Learning to Rank in the Age of Muppets: Effectiveness–Efficiency Tradeoffs in Multi-Stage Ranking

Yue Zhang,[1*] **Chengcheng Hu,**[2*] **Yuqi Liu,**[2*] **Hui Fang,**[1] **Jimmy Lin**[2]

[1] Department of Electrical and Computer Engineering, University of Delaware
[2] David R. Cheriton School of Computer Science, University of Waterloo

{zhangyue, hfang}@udel.edu, {c45hu, y899liu, jimmylin}@uwaterloo.ca

Abstract

It is well known that rerankers built on pretrained transformer models such as BERT have dramatically improved retrieval effectiveness in many tasks. However, these gains have come at substantial costs in terms of efficiency, as noted by many researchers. In this work, we show that it is possible to retain the benefits of transformer-based rerankers in a multi-stage reranking pipeline by first using feature-based learning-to-rank techniques to reduce the number of candidate documents under consideration without adversely affecting their quality in terms of recall. Applied to the MS MARCO passage and document ranking tasks, we are able to achieve the same level of effectiveness, but with up to 18× increase in efficiency. Furthermore, our techniques are orthogonal to other methods focused on accelerating transformer inference, and thus can be combined for even greater efficiency gains. A higher-level message from our work is that, even though pretrained transformers dominate the modern IR landscape, there are still important roles for "traditional" LTR techniques, and that we should not forget history.

1 Introduction

Pretrained transformers such as BERT (Devlin et al., 2019) have dramatically increased retrieval effectiveness in many tasks across a multitude of domains (Lin et al., 2020a). Nevertheless, in a standard "retrieve-then-rerank" setup, the application of pretrained transformer-based rerankers incurs large computational costs and long query latencies, making those rerankers unrealistic for many real-world applications. For example, according to the ColBERT paper (Khattab and Zaharia, 2020), reranking 1000 hits from the MS MARCO passage dataset takes 32.9 seconds per query. Other researchers have noted the computational costs of transformer-based rankers (Hofstätter and Hanbury, 2019), and this realization has compelled the field to explore other approaches, for example, simplified models (Hofstätter et al., 2020; Soldaini and Moschitti, 2020; Mitra et al., 2020; MacAvaney et al., 2020; Gao et al., 2020; Jiang et al., 2020) and learned dense representations (Xiong et al., 2020; Lin et al., 2020b).

We are also motivated by the desire to reduce the computational costs of ranking with transformers, but from a different perspective. Based on the observation that neural networks in general (and transformers in particular) have largely supplanted feature-based learning to rank (LTR) in modern information retrieval, we ask the question: What, if anything, does "traditional" feature-based learning to rank have to offer in the age of muppets?[1] The subtext of this question is that we, as a field, should not forget our own history.

There are two obvious approaches to try and answer this question. The first is to simply consider transformer-based features (e.g., BERT score, ColBERT score, etc.) as yet another feature within a learning-to-rank framework—for example, with gradient boosted decision trees (Wang et al., 2020). This is not the route that we take, because this approach has less bearing on our desire to increase the efficiency of transformer-based models. Instead, we take the alternative approach of using learning-to-rank techniques as a "filtering" stage in a multi-stage ranking architecture to reduce the number of candidates under consideration by BERT. More concretely, we find that a design based on this idea achieves the same level of effectiveness as a standard retrieve-and-rerank approach using BERT, but is up to 18× faster. Other effectiveness–efficiency tradeoffs are possible, giving developers a rich design space to build systems tailored to different application scenarios.

The contribution of this work is to demon-

* Equal contribution

[1] Muppets being a whimsical way to refer to BERT and related transformer models.

Proceedings of the 2nd Workshop on Simple and Efficient Natural Language Processing, pages 64–73
November 10, 2021. ©201 Association for Computational Linguistics

strate that by inserting a "filtering" learning-to-rank stage prior to BERT-based reranking, we can control effectiveness–efficiency tradeoffs in a manner that makes deployments in real-world applications more practical. We emphasize that this work is orthogonal to other methods that directly attempt to accelerate inference, e.g., via knowledge distillation, early exits, model simplifications, etc. It is likely that our gains are cumulative with respect to these enhancements.

