Top-Down Partitioning for List-Wise Rankers

Anonymous Author(s)*

ABSTRACT

Large Language Models (LLMs) have significantly impacted many facets of natural language processing and information retrieval. Unlike previous neural approaches, the enlarged context window of these generative models allows for ranking multiple documents at once, commonly called list-wise ranking. However, there are still limits to the number of documents that can be ranked in a single inference of the model, leading to the broad adoption of a sliding window approach to identify the k most relevant items in a ranked list. We argue that the sliding window approach is not well-suited for list-wise re-ranking because it (1) cannot be parallelized in its current form, (2) wastes compute repeatedly re-scoring the best set of documents as it works its way up the initial ranking, and (3) prioritizes the lowest-ranked items for scoring rather than the highest-ranked items by taking a bottom-up approach. We propose a novel algorithm that partitions a ranking to depth k and processes documents top-down. Unlike sliding window approaches, our algorithm is inherently parallelizable due to using a highly ranked pivot element, which can be compared to documents down to an arbitrary depth concurrently. In doing so, we reduce the number of expected inference calls by around 33% when ranking at depth 100 while matching the performance of prior approaches across multiple strong re-rankers.

CCS CONCEPTS

• **Information systems** → **Information retrieval**.

KEYWORDS

LLMs, List-wise ranking, Top-down Partitioning

ACM Reference Format:

Anonymous Author(s). 2018. Top-Down Partitioning for List-Wise Rankers. In . ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Until recently, neural approaches to ranking entailed the direct comparison of a document[1] to only one or two other distinct texts at a time: either a "point-wise" comparison to the query [12, 19] or a "pair-wise" comparison to a query and one other reference document [23]. Recent developments in Large Language Models (LLMs) improve the natural language understanding of transformer-based models and vastly increase the number of tokens a model can effectively process compared to previous models [2, 4, 35]. These two improvements allow for a "list-wise" ranking approach at inference time, where multiple documents can all be evaluated simultaneously. The key benefit of such a list-wise approach is that the ranker is directly given context to judge each document by considering other potentially relevant documents. State-of-the-art list-wise ranking approaches output a permutation of the original ranking instead of explicit scores [25, 31]. The need to encode multiple documents and output multiple tokens to determine document order makes these models computationally expensive due to the quadratic complexity of attention, meaning that encoding multiple documents inherently takes more time than current point-wise architectures.

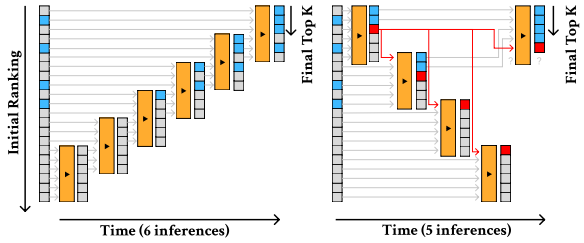

Figure 1: Bottom-up sliding window vs our top-down partitioning approach. Relevant documents are colored blue. In the case of our approach, the pivot document marked red is identified in the first pass and can be compared to other partitions concurrently with red arrows denoting the use of the pivot element.

In existing work, there is still a limit to the number of documents that can be ranked in a single pass (typically around 20 documents [25]). Therefore, a larger ranking must be 'batched' before inference. A strategy is needed to select which passages to 'batch' together and how to combine them, given that a batch permutation gives no indicator of absolute relevance across the full ranking—only the relative ranking of documents within a batch. Due to the absence of scores, this problem becomes closer to a classical selection or partitioning algorithm, such as quick select [10].

The prevailing approach in list-wise ranking uses a sliding window over a chosen ranking depth from bottom to top [25, 31, 39] as illustrated in Figure 1(a). By using an overlapping 'stride' over windows, documents that are initially lower in the ranking can bubble to the top (in principle [39]). However, in this approach, the ranking of each window is dependent on the processing of the documents below, leading to the repeated scoring of multiple documents to enter the top-k and, therefore, an inefficient partitioning scheme. We alleviate this problem with a rank-biased partitioning algorithm, which allows for parallel comparison of multiple batches since each batch only depends on a 'pivot' from the first batch (Figure 1(b)).

