ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT

Omar Khattab
Stanford University
okhattab@stanford.edu

Matei Zaharia
Stanford University
matei@cs.stanford.edu

ABSTRACT

Recent progress in Natural Language Understanding (NLU) is driving fast-paced advances in Information Retrieval (IR), largely owed to fine-tuning deep language models (LMs) for document ranking. While remarkably effective, the ranking models based on these LMs increase computational cost by orders of magnitude over prior approaches, particularly as they must feed each query–document pair through a massive neural network to compute a single relevance score. To tackle this, we present ColBERT, a novel ranking model that adapts deep LMs (in particular, BERT) for efficient retrieval. ColBERT introduces a *late interaction* architecture that independently encodes the query and the document using BERT and then employs a cheap yet powerful interaction step that models their fine-grained similarity. By delaying and yet retaining this fine-granular interaction, ColBERT can leverage the expressiveness of deep LMs while simultaneously gaining the ability to pre-compute document representations offline, considerably speeding up query processing. Beyond reducing the cost of re-ranking the documents retrieved by a traditional model, ColBERT's *pruning-friendly* interaction mechanism enables leveraging vector-similarity indexes for end-to-end retrieval directly from a large document collection. We extensively evaluate ColBERT using two recent passage search datasets. Results show that ColBERT's effectiveness is competitive with existing BERT-based models (and outperforms every non-BERT baseline), while executing two orders-of-magnitude faster and requiring four orders-of-magnitude fewer FLOPs per query.

ACM Reference format:
Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, China, July 25–30, 2020 (SIGIR '20),* 10 pages.
DOI: 10.1145/3397271.3401075

1 INTRODUCTION

Over the past few years, the Information Retrieval (IR) community has witnessed the introduction of a host of neural ranking models, including DRMM [7], KNRM [4, 36], and Duet [20, 22]. In contrast to prior learning-to-rank methods that rely on hand-crafted features, these models employ embedding-based representations of queries and documents and directly model *local interactions* (i.e., fine-granular relationships) between their contents. Among them, a recent approach has emerged that *fine-tunes* deep pre-trained language models (LMs) like ELMo [29] and BERT [5] for estimating relevance. By computing deeply-contextualized semantic representations of query–document pairs, these LMs help bridge the pervasive vocabulary mismatch [21, 42] between documents and queries [30]. Indeed, in the span of just a few months, a number of ranking models based on BERT have achieved state-of-the-art results on various retrieval benchmarks [3, 18, 25, 39] and have been proprietarily adapted for deployment by Google[1] and Bing[2].

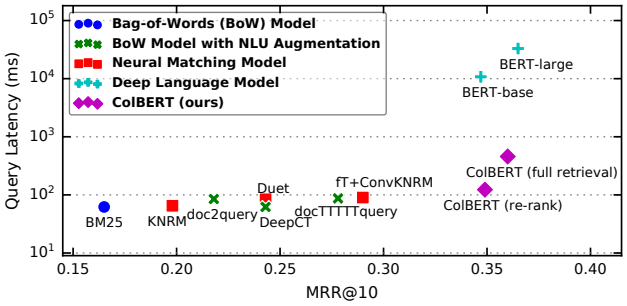

Figure 1: Effectiveness (MRR@10) versus Mean Query Latency (log-scale) for a number of representative ranking models on MS MARCO Ranking [24]. The figure also shows ColBERT. Neural re-rankers run on top of the official BM25 top-1000 results and use a Tesla V100 GPU. Methodology and detailed results are in §4.

However, the remarkable gains delivered by these LMs come at a steep increase in computational cost. Hofstätter *et al.* [9] and MacAvaney *et al.* [18] observe that BERT-based models in the literature are 100-1000× more computationally expensive than prior models—some of which are arguably *not* inexpensive to begin with [13]. This quality–cost tradeoff is summarized by Figure 1, which compares two BERT-based rankers [25, 27] against a representative set of ranking models. The figure uses MS MARCO Ranking [24], a recent collection of 9M passages and 1M queries from Bing's logs. It reports retrieval effectiveness (MRR@10) on the official validation set as well as average query latency (log-scale) using a high-end server that dedicates one Tesla V100 GPU per query for neural re-rankers. Following the *re-ranking* setup of MS MARCO, ColBERT (re-rank), the Neural Matching Models, and the Deep LMs re-rank the MS MARCO's official top-1000 documents per query.

SIGIR '20, Virtual Event, China

978-1-4503-8016-4/20/07…$15.00
DOI: 10.1145/3397271.3401075

[1] https://blog.google/products/search/search-language-understanding-bert/
[2] https://azure.microsoft.com/en-us/blog/bing-delivers-its-largest-improvement-in-search-experience-using-azure-gpus/

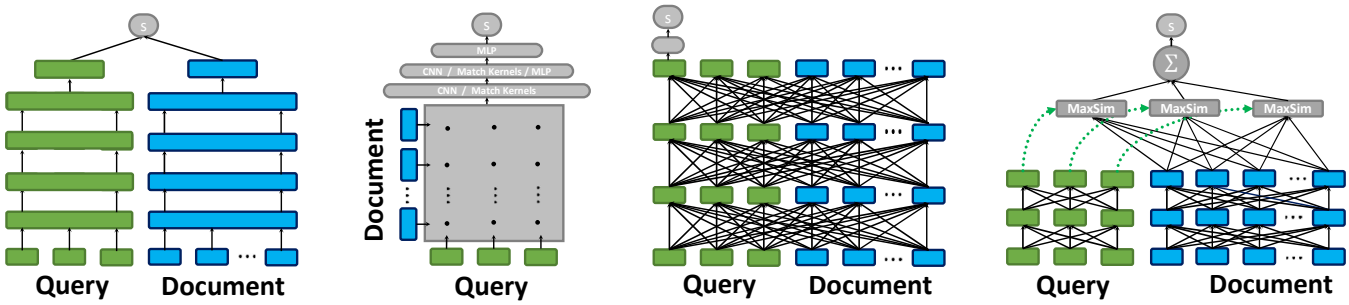

Figure 2: Schematic diagrams illustrating query–document matching paradigms in neural IR. The figure contrasts existing approaches (sub-figures (a), (b), and (c)) with the proposed late interaction paradigm (sub-figure (d)).

Other methods, including ColBERT (full retrieval), directly retrieve the top-1000 results from the entire collection.

As the figure shows, BERT considerably improves search precision, raising MRR@10 by almost 7% against the best previous methods; simultaneously, it increases latency by up to tens of thousands of milliseconds even with a high-end GPU. This poses a challenging tradeoff since raising query response times by as little as 100ms is known to impact user experience and even measurably diminish revenue [17]. To tackle this problem, recent work has started exploring using Natural Language Understanding (NLU) techniques to augment traditional retrieval models like BM25 [32]. For example, Nogueira *et al.* [26, 28] expand documents with NLU-generated queries before indexing with BM25 scores and Dai & Callan [2] replace BM25's term frequency with NLU-estimated term importance. Despite successfully reducing latency, these approaches generally reduce precision substantially relative to BERT.

