Fast Passage Re-ranking with Contextualized Exact Term Matching and Efficient Passage Expansion

Shengyao Zhuang
The University of Queensland
Brisbane, QLD, Australia
s.zhuang@uq.edu.au

Guido Zuccon
The University of Queensland
Brisbane, QLD, Australia
g.zuccon@uq.edu.au

ABSTRACT

BERT-based information retrieval models are expensive, in both time (query latency) and computational resources (energy, hardware cost), making many of these models impractical especially under resource constraints. The reliance on a query encoder that only performs tokenization and on the pre-processing of passage representations at indexing, has allowed the recently proposed TILDE method to overcome the high query latency issue typical of BERT-based models. This however is at the expense of a lower effectiveness compared to other BERT-based re-rankers and dense retrievers. In addition, the original TILDE method is characterised by indexes with a very high memory footprint, as it expands each passage into the size of the BERT vocabulary.

In this paper, we propose TILDEv2, a new model that stems from the original TILDE but that addresses its limitations. TILDEv2 relies on contextualized exact term matching with expanded passages. This requires to only store in the index the score of tokens that appear in the expanded passages (rather than all the vocabulary), thus producing indexes that are 99% smaller than those of TILDE. This matching mechanism also improves ranking effectiveness by 24%, without adding to the query latency. This makes TILDEv2 the state-of-the-art passage re-ranking method for CPU-only environments, capable of maintaining query latency below 100ms on commodity hardware.

One potential drawback of TILDEv2, compared to the original TILDE, is the extra passage expansion process required at indexing. This is an expensive process if performed using current passage expansion methods. However, we address this by adapting the original TILDE model to serve as a passage expansion method. Compared to current expansion methods, our proposed method reduces the passage expansion time by 98% with only less than 1% effectiveness loss on the MS MARCO passage ranking dataset (and even improvements on other datasets). We further show that our expansion approach generalises to other ranking methods that rely on expansion.

CCS CONCEPTS

• **Information systems → Retrieval models and ranking**; **Information retrieval query processing**.

KEYWORDS

Tokenizer-based query encoder, Contextualized exact term matching, Passage expansion

1 INTRODUCTION

Passage ranking is a core task for many web search and information retrieval applications. Traditional passage retrieval methods, such as BM25, rely on exact lexical matching signals and use frequency-based term importance estimation to calculate the matching score between queries and passages. This bag-of-words (BOW) mechanism however limits the capability of these methods of retrieving passages that are semantically relevant but have low or zero query term frequency: the well-known *vocabulary mismatch problem* [8].

Neural retrieval methods aim to address this limitation. Recent advances in neural rankers have seen the introduction of pre-trained deep language models (LMs) that are then fine-tuned on passage ranking tasks. These methods leverage the contextualized representation produced by deep LMs [31], such as BERT [6], to estimate the semantic matching score between passages and queries. For example, monoBERT [27] takes query-passage pairs as the input of BERT and the matching scores are estimated on the contextualized CLS token representation. Many empirical results obtained with monoBERT and its variants [5, 10, 28, 30] have demonstrated that these deep LMs based rankers achieve substantially better effectiveness than BOW methods on passage ranking tasks. However, this effectiveness gain does not come for free. The query latency cost of this type of neural rankers is several orders of magnitude larger than that of BOW methods [30]. In addition, GPUs are required, in place of more economical CPUs, not only for the offline training of the rankers, but also for the online (i.e. at query time) encoding of the contextualized representations. This hinders the practical adoption of these powerful rankers on small, GPU-free devices, such as mobile phones or embedded systems, or limits the number of passages that can be considered for re-ranking within a reasonable amount of time to guarantee real-time responses [13, 46].

To address the high query latency issue of these BERT-based re-rankers and to allow for the use of CPU-based systems in place of GPU-based ones, the recently proposed TILDE method [46] proposes to only use the BERT tokenizer to encode query representations at query time (online), while use BERT to pre-compute contextualized token importance scores over the BERT vocabulary for each passage in the index at indexing time (offline). Since no BERT inference is required at query time, TILDE achieves impressive re-ranking speed and GPUs are not required for inference. However, despite being very efficient, TILDE is much less effective than state-of-the-art BERT-based re-rankers and dense retrievers: TILDE trades off query representation quality, and thus effectiveness, for querying efficiency. On the other hand, because TILDE expands all passages in the collection to the size of the BERT vocabulary, it also has the drawback of a large index (and thus associated memory requirements): each passage, in fact, has a posting for every term in the BERT vocabulary.

In this paper, we propose changes to the TILDE method that tackle the current drawbacks of TILDE, while maintaining its efficiency. The result is a method, referred to as TILDEv2, that is highly efficient in both query latency (maintaining TILDE's original low latency) and index size (reducing the original TILDE index by up to 99%), and showcases effectiveness at par to state-of-the-art BERT-based methods (improving over TILDE by up to 24% on MS MARCO). This makes TILDEv2 to be production-ready for applications with limited computational power (e.g. no GPUs, start-ups). Specifically, we modify TILDE in the following aspects:

- **Exact Term Matching.** The query likelihood matching originally employed in TILDE, expands passages into the BERT vocabulary size, resulting in large indexes. To overcome this issue, we follow the recent paradigm of estimating relevance scores with contextualized exact term matching [9, 19, 25]. This allows the model to index tokens only present in the passage, thus reducing the index size. In addition to this, we replace the query likelihood loss function, with the *Noise-contrastive estimation* (NCE) loss [12] that allows to better leverage negative training samples. This loss function has been shown effective for recent dense retrievers [9, 15, 21, 33] and BERT-based re-rankers [10].
- **Passage Expansion.** To overcome the vocabulary mismatch problem that affects exact term matching methods, we use passage expansion to expand the original passage collection. Passages in the collection are expanded using deep LMs with a limited number of tokens. This requires TILDEv2 to only index a few extra tokens in addition to those in the original passages.

