Faster Learned Sparse Retrieval with Guided Traversal

Antonio Mallia
antonio.mallia@nyu.edu
New York University
New York, US

Joel Mackenzie
joel.mackenzie@uq.edu.au
The University of Queensland
Brisbane, Australia

Torsten Suel
torsten.suel@nyu.edu
New York University
New York, US

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

ABSTRACT

Neural information retrieval architectures based on transformers such as BERT are able to significantly improve system effectiveness over traditional sparse models such as BM25. Though highly effective, these neural approaches are very expensive to run, making them difficult to deploy under strict latency constraints. To address this limitation, recent studies have proposed new families of learned sparse models that try to match the effectiveness of learned dense models, while leveraging the traditional inverted index data structure for efficiency.

Current learned sparse models learn the weights of terms in documents and, sometimes, queries; however, they exploit different vocabulary structures, document expansion techniques, and query expansion strategies, which can make them slower than traditional sparse models such as BM25. In this work, we propose a novel indexing and query processing technique that exploits a traditional sparse model's "guidance" to efficiently traverse the index, allowing the more effective learned model to execute fewer scoring operations. Our experiments show that our guided processing heuristic is able to boost the efficiency of the underlying learned sparse model by a factor of four without any measurable loss of effectiveness.

CCS CONCEPTS

• **Information systems** → **Information retrieval query processing**; **Retrieval models and ranking**; **Search engine architectures and scalability**.

KEYWORDS

Query processing, Inverted Index, Learned Sparse Retrieval

1 INTRODUCTION

Neural information retrieval architectures based on transformers such as BERT [8] are able to significantly improve system effectiveness over traditional sparse models such as BM25. At the same time, they also pose new challenges, as transformers are very computationally expensive. This has motivated a lot of recent work on making transformer-based ranking more efficient. This includes the use of special-purpose hardware such as Google's Tensor Processing Units (TPUs) [14], nearest-neighbor methods for quickly identifying candidate documents in dense retrieval scenarios [13], and the design of late-interaction transformer-based rankers such as ColBERT [15].

A straightforward application of transformer-based ranking involves applying a transformer at query time to each document that is being re-ranked, leading to significant computational costs. While this can be reduced through approaches such as ColBERT [15], or by re-ranking only a small set of candidates, the resulting methods are still much more expensive than a simple ranking function that can be directly evaluated over an inverted index. On the other hand, approaches based on learned sparse representations aim to come close to the best transformer-based methods in effectiveness while preserving the efficiency of simple bag-of-words rankers such as BM25 [32].

The main goal in (most) learned sparse representations, as discussed by Lin and Ma [16], is to learn a set of terms under which a document should be indexed (document expansion), and the impact scores that should be stored in the corresponding inverted index postings (learning impacts), such that the resulting ranking function approximates the effectiveness of a full transformer-based ranker while retaining the efficiency of the fastest inverted-index based methods. Recent work has shown, however, that while effective retrieval is possible with learned sparse approaches, they are often still much slower than their traditional counterparts [20, 22, 27]. In this work, we propose a novel heuristic index traversal mechanism that closes the performance gap between learned and traditional rankers. Experiments over the MSMARCO passage collection demonstrate that our heuristic approach can accelerate DeepImpact retrieval by a factor of four without any measurable loss in effectiveness.

2 BACKGROUND & MOTIVATION

We now briefly introduce some background and related work before motivating our proposed approach.

Learned Sparse Models. While various learned sparse models have been proposed, they all improve effectiveness in two ways:

(1) *Term Expansion:* Adding *new* terms to document or query representations; and

(2) *Term Re-weighting:* Changing the *weights* of the terms in documents or queries.

To this end, learned sparse models typically make use of the contextual word representations of large language models such as BERT, allowing them to "learn" when new terms should be added, or when terms are important. It is important to note that both expansion and re-weighting can be applied to documents (before indexing), to queries (before searching), or any combination thereof.

