Efficient Document Re-Ranking for Transformers by Precomputing Term Representations

Sean MacAvaney
IR Lab, Georgetown University, USA
sean@ir.cs.georgetown.edu

Franco Maria Nardini
ISTI-CNR, Pisa, Italy
francomaria.nardini@isti.cnr.it

Raffaele Perego
ISTI-CNR, Pisa, Italy
raffaele.perego@isti.cnr.it

Nicola Tonellotto
University of Pisa, Italy
nicola.tonellotto@unipi.it

Nazli Goharian
IR Lab, Georgetown University, USA
nazli@ir.cs.georgetown.edu

Ophir Frieder
IR Lab, Georgetown University, USA
ophir@ir.cs.georgetown.edu

ABSTRACT

Deep pretrained transformer networks are effective at various ranking tasks, such as question answering and ad-hoc document ranking. However, their computational expenses deem them cost-prohibitive in practice. Our proposed approach, called PreTTR (Precomputing Transformer Term Representations), considerably reduces the query-time latency of deep transformer networks (up to a 42× speedup on web document ranking) making these networks more practical to use in a real-time ranking scenario. Specifically, we precompute part of the document term representations at indexing time (without a query), and merge them with the query representation at query time to compute the final ranking score. Due to the large size of the token representations, we also propose an effective approach to reduce the storage requirement by training a compression layer to match attention scores. Our compression technique reduces the storage required up to 95% and it can be applied without a substantial degradation in ranking performance.

ACM Reference Format:
Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. 2020. Efficient Document Re-Ranking for Transformers by Precomputing Term Representations. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401093

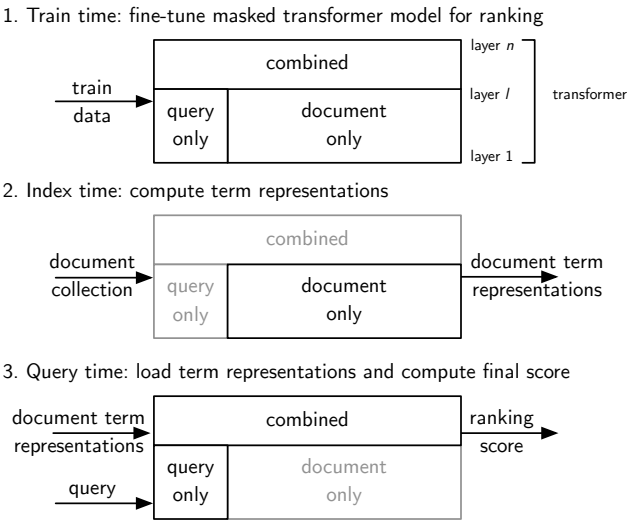

Figure 1: High-level overview of PreTTR. At query time, document representations (which were computed at index time) are loaded, which reduces the computational burden.

1 INTRODUCTION

Pretrained deep transformer networks, e.g., BERT [8], have recently been transformative for many tasks, exceeding the effectiveness of prior art in many natural language processing and information retrieval tasks [4, 27, 31, 32, 47, 48]. However, these models are huge in size, thus expensive to run. For instance, in about one year, the largest pretrained transformer model grew from about 110 million parameters (GPT [34]) to over 8.3 billion (Megatron-LM [39]), which, when applied to IR tasks like ad-hoc retrieval, have substantial impact on the query processing performance, to the point of being impractical [27]. We move these neural ranking models towards practicality.

Runtime efficiency is a central tenant in information retrieval, though as neural approaches have gained prominence, their running time has been largely ignored in favor of gains in ranking performance [16]. Recently, the natural language processing community has begun to consider and measure running time [37], albeit mostly for reasons of environmental friendliness and inclusiveness. Chiefly, model distillation approaches [22, 36, 40] are prominent, which involve training a smaller model off of the predictions of a larger model. This smaller model can then be further fine-tuned for a specific task. While this approach can exceed the performance of a smaller model when only trained on the specific task data, it inherently limits the performance of the smaller model to that of the larger model. Nevertheless, distillation is a method complementary to ours; our approach can work with a distilled transformer network. Others have explored quantization approaches to reduce model sizes, by limiting the number of bits used to represent network's parameters to 16, 8, or fewer bits. Quantization was mainly

SIGIR '20, July 25–30, 2020, Virtual Event, China

ACM ISBN 978-1-4503-8016-4/20/07…$15.00
https://doi.org/10.1145/3397271.3401093

explored to make the neural networks suitable for embedded systems [11, 38]. We employ a basic quantization technique to reduce the storage requirements of the term representations.

We propose a method for improving the efficiency of transformer-based neural ranking models. We exploit a primary characteristic of ad-hoc ranking: an initial indexing phase can be employed to pre-process documents in the collection to improve query-time performance. Specifically, we observe that much of the term interaction at query time happens locally within either the query or document, and only the last few layers of a deep transformer network are required to produce effective ranking scores once these representations are built. Thus, documents can be processed at index time through part of the network without knowledge of the query. The output of this partial network computation is a sequence of contextualised term representations. These representations can then be stored and used at query time to finish the processing in conjunction with the query. This approach can be trained end-to-end by masking the attention across the query and document during training time (i.e., disallowing the document from attending to the query and vice versa.) We call this approach PreTTR (Precomputing Transformer Term Representations). A high-level overview of PreTTR is shown in Figure 1.

At train time, a transformer network is fine-tuned for ad-hoc document ranking. This transformer network masks attention scores in the first l layers, disallowing interactions between the query and the document. At index time, each document in the collection is processed through the first l layers, and the resulting term representations are stored. At query time, the query is processed through the first l layers, and then combined with the document term representations to finish the ranking score calculation.

Since term representations of each layer can be large (e.g., 768 float values per document term in the base version of BERT), we also propose a compression approach. This approach involves training an encoding layer between two transformer layers that produces representations that can replicate the attention patterns exhibited by the original model. We experimentally show that all these processes result in a much faster network at query time, while having only a minimal impact on the ranking performance and a reasonable change in index size. The settings of PreTTR (amount of precomputation, degree of compression) can be adjusted depending on the needs of the application. These are all critical findings that are required to allow transformer networks to be used in practical search environments. Specifically, the lower computation overhead reduces query-time latency of using transformer networks for ranking, all while still yielding the substantial improvements to ranking accuracy that transformer-based rankers offer.

In summary, the contributions of the paper are the following:

- A new method for improving the efficiency of transformer-based neural ranking models (PreTTR). The approach exploits the inverted index to store a precomputed term representation of documents used to improve query-time performance;
- A novel technique for compressing the precomputed term representations to reduce the storage burden introduced by PreTTR. This is accomplished by training a compression function between transformer layers to minimize the difference between the attention scores with and without compression;
- A comprehensive experimental evaluation of PreTTR on multiple pre-trained transformer networks on two public datasets, namely, TREC WebTrack 2012 and TREC Robust 2004. Our PreTTR accelerates the document re-ranking stage by up to 42× on TREC WebTrack 2012, while maintaining comparable P@20 performance. Moreover, our results show that our compression technique can reduce the storage required by PreTTR by up to 97.5% without a substantial degradation in the ranking performance;
- For reproducibility, our code is integrated into OpenNIR [26], with instructions and trained models available at: https://github.com/Georgetown-IR-Lab/prettr-neural-ir.