Compared to the original TILDE, the only drawback introduced by TILDEv2 is an extra passage expansion process to be executed at indexing time. This passage expansion process is typical of methods that exploit contextualized exact term matching [9, 19, 25]. For passage expansion, these previous methods rely on docT5query [29] to generate related tokens that are appended to the original passage. However, docT5query is a T5-based [34] sequence-to-sequence generative model, which is very expansive for inference: it requires 320 hours on a preemptible TPU[1] to expand the whole MS MARCO passage collection. This process becomes then expansive, and often infeasible, for large-scale information retrieval applications such as web search, or for small organisations such as start-ups. In TILDEv2 instead, we introduce a new way of performing this passage expansion process by replacing docT5query with the original TILDE model (which is then used for passage expansion, but not for retrieval). Empirical evaluation demonstrates that the proposed passage expansion method requires only a fraction of the time of the previous expansion method (45 times faster than docT5query), with only less than 1% effectiveness loss, if any. In addition, we also show that our passage expansion method is generally applicable to other retrieval methods such as uniCOIL [19].

2 RELATED WORK

Transformer-based [38] pre-trained deep LMs, such as BERT [6], have been shown to provide rich contextualized information, delivering high effectiveness in many NLP [1, 35, 43] and downstream retrieval and ranking tasks [20]. Nogueira and Cho were the first to directly fine-tuned BERT for passage ranking, achieving a large performance leap over BOW methods [27]. Gao et al. further showed that BERT trained with localized NCE loss achieves better effectiveness for document re-ranking [10]. Beyond BERT, other generative pre-trained LMs such as GPT [1], BART [18] and T5 [35] also have shown promising results in text re-ranking with query likelihood [7, 17, 45].

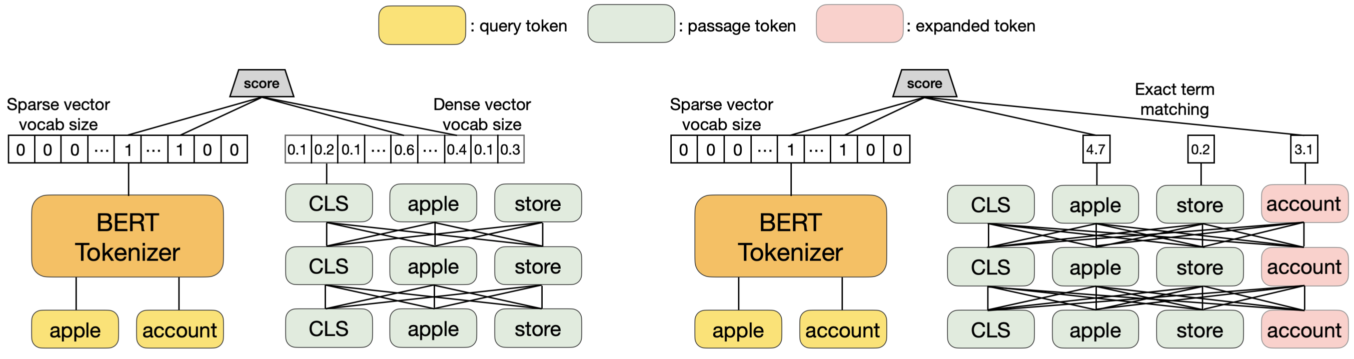
The biggest shortage of BERT-based rankers is their high query latency: several expensive BERT inferences are needed for ranking passages from a candidate pool because at query time BERT requires as input individual query-passage pairs. This makes BERT-based rankers feasible only for re-ranking, and with small rank cut-offs.

Several works have attempted to address the high query latency of BERT-based re-rankers. One direction is modifying the BERT encoder. Hofstätter et al. [13] proposed the Transformer Kernel (TK): Instead of using the full-size BERT, TK uses a limited number of transformer layers to pre-compute contextual representations of the passages' tokens, requiring only a small amount of computation to produce the query representation at query time. The encoded query tokens and document tokens are then used to compute a feature matrix with a kernel operation. MacAvaney et al. [24] proposed EPIC, where query and passages are encoded independently: this allows to pre-compute the passage representation at indexing time. The original TILDE method [46], which we build on top of in this paper, takes this idea to the extreme: TILDE only uses the BERT tokenizer to encode the query representation at query time. Since no transformer layer is involved at query time, TILDE can be ran efficiently on a CPU-only environment. A drawback of both TILDE and EPIC is the large amount of memory required to store the passage representations (index).

An alternative direction is reconsidering the representation. This is the approach followed by BERT-based dense retrievers (DRs) [11, 15, 16, 33, 41, 44]. Similar to EPIC, DRs also compute passage representations at indexing time, rather than at querying, requiring then at query time only one BERT inference for encoding the query. Passages are ranked using the similarity between the query representation and the passage representations: when powerful GPUs are used, DRs can achieve similar query latency as traditional BOW methods based on inverted index (e.g., BM25).

A third direction is represented by methods based on the supervised construction of an inverted index and the use of contextualized exact term matching. DeepCT [4], for example, uses BERT to estimate the contextualized term importance weight for terms in a passage. Then, the learnt term weights are stored in a standard inverted index. Hence, retrieval can be performed using any BOW exact term matching method, such as BM25. DeepCT's effectiveness, however, is limited by the exact term matching mechanism as it can only estimate weights for terms that appear in the passage. This drawback is solved in the recently proposed DeepImpact [25]: DeepImpact first uses docT5query [29] to generate terms to expand the original passage. Contextualized term weights are then assigned to

[1] Estimated based on generating 40 "expansion queries" per passage.



(a) TILDE

(b) TILDEv2

Figure 1: Model architectures. Left: the original TILDE model. Right: Our TILDEv2 model.

these new terms, and matching can be executed. COIL [9] uses a different contextualized exact term matching architecture. Instead of storing scalar term weights, COIL learns and stores contextualized token representations into the inverted lists, which are then used to perform exact term matching at query time. A subsequent work by Lin and Ma introduced a variation of COIL, named uniCOIL [19], which combines the idea of passage expansion from DeepImpact with the COIL architecture, so as to learn scalar weights that can be stored in the standard inverted index. Our proposed TILDEv2 builds upon this prior work by introducing contextualized exact term matching coupled with passage expansion into TILDE for the passage re-ranking task.