2 Methods

2.1 Multi-Stage Ranking

We adopt a standard formulation of multi-stage ranking, which comprises a candidate generation stage H_0 (also called first-stage retrieval), followed by a pipeline of rerankers, denoted H_1 to H_N. Candidate generation is typically accomplished via keyword search against an inverted index, which retrieves k_0 hits from the corpus. Each subsequent stage H_n receives a ranked list R_{n-1} comprising k_{n-1} candidates from the previous stage, reranks these candidates, and then passes the results to the next stage. The output of the last stage serves as the final ranked list, e.g., to be shown to the user or to be evaluated using standard tools.

In this work, we compare two designs of multi-stage ranking architectures:

BoW + BERT As a baseline, we consider the retrieve-and-rerank approach originally proposed by Nogueira and Cho (2019), which has emerged as the standard architecture for applying pretrained transformers to ranking. We notate a specific configuration of this design as BoW(k_0) + BERT, where k_0 denotes the number of candidates from bag-of-words retrieval that are then reranked by BERT. A commonly used default is BoW(1000) + BERT (Nogueira and Cho, 2019).

In addition, we also examine the docTTTTTquery document expansion technique (Nogueira et al., 2019b; Nogueira and Lin, 2019) based on predicting queries for which a text would be relevant (henceforth, just d2q for short). The predicted queries are concatenated to the end of the original text; this greatly improves BoW retrieval. We call this variant BoW_{d2q} and denote the corresponding pipeline BoW_{d2q}(1000) + BERT.

BoW + LTR + BERT This represents our proposed design of introducing a filtering stage before BERT to reduce the number of candidates under consideration. We notate a specific configuration of this design as BoW(k_0) + LTR(k_1) + BERT, where k_0 denotes the number of candidates from bag-of-words retrieval. Our LTR stage then reranks these k_0 hits to generate a new ranking of k_1 hits that are passed to the BERT stage. Similar to BoW_{d2q}, the LTR variant that extracts features from the document expansions is called LTR_{d2q}.

Given this setup, our research question then becomes an effectiveness–efficiency exploration of BoW + BERT vs. BoW + LTR + BERT (including d2q variants). We want to know, given effectiveness parity as a requirement, what degree of latency reduction we can achieve. To answer this question, we adopt the standard setting of LTR as supervised machine learning, where our objective is to maximize recall. We use features extracted from our LTR module to train a model and then apply inference on candidates from the BoW stage, passing on only the most promising ones for (expensive) neural reranking.

2.2 Learning-to-Rank Features

Our LTR features fall into four categories: term-based, score-based, proximity-based, and translation-based. These features are inspired by previous studies on LTR (Qin and Liu, 2013; Gallagher et al., 2020) and summarized in Table 1. An enumeration of all features is presented in Appendix A.

Term-based features Following previous work, we compute different term statistics including term frequency (TF), inverse document frequency (IDF), log term probability, and inverse collection term frequency. Furthermore, existing retrieval models such as BM25 are used to compute the relevance score of a term, and these scores can also be used as term-based features combining different types of statistics. To aggregate term-based statistics for all terms in a query, a number of different aggregation functions are used, e.g., max, min, sum, mean, median, and the ratio between max and min.

Score-based features Term-based statistics are not sufficient as document-level statistics such as document length are also crucial for retrieval effectiveness. Traditional BoW retrieval models (e.g., BM25, Query Likelihood, Divergence From Randomness) have proposed effective ways of combining term-based and document-based statistics, so we also include the retrieval scores of these models

Feature Category	#	Examples
Term-based	54	Max of IDF, Max of TF
Score-based	14	BM25, DFR
Proximity-based	15	Co-occurrences, BM25-TP
Translation-based	4	translation probability

Table 1: Summary of LTR features.

as features. Note that when we use sum as the aggregation function for term-based statistics, we are essentially computing the retrieval model score.

Proximity-based features Traditional retrieval models assume terms are independent and ignore their relationships, but the proximity among query terms often serves as an important relevance signal. Thus, we include features that directly capture the proximity of query terms, such as the counts of ordered and unordered co-occurrence of bigrams within different window sizes. We compute the scores of proximity-based retrieval functions, such as SDM (Metzler and Croft, 2005; Gallagher et al., 2020) and BM25-TP (Lu et al., 2015; Gallagher et al., 2020), as our features.