2 RELATED WORK

We present an overview of neural ranking techniques, pretrained transformers for ranking, and efforts to optimize the efficiency of such networks.

2.1 Neural Ranking

As neural approaches have gained prominence in other disciplines, many have investigated how deep neural networks can be applied to document ranking [10, 17, 19, 44]. These approaches typically act as a final-stage ranking function, via a *telescoping* (also referred to as *cascading*, or *multi-stage*) technique [29, 43]; that is, initial ranking is conducted with less expensive approaches (e.g., BM25), with the final ranking score calculated by the more expensive machine-learned functions. This technique is employed in commercial web search engines [35]. Neural ranking approaches can broadly be categorized into two categories: *representation-focused* and *interaction-focused* models. Representation-focused models, such as DSSM [17], aim to build a dense "semantic" representation of the query and the document, which can be compared to predict relevance. This is akin to traditional vector space models, with the catch that the vectors are learned functions from training data. Interaction models, on the other hand, learn patterns indicative of relevance. For instance, PACRR [19] learns soft n-gram matches in the text, and KNRM [44] learns matching kernels based on word similarity scores between the query and the document.

2.2 Pretrained Transformers for Ranking

Since the rise of pretrained transformer networks (e.g., BERT [8]), several have demonstrated their effectiveness on ranking tasks. Nogueira and Cho [31] demonstrated that BERT was effective at passage re-ranking (namely on the MS-MARCO and TREC CAR datasets) by fine-tuning the model to classify the query and passage pair as relevant or non-relevant. Yang et al. [47] used BERT in an end-to-end question-answering pipeline. In this setting, they predict the spans of text that answer the question (same setting as demonstrated on SQuAD in [8]). MacAvaney et al. [27] extended that BERT is effective at *document* ranking, both in the "vanilla" setting (learning a ranking score from the model directly) and when using the term representations from BERT with existing neural ranking architectures (CEDR). Dai and Callan [4] found that the additional context given by natural language queries (e.g., topic descriptions) can improve document ranking performance, when compared with keyword-based queries. Yang et al. [48] showed that

BERT scores aggregated by sentence can be effective for ranking. Doc2Query [32] employs a transformer network at index time to add terms to documents for passage retrieval. The authors also demonstrate that a BERT-based re-ranker can be employed atop this index to further improve ranking performance.

2.3 Neural Network Efficiency

Pretrained transformer networks are usually characterized by a very large numbers of parameters and very long inference times, making them unusable in production-ready IR systems such as web search engines. Several approaches were proposed to reduce the model size and the inference computation time in transformer networks [12]. Most of them focus on the compression of the neural network to reduce their complexity and, consequently, to reduce their inference time.

Neural network *pruning* consists of removing weights and activation functions in a neural network to reduce the memory needed to store the network parameters. The objective of pruning is to convert the weight matrix of a dense neural network to a sparse structure, which can be stored and processed more efficiently. Pruning techniques work both at learning time and as a post-learning step. In the first category, Pan et al. propose regularization techniques focused at removing redundant neurons at training time [33]. Alternatively, in the second category, Han et al. propose to remove the smallest weights in terms of magnitude and their associated edges to shrink the size of the network [13]. Conversely, our proposed approach does not change the dense structure of a neural network to a sparser representation, but it aims to precompute the term representation of some layers, thus completely removing the document-only portion of a transformer neural network (see Figure 1).

Another research line focuses on improving the efficiency of a network is weight *quantization*. The techniques in this area aim at reducing the number of bits necessary to represent the model weights: from the 32 bits necessary to represent a float to only a few bits [18]. The state of the art network quantization techniques [1, 45] aims at quantizing the network weights using just 2-3 bits per parameter. These approaches proved effective on convolutional and recurrent neural networks. Quantization strategies could be used in our proposed approach. However, to reduce the size of the term representations, we opt to instead focus on approaches to reduce the dimensionality of the term representations, and leave quantization of the stored embeddings to future work.

A third research line employed to speed-up neural networks is *knowledge distillation* [15]. It aims to transform the knowledge embedded in a large network (called teacher) into a smaller network (called student). The student network is trained to reproduce the results of the teacher networks using a simpler network structure, with less parameters than those used in the teacher network. Several strategies have been proposed to distill knowledge in pretrained transformer networks such as BERT [22, 36, 40].

Our PreTTR method is orthogonal to knowledge distillation of transformer network. In fact, our approach can be applied directly to any kind of transformer, including those produced by knowledge distillation.

Table 1: Table of symbols.

Symbol(s)	Definition
$\mathbf{q}$	Query
$\mathbf{d}$	Document
$R(\mathbf{q}, \mathbf{d})$	Neural ranking architecture
$T(\mathbf{s})$	Transformer network
$\mathbf{s}$	a sequence of input tokens
E	Embedding layer
L_i	Transformer encoding layer
$\mathbf{s}_i$	Transformer token representations after layer i
$\mathbf{a}_i$	Attention weights used in layer i
c	Classification representation
d	Dimension of the classification representation
m	Length of sequence $\mathbf{s}$
h	Number of attention heads per layer
n	Number of layers in T
$W_{combine}$	Vanilla BERT weight combination
l	Layer number the transformer is executed for precomputing document term vectors
e	Compressed size
$\mathbf{r}$	Compressed representation after layer l
$W/\mathbf{b}_{comp}$	Compression parameters
$W/\mathbf{b}_{decomp}$	De-compression parameters
$\hat{\mathbf{s}}_l$	De-compressed representation after layer l

2.4 Neural Ranking Efficiency

Scalability and computational efficiency are central challenges in information retrieval. While the efficiency of learning to rank solutions for document re-ranking have been extensively studied [6, 24, 41], computational efficiency concerns have largely be ignored by prior work in neural ranking, prompting some to call for more attention to this matter [16]. That being said, some efforts do exist. For instance, Zamani et al. [50] investigate learning sparse query and document representations which allow for indexing. Ji et al. [21] demonstrate that Locality-Sensitive Hashing (LSH) and other tricks can be employed to improve the performance of interaction-focused methods such as DRMM [10], KNRM [44], and ConvKNRM [5]. This approach does not work for transformer models, however, because further processing of the term embeddings is required (rather than only computing similarity scores between the query and document).

Within the realm of transformer-based models for ad-hoc ranking, to our knowledge only [27] and [32] acknowledge that retrieval speed is substantially impacted by using a deep transformer network. As a result Hofstätter and Hanbury [16] call for more attention to be paid to run time. MacAvaney et al. find that limiting the depth of the transformer network can reduce the re-ranking time while yielding comparable ranking performance [27]. Nogueira et al. find that their approach is faster than a transformer-based re-ranker, but it comes at a great cost to ranking performance: a trade-off that they state can be worthwhile in some situations [32]. In contrast with both these approaches, we employ *part* of the transformer network at index time, and the remainder at query-time (for re-ranking). We find that this can yield performance on par with the full network, while significantly reducing the query time latency.