DeepCT [6, 7] is the first example of learned sparse retrieval, exploiting the contextual word representations from BERT to re-weight term frequencies for BM25 scoring. The main limitation of DeepCT lies in the fact that it does not address the vocabulary mismatch problem [37]: only terms already appearing in the documents will receive learned weights to improve their relevance signals. DocT5Query [29] addresses the vocabulary mismatch problem by expanding documents offline via the T5 [31] sequence-to-sequence model. However, DocT5Query does not explicitly re-weight the terms within each document and, like DeepCT, relies on BM25 scoring.

A way to address the shortcomings of DeepCT and DocT5Query was first proposed by Mallia et al. [27]. Instead of using the original document collection, their model, DeepImpact, expands the documents to include new terms to address the vocabulary mismatch via the DocT5Query model. Then, instead of scoring the documents with BM25, DeepImpact directly optimizes the sum of query term impacts to maximize the score difference between relevant and non-relevant passages for the query. That is, DeepImpact actually *learns* the ranking function instead of applying an existing one.

Other approaches for learning expansion terms and term importance re-weighting include uniCOIL [16], TILDEv2 [38], and SPLADEv2 [11], all of which employ document expansion and the use of a contextualized language model to learn a ranking model. As opposed to DeepImpact, uniCOIL and SPLADEv2 also perform weighting on the query terms, such that document ranking becomes a *weighted* sum over term impacts. Furthermore, SPLADEv2 also employs query expansion, thereby adding new terms to each query rather than just re-weighting the existing ones.

In order to make use of an inverted index for querying, all of the learned term scores derived from the aforementioned models are quantized and embedded into the postings lists of the index. Then querying proceeds as usual, with document scores computed as the weighted sum of term impacts. Although TILDEv2 was initially proposed in the context of re-ranking a set of candidates, we employ it here as an effective first-stage ranker. It is also worth mentioning that Formal et al. [11] train SPLADEv2 using knowledge distillation [12], an approach which substantially increases the model accuracy with no particular effect on the retrieval latency; this could be leveraged by the other models as well, but we leave this as future work.

Table 1 summarizes the index statistics and the average query lengths of the different learned sparse models. The average number of postings per term ranges from 100 to 178 for methods exploiting

word-level tokenization, namely BM25, DeepCT, DocT5Query, and DeepImpact, and from 21K to 72K postings per term for methods exploiting BERT's WordPiece tokens [8], namely uniCOIL, TILDEv2, and SPLADEv2. The latter tokenization algorithm greedily restricts the number of tokens to subwords that are part of the BERT vocabulary. This process is very different from the more popular word-level tokenization, where tokens are generated by splitting on whitespaces and punctuation characters. These ranges, as well as the average number of query terms, have a serious impact on the query processing times, as we will show in Section 4.

Table 1: Basic statistics for the indexes of several traditional and learned sparse retrieval models.

Model	Terms	Postings	Avg. Query Length
BM25	2,660,824	266,247,718	4.5
DeepCT	989,873	128,969,826	4.5
DocT5Query	3,929,111	452,197,951	4.5
uniCOIL	27,678	587,435,995	686.3
TILDEv2	27,437	809,658,361	4.9
SPLADEv2	28,131	2,028,512,653	2037.8
DeepImpact	3,514,102	628,412,657	4.2

Efficient Index Traversal. To avoid the expensive scoring of all documents matching at least one query term in the document collection, several dynamic pruning techniques for disjunctive queries have been proposed. Safe dynamic pruning methods enrich the inverted index with additional information used to skip documents; during query processing, documents that cannot possibly score high enough to be in the list of the final top-K documents are bypassed. Thus, the final list of returned documents is the same as in exhaustive scoring of all documents, but with significantly less work. The most widely adopted dynamic pruning techniques include MaxScore [34], WAND [2], BlockMax WAND (BMW) [10], and Variable BMW [24, 25]. In general, dynamic pruning techniques exhibit good efficiency speed-ups when able to exploit the underlying distribution of scores, global upper bounds, and local upper bounds [33]. Given a query, if few composing terms have high upper bounds, while the remaining posting lists or blocks of postings have low upper bounds, there are many possibilities to skip over documents that cannot make it into the final top-K. In particular, the BM25 and DPH [1] scoring functions are known to exhibit favorable score distributions, while scores based on probabilistic language models are less amenable to skipping [30].