Translation-based features Capturing semantic relationships between a query and a document is also crucial to improving retrieval accuracy. To incorporate such features, we can use a translation model (Boytsov and Nyberg, 2020; Boytsov and Kolter, 2021) to measure the log translation probability between queries and documents. The conditional probability we need $p(q|d_n)$ is generated by the IBM Model 1 translation model, and the final query–document feature is the sum of all individual conditional query probabilities.

The extraction of all LTR features is performed at the level of tokens. In particular, both queries and documents are tokenized into a multi-field representation. They include: (1) the *raw* field, which consists of the original tokens; (2) the *stemmed* field, which includes the stemmed tokens; (3) the *subword* field, which breaks tokens into subwords; and (4) the *d2q* field, which includes the stemmed tokens from the concatenated docTTTTTquery predictions (for the d2q variants). For each query–document pair, feature extraction is repeated over all applicable fields. In total, there are 83 different features (per field) plus four translation-based features that are only available in the *raw* and *subword* fields. Table 1 summarizes our features and provides examples for each feature category.

3 Experimental Setup

Data We use the MS MARCO passage ranking dataset (Bajaj et al., 2018) for training and testing. The training set contains ∼500K queries while the development and test sets contain ∼7K queries each. On average, each query has only one positive example; negative examples are taken from BM25 results that are not otherwise judged as relevant. Since it is inefficient to use all the negative samples, we downsample to 20 negative examples per query and combine them with all the positive examples to arrive at the training data.

We additionally test on the MS MARCO document ranking task (Bajaj et al., 2018) in a zero-shot manner. For this, we segment each document into multiple passages as the neural models cannot process long documents. Specifically, we use the sliding window strategy of Pradeep et al. (2021), where the window length is ten sentences with a stride of five sentences. Retrieval is performed at the passage level, and the document score is computed based on the highest relevance score among its passages.

Implementation We use Anserini (Yang et al., 2018), an open-source IR toolkit built on Lucene, to build the indexes and retrieve top-ranked candidate passages. For first-stage retrieval, we use BM25, with parameters ($k_1 = 0.82$ and $b = 0.68$) based on the authors' recommendations to optimize for recall@1000. The retrieved candidate passages are then sent to feature extraction through Anserini's Python interface, Pyserini (Lin et al., 2021).[2]

For our LTR module, we use the LambdaMART algorithm implemented in LightGBM[3] as our model. The hyperparameters are tuned to achieve the highest recall@200 on the development set. Specifically, num_leaves is 200, learning_rate is 0.1, min_data_in_leaf is 50, max_bin is 255. We fix early stopping patience to 200 and use up to 1000 trees. Finally, we utilize batch processing and multi-threading to accelerate the processing. This helps us leverage contemporary multi-core CPUs.

For the final-stage neural reranker, we experiment with BERT-large and T5-base in the PyGaggle library fine-tuned on the MS MARCO passage data.[4] We simply use checkpoints provided by the library, as our work is not specifically focused

[2] https://github.com/castorini/pyserini
[3] https://github.com/microsoft/LightGBM
[4] https://github.com/castorini/pygaggle

Configuration	N	MRR@10	NDCG@10	Latency	
BoW(1k) + BERT	1000	0.379	0.441	9.63s	
BoW(10k) + LTR(100) + BERT	100	0.381	0.443	1.32s	(7×)
BoW(10k) + LTR$_{\text{d2q}}$(20)+ BERT	20	0.382	0.442	0.53s	(18×)
BoW(1k) + T5	1000	0.380	0.443	5.60s	
BoW(10k) + LTR(100) + T5	100	0.382	0.445	0.92s	(6×)
BoW(10k) + LTR$_{\text{d2q}}$(20) + T5	20	0.382	0.444	0.46s	(12×)
BoW$_{\text{d2q}}$(1k) + BERT	1000	0.389	0.454	9.63s	
BoW$_{\text{d2q}}$(10k) + LTR$_{\text{d2q}}$(50) + BERT	50	0.389	0.454	0.83s	(12×)
BoW$_{\text{d2q}}$(1k) + T5	1000	0.386	0.453	5.60s	
BoW$_{\text{d2q}}$(10k) + LTR$_{\text{d2q}}$(50) + T5	50	0.388	0.454	0.63s	(9×)

Table 2: The effectiveness and efficiency of different pipeline configurations on the MS MARCO passage ranking task. The effectiveness of the pipelines with additional LTR modules are statistically indistinguishable from the baselines without the LTR modules.

on final-stage neural reranking. Previous evaluations (Nogueira and Cho, 2019; Nogueira et al., 2020; Pradeep et al., 2021) have already verified that these two models serve as competitive baselines. We pad all the token sequences in the batch to have the same length and truncate them if their lengths exceed 512 tokens.