3 MOTIVATION

Let a generic transformer network $T : \mathbf{s} \mapsto c$ map a sequence $\mathbf{s}$ of m tokens (e.g., query and document terms) to a d-dimensional

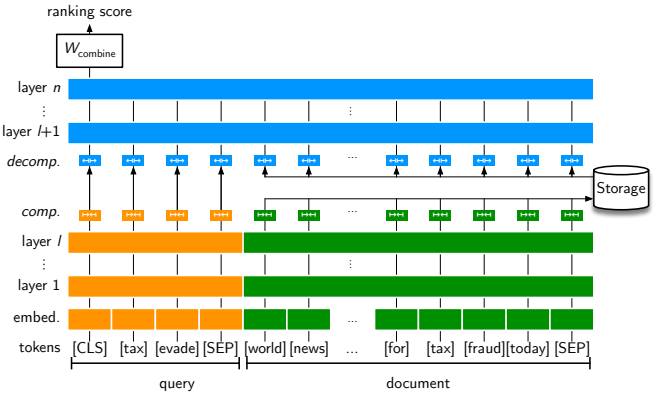

Figure 2: Overview of PreTTR. Compressed term representations for document layers 1 to l are computed and stored at index time (green segments) while term representations for query layers 1 to l (orange segments) and joint query-document representations for layers $l+1$ to n (blue segments) are computed at query time to produce the final ranking score. Compression and decompression can optionally be applied between layers l and $l+1$ to reduce the storage needed for the document term representations.

output representation $c \in \mathbb{R}^d$. As depicted in Figure 2, the transformer network is composed by an initial embedding layer E and by n layers $L_1, \ldots, L_n$. The embedding layer E maps each of the m input tokens into the initial d-dimensional token representations matrix $\mathbf{s}_0 \in \mathbb{R}^{m \times d}$. Each layer L_i takes the token representations matrix $\mathbf{s}_{i-1} \in R^{m \times d}$ from the previous layer L_{i-1} and produces a new representations matrix $\mathbf{s}_i \in \mathbb{R}^{m \times d}$. The specific representation used and operations performed in E and L_i depend on the specific transformer architecture (e.g., BERT uses token, segment, and position embeddings for the embedding layer E and self-attention, a feed-forward layer, and batch normalization in each layer L_i). However, the primary and common component of each layer L_i is the self-attention mechanism and associated procedure. When the transformer network is trained, every layer produces a self-attention tensor $\mathbf{a}_i \in \mathbb{R}^{h \times m \times m}$, where h is the number of attention heads per layer, i.e., the number of attention "representation subspaces" per layer. A general description of this process is given by Vaswani et al. [42], while different transformer architectures may have tweaks to this general structure or pre-training procedure.

We assume a special output classification token, e.g., `[CLS]` in BERT, is included as a token in c, and that the final representation of this token is used as the final output of the transformer network, i.e., $c = T(\mathbf{s})$. Without loss of generality, here we only concern ourselves with the `[CLS]` output classification token, i.e., we ignore other token representation outputs; this is the special token representation that models such as BERT use to generate ranking scores.

We illustrate how neural transformer networks are used in a ranking scenario. We follow the Vanilla BERT model proposed by MacAvaney et al. [27] and generalize it. Let a ranking function $R(\mathbf{q}, \mathbf{d}) \in \mathbb{R}$ map a query $\mathbf{q}$ and a document $\mathbf{d}$ to a real-valued ranking score. Neural rankers based on transformer networks such as Vanilla BERT compute the ranking score by feeding the query-document pair into the transformer. Given a query $\mathbf{q}$ and a document $\mathbf{d}$, their tokens are concatenated into a suitable transformer input, e.g., $\mathbf{s} = [\texttt{CLS}]; \mathbf{q}; [\texttt{SEP}]; \mathbf{d}; [\texttt{SEP}]$, where ";" represents the concatenation operator.[1] The output of the transformer network corresponding to this input is then linearly combined using a tuned weight matrix $W_{combine} \in \mathbb{R}^{d \times 1}$ to compute the final ranking score as follows:

$$R(\mathbf{q}, \mathbf{d}) = T([\texttt{CLS}]; \mathbf{q}; [\texttt{SEP}]; \mathbf{d}; [\texttt{SEP}]) W_{combine}. \quad (1)$$

The processing time of state-of-the-art neural rankers based on transformer networks is very high, e.g., approximately 50 documents ranked per second on a modern GPU, making such rankers impractical for most ad-hoc retrieval tasks.

To gain an understanding of where are the most expensive components of a transformer network such as the Vanilla BERT model, we measure the run-times of the main steps of the model. We find that most of the processing is performed in the computations involving the transformer's layers. In particular, about 50% of the total time is spent performing attention-related tasks. Moreover, the feed-forward step of the transformer (consisting of intermediate and output in diagram) accounts for about 48% of the total time, and is largely due to the large intermediate hidden representation size for each token. This breakdown motivates the investigation of possible solutions to reduce the processing time of transformer networks, in particular in reducing the time spent in traversing the transformer's layers.

4 PROPOSED SOLUTION

We discuss how our PreTTR approach improve the efficiency of processing queries using a transformer network by reducing the computational impact of the network's layers.

4.1 PreTTR: Precomputing Transformer Term Representations

We improve the query time performance of transformer models by precomputing document term representations partially through the transformer network (up to transformer layer l). We then use these representations at query time to complete the execution of the network when the query is known.

This is accomplished at model training time by applying an attention mask to layers $L_1, L_2, \ldots, L_l$, in which terms from the query are not permitted to attend to terms from the document and vice versa. In layers $L_{l+1}, \ldots, L_n$, this attention mask is removed, permitting any token to attend to any other token. Once trained, the model is used at both index and query time. At index time, documents are encoded (including the trailing `[SEP]` token)[2] by the transformer model through layers $L_1, L_2, \ldots, L_l$ without a query present (Figure 2, green segments). The token representations generated at index time at layer L_l are then stored to be reused at query time (Figure 2, document storage between layers L_l and L_{l+1}). To answer a query, candidate documents are selected, e.g., the top documents

[1] We use the BERT convention of `[CLS]` and `[SEP]` to represent the classification and separation tokens, respectively.

[2] There is evidence that the separator token performs an important function for pre-trained transformer models, by acting as a no-op for the self-attention mechanism [2].

retrieved by a first-stage simple ranking model [41], and precomputed term representations are loaded. The query terms (including the leading [CLS] and training [SEP] tokens) are encoded up to layer L_l without a document present (Figure 2, orange segments). Then, the representations from the query and the document are joined, and the remainder of the transformer network is executed over the entire sequence to produce a ranking score (Figure 2, blue segments).