3 METHOD

The proposed TILDEv2 addresses the limitations of the current TILDE model by integrating and expanding upon a number of recent advances in BERT-based rankers. Next, we discuss the key components of our TILDEv2 and its similarities and differences compared to the original TILDE.

3.1 Tokenizer-based query encoder

Unlike other BERT-based re-rankers and DRs, at query time TILDE only uses the BERT tokenizer to encode the query into a sparse vector representation: this represents one of the main innovations of TILDE. This tokenizer-based query encoder is very simple but also very efficient: it is a lookup table without any model parameters, thus eliminating the need for costly inferences that require GPU computation. In our experiments, it takes only less than 1 ms to encode the query with TILDE.

In order to achieve maximum re-ranking speed, our TILDEv2 inherits the simple query encoder which is at the basis of TILDE. As Figure 1 illustrates, both TILDE and TILDEv2 use the BERT tokenizer to encode the query. The encoded query representation is a sparse vector of dimension equal to the BERT vocabulary size, in which each element in the vector is the frequency of that token in the query. For instance, for the query 'apple account' (Figure 1), the token *'apple'* appears once in the query and its token id in the BERT vocabulary is 6207; hence, the value of the 6207th element in the query vector is 1. These non-zero elements are used to compute the matching score between the query and a passage (see next). Since queries in web search are often short [36], the vectors are sparse (only few elements are non-zero), and the matching operation only requires to consider the non-zero elements, being thus very efficient.

3.2 Re-ranking with contextualized exact term matching

The biggest difference between TILDE and TILDEv2 is the query-passage relevance matching mechanism. TILDE follows the query likelihood paradigm [32], where the probability of a passage being relevant to a query is estimated by the likelihood of generating the query text given the passage text. As Figure 1(a) illustrates, TILDE outputs the query token probabilities over the BERT vocabulary (presented as a vocabulary size dense vector) using a projection layer on top of the BERT's [CLS] token. Then, it assumes the query tokens are independent and it computes the query likelihood by summing the log probabilities of the query tokens in the dense vector. This relevance matching mechanism forces TILDE to compute and store query token likelihoods for all tokens in the BERT vocabulary at indexing time, resulting in a very large index. As an example, the TILDE index size for the MS MARCO passage collection is more than 500 GiB, compared to the standard Lucene index which is only 2.8GiB: TILDE index size is then often not practical, especially for the systems it targets (those with low computational power, i.e. no GPUs, like mobiles and embedded systems, which are often also characterised by limited disk space).

To overcome this issue, TILDEv2 abandons the use of the query likelihood matching mechanism. Inspired by recent advances in contextualized term weighting, instead, in TILDEv2 we use BERT to output a scalar importance weight for all tokens in the passage and perform exact term matching between query and passage tokens. Figure 1(b) illustrates the matching mechanism used in TILDEv2.

Specifically, we use BERT to output the contextualized token embeddings for all passage tokens, and then we feed them into a projection layer to downsize the embeddings to dimension 1:

$$v_i^p = ReLU\left(\boldsymbol{W}_{proj}^{1\times n}\boldsymbol{BERT}(p_i) + b\right) \tag{1}$$

where p_i is the i-th token in the passage and $\boldsymbol{W}_{proj}^{1\times n}$ is the projection matrix that maps the BERT's n dimensional word embedding of

Algorithm 1 Passage expansion with the original TILDE model.

```
Input: $TILDE$, stopword list $stop\_list$, passage $p$, Threshold $m$
$T = TILDE(p)$              // get token likelihood distribution $T$.
$sort(T)$            // sort by descending order of the likelihoods.
for $t \in T[: m]$ do
    if $t \notin p$ AND $t \notin stop\_list$ then
        $p.append(t)$
    end if
end for
```

p_i into a scalar; b is the learnable bias parameter of the projection layer. The scalar output by the projection layer is then passed to a ReLU operation to obtain the final contextualized term importance weight v_i^p for p_i. The ReLU operation masks out all the negative scalars to zero, thus forcing all term weights to be positive. This operation has also been used in previous work [19, 25].

We now define the exact term matching scoring function for computing the passage relevance scores given all the query tokens' frequencies encoded by the BERT tokenizer:

$$S(q, p) = \sum_{q_i \in q} \max_{q_i = p_j} \left(c(q_i) \times v_j^p \right) \tag{2}$$

where $c(q_i)$ is the count of the i-th unique query token given by the BERT tokenizer query encoder described in section 3.1. The relevance score of each query-passage pair is the sum of the contextualized term weights provided by each query token that appears in the passage. If a query token appears more than once in a passage, then its score is equal to the highest contextualized term weight ($max(.)$) for that token in the passage. With this matching mechanism, TILDEv2 only needs to pre-compute and store the tokens that appear in the passage along with the max contextualized term weight. Compared to TILDE, which needs to store the likelihood value of all tokens in the BERT vocabulary, the index size of TILDEv2 is therefore two orders of magnitude smaller (we provide more details on this aspect in section 5.4).

Finally, following previous work [9, 10, 15, 21, 33], we train our TILDEv2 with the NCE loss function [12] with the negative passage set l created by randomly sampling passages from the top 1,000 results obtained by BM25 for the query:

$$\mathcal{L} = -\log \frac{\exp(S(q, p^+))}{\exp(S(q, p^+)) + \sum_{p^- \in l} \exp(S(q, p^-))} \tag{3}$$

More training details are discussed in section 4.3

3.3 Passage expansion

Similar to traditional BOW methods, TILDEv2 can only match those query terms that appear in the passage; thus, if no other matching mechanism is put in place, its effectiveness is limited by the vocabulary mismatch problem. In order to reduce the impact of this problem, following recent advances in exact term matching models [19, 25], we use the technique of passage expansion to expand each passage in the collection at indexing time. Passage expansion appends semantically related and potentially relevant terms at the end of a passage, in the bid to increase the likelihood of retrieving the passage for queries containing those expanded terms and for which the passage is relevant.