Latency Measurements When measuring latency, we used two different servers. The latency of first-stage retrieval and the LTR filtering module is measured on a server equipped with 2 Intel Xeon Platinum 8160 CPUs. The index is located on a local SSD partition. The neural model latency is measured using a 6 core server with a single Tesla V100 GPU. All latency measurements exclude the time to load data and models. Component results are summed to yield end-to-end query latency, which we normalize into a speedup value when comparing different conditions.

4 Results and Analysis

4.1 Results on Passage Ranking

Evaluation results on MS MARCO passage ranking are shown in Table 2 for the various pipeline configurations based on the notation introduced in Section 2. N represents the number of candidates reranked by the final neural reranker, which consumes most of the query time. Note that in these experiments, our goal is to obtain effectiveness parity (i.e., same level of effectiveness) while accelerating inference (i.e., reducing query latency).

The first row in each block of the table represents a baseline configuration, with N set to the common value of 1000. We present six LTR pipelines, where we choose the smallest N (and different k's) that can reach MRR@10 parity with the baseline. We conduct two-tailed paired t-tests to confirm that there are no significant effectiveness differences between results before and after inserting LTR as the filtering stage. Depending on the pipeline setup, we only need to perform neural inference on 20 to 100 candidates—precisely because of our LTR filtering. We report the per-query latency for each configuration and compute a speedup by normalizing against the latency of each baseline. Based on which neural model we use and whether we use d2q, we observe speedups ranging from 6× to 18×. That is, we can achieve comparable effectiveness with these increases in speed. Note that our baseline BoW + BERT latency is already more than 3× faster than the values reported by Khattab and Zaharia (2020) (∼33s) using the same configuration, primarily due to batch tokenization and other engineering optimizations, so the LTR pipelines are compared against a well-optimized baseline.

Table 3 shows the query latency breakdown for a few representative models. Note that latency is dominated by final-stage neural reranking latency, which scales linearly, so smaller N values (in Table 2) are more desirable. However, this is balanced by the introduction of LTR overhead, both feature extraction as well as the prediction latency itself. Nevertheless, this is a worthwhile tradeoff as we observe large speedups overall. Since T5-base is faster than BERT-large, the effect of the LTR overhead is relatively larger and thus the speedup is lower. We can see that increasing the initial k_0 for BoW from 1k to 10k is acceptable as LTR overhead remains modest.

The most time-consuming step in LTR is feature extraction, and within that, loading the forward in-

Model	Retrieval	Feature Extraction	LTR Prediction	Neural Reranking	**Total**
BoW(1k) + BERT	15	-	-	9610	**9630**
BoW(10k) + LTR(100) + BERT	120	180	40	980	**1320**
BoW(1k) + T5	15	-	-	5580	**5600**
BoW(10k) + LTR(100) + T5	120	180	40	584	**920**

Table 3: Detailed breakdown of latency (ms/query) for a few representative pipeline configurations on the MS MARCO passage ranking task.

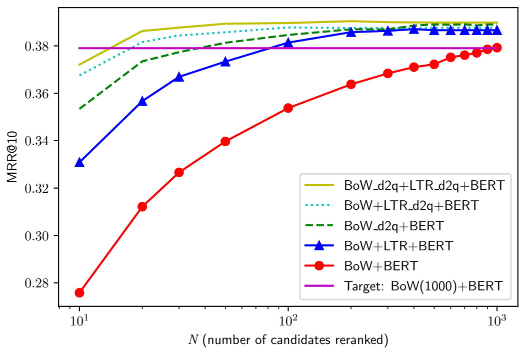

Figure 1: MRR@10 as a function of N, the number of candidates reranked by the final neural reranker, on the MS MARCO passage ranking task.