Since (1) the length of a query is typically much shorter than the length of a document, (2) the query representations can be re-used for each document being ranked, (3) each transformer layer takes about the same amount of time to execute, and (4) the time needed to perform term embedding is comparatively low, PreTTR decreases by about $\frac{n-l}{n}$ the cost of traversing the transformer network layers. With a sufficiently large value of l, this results in considerable time savings. Note that this reduction can be at most equal to $\frac{1}{n}$ because, when $l = n$, no information about the document ever contributes to the ranking score, resulting in identical scores for every document. Moreover, we show experimentally that this can be further improved by limiting the computation of the final layer to only the [CLS] representation.

4.2 Token Representation Compression

Although PreTTR can reduce the run-time cost of traversing the first l layers of the transformer network at query time, the solution proposed might be costly in terms of storage requirements because the representation size d is quite large (e.g., 1024, 768 or 512 float values per token). To address this issue, we propose a new token compression technique that involves pre-training a simple encoder-decoder network. This network is able to considerably reduce the token representation size. We opt for this approach because it can fit seamlessly into the transformer network, while reducing the number of dimensions needed to represent each token. The compressor is added as an additional component of the transformer network between layers L_l and L_{l+1}. We compress the input by using a simple feed-forward and normalization procedure, identical to the one used within a BERT layer to transform the output (but with a *smaller* internal representation rather than a larger one). We optimize the weights for the compression network in two stages: (1) an initial pre-training stage on unlabeled data, and (2) a fine-tuning stage when optimizing for relevance.

For a compressed size of e values, a two-step procedure is used. First, the compressed representations $\mathbf{r} \in \mathbb{R}^{m \times e}$ are built using $\mathbf{r} = \text{GELU}(\mathbf{s}_l W_{comp} + \mathbf{b}_{comp})$, where $\text{GELU}(\cdot)$ is a Gaussian Error Linear Unit [14], and $W_{comp} \in \mathbb{R}^{d \times e}$ and $\mathbf{b}_{comp} \in \mathbb{R}^{e}$ are the new learned weight parameters. These compressed representations $\mathbf{r}$ can be stored in place of $\mathbf{s}_l$. Second, the compressed representations $\mathbf{r}$ are then expanded back out to $\hat{\mathbf{s}}_l \in \mathbb{R}^{m \times d}$ via a second linear transformation involving the learned weight parameters W_{decomp}, $\mathbf{b}_{decomp}$, and batch normalization. The decompressed representations $\hat{\mathbf{s}_l}$ are then used in place of the original representation $\mathbf{s}_l$ for the remaining layers of the transformer.

In preliminary experiments, we found the compression and decompression parameters to be difficult to learn jointly with the ranker itself. Thus, we instead propose a pre-training approach to provide an effective initialization of these parameters. We want the transformer network with the compression mechanism to behave similarly to that of the network without such compression: we do not necessarily care about the exact representations themselves. Thus, we use an attention-based loss function. More specifically, we optimize our compression/decompression network to reduce the mean squared error of the attention scores in the last $n - l$ layers of the compressed transformer network and the original transformer network. Thus, the loss function we use to train our compression and decompression network is:

$$\mathcal{L}(\mathbf{a}_{l+1}, \ldots, \mathbf{a}_n, \hat{\mathbf{a}}_{l+1}, \ldots, \hat{\mathbf{a}}_n) = \frac{1}{n-l} \sum_{i=l+1}^{n} \text{MSE}(\mathbf{a}_i, \hat{\mathbf{a}}_i), \qquad (2)$$

where a_i represents the attention scores at layer i from the unmodified transformer network, $\hat{a_i}$ represents the attention scores at layer i from the transformer network with the compression unit, and $\text{MSE}(\cdot)$ is the mean squared error function. With this loss function, the weights can be pre-trained on a massive amount of unlabeled text. We use this procedure as an initial pre-training step; we further fine-tune the weights when optimizing the entire ranking network for relevance.

5 EXPERIMENTAL SETUP

We detail the setup employed in our experiments: the datasets, namely TREC WebTrack 2012 and TREC Robust 2004, and the transformer networks we use, i.e., Vanilla BERT and some of its variants. Then, we discuss the training procedure adopted in training the transformer networks and our proposed compression/decompression technique. Details about the evaluation metrics and the baselines used conclude the section.

5.1 Datasets

We test PreTTR on two datasets, namely TREC WebTrack 2012 and TREC Robust 2004. Table 2 summarizes some salient statistics about the two datasets.

Table 2: Datasets characteristics.

	WebTrack 2012	Robust 2004
Domain	Web	Newswire
Document collection	ClueWeb09-B	TREC Disks 4 & 5
# Queries	50	249
# Documents	$50M$	$528k$
Tokens / query	2.0	2.7
Judgments / query	321	$1.2k$

The TREC WebTrack 2012 dataset consists of web queries and relevance judgments from the ClueWeb09-B document collection. We use relevance judgments from 2012 for test and the ones from 2011 for validation. The relevance judgments available from the remaining years of the TREC WebTrack, i.e., 2009, 2010, 2013, and 2014 are used for training. Note that, while the TREC WebTrack 2009–12 have been evaluated on the ClueWeb09-B document collection, the TREC WebTrack 2013–14 have been evaluated on the ClueWeb12 [19] document collection.[3] We generate the training samples by using the corresponding document collection. This is

[3] https://lemurproject.org/clueweb09/ and https://lemurproject.org/clueweb12/.

the setup used by several other works on TREC WebTrack 2012, e.g., [19, 27].

TREC Robust 2004 consists of 249 news queries. For these experiments, we use a standard k-fold evaluation ($k = 5$) where each iteration uses three folds for training, one for validation, and a final held-out fold for testing. We perform this evaluation by using the five folds provided by Huston and Croft [20].

5.2 Transformer Networks

We use the Vanilla transformer model from [27]. This model yields comparable performance to other leading formulations, while being simpler, e.g., no paragraph segmentation required, as is needed by `FirstP`/`MaxP`/`SumP` [4], or alternative training datasets and sentence segmentation, as required by the system of Yang et al. [48]. Vanilla BERT encodes as much of the document as possible (adhering to the transformer maximum input length constraint), and averages the classification embeddings when multiple document segments are required. We employ the same optimal hyper-parameters for the model presented in [27]. For our primary experiments, we use the pretrained `bert-base-uncased` [8]. We do not test with the `large` variants of BERT because the larger model exhibits only marginal gains for ranking tasks, while being considerably more expensive to run [31]. To show the generality of our approach we present tests conducted also for other pretrained transformers in Section 6.5: a version of BERT that was more effectively pre-trained, i.e., RoBERTa [25] (`roberta-base`) and a smaller (distilled) version of BERT, i.e., DistilBERT [36] (`distilbert-base-uncased`).