Motivation. Prior work has empirically demonstrated that learned sparse models are typically slower than their traditional counterparts [20, 22, 27], suggesting that the resulting score distributions are less amenable to dynamic pruning. To illustrate this phenomenon, consider Figure 1, which plots the list-wise upper-bound scores assigned to each of the query terms in the MSMARCO dev query set. Clearly, DeepImpact scoring yields many more high impact terms than BM25 and DocT5Query, making it more difficult for dynamic pruning algorithms to avoid work. Figure 2 shows this effect, with DeepImpact scoring a much higher proportion of documents than BM25 and DocT5Query under the popular MaxScore
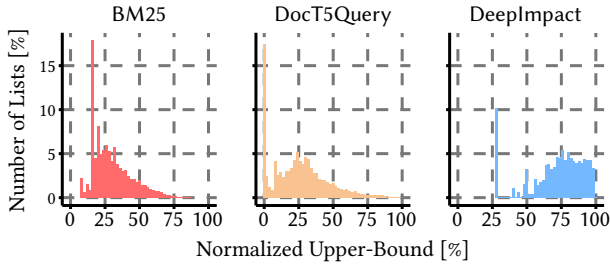
Figure 1: The distribution of list-wise upper-bound scores for all terms in the MSMARCO Dev Queries. DeepImpact gives much higher weights to terms, on average, than the others.
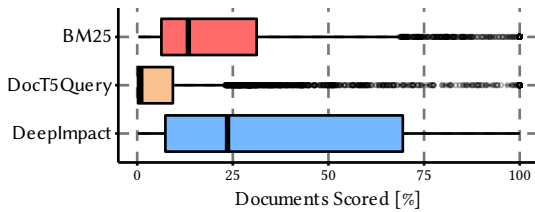
Figure 2: Relative percentage of documents scored (as compared to an exhaustive algorithm) with MaxScore for $k = 1000$ over the MSMARCO Dev Queries.

algorithm. What is the cause of these high impacts? Consider any common term such as *"the"*; BM25 will, by nature, give this term a low impact because it appears in most documents. However, the contextual nature of the language model DeepImpact is trained over means it can predict when *"the"* is an important word, such as in an article discussing the definition or origin of the term *"the"*. As such, we are interested in exploring heuristic processing techniques that can achieve the efficiency of the traditional rankers with the effectiveness of the learned sparse models.

3 FAST TOP-K GUIDED TRAVERSAL

Motivated by the analysis in Section 2, we now propose a fast heuristic query processing strategy that exploits the favorable score distributions provided by BM25. In particular, our technique employs BM25 scoring over a DocT5Query expanded index to decide which documents to process, but uses the DeepImpact model to compute the document scores, thereby running nearly as fast as plain BM25 traversal, but with significantly better effectiveness. This method relies on the fact that DeepImpact uses DocT5Query to expand the underlying corpus and also uses word-level tokenization; thus the underlying posting lists are basically the same.[1] We hypothesize that the documents visited during index traversal over the DocT5Query index are typically high-impact documents under the DeepImpact model, which should allow fast, yet effective, retrieval.

To implement this query processing strategy, the impact value for both BM25 (on the DocT5Query index) and DeepImpact must

---
[1]In practice, there may be slight differences due to different tokenization techniques.