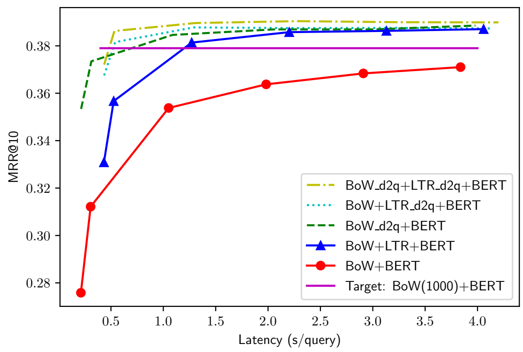

Figure 2: MRR@10 vs. reranking latency on the MS MARCO passage ranking task.

dex for each document is the most expensive single step. Our experiments show that the number of features does not affect latency substantially because we only need to load the document once; once in cache, individual feature extraction is very fast. Note that we have not spent much effort optimizing feature extraction (which is relatively inefficient Java code) and that more engineering effort, for example, optimizations proposed by Asadi and Lin (2013), are likely to further increase speedups.

Figure 1 shows the MRR@10 for our five models as a function of N (number of candidates reranked by the final neural reranker), shown on the x-axis (log scale). Effectiveness of the BoW + BERT baseline at $N = 1000$ is represented as the horizontal line—the effectiveness level we are targeting. Note that for some configurations, we needed to adjust the k_1 for the LTR stage in order to meet the desired N. We can see that our pipelines reach the target MRR@10 with much smaller values of N; reducing expensive neural reranking is where most of our speedups come from. After adding d2q predictions, we are able to achieve our target MRR@10 with an even smaller value of N.

Figure 2 shows the trade-offs between MRR@10 and query latency (the x-axis). Each curve represents a sweep over N values; we only focus on the case when N is relatively small (< 300). We can see that BoW + LTR + BERT is not helpful if the desired per-query latency is less than 0.5s, because the LTR system itself introduces overhead and leaves little time for neural reranking. For reference, without neural reranking, BoW + LTR only reaches 0.25 MRR@10 while consuming 0.34s; in contrast, spending 0.43s with BoW + BERT achieves 0.28 MRR@10. Between 0.5s and 2s, the effectiveness gap between BoW + BERT and BoW + LTR + BERT increases.

Until now, we have focused on targeting effectiveness parity. What if we're willing to sacrifice effectiveness? Figure 3 uses the BoW + LTR + BERT configuration as an example to show possible speedups and corresponding MRR@10 values if we accept lower effectiveness. We use the latency of BoW(1000) + BERT as our reference point to calculate speedups. For example, in one setting, we can achieve 0.36 MRR@10 using only 20 candidates ($N = 20$) and enjoy 17× speedup without using d2q (see Table 2).

We also conduct ablation experiments to investigate the importance of the four feature categories used in our LTR module. We use BoW(10k) + LTR(100) + BERT as our base model and assess

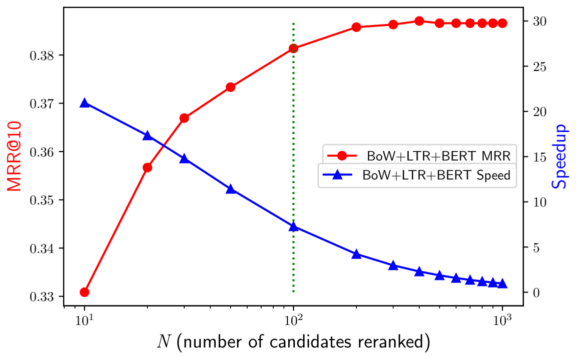

Figure 3: MRR@10 vs. speedup for different N, the number of candidates reranked by the final neural reranker, on the MS MARCO passage ranking task.

Configuration	Recall@100	MRR@10
Full Model	0.781	0.382
− Score-based	0.780 (−0.09%)	0.381 (−0.24%)
− Term-based	0.771 (−1.27%)	0.379 (−0.74%)
− Proximity-based	0.767 (−1.76%)	0.379 (−0.96%)
− Translation-based	0.743 (−4.82%)	0.371 (−3.01%)

Table 4: Feature importance ablation results on the MS MARCO passage ranking task.

feature importance by removing each category of features from the LTR module and measuring the effectiveness of the ablated reranker. Table 4 shows the absolute scores and relative differences in terms of recall@100 and MRR@10.

From these results, we can see that the score-based features appear to be the least important category of features in our model, likely because there is a lot of redundant information between score-based and term-based features. They use different arithmetic formulas to manipulate the same raw signals: term frequency, document frequency, etc. Compared with term-based features, score-based features are fewer in number. Removing score-based features has much less impact than removing term-based features.