5.3 Training

We train all transformer models using pairwise softmax loss [7] and the Adam optimizer [23] with a learning rate of 2×10^{-5}. We employ a batch size of 16 pairs of relevant and non-relevant documents with gradient accumulation. Training pairs are selected randomly from the top-ranked documents in the training set, where documents that are labeled as relevant are treated as positive, and other top-ranked documents are considered negative. Every 32 batches, the model is validated, and the model yielding the highest performance on the validation set is selected for final evaluation.

For training the document term compressor/decompressor (as described in Section 4.2), we use the Wikipedia text from the TREC Complex Answer Retrieval (CAR) dataset [9] (version 2.0 release). This dataset was chosen because it overlaps with the data on which BERT was originally trained on, i.e., Wikipedia, and was used both for evaluation of passage ranking approaches [30] and as a weak supervision dataset for training neural models [28]. We sample text pairs using combinations of headings and paragraphs. Half the pairs use the heading associated with the paragraph, and the other half use a random heading from a different article, akin to the next sentence classification used in BERT pre-training. The compression and decompression parameters (W_{comp}, $\mathbf{b}_{comp}$, W_{decomp}, and $\mathbf{b}_{decomp}$) are trained to minimize the difference in attention scores, as formulated in Eq. (2). We found that the compressor training process converged by $2M$ samples.

5.4 Evaluation

Since the transformer network is employed as a final-stage re-ranker, we evaluate the performance of our approach on each dataset using two precision-oriented metrics. Our primary metric for both datasets is P@20 (also used for model validation). Following the evaluation convention from prior work [27], we use ERR@20 for TREC WebTrack 2012 and nDCG@20 for TREC Robust 2004 as secondary metrics.

We also evaluate the query-time latency of the models. We conduct these experiments using commodity hardware: one GeForce GTX 1080 Ti GPU. To control for factors such as disk latency, we assume the model and term representations are already loaded in the main memory. In other words, we focus on the impact of the model computation itself. However, the time spent moving the data to and from the GPU memory is included in the time.

5.5 Baselines

The focus of this work is to reduce the query-time latency of using Vanilla transformer models, which are among the state-of-the-art neural ranking approaches. Thus, our primary baseline is the unmodified Vanilla transformer network. To put the results in context, we also include the BM25 results tuned on the same training data. We tune BM25 using grid search with Anserini's implementation [46], over k_1 in the range of 0.1–4.0 (by 0.1) and b in the range of 0.1–1.0 (by 0.1). We also report results for CEDR-KNRM [27], which outperform the Vanilla transformer approaches. However, it come with its own query-time challenges. Specifically, since it uses the term representations from every layer of the transformer, this would require considerably more storage. To keep our focus on the typical approach, i.e., using the `[CLS]` representation for ranking, we leave it to future work to investigate ways in which to optimize the CEDR model.[4] We also report results for Birch [49], which exploits transfer learning from the TREC Microblog dataset. To keep the focus of this work on the effect of pre-computation, we opt to evaluate in the single-domain setting.

6 RESULTS AND DISCUSSION

We report the results of a comprehensive experimental evaluation of the proposed PreTTR approach. In particular, we aim at investigating the following research questions:

- RQ1 What is the impact of PreTTR on the effectiveness of the Vanilla BERT transformer network in ad-hoc ranking? (Section 6.1)
- RQ2 What is the impact of the token representation compression on the effectiveness of PreTTR? (Section 6.2)
- RQ3 What is the impact of the proposed PreTTR approach on the efficiency of Vanilla BERT when deployed as a second stage re-ranker? (Section 6.3)
- RQ4 What is the impact of PreTTR when applied to first $n-1$ layers of a transformer network? (Section 6.4)
- RQ5 What is the impact of PreTTR when applied to different transformer networks such as RoBERTA and DistilBERT? (Section 6.5)

[4] We note that techniques such as LSH hashing can reduce the storage requirements for CEDR, as it uses the representations to compute query-document similarity matrices, as demonstrated by [21].

6.1 Precomputing Transformer Term Representations

To answer RQ1 we first evaluate the effect of the precomputation of term representations. Table 3 provides a summary of the ranking performance of PreTTR-based Vanilla BERT at layer l. At lower values of l, the ranking effectiveness remains relatively stable, despite some minor fluctuations. We note that these fluctuations are not statistically significant when compared with the base model (paired t-test, 99% confidence interval) and remain considerably higher than the tuned BM25 model. We also tested using a two one-sided equivalence (TOST) and found similar trends (i.e., typically the the significant differences did not exhibit significant equivalence.) In the case of TREC WebTrack 2012, the model achieves comparable P@20 performance w.r.t. the base model with only a single transformer layer (12), while the first 11 layers are precomputed. Interestingly, the ERR@20 suffers more than P@20 as more layers are precomputed. This suggests that the model is able to identify generally-relevant documents very effectively with only a few transformer layers, but more are required to be able to identify the subtleties that contribute to greater or lesser degrees of relevance. Although it would ideally be best to have comparable ERR@20 performance in addition to P@20, the substantial improvements that this approach offers in terms of query-time latency (see Section 6.3) may make the trade-off worth it, depending on the needs of the application.

On the TREC Robust 2004 newswire collection, precomputing the first 10 layers yields comparable P@20 performance w.r.t. the base model. Interestingly, although $l = 11$ yields a relatively effective model for WebTrack, Robust performance significantly suffers in this setting, falling well below the BM25 baseline. We also observe a significant drop in nDCG@20 performance at $l = 8$, while P@20 performance remains stable until $l = 11$. This is similar to the behavior observed on WebTrack: as more layers are precomputed, the model has a more difficult time distinguishing graded relevance.

We observe that the highest-performing models (metric in bold) are not always the base model. However, we note that these scores do not exhibit statistically significant differences when compared to the base model.

In summary, we answer RQ1 by showing that Vanilla BERT can be successfully trained by limiting the interaction between query terms and document terms, and that this can have only a minimal impact on ranking effectiveness, particularly in terms in the precision of top-ranked documents. This is an important result because it shows that document term representations can be built independently of the query at index time.

Table 3: Breakdown of ranking performance when using a PreTTR-based Vanilla BERT ranking, joining the encodings at layer l. Statistically significant differences with the base model are indicated by ↓ (paired t-test by query, $p < 0.01$).