(a) Pivot Selection   **(b) Decoding, Scoring**
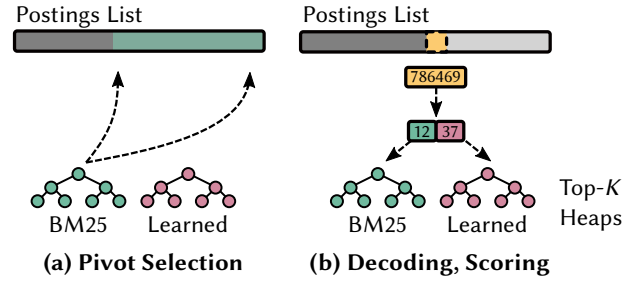
Figure 3: Guided traversal for a single-term query: (a) the threshold of the BM25 top-K min-heap is used to find the next document to process, (b) the combined impact is decoded into its two component values, and checked against the respective results sets.

be stored for *each posting in the index*.[2] Fortunately, due to the fact that a small number of bits is sufficient for representing the impact scores for these models, they can be packed into a single 32-bit integer, with each score represented with 16 bits. At query processing time, two top-K min-heaps are initialized [23]; the first heap maintains the top-K documents according to BM25, and the second heap maintains the top-K documents for DeepImpact. Query processing proceeds, as usual, using the BM25 min-heap to drive the index traversal, including any dynamic pruning. When it comes time to score a document, the combined impacts are unpacked from the index, and each min-heap is updated. At the end of processing, both top-K heaps will contain the K highest scoring documents that were visited during BM25 traversal. Figure 3 sketches one cycle of this process, which we denote *guided traversal* (GT).

An interesting extension to the guided traversal process described above is to also account for *interpolated* scoring regimes [35]. In particular, since the secondary learned top-K set is computed over exactly those documents visited by the BM25-driven traversal, it is possible to compute interpolated scores in the secondary heap. We investigate this idea in our experiments by applying an unweighted linear interpolation between BM25 and DeepImpact, denoted as *guided traversal with interpolation* (GTI).

While we expect that our DeepImpact-GT and DeepImpact-GTI methods could generalize to other models, we only apply them to DeepImpact since uniCOIL, TILDEv2, and SPLADEv2 use BERT WordPiece tokenization resulting in incompatible vocabularies (see Table 1). We leave this investigation for future work.

4 EXPERIMENTS AND RESULTS

In this section, we analyze the performance of the proposed method using the popular MSMARCO [28] dataset, consisting of 8.8M passages. To evaluate query processing effectiveness and efficiency, all models are compared using the MSMARCO Dev Queries, TREC 2019 [4] and TREC 2020 [5] queries from the TREC Deep Learning passage ranking track. All experiments were conducted in memory using a single thread of a Linux machine with two 3.50 GHz Intel Xeon Gold 6144 CPUs and 512 GiB of RAM.

Implementations. We use Anserini [36] to generate the inverted indexes of the collections. We then export the Anserini indexes

---
[2]Where the postings do not align one-to-one, the missing score is set to zero.

Table 2: Efficiency and Effectiveness for different processing strategies over the MSMARCO dev queries, with GT and GTI denoting our guided traversal without or with interpolation, respectively.

Strategy	MSMARCO Dev Queries			
	Mean	Median	P_{99}	RR@10
BM25	5.7	4.3	24.0	0.187
DeepCT	1.1	0.9	4.5	0.244
DocT5Query	3.8	2.8	16.5	0.272
TILDEv2	20.7	14.3	90.6	0.333
uniCOIL	37.9	21.4	194.5	0.352
SPLADEv2	219.9	201.3	581.6	0.369
DeepImpact	19.5	14.0	79.6	0.326
DeepImpact-GT	4.8	4.2	16.1	0.326
DeepImpact-GTI	5.0	4.4	16.7	0.341
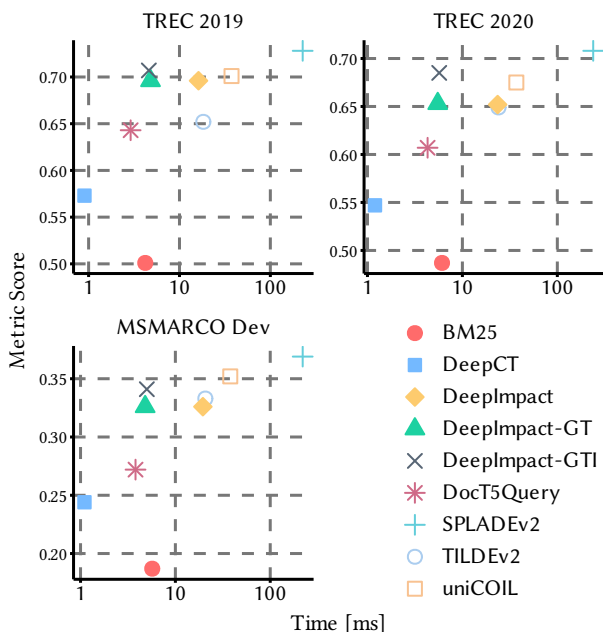
Figure 4: Effectiveness vs Mean Latency overall collections. Both TREC 2019 and TREC 2020 are evaluated with NDCG@10, whereas the MSMARCO dev queries are evaluated with RR@10. Note the log scale on the x-axis, and the differing y-axis breaks.