Our key intuition relates to the purpose of such expensive rankers; such an approach is inherently precision-orientated and, therefore,

[1] In practice, these models are conducted over shorter passages of text. But for ease of reading, we simply refer to them as "documents".

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

could find use in a broad set of domains that rely on either an exact match or a small set of highly relevant documents [11, 14]. With such applications in mind, we identify a pivot element at the k^{th} rank, which is then compared with all documents down to an arbitrary depth. Any document considered more relevant than the pivot element is necessarily a candidate for the top k results. A final scoring is performed over all candidates to yield a final top-k. We additionally constrain the number of candidates that can be selected in a given iteration to allow a budget on latency.

Our core contribution is a novel approach to list-wise ranking that matches the performance of sliding window ranking and reduces the number of LLM inferences while being parallelizable. We then analyze the precision of list-wise rankers under our approach, considering a broad set of first-stage rankers. With this work, we aim to bring attention to the inefficiencies of current approaches and how a task-specific approach can largely reduce the required inferences over computationally expensive language models. We release artifacts and experiment code to ensure reproducibility.[2]

2 BACKGROUND AND RELATED WORK

The Ranking Problem. A ranking is a permutation of a set ordered with respect to some scoring function. Within retrieval, for a corpus $C = \{d_1, \ldots, d_{|C|}\}$ and a user query text q, a ranking model returns a top-k set of documents where $k \ll |C|$, ordered by the probability of relevance to q [28].

Until the resurgence of neural networks and, more specifically, the transformer architecture [37], ad-hoc search primarily involved term weights from exact lexical matches [29]; neural architectures overcome the problem of exact term matching via soft-matching of terms using pre-trained language models [7, 27]. Multiple variants of the transformer architecture are applied in text retrieval, with most approaches falling into one of the following two paradigms. The first is a bi-encoder, which separately encodes a query and document into a dense latent space and executes a vector similarity search to determine relevance [12, 17], a variation of this architecture, learned sparse retrieval uses sparse vectors frequently projected over the vocabulary [9, 16]. The second, a cross-encoder, treats retrieval as a classification problem [19, 20, 22], jointly encoding a query and document before outputting a relevance score for the pair. List-wise ranking can be seen as a cross-encoder variant due to its jointly encoding queries and multiple documents [31][3].

List-wise Ranking. List-wise ranking is the ranking of multiple documents in a single inference step. We follow the notation convention of Tang et al. [33] in which an n-permutation of an n-ranking $X = \{X_i\}_{i=1}^{n}$ is defined as $\sigma : \{d_1, \ldots, d_n\} \rightarrow \{d_1, \ldots, d_n\}$. Given X, a list-wise model θ outputs a permutation σ ordered by relevance to q. A neural approach generally performs permutation in discrete space via the decoding of tokens to determine order. For an initial ranking X and query text q, $\sigma(X) = \text{PERMUTE}(X, q; \theta)$, where the permutation σ is generated auto-regressively through greedy decoding. Sun et al. [31] first showed the capability of GPT variants [2] as zero-shot list-wise rankers, with Ma et al. [15] achieving strong performance with a different prompt. Multiple works then distilled GPT-based rankers into smaller architectures in both a decoder-only [24, 25] and encoder-decoder setting [32], finding that the distilled model could outperform its teacher with suitable data augmentation [25]. Zhang et al. [39] then showed that comparable performance could be achieved without using the GPT API.

Dynamic Pruning. Our approach takes inspiration from classic dynamic pruning algorithms in retrieval. The key notion is that one does not need to consider documents that have no chance of entering the top-k during scoring. Such an idea has been applied to approximate a top-k ranking over sparse indices in a document-at-a-time fashion [1]. By caching terms already scored for a given query, a lower bound can be determined for the document score required to enter the top-k; as such, the scoring of many documents can be cut short or eliminated based on this estimation [30]. In recent years, dynamic pruning has been successfully applied to neural approaches, including re-ranking [13] and learned sparse retrieval [26]. What differentiates our algorithm for listwise ranking is that we cannot make in-document or term-at-a-time comparisons and, therefore, use an entire document text as a threshold.