	WebTrack 2012		Robust 2004	
Ranker	P@20	ERR@20	P@20	nDCG@20
Base	**0.3460**	0.2767	0.3784	0.4357
$l = 1$	0.3270	**0.2831**	0.3851	**0.4401**
$l = 2$	0.3170	0.2497	0.3821	0.4374
$l = 3$	0.3440	0.2268	**0.3859**	0.4386
$l = 4$	0.3280	0.2399	0.3701	0.4212
$l = 5$	0.3180	0.2170	0.3731	0.4214
$l = 6$	0.3270	0.2563	0.3663	0.4156
$l = 7$	0.3180	0.2255	0.3656	0.4139
$l = 8$	0.3140	0.2344	0.3636	↓ 0.4123
$l = 9$	0.3130	0.2297	0.3644	↓ 0.4106
$l = 10$	0.3360	0.2295	0.3579	↓ 0.4039
$l = 11$	0.3380	↓ 0.1940	↓ 0.2534	↓ 0.2590
Tuned BM25	0.2370	0.1418	0.3123	0.4140
Vanilla BERT [27]	-	-	0.4042	0.4541
CEDR-KNRM [27]	-	-	0.4667	0.5381
Birch [49]	-	-	0.4669	0.5325

6.2 Term Representation Compression

To answer RQ2, we run the Vanilla BERT model with varying sizes e of the compressed embedding representations over the combination layers l that give the most benefit to query latency time (i.e., $l = 7, 8, 9, 10, 11$). Layers $l \leq 6$ are not considered because they provide less computational benefit (taking about one second or more per 100 documents, see Section 6.3). See Table 4 for a summary of the results on TREC WebTrack 2012 and Robust 2004. We find that the representations can usually be compressed down to at least $e = 256$ (67% of the original dimension of 768) without substantial loss in ranking effectiveness. In Robust, we observe a sharp drop in performance at $e = 128$ (83% dimension compression) at layers 7–10. There is no clear pattern for which compression size is most effective for WebTrack 2012. Note that these differences are generally not statistically significant. This table shows that, to a point, there is a trade-off between the size of the stored representations and the effectiveness of the ranker.

Without any intervention, approximately 112TB of storage would be required to store the full term vectors for ClueWeb09-B (the document collection for TREC WebTrack 2012). For web collections, this can be substantially reduced by eliminating undesirable pages, such as spam. Using recommended settings for the spam filtering approach proposed by Cormack et al. [3] for ClueWeb09-B, the size can be reduced to about 34TB. Using our compression/decompression approach, the storage needed can be further reduced, depending on the trade-off of storage, query-time latency, and storage requirements. If using a dimension $e = 128$ for the compressed representation (with no statistically significant differences in effectiveness on WebTrack), the size is further reduced to 5.7TB, which yields a 95% of space reduction. We also observed that there is little performance impact by using 16-bit floating point representations, which further reduces the space to about 2.8TB. Although this is still a tall order, it is only about 2.5% of the original size, and in the realm of reasonable possibilities. We leave it to future work to investigate further compression techniques, such as kernel density estimation-based quantization [38].

Since the size scales with the number of documents, the storage requirements are far less for smaller document collections such as newswire. Document representations for the TREC Disks 4 & 5 (the document collection for the Robust 2004) can be stored in about

Table 4: Ranking performance at various compression sizes. Statistically significant increases and decreases in ranking performance (compared to the model without compression) are indicated with ↑ and ↓, respectively (paired t-test by query, $p < 0.01$). We mark columns with * to indicate cases in which the uncompressed model (none) significantly underperforms the Base model performance (from Table 3).

TREC WebTrack 2012

	P@20					ERR@20				
Compression	$l = 7$	$l = 8$	$l = 9$	$l = 10$	$l = 11$	$l = 7$	$l = 8$	$l = 9$	$l = 10$	$^*l = 11$
(none)	0.3180	0.3140	0.3130	**0.3360**	**0.3380**	0.2255	**0.2344**	0.2297	**0.2295**	0.1940
$e = 384$ (50%)	**0.3430**	**0.3260**	0.2980	**0.3360**	0.3090	0.2086	0.2338	0.1685	0.2233	**0.2231**
$e = 256$ (67%)	0.3380	0.3120	↑ **0.3440**	0.3260	0.3250	↑ **0.2716**	0.2034	↑ **0.2918**	0.1909	0.2189
$e = 128$ (83%)	0.3100	0.3210	0.3320	0.3220	0.3370	0.2114	0.2234	0.2519	0.2239	0.2130

TREC Robust 2004

	P@20					nDCG@20				
Compression	$l = 7$	$l = 8$	$l = 9$	$l = 10$	$^*l = 11$	$l = 7$	$^*l = 8$	$^*l = 9$	$^*l = 10$	$^*l = 11$
(none)	**0.3656**	**0.3636**	**0.3644**	**0.3579**	0.2534	**0.4139**	**0.4123**	**0.4106**	**0.4039**	0.2590
$e = 384$ (50%)	0.3587	↓ 0.3369	↓ 0.3435	0.3522	**0.2687**	0.4098	↓ 0.3720	↓ 0.3812	0.3895	↑ **0.2807**
$e = 256$ (67%)	↓ 0.2950	0.3623	↓ 0.2695	0.3535	0.2635	↓ 0.3130	0.4074	↓ 0.2753	0.3983	0.2694
$e = 128$ (83%)	↓ 0.2461	↓ 0.2530	↓ 0.2499	↓ 0.2607	0.2655	↓ 0.2454	↓ 0.2568	↓ 0.2533	↓ 0.2608	0.2713

Table 5: Vanilla BERT query-time latency measurements for re-ranking the top 100 documents on TREC WebTrack 2012 and TREC Robust 2004. The latency is broken down into time to compute query representations up through layer l, the time to decompress document term representations, and the time to combine the query and document representations from layer $l + 1$ to layer n. The $l = 11$ setting yields a 42× speedup for TREC WebTrack, while not significantly reducing the ranking performance.

	TREC WebTrack 2012					Robust04
Ranker	Total	Speedup	Query	Decom.	Combine	Total
Base	1.941s	(1.0×)	-	-	-	2.437s
$l = 1$	1.768s	(1.1×)	**2ms**	10ms	1.756s	2.222s
$l = 2$	1.598s	(1.2×)	3ms	10ms	1.585s	2.008s
$l = 3$	1.423s	(1.4×)	5ms	10ms	1.409s	1.792s
$l = 4$	1.253s	(1.5×)	6ms	10ms	1.238s	1.575s
$l = 5$	1.080s	(1.8×)	7ms	10ms	1.063s	1.356s
$l = 6$	0.906s	(2.1×)	9ms	10ms	0.887s	1.138s
$l = 7$	0.735s	(2.6×)	10ms	10ms	0.715s	0.922s
$l = 8$	0.562s	(3.5×)	11ms	10ms	0.541s	0.704s
$l = 9$	0.391s	(5.0×)	12ms	10ms	0.368s	0.479s
$l = 10$	0.218s	(8.9×)	14ms	10ms	0.194s	0.266s
$l = 11$	**0.046s**	**(42.2×)**	15ms	10ms	**0.021s**	**0.053s**

195GB, without any filtering and using the more effective $e = 256$ for the dimension of the compressed representation.

In summary, regarding RQ2, we show that, through our compression technique, one can reduce the storage requirements of PreTTR. With a well-trained compression and decompression weights, this can have minimal impact on ranking effectiveness.