To reconcile efficiency and contextualization in IR, we propose **ColBERT**, a ranking model based on **co**ntextualized **l**ate interaction over **BERT**. As the name suggests, ColBERT proposes a novel *late interaction* paradigm for estimating relevance between a query q and a document d. Under late interaction, q and d are separately encoded into two sets of contextual embeddings, and relevance is evaluated using cheap and *pruning-friendly* computations between both sets—that is, fast computations that enable ranking without exhaustively evaluating every possible candidate.

Figure 2 contrasts our proposed late interaction approach with existing neural matching paradigms. On the left, Figure 2 (a) illustrates *representation-focused* rankers, which independently compute an embedding for q and another for d and estimate relevance as a single similarity score between two vectors [12, 41]. Moving to the right, Figure 2 (b) visualizes typical *interaction-focused* rankers. Instead of summarizing q and d into individual embeddings, these rankers model word- and phrase-level relationships across q and d and match them using a deep neural network (e.g., with CNNs/MLPs [22] or kernels [36]). In the simplest case, they feed the neural network an *interaction matrix* that reflects the similiarity between every pair of words across q and d. Further right, Figure 2 (c) illustrates a more powerful interaction-based paradigm, which models the interactions between words *within* as well as *across* q and d at the same time, as in BERT's transformer architecture [25].

These increasingly expressive architectures are in tension. While interaction-based models (i.e., Figure 2 (b) and (c)) tend to be superior for IR tasks [8, 21], a representation-focused model—by isolating the computations among q and d—makes it possible to precompute document representations offline [41], greatly reducing the computational load per query. In this work, we observe that the fine-grained matching of interaction-based models and the precomputation of document representations of representation-based models can be combined by retaining yet judiciously *delaying* the query–document interaction. Figure 2 (d) illustrates an architecture that precisely does so. As illustrated, every query embedding interacts with all document embeddings via a MaxSim operator, which computes maximum similarity (e.g., cosine similarity), and the scalar outputs of these operators are summed across query terms. This paradigm allows ColBERT to exploit deep LM-based representations while shifting the cost of encoding documents offline and amortizing the cost of encoding the query once across all ranked documents. Additionally, it enables ColBERT to leverage vector-similarity search indexes (e.g., [1, 15]) to retrieve the top-k results directly from a large document collection, substantially improving *recall* over models that only re-rank the output of term-based retrieval.

As Figure 1 illustrates, ColBERT can serve queries in tens or few hundreds of milliseconds. For instance, when used for re-ranking as in "ColBERT (re-rank)", it delivers over 170× speedup (and requires 14,000× fewer FLOPs) relative to existing BERT-based models, while being more effective than every non-BERT baseline (§4.2 & 4.3). ColBERT's indexing—the only time it needs to feed documents through BERT—is also practical: it can index the MS MARCO collection of 9M passages in about 3 hours using a single server with four GPUs (§4.5), retaining its effectiveness with a space footprint of as little as few tens of GiBs. Our extensive ablation study (§4.4) shows that late interaction, its implementation via MaxSim operations, and crucial design choices within our BERT-based encoders are all essential to ColBERT's effectiveness.

Our main contributions are as follows.

(1) We propose *late interaction* (§3.1) as a paradigm for efficient and effective neural ranking.
(2) We present ColBERT (§3.2 & 3.3), a highly-effective model that employs novel BERT-based query and document encoders within the late interaction paradigm.

(3) We show how to leverage ColBERT both for re-ranking on top of a term-based retrieval model (§3.5) and for searching a full collection using vector similarity indexes (§3.6).
(4) We evaluate ColBERT on MS MARCO and TREC CAR, two recent passage search collections.

2 RELATED WORK

Neural Matching Models. Over the past few years, IR researchers have introduced numerous neural architectures for ranking. In this work, we compare against KNRM [4, 36], Duet [20, 22], ConvKNRM [4], and fastText+ConvKNRM [10]. KNRM proposes a differentiable kernel-pooling technique for extracting matching signals from an interaction matrix, while Duet combines signals from exact-match-based as well as embedding-based similarities for ranking. Introduced in 2018, ConvKNRM learns to match n-grams in the query and the document. Lastly, fastText+ConvKNRM (abbreviated fT+ConvKNRM) tackles the absence of rare words from typical word embeddings lists by adopting sub-word token embeddings.

In 2018, Zamani *et al.* [41] introduced SNRM, a representation-focused IR model that encodes each query and each document as a single, sparse high-dimensional vector of "latent terms". By producing a sparse-vector representation for each document, SNRM is able to use a traditional IR inverted index for representing documents, allowing fast end-to-end retrieval. Despite highly promising results and insights, SNRM's effectiveness is substantially outperformed by the state of the art on the datasets with which it was evaluated (e.g., see [18, 38]). While SNRM employs sparsity to allow using inverted indexes, we relax this assumption and compare a (dense) BERT-based representation-focused model against our late-interaction ColBERT in our ablation experiments in §4.4. For a detailed overview of existing neural ranking models, we refer the readers to two recent surveys of the literature [8, 21].

Language Model Pretraining for IR. Recent work in NLU emphasizes the importance pre-training language representation models in an unsupervised fashion before subsequently fine-tuning them on downstream tasks. A notable example is BERT [5], a bidirectional transformer-based language model whose fine-tuning advanced the state of the art on various NLU benchmarks. Nogueira *et al.* [25], MacAvaney *et al.* [18], and Dai & Callan [3] investigate incorporating such LMs (mainly BERT, but also ELMo [29]) on different ranking datasets. As illustrated in Figure 2 (c), the common approach (and the one adopted by Nogueira *et al.* on MS MARCO and TREC CAR) is to feed the query–document pair through BERT and use an MLP on top of BERT's [CLS] output token to produce a relevance score. Subsequent work by Nogueira *et al.* [27] introduced duoBERT, which fine-tunes BERT to compare the relevance of a *pair* of documents given a query. Relative to their single-document BERT, this gives duoBERT a 1% MRR@10 advantage on MS MARCO while increasing the cost by at least 1.4×.

BERT Optimizations. As discussed in §1, these LM-based rankers can be highly expensive in practice. While ongoing efforts in the NLU literature for distilling [14, 33], compressing [40], and pruning [19] BERT can be instrumental in narrowing this gap, they generally achieve significantly smaller speedups than our redesigned architecture for IR, due to their generic nature, and more aggressive optimizations often come at the cost of lower quality.