3 PROPOSED ALGORITHM

Our core idea is to remove the dependence of a sequence of partitions of a ranking on each other. Currently, under a sliding window w with a stride s, dependencies prevent parallelism and other efficiency improvements due to the bottom-up nature of such an approach. To improve efficiency, we partition a ranking in a top-down fashion compared to a highly ranked pivot element p to select candidates under a budget b for each iteration.

As outlined in Algorithm 1, given a set of documents $A_{i-1} \subseteq X$ being the previous iteration or the full ranked list, we process the top-w candidates L and take a pivot element[4] p at rank k with all elements above the pivot added to a candidate set A_i with maximum size b, which is modified over each iteration; all other elements are added to a backfill set B, which persists throughout execution to retain all documents which are estimated to be less relevant than the pivot. The remainder of the ranked list X can be processed in parallel in partitions of size of $w - 1$.

The pivot p is prepended to the next set L, which is then ordered by model θ to find documents ranked above p. Any elements of L ranked above p are added to A, and the remainder is added to B. If no element in R is greater than the current pivot, we know no element can enter the top-k; therefore, our top set is already sorted as L. If candidates are found, the algorithm is executed on A_i constrained to budget size b (Algorithm 1 Line 14). For our main evaluation, the candidate size equals the window size. Therefore, a single extra iteration is required after traversing a full ranked list. Our algorithm incorporates the desirable rank bias of favoring the top-ranked documents because it works top-down to select a candidate pool constrained by budget b. As a document at rank 20 is more likely to be relevant than a document at rank 100, when a set of b candidates is found, we posit that there may be no need to search the remaining partitions.

For $|X| > w$, a sliding window approach always has a worst-case number of inferences, $\text{inferences}(X) = \frac{|X|}{s} - 1$. As top-down partitioning does not use a stride over a sliding window, our expectation

[2] Anonymous Github Repository

[3] Though a list-wise loss criterion has been applied in Learn-To-Rank [3] before neural retrieval, due to context window constraints, neural rankers were trained with point-wise or pair-wise loss functions.

[4] we choose $k = \frac{w}{2}$ for comparison to the commonly used stride of 10.

Algorithm 1: Top-Down Partitioning

Input: Candidate set: A_{i-1}, Backfill set: B, Query: q, Window size: w, Budget: b, Optimization cutoff: c, List-wise ranker: θ
Output: $\sigma(A_{i-1})$: A_{i-1} ordered by relevance to q

begin
  $L \leftarrow \{d_1, \ldots, d_w, d \in A_{i-1}\}$ // First w documents
  $R \leftarrow A_{i-1} - L$ // Remaining documents
  $L \leftarrow \text{PERMUTE}(L, q; \theta)$ // Order current partition
  $p \leftarrow L[k]$ // Take pivot element
  $A_i \leftarrow L[1:k]$
  $B \leftarrow B \cup L[k+1:|L|]$
  while $|A_{i-1}| > 0$ *and* $|A_i| < b$ **do**
    $L \leftarrow \{d_1, \ldots, d_{w-1}\}, d \in R$
    $R \leftarrow R - L$
    $L \leftarrow \text{PERMUTE}(p \cup L, q; \theta)$
    $A_i \leftarrow A_i \cup L[1:p]$ // Elements $> p$
    $B \leftarrow B \cup L[p+1:|L|]$ // Elements $< p$
  return $(|A_i| = k-1)$? $A_i \cup p \cup B$: $\text{pivot}(A_i) \cup p \cup B$

over the number of inferences is a function of the desired search depth, the window size, and the budget b. Window size beyond the top-w is reduced due to the need for the pivot element to be included in each inference. As we do not know how many documents will be considered more relevant than the pivot, identifying p_i at the beginning of the i^{th} iteration, we define the set of possible ranking iterations A as follows. Given the 0^{th} element $A_0 = X$:

$$A_i = \{A_i : |A_i| < b \wedge \forall d \in A_{i-1}, \text{rank}(d) < \text{rank}(p_{i-1})\} \quad (1)$$

We can then estimate a worst-case number of inferences by taking the sum of inferences over all iterations defined in Equation 1. Empirically, this does not occur as the algorithm either finds b candidates before processing all partitions or the ranker is sufficiently precise to meet the condition $|A_i| = k - 1$ early.

$$\text{inferences}(R) = \sum_{i=0}^{|A|} 1 + \frac{|A_i| - w}{w - 1} \quad (2)$$

Equation 2 degenerates to $\text{inferences}(X) = 2 + \frac{|X|-w}{w-1}$ in the case of $b = w$. We need a single inference to find the initial pivot at each stage; we then have $|A_i| - w$ documents remaining to process with a reduced window size $w - 1$ as we must account for the pivot element. We then require a final inference to order the set of w candidates.

