re-building an index for neural retrieval can be avoided as much as possible so that researchers can shift their attention back to creating versatile retrieval functions.

Currently, SEINE does not support MonoBERT, an all-in-all, dense interaction-based method, the most effective retriever at the moment. In all-in-all interaction, terms interact within and across a pair of query and document. To implement SEINE for such methods, we need to understand how to decompose this all-in-all interactions within the transformer blocks into some function of independent vocabulary entries. We leave this as future work.

REFERENCES

[1] Yang Bai, Xiaoguang Li, Gang Wang, Chaoliang Zhang, Lifeng Shang, Jun Xu, Zhaowei Wang, Fangshan Wang, and Qun Liu. 2020. SparTerm: Learning term-based sparse representation for fast text retrieval. *arXiv preprint arXiv:2010.00768* (2020).

[2] Stéphane Clinchant and Florent Perronnin. 2013. Aggregating continuous word embeddings for information retrieval. In *Proceedings of the workshop on continuous vector space models and their compositionality*. 100–109.

[3] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of Recommender Algorithms on Top-n Recommendation Tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems* (Barcelona, Spain) *(RecSys '10)*. Association for Computing Machinery, New York, NY, USA, 39–46. https://doi.org/10.1145/1864708.1864721

[4] Zhuyun Dai and Jamie Callan. 2019. Context-aware sentence/passage term importance estimation for first stage retrieval. *arXiv preprint arXiv:1910.10687* (2019).

[5] Zhuyun Dai and Jamie Callan. 2020. *Context-Aware Document Term Weighting for Ad-Hoc Search*. Association for Computing Machinery, New York, NY, USA, 1897–1907. https://doi.org/10.1145/3366423.3380258

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

[7] Yixing Fan, Jiafeng Guo, Yanyan Lan, Jun Xu, Chengxiang Zhai, and Xueqi Cheng. 2018. Modeling diverse relevance patterns in ad-hoc retrieval. In *The 41st international ACM SIGIR conference on research & development in information retrieval*. 375–384.

[8] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. *SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking*. Association for Computing Machinery, New York, NY, USA, 2288–2292. https://doi.org/10.1145/3404835.3463098

[9] Debasis Ganguly, Dwaipayan Roy, Mandar Mitra, and Gareth JF Jones. 2015. Word embedding based generalized language model for information retrieval. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 795–798.

[10] Luyu Gao and Jamie Callan. 2021. Condenser: a Pre-training Architecture for Dense Retrieval. *arXiv preprint arXiv:2104.08253* (2021).

[11] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 3030–3042. https://doi.org/10.18653/v1/2021.naacl-main.241

[12] Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, and Narayan Bhamidipati. 2015. Search retargeting using directed query embeddings. In *Proceedings of the 24th international conference on world wide web*. 37–38.

[13] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 55–64.

[14] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 3887–3896. https://proceedings.mlr.press/v119/guo20h.html

[15] Parth Gupta, Kalika Bali, Rafael E Banchs, Monojit Choudhury, and Paolo Rosso. 2014. Query expansion for mixed-script information retrieval. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. 677–686.

[16] Marti A Hearst. 1994. Multi-paragraph segmentation of expository text. *arXiv preprint cmp-lg/9406037* (1994).

[17] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 2333–2338.

[18] Gautier Izacard and Edouard Grave. 2020. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282* (2020).

[19] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[20] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[21] I.T. Jolliffe. 1986. *Principal Component Analysis*. Springer Verlag.

[22] Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia. 2015. *Learning spark: lightning-fast big data analysis*. " O'Reilly Media, Inc.".

[23] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906* (2020).

[24] Omar Khattab and Matei Zaharia. 2020. *ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT*. Association for Computing Machinery, New York, NY, USA, 39–48. https://doi.org/10.1145/3397271.3401075

[25] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent Retrieval for Weakly Supervised Open Domain Question Answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 6086–6096. https://doi.org/10.18653/v1/P19-1612

[26] Zhengdong Lu and Hang Li. 2013. A deep architecture for matching short texts. *Advances in neural information processing systems* 26 (2013), 1367–1375.