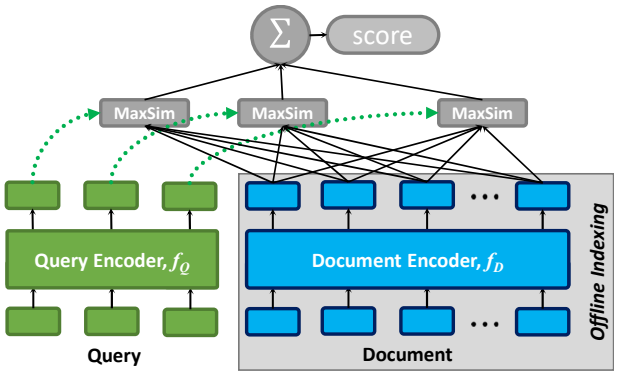

Figure 3: The general architecture of ColBERT given a query q and a document d.

Efficient NLU-based Models. Recently, a direction emerged that employs expensive NLU computation offline. This includes doc2query [28] and DeepCT [2]. The doc2query model expands each document with a pre-defined number of synthetic queries queries generated by a seq2seq transformer model that is trained to generate *queries* given a document. It then relies on a BM25 index for retrieval from the (expanded) documents. DeepCT uses BERT to produce the *term frequency* component of BM25 in a context-aware manner, essentially representing a feasible realization of the term-independence assumption with neural networks [23]. Lastly, docTTTTTquery [26] is identical to doc2query except that it fine-tunes a pre-trained model (namely, T5 [31]) for generating the predicted queries.

Concurrently with our drafting of this paper, Hofstätter *et al.* [11] published their Transformer-Kernel (TK) model. At a high level, TK improves the KNRM architecture described earlier: while KNRM employs kernel pooling on top of word-embedding-based interaction, TK uses a Transformer [34] component for contextually encoding queries and documents before kernel pooling. TK establishes a new state-of-the-art for non-BERT models on MS MARCO (Dev); however, the best non-ensemble MRR@10 it achieves is 31% while ColBERT reaches up to 36%. Moreover, due to indexing document representations offline and employing a MaxSim-based late interaction mechanism, ColBERT is much more scalable, enabling end-to-end retrieval which is not supported by TK.

3 COLBERT

ColBERT prescribes a simple framework for balancing the quality and cost of neural IR, particularly deep language models like BERT. As introduced earlier, delaying the query–document interaction can facilitate cheap neural re-ranking (i.e., through pre-computation) and even support practical end-to-end neural retrieval (i.e., through pruning via vector-similarity search). ColBERT addresses how to do so while still preserving the effectiveness of state-of-the-art models, which condition the bulk of their computations on the joint query–document pair.

Even though ColBERT's late-interaction framework can be applied to a wide variety of architectures (e.g., CNNs, RNNs, transformers, etc.), we choose to focus this work on bi-directional transformer-based encoders (i.e., BERT) owing to their state-of-the-art effectiveness yet very high computational cost.

3.1 Architecture

Figure 3 depicts the general architecture of ColBERT, which comprises: (a) a query encoder f_Q, (b) a document encoder f_D, and (c) the late interaction mechanism. Given a query q and document d, f_Q encodes q into a bag of fixed-size embeddings E_q while f_D encodes d into another bag E_d. Crucially, each embeddings in E_q and E_d is *contextualized* based on the other terms in q or d, respectively. We describe our BERT-based encoders in §3.2.

Using E_q and E_d, ColBERT computes the relevance score between q and d via late interaction, which we define as a summation of maximum similarity (MaxSim) operators. In particular, we find the maximum cosine similarity of each $v \in E_q$ with vectors in E_d, and combine the outputs via summation. Besides cosine, we also evaluate squared L2 distance as a measure of vector similarity. Intuitively, this interaction mechanism softly *searches* for each query term t_q—in a manner that reflects its context in the query—against the document's embeddings, quantifying the strength of the "match" via the largest similarity score between t_q and a document term t_d. Given these term scores, it then estimates the document relevance by summing the matching evidence across all query terms.

While more sophisticated matching is possible with other choices such as deep convolution and attention layers (i.e., as in typical interaction-focused models), a summation of maximum similarity computations has two distinctive characteristics. First, it stands out as a particularly cheap interaction mechanism, as we examine its FLOPs in §4.2. Second, and more importantly, it is amenable to highly-efficient pruning for top-k retrieval, as we evaluate in §4.3. This enables using vector-similarity algorithms for skipping documents without materializing the full interaction matrix or even considering each document in isolation. Other cheap choices (e.g., a summation of *average* similarity scores, instead of maximum) are possible; however, many are less amenable to pruning. In §4.4, we conduct an extensive ablation study that empirically verifies the advantage of our MaxSim-based late interaction against alternatives.

3.2 Query & Document Encoders

Prior to late interaction, ColBERT encodes each query or document into a bag of embeddings, employing BERT-based encoders. We share a single BERT model among our query and document encoders but distinguish input sequences that correspond to queries and documents by prepending a special token `[Q]` to queries and another token `[D]` to documents.

Query Encoder. Given a textual query q, we tokenize it into its BERT-based WordPiece [35] tokens $q_1 q_2 \ldots q_l$. We prepend the token `[Q]` to the query. We place this token right after BERT's sequence-start token `[CLS]`. If the query has fewer than a pre-defined number of tokens N_q, we pad it with BERT's special `[mask]` tokens up to length N_q (otherwise, we truncate it to the first N_q tokens). This padded sequence of input tokens is then passed into BERT's deep transformer architecture, which computes a contextualized representation of each token.

We denote the padding with masked tokens as **query augmentation**, a step that allows BERT to produce query-based embeddings at the positions corresponding to these masks. Query augmentation is intended to serve as a soft, differentiable mechanism for learning to expand queries with new terms or to re-weigh existing terms based on their importance for matching the query. As we show in §4.4, this operation is essential for ColBERT's effectiveness.

Given BERT's representation of each token, our encoder passes the contextualized output representations through a linear layer with no activations. This layer serves to control the dimension of ColBERT's embeddings, producing m-dimensional embeddings for the layer's output size m. As we discuss later in more detail, we typically fix m to be much smaller than BERT's fixed hidden dimension.

While ColBERT's embedding dimension has limited impact on the efficiency of query encoding, this step is crucial for controlling the space footprint of documents, as we show in §4.5. In addition, it can have a significant impact on query execution time, particularly the time taken for transferring the document representations onto the GPU from system memory (where they reside before processing a query). In fact, as we show in §4.2, gathering, stacking, and transferring the embeddings from CPU to GPU can be the most expensive step in re-ranking with ColBERT. Finally, the output embeddings are normalized so each has L2 norm equal to one. The result is that the dot-product of any two embeddings becomes equivalent to their cosine similarity, falling in the $[-1, 1]$ range.