6.3 Re-ranking Efficiency

The reduction of the re-ranking latency achieved by our proposed PreTTR is considerable. To answer RQ3, in Table 5 we report an analysis of the re-ranking latency of PreTTR-based Vanilla BERT when precomputing the token representations at a specific layer l and a comparison against the base model, i.e., Vanilla BERT. Without our approach, re-ranking the top 100 results for a query using Vanilla BERT takes around 2 seconds. Instead, when using PreTTR-based Vanilla BERT at layer $l = 11$, which yields comparable P@20 performance to the base model on the TREC WebTrack 2012 collection, the re-ranking process takes 46 milliseconds for 100 documents, i.e., we achieve a 42.0× speedup. One reason this performance is achievable is because the final layer of the transformer network does not need to compute the representations for each token; only the representations for the `[CLS]` token are needed, since it is the only token used to compute the final ranking score. Thus, the calculation of a full self-attention matrix is not required. Since the `[CLS]` representation is built in conjunction with the query, it alone can contain a summary of the query terms. Furthermore, since the query representation in the first l layers is independent of the document, these representations are re-used among all the documents that are re-ranked. Of the time spent during re-ranking for $l = 11$, 32% of the time is spent building the query term representation, 21% of the time is spent decompressing the document term representations, and the remainder of the time is spent combining the query and document representations. Moreover, when using PreTTR-based Vanilla BERT at layer $l = 10$, the transformer network needs to perform a round of computations on all the term representations. Nevertheless, in this case, our PreTTR approach leads to a substantial speedup of 8.9× w.r.t. Vanilla BERT. We also observe that the time to decompress the term representations (with $e = 256$) remains a constant overhead, as expected. We observe a similar trend when timing the performance of Robust 2004, though we would recommend using $l \leq 10$ for this dataset, as $l = 11$ performs poorly in terms of ranking effectiveness. Nonetheless, at $l = 10$, Robust achieves a 9.2× speedup, as compared to the full model.

In summary, regarding RQ3, we show that the PreTTR approach can save a considerable amount of time at query-time, as compared to the full Vanilla BERT model. These time savings can make it practical to run transformer-based rankers in a real-time query environment.

6.4 Single Layer Ranking ($l = 11$)

We answer RQ4 by highlighting a first interesting difference between the WebTrack and the Robust ranking performance: the effectiveness at $l = 11$ (Table 3). For WebTrack, the performance is comparable in terms of P@20, but suffers in terms of ERR@20. For Robust, the performance suffers drastically. We attribute this to differences in the dataset characteristics. First, let us consider what happens in the $l = 11$ case. Since it is the final layer and only the representation of the [CLS] token is used for ranking, the only attention comparisons that matter are between the [CLS] token and every other token (not a full comparison between every pair of tokens, as is done in other layers). Thus, a representation of the entire query must be stored in the [CLS] representation from layer 11 to provide an effective comparison with the remainder of the document, which will have no contribution from the query. Furthermore, document token representations will need to have their context be fully captured in a way that is effective for the matching of the [CLS] representation. Interestingly, this setting blurs the line between representation-focused and interaction-focused neural models.

Now we will consider the characteristics of each dataset. From Table 2, we find that the queries in the TREC WebTrack 2012 are typically shorter (mean: 2.0, median: 2, stdev: 0.8) than those from Robust (mean: 2.7, median: 3, stdev: 0.7). This results in queries that are more qualified, and may be more difficult to successfully represent in a single vector.

To answer RQ4, we observe that the ranking effectiveness when combining with only a single transformer layer can vary depending on dataset characteristics. We find that in web collections (an environment where query-time latency is very important), it may be practical to use PreTTR in this way while maintaining high precision of the top-ranked documents.

6.5 PreTTR for Other Transformers

Numerous pre-trained transformer architectures exist. We now answer RQ5 by showing that PreTTR is not only effective on BERT, but its ability of reducing ranking latency by preserving quality holds also on other transformer variants. We investigate both the popular RoBERTa [25] model and the DistilBERT [36] model. These represent a model that uses a more effective pre-training process, and a smaller network size (via model distillation), respectively. Results for this experiment are shown in Table 6. We first observe that the unmodified RoBERTa model performs comparably with the BERT model, while the DistilBERT model performs slightly worse. This suggests that model distillation alone may not be a suitable solution to address the poor query-time ranking latency of transformer networks. With each value of l, we observe similar behavior to BERT: P@20 remains relatively stable, while ERR@20 tends to degrade. Interestingly, at $l = 2$ DistilBERT's ERR@20 performance peaks at 0.2771. However, this difference is not statistically significant, and thus we cannot assume it is not due to noise.

We tested the query-time latency of RoBERTa and DistilBERT in the same manner as described in Section 6.3. With 12 layers and a similar neural architecture, RoBERTa exhibited similar speedups as BERT, with up to a 56.3× speedup at $l = 11$ (0.041s per 100 documents, down from 1.89s). With only 6 layers, the base DistilBERT model was faster (0.937s), and was able to achieve a speedup of 24.1× with $l = 5$ (0.035s).

Table 6: WebTrack 2012 using two other Vanilla transformer architectures: RoBERTa and DistilBERT. Note that DistilBERT only has 6 layers; thus we only evaluate $l \in [1, 5]$ for this model. There are no statistically significant differences between the Base Model and any of the PreTTR variants (paired t-test, $p < 0.01$).

	RoBERTA [25]		DistilBERT [36]	
Ranker	P@20	ERR@20	P@20	ERR@20
Base	0.3370	0.2609	0.3110	0.2293
$l = 1$	0.3380	**0.2796**	0.3220	0.1989
$l = 2$	0.3370	0.2207	0.3340	**0.2771**
$l = 3$	0.3530	0.2669	0.3070	0.1946
$l = 4$	**0.3620**	0.2647	0.3350	0.2281
$l = 5$	0.2950	0.1707	**0.3350**	0.2074
$l = 6$	0.3000	0.1928	-	-
$l = 7$	0.3350	0.2130	-	-
$l = 8$	0.3220	0.2460	-	-
$l = 9$	0.3180	0.2256	-	-
$l = 10$	0.3140	0.1603	-	-
$l = 11$	0.3210	0.2241	-	-

In summary, we show that the PreTTR approach can be successfully generalized to other transformer networks (RQ5). We observed similar trends to those we observed with BERT in two transformer variants, both in terms of ranking effectiveness and efficiency.

7 CONCLUSIONS AND FUTURE WORK

Transformer networks, such as BERT, present a considerable opportunity to improve ranking effectiveness [4, 27, 31]. However, relatively little attention has been paid to the effect that these approaches have on query execution time. In this work, we showed that these networks can be trained in a way that is more suitable for query-time latency demands. Specifically, we showed that web query execution time can be improved by up to 42× for web document ranking, with minimal impact on P@20. Although this approach requires storing term representations for documents in the collection, we proposed an approach to reduce this storage required by 97.5% by pre-training a compression/decompression function and using reduced-precision (16 bits) floating point arithmetic. We experimentally showed that the approach works across transformer architectures, and we demonstrated its effectiveness on both web and news search. These findings are particularly important for large-scale search settings, such as web search, where query-time latency is critical.