Translation-based features appear to be the most important feature category: they make the biggest difference in effectiveness with the smallest number of features. The translation model bridges the query–document gap by modeling alternative expressions of query terms and enables a document to match query terms that are not present in the document (i.e., semantic matching).

We also evaluate each feature's importance by the total number of splits and the total gain of splits using a built-in feature in the LightGBM library. We find that all translation-based features rank in the top 10 features with the highest number of splits. This again confirms that matching alternative expressions is an important factor of why our method works. The GL2 model from the Divergence From Randomness family of scoring functions ranks first in terms of total gain of splits, suggesting that traditional bag-of-words retrieval models still form the backbone of learning to rank.

4.2 Results on Document Ranking

To explore the generality of our LTR approach, we also conduct experiments on the MS MARCO document ranking task. We emphasize here that all experiments are conducted in a zero-shot manner, over paragraph extracts from the collection, what is commonly known as the MaxP approach (Bendersky and Kurland, 2009; Dai and Callan, 2019; Akkalyoncu Yilmaz et al., 2019; Sheetrit et al., 2020). Both the final-stage neural reranker and our LTR module are trained on MS MARCO *passage* data only. However, comparing our effectiveness results with the official leaderboard reveals that our configurations are competitive compared to other single-stage rerankers.

Results are shown in Table 5, organized in the same manner as Table 2. Recall again that the goal here is also to retain effectiveness parity with respect to the reference reranker (T5 in this case). We conduct two-tailed paired t-tests to confirm that there are no statistically significant effectiveness differences between all model configurations. We see that in a zero-shot setting, speedups of nearly $3\times$ can be observed.

We have a few explanations of why the speedups here are not as impressive as in the passage ranking case. First, because we are not training a corpus-specific model, the power of the LTR module is weaker, and hence the pipeline needs to consider more candidates—larger N means more time in final-stage neural reranking, and hence less savings. Additionally, input to the neural models are on average much longer than the candidates from the passage corpus. This means that there is more text and richer signals for the transformers to extract, which correspondingly means that LTR is more impoverished. Furthermore, longer documents translate into more time spent on feature extraction in LTR. Taken together, all of these issues result in smaller speedups.

Configurlation	N	MRR@100	NDCG@10	Latency	
BoW(1k) + T5	1000	0.405	0.470	18.2s	
BoW(10k) + LTR(550) + T5	550	0.406	0.474	10.0s	(1.8×)
BoW(10k) + LTR_{d2q}(750)+ T5	750	0.405	0.470	13.7s	(1.4×)
BoW_{d2q}(1k) + T5	1000	0.409	0.476	18.2s	
BoW_{d2q}(1k) + LTR_{d2q}(350)+ T5	350	0.409	0.476	6.4s	(2.8×)

Table 5: The effectiveness and efficiency of different pipeline configurations on the MS MARCO document ranking task. The effectiveness of the pipelines with additional LTR modules are statistically indistinguishable from the baselines without the LTR modules.

5 Related Work

At a high level, the entire premise of our work *is* the point of multi-stage ranking, in that the architecture evolved to achieve a good balance between effectiveness and efficiency in end-to-end retrieval. Motivated by the observation, dating back more than a decade, that effective techniques are often computationally expensive, multi-stage retrieval architectures control latency by applying expensive techniques over only the most promising candidates (Wang et al., 2011). This is often operationalized as optimizing for recall in the earlier stages of the pipeline. Specifically in the context of transformers, multi-stage neural pipelines have been explored in the past by many researchers (Nogueira et al., 2019a; Soldaini and Moschitti, 2020; Matsubara et al., 2020; Pradeep et al., 2021). The key difference in our work is the (re-)introduction of "traditional" feature-based learning-to-rank approaches alongside neural models. This aligns with our broader goal of investigating how learning to rank might contribute to modern retrieval approaches dominated by neural models.

The computational costs associated with ranking using pretrained transformers can be reduced in various ways. We can accelerate inference using smaller or simpler models. Gao et al. (2020) use distillation to transfer knowledge captured in a larger model into a smaller model, achieving substantial speedups with minimal effectiveness loss. Hofstätter et al. (2020) propose a simpler transformer model to capture contextual information that trades effectiveness for much faster inference. Additional examples of this approach include Mitra et al. (2020) and MacAvaney et al. (2020). An alternative is to introduce early-exit optimizations, as in Soldaini and Moschitti (2020) and Xin et al. (2020). Further speedups can be gained by making modifications to the backbone transformer model, as in Sanh et al. (2020). The key point is that our proposed LTR filtering module achieves speedups in a manner that is orthogonal to the methods discussed here, which focus on directly accelerating transformer inference. Thus, these approaches can be combined with our method for even greater efficiency gains.