Document Encoder. Our document encoder has a very similar architecture. We first segment a document d into its constituent tokens $d_1 d_2 \ldots d_m$, to which we prepend BERT's start token `[CLS]` followed by our special token `[D]` that indicates a document sequence. Unlike queries, we do not append `[mask]` tokens to documents. After passing this input sequence through BERT and the subsequent linear layer, the document encoder filters out the embeddings corresponding to punctuation symbols, determined via a pre-defined list. This filtering is meant to reduce the number of embeddings per document, as we hypothesize that (even contextualized) embeddings of punctuation are unnecessary for effectiveness.

In summary, given $q = q_0 q_1 \ldots q_l$ and $d = d_0 d_1 \ldots d_n$, we compute the bags of embeddings E_q and E_d in the following manner, where # refers to the `[mask]` tokens:

$$E_q := \texttt{Normalize}(\ \texttt{CNN}(\ \texttt{BERT}(\text{“}[Q] q_0 q_1 \ldots q_l \#\# \ldots \#\text{”})\ )\ ) \quad (1)$$

$$E_d := \texttt{Filter}(\ \texttt{Normalize}(\ \texttt{CNN}(\ \texttt{BERT}(\text{“}[D] d_0 d_1 \ldots d_n\text{”})\ )\ )\ ) \quad (2)$$

3.3 Late Interaction

Given the representation of a query q and a document d, the relevance score of d to q, denoted as $S_{q,d}$, is estimated via late interaction between their bags of contextualized embeddings. As mentioned before, this is conducted as a sum of maximum similarity computations, namely cosine similarity (implemented as dot-products due to the embedding normalization) or squared L2 distance.

$$S_{q,d} := \sum_{i \in [|E_q|]} \max_{j \in [|E_d|]} E_{q_i} \cdot E_{d_j}^T \quad (3)$$

ColBERT is differentiable end-to-end. We fine-tune the BERT encoders and train from scratch the additional parameters (i.e., the linear layer and the [Q] and [D] markers' embeddings) using the Adam [16] optimizer. Notice that our interaction mechanism has no trainable parameters. Given a triple $\langle q, d^+, d^- \rangle$ with query q, positive document d^+ and negative document d^-, ColBERT is used to produce a score for each document individually and is optimized via pairwise softmax cross-entropy loss over the computed scores of d^+ and d^-.

3.4 Offline Indexing: Computing & Storing Document Embeddings

By design, ColBERT isolates almost all of the computations between queries and documents, largely to enable pre-computing document representations offline. At a high level, our indexing procedure is straight-forward: we proceed over the documents in the collection in batches, running our document encoder f_D on each batch and storing the output embeddings per document. Although indexing a set of documents is an offline process, we incorporate a few simple optimizations for enhancing the throughput of indexing. As we show in §4.5, these optimizations can considerably reduce the offline cost of indexing.

To begin with, we exploit multiple GPUs, if available, for faster encoding of batches of documents in parallel. When batching, we pad all documents to the maximum length of a document *within* the batch.[3] To make capping the sequence length on a per-batch basis more effective, our indexer proceeds through documents in groups of B (e.g., $B = 100{,}000$) documents. It sorts these documents by length and then feeds batches of b (e.g., $b = 128$) documents of comparable length through our encoder. This length-based bucketing is sometimes refered to as a `BucketIterator` in some libraries (e.g., allenNLP). Lastly, while most computations occur on the GPU, we found that a non-trivial portion of the indexing time is spent on pre-processing the text sequences, primarily BERT's WordPiece tokenization. Exploiting that these operations are independent across documents in a batch, we parallelize the pre-processing across the available CPU cores.

Once the document representations are produced, they are saved to disk using 32-bit or 16-bit values to represent each dimension. As we describe in §3.5 and 3.6, these representations are either simply loaded from disk for ranking or are subsequently indexed for vector-similarity search, respectively.

3.5 Top-k Re-ranking with ColBERT

Recall that ColBERT can be used for re-ranking the output of another retrieval model, typically a term-based model, or directly for end-to-end retrieval from a document collection. In this section, we discuss how we use ColBERT for ranking a small set of k (e.g., $k = 1000$) documents given a query q. Since k is small, we rely on batch computations to exhaustively score each document (unlike our approach in §3.6). To begin with, our query serving subsystem loads the indexed documents representations into memory, representing each document as a matrix of embeddings.

Given a query q, we compute its bag of contextualized embeddings E_q (Equation 1) and, concurrently, gather the document representations into a 3-dimensional tensor D consisting of k document matrices. We pad the k documents to their maximum length to facilitate batched operations, and move the tensor D to the GPU's memory. On the GPU, we compute a batch dot-product of E_q and D, possibly over multiple mini-batches. The output materializes a 3-dimensional tensor that is a collection of cross-match matrices between q and each document. To compute the score of each document, we reduce its matrix across document terms via a max-pool (i.e., representing an exhaustive implementation of our MaxSim computation) and reduce across query terms via a summation. Finally, we sort the k documents by their total scores.

Relative to existing neural rankers (especially, but not exclusively, BERT-based ones), this computation is very cheap that, in fact, its cost is dominated by the cost of gathering and transferring the pre-computed embeddings. To illustrate, ranking k documents via typical BERT rankers requires feeding BERT k different inputs each of length $l = |q| + |d_i|$ for query q and documents d_i, where attention has quadratic cost in the length of the sequence. In contrast, ColBERT feeds BERT only a single, much shorter sequence of length $l = |q|$. Consequently, ColBERT is not only cheaper, it also scales much better with k as we examine in §4.2.

3.6 End-to-end Top-k Retrieval with ColBERT

As mentioned before, ColBERT's late-interaction operator is specifically designed to enable end-to-end retrieval from a large collection, largely to improve recall relative to term-based retrieval approaches. This section is concerned with cases where the number of documents to be ranked is too large for exhaustive evaluation of each possible candidate document, particularly when we are only interested in the highest scoring ones. Concretely, we focus here on retrieving the top-k results directly from a large document collection with N (e.g., $N = 10{,}000{,}000$) documents, where $k \ll N$.

To do so, we leverage the pruning-friendly nature of the MaxSim operations at the backbone of late interaction. Instead of applying MaxSim between one of the query embeddings and all of one document's embeddings, we can use fast vector-similarity data structures to efficiently conduct this search between the query embedding and *all* document embeddings across the full collection. For this, we employ an off-the-shelf library for large-scale vector-similarity search, namely `faiss` [15] from Facebook.[4] In particular, at the end of offline indexing (§3.4), we maintain a mapping from each embedding to its document of origin and then index all document embeddings into `faiss`.

Subsequently, when serving queries, we use a two-stage procedure to retrieve the top-k documents from the entire collection. Both stages rely on ColBERT's scoring: the first is an approximate stage aimed at filtering while the second is a refinement stage. For the first stage, we concurrently issue N_q vector-similarity queries (corresponding to each of the embeddings in E_q) onto our `faiss` index. This retrieves the top-k' (e.g., $k' = k/2$) matches for that vector

[3] The public BERT implementations we saw simply pad to a pre-defined length.