Existing approaches use docT5query [29] to perform passage expansion [19, 25]. docT5query is a T5-based [35] sequence-to-sequence generative language model, which can only generate one token at a time. Thus, multiple inferences from docT5query are needed to obtain several tokens for passage expansion. Provided that T5 is a large transformer model, passage expansion with docT5query requires a large amount of computational resources. According to the statistic provided by the docT5query authors [29], sampling 40 queries[2] per passage for each of the ≈8.8 million passages in the MS MARCO collection requires ≈320 hours on a single TPU, and ≈5,000 hours are required for expanding the MS MARCO v2's 138.3 million passages[3]. For large-scale information retrieval applications such as web search, this is a very expensive process.

For TILDEv2 we take a different approach to passage expansion: we adapt the original TILDE method to perform the passage expansion. This idea is based on the observation that TILDE actually outputs a query token likelihood distribution over the vocabulary. This distribution can be considered as an estimation of term importance given the passage context. In addition, unlike docT5query, the original TILDE model assumes query terms are independent, so that it only needs a single inference step to get the distribution output for all tokens.

The main algorithm that exploits the original TILDE model for passage expansion is described in Algorithm 1. For a given passage p, we use TILDE to get the likelihood distribution T. Note, each element in T is a token-likelihood pair. We then sort all the tokens in T in descending order according to their corresponding likelihoods. Next, for each top-m token t in the sorted list T, if t is not in the original passage p and it is not in a pre-defined stopword list, we then append it to the original passage. We do this expansion for all passages in the collection. In our experiments, passage expansion with TILDE can expand the whole MS MARCO passage collection in 7.3 hours on a single GPU. More details are provided in section 5.4. Note, the use of TILDE for passage expansion was not present in the original work of Zhuang and Zuccon [46], and thus is a novel contribution of our work.

4 EXPERIMENTAL SETTINGS

Next, we describe the experiment settings we use to investigate the performance of our TILDEv2, to compare it to current, relevant methods in the literature, and to answer the research questions:

- **RQ1**: Which matching mechanism is more effective and more efficient: the query likelihood matching used in the original TILDE, or the contextualized exact term matching used in TILDEv2?
- **RQ2**: How does TILDEv2 compare to current methods for passage ranking in terms of effectiveness and efficiency?
- **RQ3**: How does the effectiveness-efficiency trade-off of TILDEv2, allowed by the setting of the rank cut-off parameter, compare to that of the BERT re-ranker?
- **RQ4**: How effective and how efficient is our passage expansion based on TILDE, compared to the current state-of-the-art method (docT5query)?

[2] This is the common number used in previous work.

[3] https://microsoft.github.io/msmarco/TREC-Deep-Learning.html

4.1 Datasets and evaluation metrics

We experiment with three commonly used publicly available large-scale passage ranking datasets: MS MARCO [26], TREC Deep Learning 2019 [3] and TREC Deep Learning 2020 [2] (DL2019, DL2020). These datasets share the same set of passages, the MS MARCO passage corpus[4], which consists of approximately 8.8 million passages (average length: 73.1 terms) crawled by the Bing search engine, but differ in terms of queries (and relevance assessments).

The MS MARCO dataset provides approximately 1 million queries. Queries are split into train, dev, and eval sets. Each query is associated with shallowly annotated judgments, where on average only one passage is marked as relevant and no irrelevant passages are identified. Following standard practice from the dataset instructions, we use queries along with their relevance judgments in the train set to train our model; while we evaluate the model on the dev set. Evaluation is performed with respect to the official evaluation measure MRR@10.

Unlike MS MARCO, the TREC DL2019 and DL2020 datasets provide small query sets (43 for DL 2019, 54 for DL2020), with deep judgments on a four-point scale (i.e. graded). Following TREC DL practice, we use nDCG@10 and MAP as evaluation measures, so we can more easily compare our method to past and future work.

For all evaluation measures, differences between methods are tested for statistical significance using a paired two-tailed t-test with Bonferroni correction.

Along with effectiveness, we also report the query latency achieved within a CPU environment and within a GPU environment. For this, we randomly sampled 200 queries from the dev queries of MS MARCO and issued them one by one to each model, and report the average query latency measured. For the CPU environment, we conducted experiments on a consumer-grade 3.2GHz 6-core Intel Core i7 CPU with 64GB DDR4 memory (2018 Apple Mac Mini). For the GPU environment, we used an NVIDIA Tesla V100 16G GPU.

4.2 Baselines

BOW retrievers: We consider the traditional BOW approach BM25 and the commonly used strong BOW baseline docT5query [29]. docT5query uses BM25 for ranking, but it performs passage expansion using the T5 deep LM as a collection pre-processing step. We also use these two methods as first stage retrievers, on top of which we apply TILDEv2 (and other deep LM re-rankers). For both methods, we use the Anserini [42] implementation with its default parameter setting.

Contextualized exact match: These methods use deep LMs to assign contextualized term weight and perform exact term matching with an inverted index. We use the recent DeepImpact [25] and uniCOIL [19] methods. At query time, Deepimpact uses the BERT tokenizer to "clean" the query tokens. uniCOIL, instead, performs a BERT inference to compute the contextualized term weights for the query tokens. For uniCOIL, we use the GitHub code[5] provided by the authors to train the model and use Anserini to index the collection. For DeepImpact, we directly use the Anserini implementation.

Dense Retrievers: We also consider dense retrievers, and specifically RepBERT [44] and ANCE [41], as means of very efficient neural methods for retrieval. RepBERT uses BERT to encode the query and the passages and is trained with BM25 hard negatives. ANCE uses RoBERTa [22], a more robust version of BERT, as the encoder. For both methods, we use the model checkpoints provided by the authors and the FAISS [14] Python toolkit to build a dense vector index.

BERT-based re-rankers: We consider two types of BERT-based re-rankers. EPIC [24] is a fast re-ranker that uses BERT to pre-encode passages and at query time it performs a single BERT inference to encode the query. The re-ranking is then performed using similarity matching between the query and passage representations. For EPIC, we use the implementation available in the OpenNIR toolkit [23]. The BERT-base/BERT-large re-ranker [27] (also known as monoBERT) are strong BERT-based re-ranker baselines. This approach requires that both the query and the passage are jointly provided at query time as inputs to BERT; the output is the matching score. The BERT-large re-ranker differs from the BERT-base from the (larger) number of parameters. We use the model checkpoints made publicly available by Huggingface's model hub [40] and provided by the NBoost IR platform [37].