4 EVALUATION

We now describe the evaluation of top-down partitioning for list-wise re-ranking and our observations from empirical evidence.

4.1 Experiment Setup

We outline the following research questions to assess our novel algorithm:
RQ-1: How does efficiency trade-off affect performance in list-wise ranking?
RQ-2: How sensitive is top-down partitioning to the relevance of the contents of the first window?
RQ-3: How does a larger candidate pool affect ranking performance?

Listwise Ranking Approaches. We compare approaches to ranking with sequence-to-sequence list-wise rankers. We do not consider variants of pairwise re-ranking as their time complexity and performance are inferior to the following approaches [24].
Single Window (Single): Re-ranks the top-w documents from a first stage ranking where w is window size.
Sliding Window (Sliding): For a given window size w and stride s re-ranks documents to depth D using a sliding window.
Top-Down Partitioning (TDPart): Our approach outlined in Algorithm 1. Performs an initial search over a top-w window to find a pivot at cutoff k before searching to depth D comparing each window to the pivot.

Datasets. We conduct all experiments on the MSMARCO corpus [18], a collection of over 8.8 million documents commonly used to train neural ranking models. As we are primarily concerned with the precision of the top-k re-ranked documents, we choose to evaluate our approach on the TREC Deep Learning 2019 and 2020 test queries [5, 6], which have densely annotated queries in contrast to either the official MSMARCO DEV set [18] or the zero-shot BEIR benchmark [34], we refer the reader to the original model papers for an overview of each models performance on sparse benchmarks. We evaluate nDCG@1, 5, 10, and P@10 using the ir_measures library; following standard MSMARCO conventions, we consider a relevance judgment of 2 or more as relevant. Our algorithm aims to reduce the inferences required when applying list-wise rankers while maintaining effectiveness, so we conduct equivalence tests (paired TOST) with $p < 0.05$ and a 5% bounds between our methods and baselines.

Models. We apply our approach to multiple state-of-the-art list-wise rankers that execute the PERMUTE function described in Section 3. RankGPT [31], which directly prompts GPT variants[5], a decoder-only distilled ranker RankZephyr [25] based on the Zephyr-7B [36] architecture and LiT5 [32] based on a T5 [27] fusion-in-decoding model [11]. We allow a maximum of 4096 tokens per input in all cases. We also employ an oracle approach with access to test relevance judgments returning a permutation sorted by these judgments. Due to tie breaks inherent when sorting by discrete relevance judgments, precision can vary. As our approach is fundamentally rank-biased, we investigate the effect of first-stage rankers on downstream precision. We re-rank BM25 [29], a common lexical model, and we use the Terrier implementation with default parameters [21]; SPLADEv2 [8], a teacher-distilled learned sparse model and RetroMAE [38], a teacher-distilled bi-encoder using MAE pre-training over a brute force index.

4.2 Results and Discussion

Inferences are Empirically Reduced. Table 1 presents our core results, contrasting our algorithm with existing approaches. Addressing **RQ-1**, as can be observed in the last column of Table 1, we consistently observe a reduced number of inferences across all cases when applying top-down partitioning, as expected by the analysis of our algorithm. Additionally, our approach exceeds or matches a sliding window's performance measured by nDCG@10 in 79% of cases, including RankGPT, which is zero-shot and known to have variable output [24] and our Oracle approach. Crucially,

[5]We use GPT 3.5 (`gpt-3.5-turbo-0125`)

Table 1: Comparing 'Single Window', 'Sliding Window', and 'TDPart' (ours) approaches. We report TOST (p<0.05), equivalence to our approach denoted with underline. We report mean inferences with the mean number of inferences run in parallel in parentheses.