[4] https://github.com/facebookresearch/faiss

over all document embeddings. We map each of those to its document of origin, producing $N_q \times k'$ document IDs, only $K \leq N_q \times k'$ of which are unique. These K documents likely contain one or more embeddings that are highly similar to the query embeddings. For the second stage, we refine this set by exhaustively re-ranking *only* those K documents in the usual manner described in §3.5.

In our `faiss`-based implementation, we use an `IVFPQ` index ("inverted file with product quantization"). This index partitions the embedding space into P (e.g., $P = 1000$) cells based on k-means clustering and then assigns each document embedding to its nearest cell based on the selected vector-similarity metric. For serving queries, when searching for the top-k' matches for a single query embedding, only the nearest p (e.g., $p = 10$) partitions are searched. To improve memory efficiency, every embedding is divided into s (e.g., $s = 16$) sub-vectors, each represented using one byte. Moreover, the index conducts the similarity computations in this compressed domain, leading to cheaper computations and thus faster search.

4 EXPERIMENTAL EVALUATION

We now turn our attention to empirically testing ColBERT, addressing the following research questions.

RQ$_1$: In a typical re-ranking setup, how well can ColBERT bridge the existing gap (highlighted in §1) between highly-efficient and highly-effective neural models? (§4.2)

RQ$_2$: Beyond re-ranking, can ColBERT effectively support end-to-end retrieval directly from a large collection? (§4.3)

RQ$_3$: What does each component of ColBERT (e.g., late interaction, query augmentation) contribute to its quality? (§4.4)

RQ$_4$: What are ColBERT's indexing-related costs in terms of offline computation and memory overhead? (§4.5)

4.1 Methodology

4.1.1 Datasets & Metrics. Similar to related work [2, 27, 28], we conduct our experiments on the MS MARCO Ranking [24] (henceforth, MS MARCO) and TREC Complex Answer Retrieval (TREC-CAR) [6] datasets. Both of these recent datasets provide large training data of the scale that facilitates training and evaluating deep neural networks. We describe both in detail below.

MS MARCO. MS MARCO is a dataset (and a corresponding competition) introduced by Microsoft in 2016 for reading comprehension and adapted in 2018 for retrieval. It is a collection of 8.8M passages from Web pages, which were gathered from Bing's results to 1M real-world queries. Each query is associated with *sparse* relevance judgements of one (or very few) documents marked as relevant and no documents explicitly indicated as irrelevant. Per the official evaluation, we use MRR@10 to measure effectiveness.

We use three sets of queries for evaluation. The official development and evaluation sets contain roughly 7k queries. However, the relevance judgements of the evaluation set are held-out by Microsoft and effectiveness results can only be obtained by submitting to the competition's organizers. We submitted our main re-ranking ColBERT model for the results in §4.2. In addition, the collection includes roughly 55k queries (with labels) that are provided as additional validation data. We re-purpose a random sample of 5k queries among those (i.e., ones not in our development or training sets) as a "local" evaluation set. Along with the official development set, we use this held-out set for testing our models as well as baselines in §4.3. We do so to avoid submitting multiple variants of the same model at once, as the organizers discourage too many submissions by the same team.

TREC CAR. Introduced by Dietz [6] *et al.* in 2017, TREC CAR is a synthetic dataset based on Wikipedia that consists of about 29M passages. Similar to related work [25], we use the first four of five pre-defined folds for training and the fifth for validation. This amounts to roughly 3M queries generated by concatenating the title of a Wikipedia page with the heading of one of its sections. That section's passages are marked as relevant to the corresponding query. Our evaluation is conducted on the test set used in TREC 2017 CAR, which contains 2,254 queries.

4.1.2 Implementation. Our ColBERT models are implemented using Python 3 and PyTorch 1. We use the popular `transformers`[5] library for the pre-trained BERT model. Similar to [25], we fine-tune all ColBERT models with learning rate 3×10^{-6} with a batch size 32. We fix the number of embeddings per query at $N_q = 32$. We set our ColBERT embedding dimension m to be 128; §4.5 demonstrates ColBERT's robustness to a wide range of embedding dimensions.

For MS MARCO, we initialize the BERT components of the ColBERT query and document encoders using Google's official pre-trained $\text{BERT}_{\text{base}}$ model. Further, we train all models for 200k iterations. For TREC CAR, we follow related work [2, 25] and use a different pre-trained model to the official ones. To explain, the official BERT models were pre-trained on Wikipedia, which is the source of TREC CAR's training and test sets. To avoid leaking test data into train, Nogueira and Cho's [25] pre-train a randomly-initialized BERT model on the Wiki pages corresponding to training subset of TREC CAR. They release their $\text{BERT}_{\text{large}}$ pre-trained model, which we fine-tune for ColBERT's experiments on TREC CAR. Since fine-tuning this model is significantly slower than $\text{BERT}_{\text{base}}$, we train on TREC CAR for only 125k iterations.

In our re-ranking results, unless stated otherwise, we use 4 bytes per dimension in our embeddings and employ cosine as our vector-similarity function. For end-to-end ranking, we use (squared) L2 distance, as we found our `faiss` index was faster at L2-based retrieval. For our `faiss` index, we set the number of partitions to P =2,000, and search the nearest $p = 10$ to each query embedding to retrieve $k' = k = 1000$ document vectors per query embedding. We divide each embedding into $s = 16$ sub-vectors, each encoded using one byte. To represent the index used for the second stage of our end-to-end retrieval procedure, we use 16-bit values per dimension.

4.1.3 Hardware & Time Measurements. To evaluate the latency of neural re-ranking models in §4.2, we use a single Tesla V100 GPU that has 32 GiBs of memory on a server with two Intel Xeon Gold 6132 CPUs, each with 14 physical cores (24 hyperthreads), and 469 GiBs of RAM. For the mostly CPU-based retrieval experiments in §4.3 and the indexing experiments in §4.5, we use another server with the same CPU and system memory specifications but which has four Titan V GPUs attached, each with 12 GiBs of memory. Across all experiments, only one GPU is dedicated per query for

[5] https://github.com/huggingface/transformers

Method	MRR@10 (Dev)	MRR@10 (Eval)	Re-ranking Latency (ms)	FLOPs/query
BM25 (official)	16.7	16.5	-	-
KNRM	19.8	19.8	3	592M (0.085×)
Duet	24.3	24.5	22	159B (23×)
fastText+ConvKNRM	29.0	27.7	28	78B (11×)
$\text{BERT}_{\text{base}}$ [25]	34.7	-	10,700	97T (13,900×)
$\text{BERT}_{\text{base}}$ (our training)	36.0	-	10,700	97T (13,900×)
$\text{BERT}_{\text{large}}$ [25]	36.5	35.9	32,900	340T (48,600×)
ColBERT (over $\text{BERT}_{\text{base}}$)	34.9	34.9	61	7B (1×)

Table 1: "Re-ranking" results on MS MARCO. Each neural model re-ranks the official top-1000 results produced by BM25. Latency is reported for re-ranking only. To obtain the end-to-end latency in Figure 1, we add the BM25 latency from Table 2.