Tokenizer-based re-rankers: At query time, both the original TILDE [46] and TILDEv2 only use the BERT tokenizer to encode the query. The key difference between the two methods is their matching mechanism. By comparing our TILDEv2 with the original TILDE we can directly measure the impact of our additions in TILDEv2 with respect to effectiveness, query latency and index size. For the original TILDE model, we use the model checkpoint made available by the authors on the Huggingface model hub.

4.3 TILDEv2 implementation and training

We implemented TILDEv2 using Pytorch and the Huggingface transformers library [40]. We used the bert-base-uncased, which has 110M parameters, as BERT model in TILDEv2. The contextualized word embeddings output by BERT have a dimension of 768; they are then projected to scalars (dimension of 1) by a projection layer. As in the original TILDE, we filtered out the same set of stopwords when encoding queries with the BERT tokenizer.

We trained TILDEv2 for 204,000 update steps with the AdamW optimizer. The learning rate was set to 3e-6 with a linear warm-up schedule. Following previous work [9, 10, 19] we used both in-batch negatives and hard negatives for the NCE loss function; these were sampled from the BM25 top 1,000 results. More specifically, for each query we sampled 7 hard negatives from BM25 and one positive. We set the batch size to 8, resulting in a total of 63 negatives per query (7 hard negatives + 56 in-batch negatives). The model was trained on a single NVIDIA Tesla V100 16G GPU; training took approximately 10 hours. We used the Python built-in dictionary class (Hashtable) to implement the index, which is used at re-ranking to search the stored contextualized term weights for the tokens in the passages.

[4] https://github.com/microsoft/MSMARCO-Passage-Ranking

[5] https://github.com/luyug/COIL/tree/main/uniCOIL

Table 1: Effectiveness and efficiency of TILDEv2 and baselines. Statistical significant differences ($p < 0.05$) in effectiveness between TILDEv2 and the baselines is reported with subscripts. The average query latency is measured in milliseconds. The latency of the re-ranking methods includes that of the first stage retrieval. The BOW models, TILDE and TILDEv2 do not run on GPU; while executing the BERT-base/large re-rankers in a CPU environment is infeasible.

	MS MARCO	TREC DL2019		TREC DL2020		Latency (ms)	
Method	**MRR@10**	**nDCG@10**	**MAP**	**nDCG@10**	**MAP**	**GPU**	**CPU**
(i) BOW retriever							
a) BM25	0.187	0.506	0.377	0.480	0.286	n.a.	70
b) docT5query (d2q)	0.277	0.648	0.463	0.616	0.408	n.a.	75
(i) Contextualized exact match							
c) DeepImpact	0.326	0.696	0.472	0.650	0.426	n.a.	235
d) uniCOIL	0.351	0.693	0.476	0.666	0.445	240	276
(ii) Dense Retrievers							
e) RepBERT	0.304	0.610	0.331	0.662	0.370	152	1,633
f) ANCE	0.330	0.645	0.361	0.642	0.405	152	1,633
(iii) Bert-based Re-rankers							
g) EPIC+BM25-top100	0.274	0.608	0.411	0.573	0.349	96	113
h) EPIC+d2q-top15	0.303	0.691	0.473	0.628	0.406	101	116
i) BERT-base+BM25-top1000	0.350	0.706	0.483	0.686	0.454	3, 815	n.a.
j) BERT-large+BM25-top1000	0.370	0.738	0.506	0.705	0.493	11, 594	n.a.
(iv) Tokenizer-based Re-rankers							
k) TILDE+BM25-top1000	0.269	0.579	0.406	0.620	0.406	n.a.	76.6
l) TILDE+d2q-top10	0.285	0.650	0.467	0.624	0.417	n.a.	75.3
TILDEv2 (ours)+BM25-top1000	$0.333^{abeghjkl}$	0.676^{a}	0.448^{ae}	0.659^{ag}	0.433^{agj}	n.a.	80.8
TILDEv2 (ours)+d2q-top100	$0.341^{abeghjkl}$	0.703^{agk}	0.498^{abefgk}	0.669^{ag}	0.449^{ag}	n.a.	76.4

5 RESULTS

5.1 RQ1: Effectiveness and efficiency of TILDEv2 vs. TILDE

Table 1 reports the results obtained with respect to effectiveness (MRR@10, nDCG@10, MAP) and efficiency (query latency) across the three studied datasets. For the baseline re-rankers, we use the best re-ranking cut-off reported in the respective original papers. For TILDEv2, we tune the cut-off on a subset of dev queries from MS MARCO, and use docT5query as the passage expansion method for fair comparison with DeepImpact and uniCOIL, which also use docT5query for passage expansion.

We first start by comparing the original TILDE and our TILDEv2 (block iv in Table 1), thus answering RQ1: which matching mechanism is more effective. Both methods in fact only use the BERT tokenizer to encode the query, but rely on different matching mechanisms. The results indicate that the contextualized exact term matching employed in TILDEv2 leads to higher effectiveness than the reliance on query likelihood of the original TILDE; and this is regardless of the first stage of retrieval (BM25 vs docT5query), though docT5query leads to better results than BM25. These improvements are especially significant for the MS MARCO dataset, with 24% when re-ranking BM25 and 20% when re-ranking docT5query.

When considering query latency, we observe that both methods only require less than 100 ms to generate the final ranking. In addition, TILDEv2 is only 4.2 ms slower than the original TILDE when re-ranking BM25, and 1.1 ms slower when re-ranking docT5query. We note that when tuning the rank cut-off on a random sample of dev queries (see above), TILDEv2 was found to be most effective when re-ranking the top 100 passages, while the original TILDE used the top 10 passages: that is, although TILDEv2 re-ranking takes a handful of milliseconds more than TILDE, it does re-rank more passages. Furthermore, an additional reduction in runtime could be achieved by optimizing the index structure used by TILDEv2.