[27] Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. 2021. Sparse, Dense, and Attentional Representations for Text Retrieval. *Transactions of the Association for Computational Linguistics* 9 (04 2021), 329–345. https://doi.org/10.1162/tacl_a_00369 arXiv:https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00369/1924040/tacl_a_00369.pdf

[28] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. 2020. *Expansion via Prediction of Importance with Contextualization*. Association for Computing Machinery, New York, NY, USA, 1573–1576. https://doi.org/10.1145/3397271.3401262

[29] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (apr 2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[30] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. Learning passage impacts for inverted indexes. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1723–1727.

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013), 3111–3119.

[32] Bhaskar Mitra. 2015. Exploring session context using distributed representations of queries and reformulations. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 3–12.

[33] Bhaskar Mitra, Nick Craswell, et al. 2018. *An introduction to neural information retrieval*. Now Foundations and Trends.

[34] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).

[35] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. 2020. Document ranking with a pretrained sequence-to-sequence model. *arXiv preprint arXiv:2003.06713* (2020).

[36] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, and Xueqi Cheng. 2016. A study of matchpyramid models on ad-hoc retrieval. *arXiv preprint arXiv:1606.04648* (2016).

[37] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2016. Text matching as image recognition. *arXiv preprint arXiv:1602.06359* (2016).

[38] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. 2017. Deeprank: A new deep architecture for relevance ranking in information retrieval. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 257–266.

[39] Biswajit Paria, Chih-Kuan Yeh, Ian EH Yen, Ning Xu, Pradeep Ravikumar, and Barnabás Póczos. 2020. Minimizing flops to learn efficient sparse representations. *arXiv preprint arXiv:2004.05665* (2020).

[40] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 datasets. *arXiv preprint arXiv:1306.2597* (2013).

[41] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2020. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2010.08191* (2020).

[42] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[43] Stephen Robertson and Hugo Zaragoza. 2009. *The probabilistic relevance framework: BM25 and beyond.* Now Publishers Inc.

[44] Peter Schäuble. 1993. SPIDER: A Multiuser Information Retrieval System for Semistructured and Dynamic Data. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Pittsburgh, Pennsylvania, USA) *(SIGIR '93)*. Association for Computing Machinery, New York, NY, USA, 318–327. https://doi.org/10.1145/160688.160756

[45] Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval.* 373–382.

[46] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd international conference on world wide web.* 373–374.

[47] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). *Advances in neural information processing systems* 27 (2014).

[48] Zhiwen Tang and Grace Hui Yang. 2019. Deeptilebars: Visualizing term distribution for neural information retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 289–296.

[49] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605. http://www.jmlr.org/papers/v9/vandermaaten08a.html

[50] Shuai Wang, Shengyao Zhuang, and Guido Zuccon. 2021. *BERT-Based Dense Retrievers Require Interpolation with BM25 for Effective Passage Retrieval.* Association for Computing Machinery, New York, NY, USA, 317–324. https://doi.org/10.1145/3471158.3472233

[51] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[52] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR conference on research and development in information retrieval.* 55–64.

[53] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808* (2020).

[54] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-end open-domain question answering with bertserini. *arXiv preprint arXiv:1902.01718* (2019).

[55] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.

[56] Hamed Zamani, Mostafa Dehghani, W Bruce Croft, Erik Learned-Miller, and Jaap Kamps. 2018. 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.* 497–506.

[57] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. 2020. Repbert: Contextualized text embeddings for first-stage retrieval. *arXiv preprint arXiv:2006.15498* (2020).

[58] Tiancheng Zhao, Xiaopeng Lu, and Kyusong Lee. 2020. Sparta: Efficient open-domain question answering via sparse transformer matching retrieval. *arXiv preprint arXiv:2009.13013* (2020).

[59] Guoqing Zheng and Jamie Callan. 2015. Learning to reweight terms with distributed representations. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval.* 575–584.

[60] Mu Zhu. 2004. Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo* 2, 30 (2004), 6.

[61] Shengyao Zhuang and Guido Zuccon. 2021. *TILDE: Term Independent Likelihood MoDEl for Passage Re-Ranking.* Association for Computing Machinery, New York, NY, USA, 1483–1492. https://doi.org/10.1145/3404835.3462922