Method	MRR@10 (Dev)	MRR@10 (Local Eval)	Latency (ms)	Recall@50	Recall@200	Recall@1000
BM25 (official)	16.7	-	-	-	-	81.4
BM25 (Anserini)	18.7	19.5	62	59.2	73.8	85.7
doc2query	21.5	22.8	85	64.4	77.9	89.1
DeepCT	24.3	-	62 *(est.)*	69 [2]	82 [2]	91 [2]
docTTTTTquery	27.7	28.4	87	75.6	86.9	94.7
$\text{ColBERT}_{\text{L2}}$ (re-rank)	34.8	36.4	-	75.3	80.5	81.4
$\text{ColBERT}_{\text{L2}}$ (end-to-end)	36.0	36.7	458	82.9	92.3	96.8

Table 2: End-to-end retrieval results on MS MARCO. Each model retrieves the top-1000 documents per query *directly* from the entire 8.8M document collection.

retrieval (i.e., for methods with neural computations) but we use up to all four GPUs during indexing.

4.2 Quality–Cost Tradeoff: Top-k Re-ranking

In this section, we examine ColBERT's efficiency and effectiveness at re-ranking the top-k results extracted by a bag-of-words retrieval model, which is the most typical setting for testing and deploying neural ranking models. We begin with the MS MARCO dataset. We compare against KNRM, Duet, and fastText+ConvKNRM, a representative set of neural matching models that have been previously tested on MS MARCO. In addition, we compare against the natural adaptation of BERT for ranking by Nogueira and Cho [25], in particular, $\text{BERT}_{\text{base}}$ and its deeper counterpart $\text{BERT}_{\text{large}}$. We also report results for "$\text{BERT}_{\text{base}}$ (our training)", which is based on Nogueira and Cho's base model (including hyperparameters) but is trained with the same loss function as ColBERT (§3.3) for 200k iterations, allowing for a more direct comparison of the results.

We report the competition's official metric, namely MRR@10, on the validation set (Dev) and the evaluation set (Eval). We also report the re-ranking latency, which we measure using a single Tesla V100 GPU, and the FLOPs per query for each neural ranking model. For ColBERT, our reported latency subsumes the entire computation from gathering the document representations, moving them to the GPU, tokenizing then encoding the query, and applying late interaction to compute document scores. For the baselines, we measure the scoring computations on the GPU and exclude the CPU-based text preprocessing (similar to [9]). In principle, the baselines can pre-compute the majority of this preprocessing (e.g., document tokenization) offline and parallelize the rest across documents online, leaving only a negligible cost. We estimate the FLOPs per query of each model using the torchprofile[6] library.

We now proceed to study the results, which are reported in Table 1. To begin with, we notice the fast progress from KNRM in 2017 to the BERT-based models in 2019, manifesting itself in over 16% increase in MRR@10. As described in §1, the simultaneous increase in computational cost is difficult to miss. Judging by their rather monotonic pattern of increasingly larger cost and higher effectiveness, these results appear to paint a picture where expensive models are necessary for high-quality ranking.

In contrast with this trend, ColBERT (which employs late interaction over $\text{BERT}_{\text{base}}$) performs no worse than the original adaptation of $\text{BERT}_{\text{base}}$ for ranking by Nogueira and Cho [25, 27] and is only marginally less effective than $\text{BERT}_{\text{large}}$ and our training of $\text{BERT}_{\text{base}}$ (described above). While highly competitive in effectiveness, ColBERT is orders of magnitude cheaper than $\text{BERT}_{\text{base}}$, in particular, by over 170× in latency and 13,900× in FLOPs. This highlights the expressiveness of our proposed late interaction mechanism, particularly when coupled with a powerful pre-trained LM like BERT. While ColBERT's re-ranking latency is slightly higher than the non-BERT re-ranking models shown (i.e., by 10s of milliseconds), this difference is explained by the time it takes to gather, stack, and transfer the document embeddings to the GPU. In particular, the query encoding and interaction in ColBERT consume only 13 milliseconds of its total execution time. We note that ColBERT's latency and FLOPs can be considerably reduced by padding queries to a shorter length, using smaller vector dimensions (the MRR@10 of which is tested in §4.5), employing quantization of the document

[6] https://github.com/mit-han-lab/torchprofile

vectors, and storing the embeddings on GPU if sufficient memory exists. We leave these directions for future work.

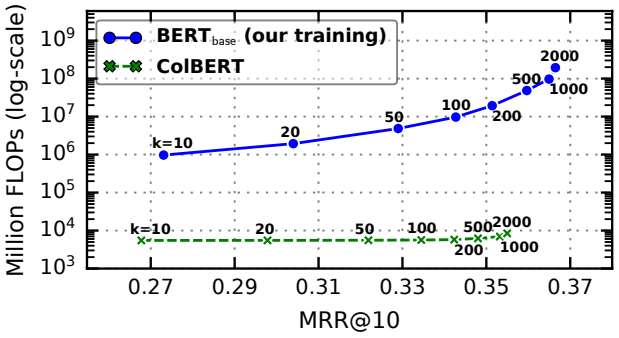

Figure 4: FLOPs (in millions) and MRR@10 as functions of the re-ranking depth k. Since the official BM25 ranking is not ordered, the initial top-k retrieval is conducted with Anserini's BM25.

Diving deeper into the quality–cost tradeoff between BERT and ColBERT, Figure 4 demonstrates the relationships between FLOPs and effectiveness (MRR@10) as a function of the re-ranking depth k when re-ranking the top-k results by BM25, comparing ColBERT and BERT$_{base}$ (our training). We conduct this experiment on MS MARCO (Dev). We note here that as the official top-1000 ranking does not provide the BM25 order (and also lacks documents beyond the top-1000 per query), the models in this experiment re-rank the Anserini [37] toolkit's BM25 output. Consequently, both MRR@10 values at $k = 1000$ are slightly higher from those reported in Table 1.

Studying the results in Figure 4, we notice that not only is ColBERT much cheaper than BERT for the same model size (i.e., 12-layer "base" transformer encoder), it also scales better with the number of ranked documents. In part, this is because ColBERT only needs to process the query once, irrespective of the number of documents evaluated. For instance, at $k = 10$, BERT requires nearly 180× more FLOPs than ColBERT; at $k = 1000$, BERT's overhead jumps to 13,900×. It then reaches 23,000× at $k = 2000$. In fact, our informal experimentation shows that this orders-of-magnitude gap in FLOPs makes it practical to run ColBERT entirely on the CPU, although CPU-based re-ranking lies outside our scope.