In summary, in answer to RQ1, we conclude that the contextualized exact term matching of TILDEv2 leads to better effectiveness (significantly on MS MARCO) than the query likelihood matching used by the original TILDE, at no or minor expense of query latency.

5.2 RQ2: Effectiveness and efficiency of TILDEv2 vs. baselines

We now compare TILDEv2 with other baselines in terms of effectiveness (RQ2). For MS MARCO, TILDEv2 outperforms, significantly, most comparison methods, although it is outperformed by uniCOIL and the BERT-base/large re-rankers. Differences between TILDEv2 and uniCOIL and BERT-base are not statistically significant; while those with BERT-large are. Similar results are observed for the two TREC DL datasets, with the difference that for these datasets TILDEv2 displays better effectiveness than uniCOIL when re-ranking docT5query (no statistical significance).

The biggest advantage of TILDEv2 is however the low query latency. TILDEv2 does not require GPUs and only adds a couple of milliseconds on top of the first stage retrieval (BM25 or docT5query), overall resulting in a query latency of ≤ 80 ms on CPU. The other neural methods, instead, either require GPUs to achieve acceptable query latency (and in any case higher than that of TILDEv2) or, if they can feasibly be used on CPUs, display higher query latency. Table 2 further details the breakdown of query latency for TILDEv2 and some strong baselines in the CPU environment. For uniCOIL, ANCE and EPIC, even just the query processing amounts to about 50% of the total latency of TILDE and TILDEv2. For DRs (ANCE), the retrieval time is much higher on CPU than GPU. The latency of uniCOIL is acceptable on CPU; yet, it is more than 4 times higher than that of the original TILDE and of TILDEv2. EPIC, TILDE and TILDEv2 have comparable query latency, but EPIC and TILDE are much less effective than TILDEv2 (as shown in Table 1).

In summary, in answer to RQ2, TILDEv2's effectiveness is on par with or better than the considered baselines, while achieving much higher efficiency.

5.3 RQ3: Effectiveness-efficiency trade-off

In this section, we investigate the effectiveness-efficiency trade-off of TILDEv2 (RQ3). This trade-off appears in many second-stage re-ranking methods with respect to the rank cut-off used for the re-ranking: higher cut-offs translate to more passages needing to be evaluated, and thus higher query latency. For this analysis, we compare our TILDEv2, which ran on a CPU environment, with the BERT-large re-ranker, ran on a GPU environment, on the task of re-ranking the top k results from BM25. The BERT-large re-ranker is the strongest re-ranker baseline we considered.

In Figure 2 we report the analysis of this effectiveness-efficiency trade-off in terms of nDCG@10 vs. query latency measured on TREC DL 2019 (similar results on other datasets), exploring the cut-offs $k = \{0, 10, 20, 50, 100, 200, 500, 1000\}$ ($k = 0$: no re-ranking).

The BERT-large re-ranker (blue line) achieves higher effectiveness when considering deeper cut-offs, as reported before by others [20, 39]. This is problematic because the BERT-large re-ranker is very inefficient when evaluating a passage (requiring on average 12ms): thus the cut-off conditions which make the BERT-large re-ranker highly effective are the same that make it largely inefficient – it takes around 12,000 ms to re-rank the top 1,000 passages.

In contrast, TILDEv2 (green line) only needs a few milliseconds to re-rank the top 1,000 passages, and re-ranking less than that leads to negligible time-saving. What is more, in less than 80ms TILDEv2 can achieve the same effectiveness that the BERT-large re-ranker achieves after spending more than 500 milliseconds (cut-off k=50).

Finally, we also consider a three-stage re-ranking system (orange line), where the top 1,000 BM25 results are first re-ranked using TILDEv2 and that ranking is further processed using the BERT-large re-ranker on the top k passages. The use on the intermediate re-ranking step with TILDEv2 adds very little to the overall latency: just 10.8 ms. As shown in Figure 2, the use of the intermediate TILDEv2 step allows the final BERT re-ranker to produce highly effective results using smaller rank cut-offs (i.e. $10 \leq k \leq 50$), thus overall producing substantial savings in query latency compared to when TILDEv2 is not added to the pipeline (blue line). For example, to reach the same level of effectiveness reached by the BERT-large re-ranker with $k = 50$ (latency 605 ms), the three-stage pipeline with TILDEv2 only requires the BERT-large re-ranker to re-rank the top 10 TILDEv2 results (latency 175 ms), with a saving of 430 ms. Similarly, the effectiveness reached by the three-stage pipeline with $k = 50$ is not statistically significantly different from that reached by the BERT-large re-ranker alone with $k = 200, 500, 1000$.

Table 2: Detailed query latency in CPU environment, in milliseconds. Each method is set to use the parameters that best optimise MRR@10 on MS MARCO. ANCE and uniCOIl do not perform re-ranking.

Methods	Query process	Retrieval	Re-rank	Total
uniCOIL	46	230	n.a.	276
ANCE	63	1,570	n.a.	1,633
EPIC+d2q-top15	40	75	1	116
TILDE+d2q-top10	0.1	75	0.2	75.3
TILDEv2+BM25-top1000	0.1	70	10.7	80.8
TILDEv2+d2q-top100	0.1	75	1.3	76.4

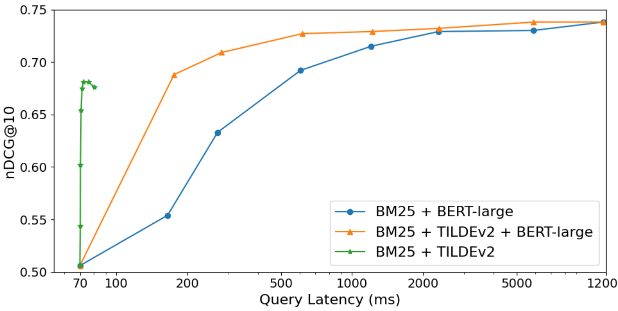


Figure 2: Query latency analysis vs. nDCG@10 on TREC DL2019. Points from left to right are re-ranking cut-offs k = 0 (no re-rank), 10, 20, 50, 100, 200, 500 and 1,000.