using the common index file format [17], and process them with PISA [26] using the MaxScore query processing algorithm [34]. All indexes are pre-quantized using 8-bit linear quantization to allow for "sum of impact" scoring [3] and reordered using BP [9, 19, 21].

Ranking Models. We use the BM25 scoring method provided by Anserini with the recommended parameters $k_1 = 0.82$ and $b = 0.68$ [18]. For DeepCT, we used the source code and data[3] provided by Mackenzie et al. [20]. DeepCT also uses BM25 to score

[3]https://github.com/jmmackenzie/term-weighting-efficiency

documents, with $k_1 = 8.0$ and $b = 0.9$ [6, 20]. For DocT5Query we use the predicted queries available online,[4] using 40 concatenated predictions for each passage in the corpus, as recommended by Nogueira and Lin [29]; documents are scored with BM25 using $k_1 = 0.82$ and $b = 0.68$. For uniCOIL we use the official implementation[5] which makes use of a DocT5Query expanded index. For TILDEv2 we use the official implementation[6] which first expands the whole MSMARCO passage collection with 200 expansion terms per document, and then indexes the expanded collection to generate an inverted index. This goes beyond the methodology adopted by Zhuang and Zuccon [38], where TILDE was used as a re-ranker. For SPLADEv2 we used the model provided by Formal et al. [11].[7] We refer here to their best performing approach, also named DistilSPLADE-max. For DeepImpact we use the model provided by Mallia et al. [27].[8]

Metrics. To evaluate retrieval effectiveness, we follow common practice and report RR@10 for the MSMARCO dev queries and NDCG@10 for TREC 2019 and TREC 2020. To carefully evaluate the retrieval performance of guided traversal with respect to the other strategies, we report the mean, the median, and the 99 th percentile of the response time in ms. Values are taken as the mean of three independent runs.

Results Analysis. In Table 2 we report our results on the large MSMARCO Dev Queries, and in Figure 4 we show the performance of the various strategies on all three query sets in terms of both effectiveness and efficiency, where the optimal region is the top left corner (minimal latency and maximal effectiveness).

Among the query processing strategies employing BM25 as the underlying ranking model, namely BM25, DeepCT, and DocT5Query, DeepCT is the most efficient strategy, with a mean response time of 1.1ms and a tail response time of 4.5ms. However, the most effective strategy is DocT5Query across all query sets (see Figure 4). Among the baseline learned sparse models, namely TILDEv2, uni-COIL, SPLADEv2, and DeepImpact, SPLADEv2 is consistently the most effective one, but it is also the least efficient. With respect to the most efficient competitor, DeepImpact, its mean response time is 10 to 14 times larger, and its 99 th percentile response time is 7 to 8 times larger, while SPLADEv2 effectiveness is 13.2% higher on MSMARCO Dev Queries, 4.6% higher on TREC 2019, and 8.4% higher on TREC 2020.

Our proposed DeepImpact-GT strategy, deployed with DeepImpact, improves its mean response time by up to 4.3×, its median response time by up to 3.6×, and its tail response time by up to 5.8×, with no measurable impact on effectiveness. Our DeepImpact-GTI strategy, interpolating BM25 scores and learned DeepImpact impacts, slightly increases the mean response time of DeepImpact-GT by 0.2 ms, but it improves the overall effectiveness: +4.6% on MSMARCO Dev Queries, +1.6% on TREC 2019, and +4.9% on TREC 2020. A pairwise t-test reported these improvements to be significant ($p \approx 10^{-10}$) for only the dev queries, with $p = 0.46$ and $p = 0.049$ reported for TREC 2019 and TREC 2020, respectively.