			nDCG@1		nDCG@5		nDCG@10		P@10		
Stage	Model	Mode	DL19	DL20	DL19	DL20	DL19	DL20	DL19	DL20	N. Inf.
SPLADEv2	Oracle	Single	0.977	1.000	0.936	0.977	0.890	0.916	0.765	0.748	1.0 (1.0)
	Oracle	Sliding	0.984	1.000	0.972	0.995	0.957	0.978	0.872	0.822	9.0 (1.0)
	Oracle	TDPart	0.984	1.000	0.972	0.995	0.956	0.976	0.872	0.820	7.05 (5.05)
	Zephyr	Single	0.849	0.855	0.808	0.831	0.777	0.795	0.672	0.637	1.0 (1.0)
	Zephyr	Sliding	0.849	0.849	0.791	0.826	0.777	0.802	0.691	0.631	9.0 (1.0)
	Zephyr	TDPart	0.833	0.852	0.794	0.828	0.780	0.805	0.679	0.639	7.35 (5.35)
	LiT5	Single	0.802	0.827	0.795	0.797	0.763	0.763	0.665	0.598	1.0 (1.0)
	LiT5	Sliding	0.802	0.818	0.798	0.771	0.774	0.763	0.691	0.600	9.0 (1.0)
	LiT5	TDPart	0.787	0.827	0.795	0.791	0.766	0.768	0.684	0.604	6.26 (4.26)
	GPT 3.5	Single	0.833	0.787	0.783	0.775	0.760	0.752	0.644	0.607	1.0 (1.0)
	GPT 3.5	Sliding	0.837	0.781	0.786	0.779	0.754	0.766	0.642	0.617	9.0 (1.0)
	GPT 3.5	TDPart	0.802	0.806	0.779	0.788	0.753	0.752	0.642	0.598	7.44 (5.44)
RetroMAE	Oracle	Single	0.953	0.981	0.925	0.934	0.863	0.874	0.749	0.696	1.0 (1.0)
	Oracle	Sliding	0.992	1.000	0.968	0.993	0.948	0.958	0.863	0.800	9.0 (1.0)
	Oracle	TDPart	0.992	1.000	0.967	0.993	0.945	0.958	0.853	0.800	7.07 (5.07)
	Zephyr	Single	0.810	0.867	0.792	0.815	0.758	0.778	0.665	0.607	1.0 (1.0)
	Zephyr	Sliding	0.810	0.867	0.796	0.818	0.769	0.785	0.686	0.607	9.0 (1.0)
	Zephyr	TDPart	0.826	0.864	0.797	0.814	0.771	0.788	0.686	0.619	7.22 (5.22)
	LiT5	Single	0.787	0.858	0.773	0.776	0.739	0.748	0.653	0.585	1.0 (1.0)
	LiT5	Sliding	0.756	0.855	0.770	0.781	0.741	0.750	0.665	0.578	9.0 (1.0)
	LiT5	TDPart	0.740	0.867	0.763	0.798	0.743	0.761	0.684	0.591	6.56 (4.56)
	GPT 3.5	Single	0.841	0.827	0.802	0.789	0.741	0.744	0.630	0.576	1.0 (1.0)
	GPT 3.5	Sliding	0.802	0.799	0.779	0.752	0.740	0.727	0.649	0.561	9.0 (1.0)
	GPT 3.5	TDPart	0.837	0.796	0.787	0.758	0.736	0.739	0.637	0.578	7.2 (5.2)
BM25	Oracle	Single	0.907	0.935	0.823	0.811	0.719	0.715	0.560	0.489	1.0 (1.0)
	Oracle	Sliding	0.953	0.981	0.924	0.928	0.879	0.880	0.788	0.702	8.87 (1.0)
	Oracle	TDPart	0.946	0.981	0.913	0.925	0.858	0.872	0.765	0.687	7.41 (5.41)
	Zephyr	Single	0.713	0.784	0.681	0.689	0.625	0.628	0.500	0.435	1.0 (1.0)
	Zephyr	Sliding	0.713	0.846	0.715	0.770	0.707	0.722	0.637	0.535	8.87 (1.0)
	Zephyr	TDPart	0.736	0.858	0.731	0.772	0.681	0.723	0.577	0.543	7.43 (5.43)
	LiT5	Single	0.748	0.799	0.685	0.678	0.626	0.604	0.507	0.413	1.0 (1.0)
	LiT5	Sliding	0.787	0.833	0.751	0.735	0.723	0.700	0.640	0.531	8.87 (1.0)
	LiT5	TDPart	0.787	0.836	0.724	0.746	0.687	0.679	0.593	0.489	5.85 (3.85)
	GPT 3.5	Single	0.698	0.765	0.647	0.666	0.602	0.588	0.484	0.393	1.0 (1.0)
	GPT 3.5	Sliding	0.709	0.809	0.727	0.704	0.686	0.654	0.588	0.480	8.87 (1.0)
	GPT 3.5	TDPart	0.686	0.802	0.693	0.708	0.654	0.637	0.556	0.459	7.54 (5.54)