This work is orthogonal to other efforts to reign in the execution time of transformer networks. There are challenges related to the application of more advanced networks, such as CEDR [27], which require the computation or storage of additional term representations. Future work could investigate how approaches like LSH-hashing [21] could be used to help accomplish this. Furthermore, our observation that comparable ranking performance can be achieved using a compression layer raises questions about the importance of the feed-forward step in each transformer layer.

ACKNOWLEDGMENTS

Work partially supported by the ARCS Foundation. Work partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence). Work partially supported by the BIGDATAGRAPES project funded by the EU Horizon 2020 research and innovation programme under grant agreement No. 780751, and by the OK-INSAID project funded by the Italian Ministry of Education and Research (MIUR) under grant agreement No. ARS01_00917.

REFERENCES

[1] Arash Ardakani, Zhengyun Ji, Sean C Smithson, Brett H Meyer, and Warren J Gross. 2019. Learning Recurrent Binary/Ternary Weights. In *International Conference on Learning Representations*. http://arxiv.org/abs/1809.11086

[2] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What Does BERT Look At? An Analysis of BERT's Attention. In *BlackBoxNLP @ ACL*. http://arxiv.org/abs/1906.04341

[3] Gordon V. Cormack, Mark D. Smucker, and Charles L. A. Clarke. 2010. Efficient and effective spam filtering and re-ranking for large web datasets. *Information Retrieval* 14 (2010), 441–465.

[4] Zhuyun Dai and Jamie Callan. 2019. Deeper Text Understanding for IR with Contextual Neural Language Modeling. In *SIGIR*.

[5] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. 2018. Convolutional Neural Networks for Soft-Matching N-Grams in Ad-hoc Search. In *WSDM*. ACM Press, Marina Del Rey, CA, USA, 126–134. http://dl.acm.org/citation.cfm?doid=3159652.3159659

[6] Domenico Dato, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2016. Fast Ranking with Additive Ensembles of Oblivious and Non-Oblivious Regression Trees. *ACM Transactions on Information Systems* 35, 2 (2016), 15:1–15:31.

[7] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. 2017. Neural Ranking Models with Weak Supervision. In *SIGIR*.

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

[9] Laura Dietz and Ben Gamari. 2017. TREC CAR: A Data Set for Complex Answer Retrieval. (2017). http://trec-car.cs.unh.edu Version 2.0.

[10] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *CIKM*. 55–64. http://arxiv.org/abs/1711.08611

[11] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *ICLR*.

[12] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. http://arxiv.org/abs/1510.00149

[13] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

[14] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).

[15] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *CoRR* abs/1503.02531 (2015). arXiv:1503.02531 http://arxiv.org/abs/1503.02531

[16] Sebastian Hofstätter and Allan Hanbury. 2019. Let's measure run time! Extending the IR replicability infrastructure to include performance aspects. In *OSIRRC@SIGIR*.

[17] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*.

[18] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.

[19] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. 2017. PACRR: A Position-Aware Neural IR Model for Relevance Matching. In *EMNLP*.

[20] Samuel Huston and W Bruce Croft. 2014. Parameters learned in the comparison of retrieval models using term dependencies. *Technical Report* (2014).

[21] Shiyu Ji, Jinjin Shao, and Tao Yang. 2019. Efficient Interaction-based Neural Ranking with Locality Sensitive Hashing. In *WWW*.

[22] Xiaoqi Jiao, Y. Yin, Lifeng Shang, Xin Jiang, Xusong Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. TinyBERT: Distilling BERT for Natural Language Understanding. *ArXiv* abs/1909.10351 (2019).

[23] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[24] Francesco Lettich, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2018. Parallel Traversal of Large Ensembles of Decision Trees. *IEEE Transactions on Parallel and Distributed Systems* (2018), 14. https://doi.org/10.1109/TPDS.2018.2860982

[25] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar S. Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke S. Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *ArXiv* abs/1907.11692 (2019).

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

[27] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. CEDR: Contextualized Embeddings for Document Ranking. In *SIGIR*.

[28] Sean MacAvaney, Andrew Yates, Kai Hui, and Ophir Frieder. 2019. Content-Based Weak Supervision for Ad-Hoc Re-Ranking. In *SIGIR*.

[29] Irina Matveeva, Christopher J. C. Burges, Timo Burkard, Andy Laucius, and Leon Wong. 2006. High accuracy retrieval with multiple nested ranker. In *SIGIR*.

[30] Federico Nanni, Bhaskar Mitra, Matt Magnusson, and Laura Dietz. 2017. Benchmark for Complex Answer Retrieval. In *ICTIR*.

[31] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *ArXiv* abs/1901.04085 (2019).

[32] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document Expansion by Query Prediction. *ArXiv* abs/1904.08375 (2019).

[33] Wei Pan, Hao Dong, and Yike Guo. 2016. DropNeuron: Simplifying the Structure of Deep Neural Networks. *CoRR* abs/1606.07326 (2016). arXiv:1606.07326 http://arxiv.org/abs/1606.07326

[34] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. *Improving language understanding by generative pre-training*. Technical Report. OpenAI.

[35] Corby Rosset, Damien Jose, Gargi Ghosh, Bhaskar Mitra, and Saurabh Tiwary. 2018. Optimizing Query Evaluations Using Reinforcement Learning for Web Search. In *SIGIR*.

[36] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *Workshop on Energy Efficient Machine Learning and Cognitive Computing @ NeuIPS*.

[37] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2019. Green AI. *ArXiv* abs/1907.10597 (2019).

[38] Sanghyun Seo and Juntae Kim. 2019. Efficient Weights Quantization of Convolutional Neural Networks Using Kernel Density Estimation based Non-uniform Quantizer. *Appl. Sci* (2019).

[39] Mohammad Shoeybi, Mostofa Ali Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *ArXiv* abs/1909.08053 (2019).

[40] 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* abs/1903.12136 (2019).

[41] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2018. Efficient Query Processing for Scalable Web Search. *Foundations and Trends in Information Retrieval* 12, 4–5 (2018), 319–492.

[42] 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 *NeuIPS*. http://arxiv.org/abs/1706.03762

[43] Lidan Wang, Jimmy Lin, and Donald Metzler. 2011. A cascade ranking model for efficient ranked retrieval. In *SIGIR*.

[44] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. In *SIGIR*. 55–64. http://arxiv.org/abs/1706.06613 arXiv: 1706.06613.

[45] Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. 2018. Alternating Multi-bit Quantization for Recurrent Neural Networks. In *International Conference on Learning Representations*. https://arxiv.org/abs/1802.00150

[46] Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the Use of Lucene for Information Retrieval Research. In *SIGIR*.

[47] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-End Open-Domain Question Answering with BERTserini. In *NAACL-HLT*.

[48] Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Simple Applications of BERT for Ad Hoc Document Retrieval. *ArXiv* abs/1903.10972 (2019).

[49] Zeynep Akkalyoncu Yilmaz, Shengjin Wang, Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Applying BERT to Document Retrieval with Birch. In *EMNLP/IJCNLP*.

[50] 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 *CIKM*.