[4]https://github.com/castorini/docTTTTTquery
[5]https://github.com/luyug/COIL
[6]https://github.com/ielab/TILDE/tree/main/TILDEv2
[7]https://github.com/naver/splade
[8]https://github.com/DI4IR/SIGIR2021

Figure 4 allows us to easily detect the characteristics of the several approaches and quickly identify the suitability of a method for a precise use case. We observe that DeepCT, DocT5Query, SPLADEv2, DeepImpact-GT, and DeepImpact-GTI are on the Pareto frontier in all the plots, reducing the number of strategies to choose from depending on the efficiency-effectiveness constraints.

5 CONCLUSIONS

Learned sparse models result in substantial retrieval quality improvements while reducing the efficiency gap between neural retrieval and the faster traditional sparse models based on inverted indexes. This gap correlates with vocabulary structures, document expansion techniques, and query expansion strategies, making the several learned sparse models quite different efficiency-wise. In this work, we have proposed a "guided traversal" approach to accelerate index traversal by coupling a learned sparse ranking model with a traditional ranking model. Our proposed approach employs BM25 ranking over a DocT5Query expanded index to *lead* the index traversal, but uses the DeepImpact ranking impacts to compute document scores. Our preliminary results on top of DeepImpact show that our guided traversal approach is almost able to match the processing efficiency of traditional sparse models, while also improving the retrieval effectiveness of the learned sparse models through interpolation of the scores of the traditional and learned sparse ranking models. In future work, we plan to explore whether our guided traversal heuristic is practical for other learned sparse models, as well as different efficient query processing strategies.

Software. In the interest of reproducibility, software is available for generating our guided traversal runs. See https://github.com/DI4IR/dual-score for more information.