the latency of our approach would be dramatically reduced due to the majority of inferences being parallelizable, as observed in the bracket of the final column of each Table. Due to the reduced window size required by our algorithm, we frequently observe a smaller final window as in our experiments, depth 100 does not cleanly divide by 19 ($w-1$); simply searching to an approximate depth which has the factor $w-1$ would further reduce inferences with minimal effect on performance.

Sensitivity to Initial Ranking. Addressing **RQ-2**, when applying our algorithm to strong first stages such as RetroMAE and SPLADEv2, it can be observed that in all cases, our approach is either statistically equivalent or outperforms the strongest baseline, being either a single window in the case that the top-w documents are already sufficiently precise or a sliding window. Of note is the statistical equivalence of all approaches when using a strong first stage and list-wise ranker; however, when considering a list-wise ranker, one would want to maximize performance such that

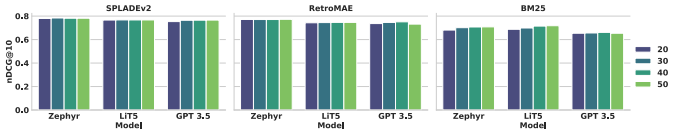

Figure 2: Ablation of Budget across each first-stage ranker on the TREC Deep Learning 2019 test queries.

a deeper search is essential. We propose that the multiple scoring of documents in a sliding window approach, which can 'miss' relevant documents due to the need for multiple scoring of the same document as observed by Zhang et al. [39] may inversely pull non-relevant documents into the top-k, explaining performance degradation over a single window as can be observed particularly under RankZephyr. Alternatively, when applying a weaker first stage, such as BM25, though our approach improves over a single window, we observe non-equivalent performance in P@10 when applying our approach versus a sliding window. A likely cause of this degradation is that due to the imprecision of the first w elements of a BM25 ranking, the chosen pivot element is insufficiently relevant to distinguish suitable candidates, and therefore, the candidate pool is polluted with partially or non-relevant items. Applying a list-wise ranker with our approach saves computation; therefore, allocating a small fraction of these savings to a stronger first stage, such as SPLADEv2 is worthwhile.

Ablation of Budget. Addressing **RQ-3**, we conduct an ablation of our approach to assess how a larger budget and, therefore, the opportunity to re-rank the top-k multiple times can affect retrieval performance. We assess budgets in the range [20, 30, 40, 50]. In Figure 2, observe that for precise first stages, we see minimal change in effectiveness when the budget is increased apart from RankGPT, which exhibits high variance. Akin to the multiple re-ranking stages proposed by Pradeep et al. [25] in which the previous ranking by the list-wise ranker is again re-ranked; however, we find a budget increase unnecessary given a first-stage dense or sparse neural retriever. However, in the case of BM25, we see that increasing the budget improves performance across all models, particularly in the case of LiT5, where nDCG@10 is improved from 0.687 to 0.715 when the budget is increased. Generally, we observe improvements of around 2 points of nDCG@10. We attribute this to the low precision of BM25, meaning that a greater budget is required to recover from a poor initial pivot value such that a progressive re-ranking is helpful. This explains small albeit insignificant degradations in nDCG@10 performance compared to a sliding window in Table 1.