Method	MAP	MRR@10
BM25 (Anserini)	15.3	-
doc2query	18.1	-
DeepCT	24.6	33.2
BM25 + BERT$_{base}$	31.0	-
BM25 + BERT$_{large}$	33.5	-
BM25 + ColBERT	31.3	44.3

Table 3: Results on TREC CAR.

Having studied our results on MS MARCO, we now consider TREC CAR, whose official metric is MAP. Results are summarized in Table 3, which includes a number of important baselines (BM25, doc2query, and DeepCT) in addition to re-ranking baselines that have been tested on this dataset. These results directly mirror those with MS MARCO.

4.3 End-to-end Top-k Retrieval

Beyond cheap re-ranking, ColBERT is amenable to top-k retrieval directly from a full collection. Table 2 considers full retrieval, wherein each model retrieves the top-1000 documents directly from MS MARCO's 8.8M documents per query. In addition to MRR@10 and latency in milliseconds, the table reports Recall@50, Recall@200, and Recall@1000, important metrics for a full-retrieval model that essentially filters down a large collection on a per-query basis.

We compare against BM25, in particular MS MARCO's official BM25 ranking as well as a well-tuned baseline based on the Anserini toolkit.[7] While many other traditional models exist, we are not aware of any that substantially outperform Anserini's BM25 implementation (e.g., see RM3 in [28], LMDir in [2], or Microsoft's proprietary feature-based RankSVM on the leaderboard).

We also compare against doc2query, DeepCT, and docTTTTTquery. All three rely on a traditional bag-of-words model (primarily BM25) for retrieval. Crucially, however, they re-weigh the frequency of terms per document and/or expand the set of terms in each document before building the BM25 index. In particular, doc2query expands each document with a pre-defined number of synthetic queries generated by a seq2seq transformer model (which docTTTTTquery replaced with a pre-trained language model, T5 [31]). In contrast, DeepCT uses BERT to produce the term frequency component of BM25 in a context-aware manner.

For the latency of Anserini's BM25, doc2query, and docTTTTTquery, we use the authors' [26, 28] Anserini-based implementation. While this implementation supports multi-threading, it only utilizes parallelism across different queries. We thus report single-threaded latency for these models, noting that simply parallelizing their computation over *shards* of the index can substantially decrease their already-low latency. For DeepCT, we only estimate its latency using that of BM25 (as denoted by *(est.)* in the table), since DeepCT re-weighs BM25's term frequency without modifying the index otherwise.[8] As discussed in §4.1, we use ColBERT$_{L2}$ for end-to-end retrieval, which employs negative squared L2 distance as its vector-similarity function. For its latency, we measure the time for `faiss`-based candidate filtering and the subsequent re-ranking. In this experiment, `faiss` uses all available CPU cores.

Looking at Table 2, we first see Anserini's BM25 baseline at 18.7 MRR@10, noticing its very low latency as implemented in Anserini (which extends the well-known Lucene system), owing to both very cheap operations and decades of bag-of-words top-k retrieval optimizations. The three subsequent baselines, namely doc2query, DeepCT, and docTTTTTquery, each brings a decisive enhancement to effectiveness. These improvements come at negligible overheads in latency, since these baselines ultimately rely on BM25-based retrieval. The most effective among these three, docTTTTTquery, demonstrates a massive 9% gain over vanilla BM25 by fine-tuning the recent language model T5.

[7] http://anserini.io/

[8] In practice, a myriad of reasons could still cause DeepCT's latency to differ slightly from BM25's. For instance, the top-k pruning strategy employed, if any, could interact differently with a changed distribution of scores.

Shifting our attention to ColBERT's end-to-end retrieval effectiveness, we see its major gains in MRR@10 over all of these end-to-end models. In fact, using ColBERT in the end-to-end setup is superior in terms of MRR@10 to re-ranking with the same model due to the improved recall. Moving beyond MRR@10, we also see large gains in Recall@k for k equals to 50, 200, and 1000. For instance, its Recall@50 actually exceeds the official BM25's Recall@1000 and even all but docTTTTTquery's Recall@200, emphasizing the value of end-to-end retrieval (instead of just re-ranking) with ColBERT.

4.4 Ablation Studies

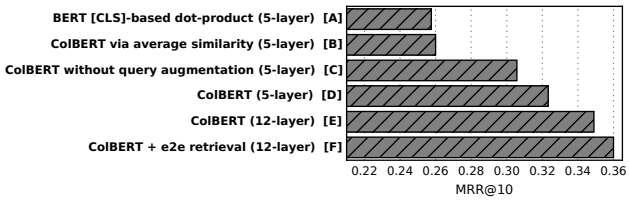

Figure 5: Ablation results on MS MARCO (Dev). Between brackets is the number of BERT layers used in each model.

The results from §4.2 indicate that ColBERT is highly effective despite the low cost and simplicity of its late interaction mechanism. To better understand the source of this effectiveness, we examine a number of important details in ColBERT's interaction and encoder architecture. For this ablation, we report MRR@10 on the validation set of MS MARCO in Figure 5, which shows our main *re-ranking* ColBERT model [E], with MRR@10 of 34.9%.

Due to the cost of training all models, we train a copy of our main model that retains only the first 5 layers of BERT out of 12 (i.e., model [D]) and similarly train all our ablation models for 200k iterations with five BERT layers. To begin with, we ask if the fine-granular *interaction* in late interaction is necessary. Model [A] tackles this question: it uses BERT to produce a single embedding vector for the query and another for the document, extracted from BERT's [CLS] contextualized embedding and expanded through a linear layer to dimension 4096 (which equals $N_q \times 128 = 32 \times 128$). Relevance is estimated as the inner product of the query's and the document's embeddings, which we found to perform better than cosine similarity for single-vector re-ranking. As the results show, this model is considerably less effective than ColBERT, reinforcing the importance of late interaction.

Subsequently, we ask if our MaxSim-based late interaction is better than other simple alternatives. We test a model [B] that replaces ColBERT's maximum similarity with *average* similarity. The results suggest the importance of individual terms in the query paying special attention to particular terms in the document. Similarly, the figure emphasizes the importance of our query augmentation mechanism: without query augmentation [C], ColBERT has a noticeably lower MRR@10. Lastly, we see the impact of end-to-end retrieval not only on recall but also on MRR@10. By retrieving directly from the full collection, ColBERT is able to retrieve to the top-10 documents missed entirely from BM25's top-1000.

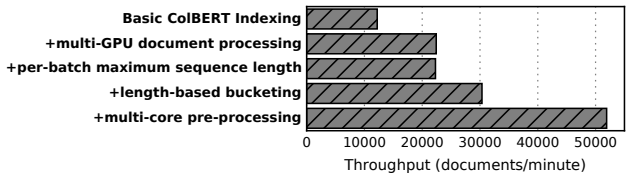

Figure 6: Effect of ColBERT's indexing optimizations on the offline indexing throughput.