REFERENCES

[1] G. Amati, E. Ambrosi, M. Bianchi, C. Gaibisso, and G. Gambosi. FUB, IASI-CNR and University of Tor Vergata at TREC 2007 blog track. In *Proc. TREC*, 2007.
[2] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proc. CIKM*, pages 426–434, 2003.
[3] M. Crane, J. S. Culpepper, J. Lin, J. Mackenzie, and A. Trotman. A comparison of Document-at-a-Time and Score-at-a-Time query evaluation. In *Proc. WSDM*, pages 201–210, 2017.
[4] N. Craswell, B. Mitra, E. Yilmaz, D. Campos, and E. M. Voorhees. Overview of the TREC 2019 deep learning track. *Preprint: arXiv:2003.07820*, 2020.
[5] N. Craswell, B. Mitra, E. Yilmaz, and D. Campos. Overview of the TREC 2020 deep learning track. *Preprint: arXiv:2102.07662*, 2021.
[6] Z. Dai and J. Callan. Context-aware sentence/passage term importance estimation for first stage retrieval. *Preprint: arXiv:1910.10687*, 2019.
[7] Z. Dai and J. Callan. Context-aware document term weighting for ad-hoc search. In *Proc. WWW*, pages 1897–1907, 2020.
[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL*, pages 4171–4186, 2019.
[9] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita. Compressing graphs and indexes with recursive graph bisection. In *Proc. SIGKDD*, pages 1535–1544, 2016.
[10] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proc. SIGIR*, pages 993–1002, 2011.
[11] T. Formal, B. Piwowarski, and S. Clinchant. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *Proc. SIGIR*, page 2288–2292, 2021.
[12] S. Hofstätter, S. Althammer, M. Schröder, M. Sertkan, and A. Hanbury. Improving efficient neural ranking models with cross-architecture knowledge distillation. *Preprint: arXiv:2010.02666*, 2020.
[13] J. Johnson, M. Douze, and H. Jegou. Billion-scale similarity search with GPUs. *IEEE Trans. Big Data*, 7(03):535–547, 2021.
[14] N. P. Jouppi *et al.* In-datacenter performance analysis of a tensor processing unit. In *Proc. ISCA*, page 1–12, 2017.
[15] O. Khattab and M. Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proc. SIGIR*, pages 39–48, 2020.
[16] J. Lin and X. Ma. A few brief notes on DeepImpact, COIL, and a conceptual framework for information retrieval techniques. *Preprint: arXiv:2106.14807*, 2021.
[17] J. Lin, J. Mackenzie, C. Kamphuis, C. Macdonald, A. Mallia, M. Siedlaczek, A. Trotman, and A. de Vries. Supporting interoperability between open-source search engines with the common index file format. In *Proc. SIGIR*, pages 2149–2152, 2020.
[18] J. Lin, X. Ma, S.-C. Lin, J.-H. Yang, R. Pradeep, and R. Nogueira. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proc. SIGIR*, pages 2356–2362, 2021.
[19] J. Mackenzie, A. Mallia, M. Petri, J. S. Culpepper, and T. Suel. Compressing inverted indexes with recursive graph bisection: A reproducibility study. In *Proc. ECIR*, pages 339–352. Springer, 2019.
[20] J. Mackenzie, Z. Dai, L. Gallagher, and J. Callan. Efficiency implications of term weighting for passage retrieval. In *Proc. SIGIR*, pages 1821–1824, 2020.
[21] J. Mackenzie, M. Petri, and A. Moffat. Faster index reordering with bipartite graph partitioning. In *Proc. SIGIR*, pages 1910–1914, 2021.
[22] J. Mackenzie, A. Trotman, and J. Lin. Wacky weights in learned sparse representations and the revenge of Score-at-a-Time query evaluation. *Preprint: arXiv:2110.11540*, 2021.
[23] J. Mackenzie, M. Petri, and A. Moffat. Efficient query processing techniques for next-page retrieval. *Information Retrieval*, 25:27–43, 2022.
[24] A. Mallia and E. Porciani. Faster blockmax wand with longer skipping. In *Proc. ECIR*, pages 771–778. Springer, 2019.
[25] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini. Faster blockmax wand with variable-sized blocks. In *Proc. SIGIR*, pages 625–634, 2017.
[26] A. Mallia, M. Siedlaczek, J. Mackenzie, and T. Suel. PISA: performant indexes and search for academia. *OSIRRC@SIGIR*, 2019.
[27] A. Mallia, O. Khattab, T. Suel, and N. Tonellotto. Learning passage impacts for inverted indexes. In *Proc. SIGIR*, page 1723–1727, 2021.
[28] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng. MS MARCO: A human-generated machine reading comprehension dataset. 2016.
[29] R. Nogueira and J. Lin. From doc2query to doctttttquery. *Online preprint*, 2019.
[30] M. Petri, J. S. Culpepper, and A. Moffat. Exploring the magic of WAND. In *Proc. ADCS*, pages 58–65, 2013.
[31] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Preprint: arXiv:1910.10683*, 2019.
[32] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009.
[33] N. Tonellotto, C. Macdonald, and I. Ounis. Efficient query processing for scalable web search. *Found. Trends in Inf. Retr.*, 12(4–5):319–492, 2018.
[34] H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Information Processing & Management*, 31(6):831–850, 1995.
[35] S. Wang, S. Zhuang, and G. Zuccon. BERT-based dense retrievers require interpolation with BM25 for effective passage retrieval. In *Proc. SIGIR*, pages 317–324, 2021.
[36] P. Yang, H. Fang, and J. Lin. Anserini: Enabling the use of Lucene for information retrieval research. In *Proc. SIGIR*, pages 1253–1256, 2017.
[37] L. Zhao. *Modeling and solving term mismatch for full-text retrieval*. PhD thesis, Carnegie Mellon University, 2012.
[38] S. Zhuang and G. Zuccon. Fast passage re-ranking with contextualized exact term matching and efficient passage expansion. *Preprint: arXiv:2108.08513*, 2021.