5 CONCLUSION

We have introduced a novel algorithm for list-wise ranking that empirically matches the performance of commonly used sliding window approaches across multiple state-of-the-art ranking models. We reduce the number of LLM inferences required by taking a top-down approach with a highly-ranked pivot element while also allowing parallel computation, as the majority of inferences do not depend on each other, unlike prior works. In doing so, we show that current approaches are sub-optimal, and further consideration must be given to the efficiency of list-wise ranking at an algorithmic level.

REFERENCES

[1] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Y. Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*. ACM, 426–434. https://doi.org/10.1145/956863.956944

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[3] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning* (Corvalis, Oregon, USA) *(ICML '07)*. Association for Computing Machinery, New York, NY, USA, 129–136. https://doi.org/10.1145/1273496.1273513

[4] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling Instruction-Finetuned Language Models. *CoRR* abs/2210.11416 (2022). https://doi.org/10.48550/ARXIV.2210.11416 arXiv:2210.11416

[5] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2020. Overview of the TREC 2020 Deep Learning Track. In *Proceedings of the Twenty-Ninth Text REtrieval Conference, TREC 2020, Virtual Event [Gaithersburg, Maryland, USA], November 16-20, 2020 (NIST Special Publication, Vol. 1266)*, Ellen M. Voorhees and Angela Ellis (Eds.). National Institute of Standards and Technology (NIST). https://trec.nist.gov/pubs/trec29/papers/OVERVIEW.DL.pdf

[6] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. 2020. Overview of the TREC 2019 deep learning track. *CoRR* abs/2003.07820 (2020). arXiv:2003.07820 https://arxiv.org/abs/2003.07820

[7] 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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/V1/N19-1423

[8] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE v2: Sparse Lexical and Expansion Model for Information Retrieval. *CoRR* abs/2109.10086 (2021). arXiv:2109.10086 https://arxiv.org/abs/2109.10086

[9] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. *CoRR* abs/2107.05720 (2021). arXiv:2107.05720 https://arxiv.org/abs/2107.05720

[10] C. A. R. Hoare. 1961. Algorithm 65: find. *Commun. ACM* 4, 7 (jul 1961), 321–322. https://doi.org/10.1145/366622.366647

[11] Gautier Izacard, Patrick S. H. Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Few-shot Learning with Retrieval Augmented Language Models. *CoRR* abs/2208.03299 (2022). https://doi.org/10.48550/ARXIV.2208.03299 arXiv:2208.03299

[12] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 6769–6781. https://doi.org/10.18653/V1/2020.EMNLP-MAIN.550

[13] Jurek Leonhardt, Henrik Müller, Koustav Rudra, Megha Khosla, Abhijit Anand, and Avishek Anand. 2023. Efficient Neural Ranking using Forward Indexes and Lightweight Encoders. *ACM Transactions on Information Systems* (2023). https://api.semanticscholar.org/CorpusID:264935522

[14] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[15] Xueguang Ma, Xinyu Zhang, Ronak Pradeep, and Jimmy Lin. 2023. Zero-Shot Listwise Document Reranking with a Large Language Model. *CoRR* abs/2305.02156 (2023). https://doi.org/10.48550/ARXIV.2305.02156 arXiv:2305.02156

[16] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. 2020. Expansion via Prediction of Importance with Contextualization. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, Jimmy X. Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu (Eds.). ACM, 1573–1576. https://doi.org/10.1145/3397271.3401262

[17] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. CEDR: Contextualized Embeddings for Document Ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, and Falk Scholer (Eds.). ACM, 1101–1104. https://doi.org/10.1145/3331184.3331317

[18] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *CoRR* abs/1611.09268 (2016). arXiv:1611.09268 http://arxiv.org/abs/1611.09268

[19] Rodrigo Frassetto Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *CoRR* abs/1901.04085 (2019). arXiv:1901.04085 http://arxiv.org/abs/1901.04085

[20] Rodrigo Frassetto 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, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020)*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 708–718. https://doi.org/10.18653/V1/2020.FINDINGS-EMNLP.63

[21] Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Douglas Johnson. 2005. Terrier Information Retrieval Platform. In *Advances in Information Retrieval, 27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3408)*, David E. Losada and Juan M. Fernández-Luna (Eds.). Springer, 517–519. https://doi.org/10.1007/978-3-540-31865-1_37