6 Conclusions

The "retrieve-then-rerank" approach with transformers has been demonstrated to be effective in many IR tasks, but poor efficiency makes it less attractive for real-world applications. Our goal is to increase the efficiency of the entire pipeline but at the same time maintain the same level of effectiveness: this is achieved by a feature-based learning-to-rank module that filters candidates prior to neural reranking. On the MS MARCO passage ranking task, we observe up to 18× speedup without degradation in terms of MRR@10. In a zero-shot setting on the MS MARCO document ranking task, we can achieve 3× speedup. These results demonstrate that in this age of muppets dominated by transformers and other neural models, learning-to-rank techniques can still be useful. Despite LTR "falling out of fashion", we should not "throw the baby out with the bath water".

One key point that bears emphasis, and one promising direction for future research is that our work can be combined with other approaches that directly accelerate inference in neural models. We expect speedups to be cumulative since we tackle efficiency issues from an orthogonal perspective.

Acknowledgements

The research conducted at the University of Waterloo was supported in part by the Canada First Research Excellence Fund and the Natural Sciences and Engineering Research Council (NSERC) of Canada; computational resources were provided by Compute Canada. The research conducted at the University of Delaware was supported as part of the Center for Plastics Innovation, an Energy Frontier Research Center, funded by the U.S. Department of Energy (DOE), Office of Science, Basic Energy Sciences (BES), under Award Number DE-SC0021166.

References

Zeynep Akkalyoncu Yilmaz, Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Cross-domain modeling of sentence-level evidence for document retrieval. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3490–3496, Hong Kong, China.

Nima Asadi and Jimmy Lin. 2013. Document vector representations for feature extraction in multi-stage document ranking. *Information Retrieval*, 16(6):747–768.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2018. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *arXiv:1611.09268v3*.

Michael Bendersky and Oren Kurland. 2009. Utilizing passage-based language models for ad hoc document retrieval. *Information Retrieval*, 13:157–187.

Leonid Boytsov and Zico Kolter. 2021. Exploring classic and neural lexical translation models for information retrieval: Interpretability, effectiveness, and efficiency benefits. *arXiv:2102.06815*.

Leonid Boytsov and Eric Nyberg. 2020. Flexible retrieval with NMSLIB and FlexNeuART. *arXiv:2010.14848*.

Zhuyun Dai and Jamie Callan. 2019. Deeper text understanding for IR with contextual neural language modeling. In *Proceedings of the 42nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2019)*, pages 985–988, Paris, France.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota.

Luke Gallagher, Antonio Mallia, J. Shane Culpepper, Torsten Suel, and B. Barla Cambazoglu. 2020. Feature extraction for large-scale text collections. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, pages 3015–3022.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2020. Understanding BERT rankers under distillation. In *Proceedings of the 2020 ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR 2020)*.

Sebastian Hofstätter and Allan Hanbury. 2019. Let's measure run time! Extending the IR replicability infrastructure to include performance aspects. In *Proceedings of the Open-Source IR Replicability Challenge (OSIRRC 2019): CEUR Workshop Proceedings Vol-2409*, pages 12–16, Paris, France.

Sebastian Hofstätter, Markus Zlabinger, and Allan Hanbury. 2020. Interpretable & time-budget-constrained contextualization for re-ranking. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, Santiago de Compostela, Spain.

Jyun-Yu Jiang, Chenyan Xiong, Chia-Jung Lee, and Wei Wang. 2020. Long document ranking with query-directed sparse transformer. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4594–4605.

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 (SIGIR 2020)*, pages 39–48.

Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 2356–2362.

Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2020a. Pretrained transformers for text ranking: BERT and beyond. *arXiv:2010.06467*.

Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2020b. Distilling dense representations for ranking using tightly-coupled teachers. *arXiv:2010.11386*.

Xiaolu Lu, Alistair Moffat, and J. Shane Culpepper. 2015. On the cost of extracting proximity features for term-dependency models. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 293–302.

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 2020)*, pages 49–58.