In summary, in answer to RQ3, we find that the rank cut-off k controls the efficiency vs. effectiveness trade-off for the BERT-large re-ranker: low values of k result in lower latency and lower effectiveness, while larger k values yield more effective results but substantially higher latencies. The rank cut-off k, however, while impacting TILDEv2's effectiveness (higher values lead to higher effectiveness), barely has any effect on query latency. Furthermore, we find that injecting TILDEv2 into the BERT-large re-ranker pipeline allows to reach the best effectiveness produced by the BERT-large re-ranker alone, for a fraction of the latency.

5.4 RQ4: Impact of passage expansion

In this section, we investigate the impact of different passage expansion methods (RQ4), and in particular the docT5query and the proposed TILDE for passage expansion, with respect to the cost of the passage expansion process (time and money), the quality of expanded terms and the impact on index size. To study these aspects, we consider passage expansion in the context of our TILDEv2 when re-ranking the top 1,000 passages from BM25, and of uniCOIL for full index retrieval [19]. Both methods require expanded passages at training and ranking; the use of uniCOIL allows us to verify the generalisability of the TILDE passage expansion method.

Table 3: Impact of different expansion methods. The index size estimation of TILDEv2 is included BM25 index size and contextualized term weight index size. The expansion cost is estimated base on Google cloud service.

	no Expansion		**docT5query**		**TILDE,** $m = 128$		**TILDE,** $m = 150$		**TILDE,** $m = 200$	
	uniCOIL	TILDEv2	uniCOIL	TILDEv2	uniCOIL	TILDEv2	uniCOIL	TILDEv2	uniCOIL	TILDEv2
MS MARCO, MRR@10	0.319	0.299	0.351	0.333	0.343	0.326	0.346	0.327	0.349	0.330
DL2019, nDCG@10	0.653	0.613	0.693	0.676	0.682	0.680	0.690	0.679	0.707	0.670
DL2019, MAP	0.418	0.417	0.476	0.448	0.464	0.457	0.470	0.452	0.474	0.447
Index size	4.3G	4.8G	6.0G	5.2G	5.6G	5.2G	6.7G	5.6G	9.81G	6.9G
Avg added token	0		19.0		13.0		25.2		61.6	
Expansion cost	0		320 hours/768$		7.22 hours/5.34$		7.25 hours/5.37$		7.33 hours/5.42$	

Table 4: Tokens generated by docT5query and TILDE for the first passage in the MS MARCO dataset (pid=0).

Original passage	**docT5query**	**TILDE, m=128**	**TILDE, m=200**
the presence of communication amid scientific minds was equally important to the success of the manhattan project as scientific intellect was. the only cloud hanging over the impressive achievement of the atomic researchers and engineers is what their success truly meant; hundreds of thousands of innocent lives obliterated.	amongst scientists why? about so a importance purpose how significant in for believe who did	importance purpose quiz scientists bomb genius development solving significance successful intelligence solve effect objective research accomplish brains progress scientist	... future impact strategic develop necessary ni role involved developing needed theory significant technology achievements accomplished science achieve intellectual new breakthrough help keypower effects effort human work engineer concept invention idea problem process ability communicate developed would affect solved decision use deal society reason effective franklin problems great goals opportunity secret considered

The methods are tested with no expansion, docT5query expansion, and TILDE expansion. For docT5query we generate 40 expansion queries, as done in previous work; larger values are infeasible (very long generation time). For TILDE, we generate $m = 128, 150, 200$ expansion terms. Recall that not all expansion terms are added to a passage: only new expansion terms are added.

The results are reported in Table 3. Any form of expansion improves over the not expanded results, showing that passage expansion is crucial for both TILDEv2 and uniCOIL. The docT5query method produces the most effective expansions for both methods on MS MARCO, although differences between docT5query and TILDE $m = 200$ are marginal (0.349 vs. 0.351 for uniCOIL, 0.330 vs. 0.333 for TILDEv2) and not statistically significant ($p = 0.848$ and $p = 0.663$, respectively). TILDE $m = 200$ provides the most effective expansions for uniCOIL on TREC DL2019 (for nDCG@10), and TILDE $m = 128$ provides the most effective expansions for TILDEv2. For MAP on this dataset, docT5query provides the most effective expansions for uniCOIL, while TILDE $m = 128$ provides the most effective ones for TILDEv2. Overall, the effectiveness achieved by the TILDE expansion method is on par with that of docT5query.

Next, we consider the size of the index produced by the methods (for TILDEv2, this is the size of the inverted index for BM25 and the term weight index, implemented using HashTable). When TILDE $m = 128$ is used, the index size produced by uniCOIL is smaller than when using docT5query, while the one produced by TILDEv2 is the same as if docT5query was used for expansion. When m increases, the index size increases for both uniCOIL and TILDEv2: this is expected as an increase in m will make TILDE produce more tokens that are added to the passages. This can be seen by observing the average added token row in Table 3. However, two observations can be made: (1) the index produced by TILDEv2 is always smaller than that produced by uniCOIL, despite the same number of tokens being added, and (2) the size of the TILDEv2 index is two orders of magnitude smaller than the size of the original TILDE index (not reported in the table), which is ≈500Gib [46].

We then consider the cost of executing the two passage expansion processes on the whole MS MARCO passage collection. The TILDE expansion process is two orders of magnitude faster to run than the docT5query (320 hours vs. ≈7 hours). In addition, the TILDE expansion process can be run on a preemptible GPU environment, which is much cheaper than the preemptible TPU environment used by Nogueira et al. [29] to run the docT5query expansion[6] (costs estimated based on the Google cloud service). Thus, using our TILDE expansion method is not just faster than docT5query, but also cheaper. These advantages become even more obvious when considering expanding larger collections such as the MS MARCO v2 (138.3 million passages): it would take docT5query ≈ 5,000 hours and $12,000, while only requiring TILDE ≈114 hours and $85.

Table 4 reports the tokens added by different expansion methods for an example passage. The expanded tokens from docT5query contain symbols such as the question mark, and common stopwords, e.g., ‘a’, ‘for’. On the other hand, TILDE removes stopwords and symbols before appending to the passage. Interestingly, both docT5query and TILDE add similar tokens such as ‘scientists’, ‘importance’, and ‘purpose’. In general, the added tokens from both methods are on-topic.