[22] Ronak Pradeep, Yuqi Liu, Xinyu Zhang, Yilin Li, Andrew Yates, and Jimmy Lin. 2022. Squeezing Water from a Stone: A Bag of Tricks for Further Improving Cross-Encoder Effectiveness for Reranking. In *Advances in Information Retrieval - 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10-14, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13185)*, Matthias Hagen, Suzan Verberne, Craig Macdonald, Christin Seifert, Krisztian Balog, Kjetil Nørvåg, and Vinay Setty (Eds.). Springer, 655–670. https://doi.org/10.1007/978-3-030-99736-6_44

[23] Ronak Pradeep, Rodrigo Frassetto Nogueira, and Jimmy Lin. 2021. The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models. *CoRR* abs/2101.05667 (2021). arXiv:2101.05667 https://arxiv.org/abs/2101.05667

[24] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankVicuna: Zero-Shot Listwise Document Reranking with Open-Source Large Language Models. *CoRR* abs/2309.15088 (2023). https://doi.org/10.48550/ARXIV.2309.15088 arXiv:2309.15088

[25] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankZephyr: Effective and Robust Zero-Shot Listwise Reranking is a Breeze! *CoRR* abs/2312.02724 (2023). https://doi.org/10.48550/ARXIV.2312.02724 arXiv:2312.02724

[26] Yifan Qiao, Yingrui Yang, Haixin Lin, and Tao Yang. 2023. Optimizing Guided Traversal for Fast Learned Sparse Retrieval. *Proceedings of the ACM Web Conference 2023* (2023). https://api.semanticscholar.org/CorpusID:258333719

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

[28] Stephen E Robertson. 1977. The probability ranking principle in IR. *Journal of documentation* 33, 4 (1977), 294–304.

[29] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994 (NIST Special Publication, Vol. 500-225)*, Donna K. Harman (Ed.). National Institute of Standards and Technology (NIST), 109–126. http://trec.nist.gov/pubs/trec3/papers/city.ps.gz

[30] Trevor Strohman and W. Bruce Croft. 2007. Efficient document retrieval in main memory. In *SIGIR 2007: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, July 23-27, 2007*, Wessel Kraaij, Arjen P. de Vries, Charles L. A. Clarke, Norbert Fuhr, and Noriko Kando (Eds.). ACM, 175–182. https://doi.org/

10.1145/1277741.1277774

[31] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 14918–14937. https://aclanthology.org/2023.emnlp-main.923

[32] Manveer Singh Tamber, Ronak Pradeep, and Jimmy Lin. 2023. Scaling Down, LiTting Up: Efficient Zero-Shot Listwise Reranking with Seq2seq Encoder-Decoder Models. *CoRR* abs/2312.16098 (2023). https://doi.org/10.48550/ARXIV.2312.16098 arXiv:2312.16098

[33] Raphael Tang, Xinyu Zhang, Xueguang Ma, Jimmy Lin, and Ferhan Ture. 2023. Found in the Middle: Permutation Self-Consistency Improves Listwise Ranking in Large Language Models. *CoRR* abs/2310.07712 (2023). https://doi.org/10.48550/ARXIV.2310.07712 arXiv:2310.07712

[34] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models. *CoRR* abs/2104.08663 (2021). arXiv:2104.08663 https://arxiv.org/abs/2104.08663

[35] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023). https://doi.org/10.48550/ARXIV.2302.13971 arXiv:2302.13971

[36] Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. 2023. Zephyr: Direct Distillation of LM Alignment. arXiv:2310.16944 [cs.LG]

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[38] Shitao Xiao, Zheng Liu, Yingxia Shao, and Zhao Cao. 2022. RetroMAE: Pre-Training Retrieval-oriented Language Models Via Masked Auto-Encoder. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, 538–548. https://doi.org/10.18653/V1/2022.EMNLP-MAIN.35

[39] Xinyu Zhang, Sebastian Hofstätter, Patrick Lewis, Raphael Tang, and Jimmy Lin. 2023. Rank-without-GPT: Building GPT-Independent Listwise Rerankers on Open-Source Large Language Models. *CoRR* abs/2312.02969 (2023). https://doi.org/10.48550/ARXIV.2312.02969 arXiv:2312.02969

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009