Yoshitomo Matsubara, Thuy Vu, and Alessandro Moschitti. 2020. Reranking for efficient transformer-based answer selection. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2020)*, pages 1577–1580.

Donald Metzler and W. Bruce Croft. 2005. A Markov random field model for term dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2005)*, pages 472–479, Salvador, Brazil.

Bhaskar Mitra, Sebastian Hofstätter, Hamed Zamani, and Nick Craswell. 2020. Conformer-kernel with query term independence for document retrieval. *arXiv:2007.10434*.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with BERT. *arXiv:1901.04085*.

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document ranking with a pretrained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 708–718.

Rodrigo Nogueira and Jimmy Lin. 2019. From doc2query to docTTTTTquery.

Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019a. Multi-stage document ranking with BERT. *arXiv:1910.14424*.

Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019b. Document expansion by query prediction. *arXiv:1904.08375*.

Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2021. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models. *arXiv:2101.05667*.

Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 datasets. *arXiv:1306.2597*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. In *Proceedings of the 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing at NeurIPS 2019*, Vancouver, British Columbia, Canada.

Eilon Sheetrit, Anna Shtok, and Oren Kurland. 2020. A passage-based approach to learning to rank documents. *Information Retrieval Journal*, 23:159–186.

Luca Soldaini and Alessandro Moschitti. 2020. The cascade transformer: An application for efficient answer sentence selection. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5697–5708.

Lidan Wang, Jimmy Lin, and Donald Metzler. 2011. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2011)*, pages 105–114, Beijing, China.

Xiao Wang, Yaxiong Wu, Xi Wang, Craig Macdonald, and Iadh Ounis. 2020. University of Glasgow Terrier team at the TREC 2020 deep learning track. In *Proceedings of the Twenty-Ninth Text REtrieval Conference (TREC 2020)*.

Ji Xin, Rodrigo Nogueira, Yaoliang Yu, and Jimmy Lin. 2020. Early exiting BERT for efficient document ranking. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 83–88.

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

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

A Appendix

A.1 Full List of LTR Features

Term-based				
				Sum of ICTF
Average of BM25	Average of DFR_GL2	Average of DFR_in_expB2	Average of DPH	Average of ICTF
Median of BM25	Median of DFR_GL2	Median of DFR_in_expB2	Median of DPH	Median of ICTF
Max of BM25	Max of DFR_GL2	Max of DFR_in_expB2	Max of DPH	Max of ICTF
Min of BM25	Min of DFR_GL2	Min of DFR_in_expB2	Min of DPH	Min of ICTF
MaxMinRatio of BM25	MaxMinRatio of DFR_GL2	MaxMinRatio of DFR_in_expB2	MaxMinRatio of DPH	MaxMinRatio of ICTF
Sum of IDF		Sum of Normalized TF	Sum of TF	
Average of IDF	Average of LMDir	Average of Normalized TF	Average of TF	Average of TFIDF
Median of IDF	Median of LMDir	Median of Normalized TF	Median of TF	Median of TFIDF
Max of IDF	Max of LMDir	Max of Normalized TF	Max of TF	Max of TFIDF
Min of IDF	Min of LMDir	Min of Normalized TF	Min of TF	Min of TFIDF
MaxMinRatio of IDF	MaxMinRatio of LMDir	MaxMinRatio of Normalized TF	MaxMinRatio of TF	MaxMinRatio of TFIDF
Score-based				
SCS	Probablity Sum	Doc Size	Query Length	Query Coverage Ratio
Unique Term Count in Query	Matching Term Count	Normalized TFIDF	BM25	LMDir
DFR_GL2	DFR_in_expB2	DPH	TFIDF	
Proximity-based				
UnorderedSequentialPairs with gap 3	OrderedSequentialPairs with gap 3	UnorderedQueryPairs with gap 3	OrderedQueryPairs with gap 3	BM25-TP
UnorderedSequentialPairs with gap 8	OrderedSequentialPairs with gap 8	UnorderedQueryPairs with gap 8	OrderedQueryPairs with gap 8	Proximity
UnorderedSequentialPairs with gap 15	OrderedSequentialPairs with gap 15	UnorderedQueryPairs with gap 15	OrderedQueryPairs with gap 15	TP distance
Translation-based				
title IBM Model1(*raw* field)	url IBM Model1(*raw* field)	body IBM Model1(*raw* field)	body IBM Model (*subword* field)	