[6] Using GPUs would result in a larger runtime.

In summary, in answer to RQ4, our TILDE passage expansion method achieves similar effectiveness as the state-of-the-art docT5query, but it requires far fewer computational resources.

6 CONCLUSION

We proposed the TILDEv2 model for passage re-ranking, which builds on top of the recently proposed TILDE by integrating the best-of-breed from recent advances in neural retrieval. We further proposed a novel use of the original TILDE as an effective and efficient passage expansion technique. Our TILDEv2 aims to solve some of the drawbacks of the original TILDE model (effectiveness, large indexes) by integrating the contextualized exact term matching approach. While, our passage expansion technique aims to address the scalability issues of current methods for passage expansion.

The empirical results show that TILDEv2 significantly improves the effectiveness of the original TILDE and largely reduces its index size, while maintaining its efficiency and without resorting to expensive computational environments (TILDEv2 runs on CPU). We also find that the proposed TILDE-based passage expansion method delivers computational cost savings of up to 98% compared to other passage expansion methods, while experiencing effectiveness drops of less than 1% (and improving effectiveness in certain settings). The proposed passage expansion method can be used not just with TILDEv2: we show its performance generalises to other methods such as uniCOIL. These results make TILDEv2 a production-ready method of great appeal in search settings that require low query latency and have limited computation resources available. The code that implements TILDEv2 and that can be used to reproduce the results in this paper is available at https://github.com/anonymous.

REFERENCES

[1] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[2] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2021. Overview of the TREC 2020 deep learning track. *arXiv preprint arXiv:2102.07662* (2021).

[3] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2020. Overview of the trec 2019 deep learning track. *arXiv preprint arXiv:2003.07820* (2020).

[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. 2019. Deeper text understanding for IR with contextual neural language modeling. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval.* 985–988.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT (1).*

[7] Cicero dos Santos, Xiaofei Ma, Ramesh Nallapati, Zhiheng Huang, and Bing Xiang. 2020. Beyond [CLS] through ranking by generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP).* 1722–1727.

[8] George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. 1987. The vocabulary problem in human-system communication. *Commun. ACM* 30, 11 (1987), 964–971.

[9] 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.* 3030–3042.

[10] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. Rethink Training of BERT Rerankers in Multi-Stage Retrieval Pipeline. In *The 43rd European Conference On Information Retrieval (ECIR).*

[11] Luyu Gao, Zhuyun Dai, Tongfei Chen, Zhen Fan, Benjamin Van Durme, and Jamie Callan. 2021. Complementing lexical retrieval with semantic residual embedding. In *The 43rd European Conference On Information Retrieval (ECIR).*

[12] Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics.* JMLR Workshop and Conference Proceedings, 297–304.

[13] Sebastian Hofstätter, Markus Zlabinger, and Allan Hanbury. 2020. Interpretable & Time-Budget-Constrained Contextualization for Re-Ranking. In *ECAI 2020.* IOS Press, 513–520.

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

[15] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP).* 6769–6781.

[16] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval.* 39–48.

[17] Oleg Lesota, Navid Rekabsaz, Daniel Cohen, Klaus Antonius Grasserbauer, Carsten Eickhoff, and Markus Schedl. 2021. A Modern Perspective on Query Likelihood with Deep Generative Retrieval Models. *Proceedings of ACM ICTIR* (2021).

[18] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.* 7871–7880.

[19] Jimmy Lin and Xueguang Ma. 2021. A Few Brief Notes on DeepImpact, COIL, and a Conceptual Framework for Information Retrieval Techniques. *arXiv preprint arXiv:2106.14807* (2021).

[20] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2020. Pretrained transformers for text ranking: Bert and beyond. *arXiv preprint arXiv:2010.06467* (2020).

[21] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2021. In-Batch Negatives for Knowledge Distillation with Tightly-Coupled Teachers for Dense Retrieval. In *Proceedings of the 6th Workshop on Representation Learning for NLP.*

[22] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[23] Sean MacAvaney. 2020. OpenNIR: A Complete Neural Ad-Hoc Ranking Pipeline. In *WSDM 2020.*

[24] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. 2020. Expansion via prediction of importance with contextualization. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval.* 1573–1576.

[25] 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.

[26] 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. In *CoCo@ NIPS.*

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

[28] Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document Ranking with a Pretrained Sequence-to-Sequence Model. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings.* 708–718.

[29] Rodrigo Nogueira, Jimmy Lin, and AI Epistemic. 2019. From doc2query to docTTTTTquery. *Online preprint* (2019).

[30] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019. Multi-stage document ranking with bert. *arXiv preprint arXiv:1910.14424* (2019).

[31] 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).

[32] Jay M Ponte and W Bruce Croft. 1998. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval.* 275–281.

[33] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* 5835–5847.

[34] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. http://jmlr.org/papers/v21/20-074.html

[35] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21 (2020), 1–67.

[36] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. 1999. Analysis of a very large web search engine query log. In *Acm sigir forum*, Vol. 33. ACM New York, NY, USA, 6–12.

[37] Cole Thienes and Jack Pertschuk. 2019. NBoost: Neural Boosting Search Results. https://github.com/koursaros-ai/nboost.

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

[39] Shuai Wang, Shengyao Zhuang, and Guido Zuccon. 2021. BERT-based Dense Retrievers Require Interpolation with BM25 for Effective Passage Retrieval. *Proceedings of ACM ICTIR* (2021).

[40] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.

[41] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *International Conference on Learning Representations*.

[42] 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.

[43] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems* 32 (2019).

[44] 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).

[45] Shengyao Zhuang, Hang Li, and Guido Zuccon. 2021. Deep Query Likelihood Model for Information Retrieval. In *The 43rd European Conference On Information Retrieval (ECIR)*.

[46] Shengyao Zhuang and Guido Zuccon. 2021. TILDE: Term Independent Likelihood MoDEl for Passage Re-Ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21)*. 1483–1492.