4.5 Indexing Throughput & Footprint

Lastly, we examine the indexing throughput and space footprint of ColBERT. Figure 6 reports indexing throughput on MS MARCO documents with ColBERT and four other ablation settings, which individually enable optimizations described in §3.4 on top of basic batched indexing. Based on these throughputs, ColBERT can index MS MARCO in about three hours. Note that any BERT-based model must incur the computational cost of processing each document at least once. While ColBERT encodes each document with BERT exactly once, existing BERT-based rankers would repeat similar computations on possibly hundreds of documents for each query.

Setting	Dimension(m)	Bytes/Dim	Space(GiBs)	MRR@10
Re-rank Cosine	128	4	286	34.9
End-to-end L2	128	2	154	36.0
Re-rank L2	128	2	143	34.8
Re-rank Cosine	48	4	54	34.4
Re-rank Cosine	24	2	27	33.9

Table 4: Space Footprint vs MRR@10 (Dev) on MS MARCO.

Table 4 reports the space footprint of ColBERT under various settings as we reduce the embeddings dimension and/or the bytes per dimension. Interestingly, the most space-efficient setting, that is, re-ranking with cosine similarity with 24-dimensional vectors stored as 2-byte floats, is only 1% worse in MRR@10 than the most space-consuming one, while the former requires only 27 GiBs to represent the MS MARCO collection.

5 CONCLUSIONS

In this paper, we introduced ColBERT, a novel ranking model that employs *contextualized late interaction* over deep LMs (in particular, BERT) for efficient retrieval. By independently encoding queries and documents into fine-grained representations that interact via cheap and pruning-friendly computations, ColBERT can leverage the expressiveness of deep LMs while greatly speeding up query processing. In addition, doing so allows using ColBERT for end-to-end neural retrieval directly from a large document collection. Our results show that ColBERT is more than 170× faster and requires 14,000× fewer FLOPs/query than existing BERT-based models, all while only minimally impacting quality and while outperforming every non-BERT baseline.

Acknowledgments. OK was supported by the Eltoukhy Family Graduate Fellowship at the Stanford School of Engineering. This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Ant Financial, Facebook, Google, Infosys, NEC, and VMware—as well as Cisco, SAP, and the

NSF under CAREER grant CNS-1651570. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Firas Abuzaid, Geet Sethi, Peter Bailis, and Matei Zaharia. 2019. To Index or Not to Index: Optimizing Exact Maximum Inner Product Search. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1250–1261.

[2] Zhuyun Dai and Jamie Callan. 2019. Context-Aware Sentence/Passage Term Importance Estimation For First Stage Retrieval. *arXiv preprint arXiv:1910.10687* (2019).

[3] Zhuyun Dai and Jamie Callan. 2019. Deeper Text Understanding for IR with Contextual Neural Language Modeling. *arXiv preprint arXiv:1905.09217* (2019).

[4] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. 2018. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of the eleventh ACM international conference on web search and data mining*. 126–134.

[5] 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).

[6] Laura Dietz, Manisha Verma, Filip Radlinski, and Nick Craswell. 2017. TREC Complex Answer Retrieval Overview.. In *TREC*.

[7] 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*. ACM, 55–64.

[8] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W Bruce Croft, and Xueqi Cheng. 2019. A deep look into neural ranking models for information retrieval. *arXiv preprint arXiv:1903.06902* (2019).

[9] Sebastian Hofstätter and Allan Hanbury. 2019. Let's measure run time! Extending the IR replicability infrastructure to include performance aspects. *arXiv preprint arXiv:1907.04614* (2019).

[10] Sebastian Hofstätter, Navid Rekabsaz, Carsten Eickhoff, and Allan Hanbury. 2019. On the effect of low-frequency terms on neural-IR models. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1137–1140.

[11] Sebastian Hofstätter, Markus Zlabinger, and Allan Hanbury. 2019. TU Wien@ TREC Deep Learning'19–Simple Contextualization for Re-ranking. *arXiv preprint arXiv:1912.01385* (2019).

[12] 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.

[13] Shiyu Ji, Jinjin Shao, and Tao Yang. 2019. Efficient Interaction-based Neural Ranking with Locality Sensitive Hashing. In *The World Wide Web Conference*. ACM, 2858–2864.

[14] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351* (2019).

[15] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).

[16] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[17] Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Nils Pohlmann. 2013. Online controlled experiments at large scale. In *SIGKDD*.

[18] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. Cedr: Contextualized embeddings for document ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1101–1104.

[19] Paul Michel, Omer Levy, and Graham Neubig. 2019. Are Sixteen Heads Really Better than One?. In *Advances in Neural Information Processing Systems*. 14014–14024.

[20] Bhaskar Mitra and Nick Craswell. 2019. An Updated Duet Model for Passage Re-ranking. *arXiv preprint arXiv:1903.07666* (2019).

[21] Bhaskar Mitra, Nick Craswell, et al. 2018. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval* 13, 1 (2018), 1–126.

[22] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1291–1299.

[23] Bhaskar Mitra, Corby Rosset, David Hawking, Nick Craswell, Fernando Diaz, and Emine Yilmaz. 2019. Incorporating query term independence assumption for efficient retrieval and ranking using deep neural networks. *arXiv preprint arXiv:1907.03693* (2019).

[24] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human-Generated MAchine Reading COmprehension Dataset. (2016).

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

[26] Rodrigo Nogueira, Jimmy Lin, and AI Epistemic. 2019. From doc2query to docTTTTTquery. (2019).

[27] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019. Multi-Stage Document Ranking with BERT. *arXiv preprint arXiv:1910.14424* (2019).

[28] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document Expansion by Query Prediction. *arXiv preprint arXiv:1904.08375* (2019).

[29] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365* (2018).

[30] Yifan Qiao, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. 2019. Understanding the Behaviors of BERT in Ranking. *arXiv preprint arXiv:1904.07531* (2019).

[31] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683* (2019).

[32] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at TREC-3. *NIST Special Publication* (1995).

[33] Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from BERT into simple neural networks. *arXiv preprint arXiv:1903.12136* (2019).

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[35] 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).

[36] 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.

[37] Peilin Yang, Hui Fang, and Jimmy Lin. 2018. Anserini: Reproducible ranking baselines using Lucene. *Journal of Data and Information Quality (JDIQ)* 10, 4 (2018), 1–20.

[38] Wei Yang, Kuang Lu, Peilin Yang, and Jimmy Lin. 2019. Critically Examining the" Neural Hype" Weak Baselines and the Additivity of Effectiveness Gains from Neural Ranking Models. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1129–1132.

[39] Zeynep Akkalyoncu Yilmaz, Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Cross-domain modeling of sentence-level evidence for document retrieval. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 3481–3487.

[40] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188* (2019).

[41] 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*. ACM, 497–506.

[42] Le Zhao. 2012. *Modeling and solving term mismatch for full-text retrieval*. Ph.D. Dissertation. Carnegie Mellon University.