

Faster Learned Sparse Retrieval with Block-Max Pruning

Antonio Mallia
Pinecone, Italy


Torsten Suel
New York University, USA

Nicola Tonellotto
University of Pisa, Italy

ABSTRACT

Learned sparse retrieval systems aim to combine the effectiveness of contextualized language models with the scalability of conventional data structures such as inverted indexes. Nevertheless, the indexes generated by these systems exhibit significant deviations from the ones that use traditional retrieval models, leading to a discrepancy in the performance of existing query optimizations that were specifically developed for traditional structures. These disparities arise from structural variations in query and document statistics, including sub-word tokenization, leading to longer queries, smaller vocabularies, and different score distributions within posting lists.

This paper introduces Block-Max Pruning (BMP), an innovative dynamic pruning strategy tailored for indexes arising in learned sparse retrieval environments. BMP employs a block filtering mechanism to divide the document space into small, consecutive document ranges, which are then aggregated and sorted on the fly, and fully processed only as necessary, guided by a defined safe early termination criterion or based on approximate retrieval requirements. Through rigorous experimentation, we show that BMP substantially outperforms existing dynamic pruning strategies, offering unparalleled efficiency in safe retrieval contexts and improved trade-offs between precision and efficiency in approximate retrieval tasks.

 <https://github.com/pisa-engine/BMP>

CCS CONCEPTS

• **Information systems** → **Information retrieval query processing**.

KEYWORDS

Efficiency, Learned Sparse Retrieval, Pruning

ACM Reference Format:

Antonio Mallia, Torsten Suel, and Nicola Tonellotto. 2024. Faster Learned Sparse Retrieval with Block-Max Pruning. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24)*, July 14–18, 2024, Washington, DC, USA. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3626772.3657906>

1 INTRODUCTION

Information retrieval (IR) systems have increasingly turned towards learned sparse models for their ability to efficiently handle the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGIR '24, July 14–18, 2024, Washington, DC, USA.

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0431-4/24/07
<https://doi.org/10.1145/3626772.3657906>

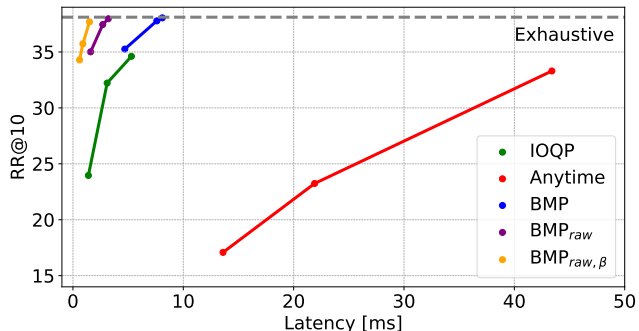


Figure 1: Effectiveness-efficiency graph for different query processing algorithms. Every line corresponds to an algorithm, every point corresponds to a different configuration.

complexity of query and document representations used in state-of-the-art ranking methods [7, 12, 20]. Learned sparse models operate by mapping queries and documents into sparse vectors, and then leveraging the traditional inverted index – a data structure that is still crucial for scalable text retrieval [30]. This use of inverted indexes gives learned sparse model an advantage over dense vector search methods [10, 33], leading to faster query processing and reduced storage requirements.

Within the domain of learned sparse models, SPLADE [7] and uniCOIL [12] have emerged as significant advancements. SPLADE operates on the principle of sparse lexical decomposition, effectively capturing the nuanced relationship between terms and documents. In contrast, uniCOIL incorporates token-level contextualization into the sparse modeling process. These models differ from methods such as DeepImpact [20], which focuses more on direct impact score predictions for terms in (suitably expanded) documents. Among the learned sparse models that have been proposed, we focus on SPLADE [7] and uniCOIL [12] for their good effectiveness, and aim at optimizing their retrieval efficiency. Learned sparse models are used to produce inverted indexes, a data structure which contains one inverted list for each distinct term, where each inverted list describes which documents are relevant to the term and how relevant they are. In particular, each inverted list consists of postings of the form (d_i, s_i) where d_i is the document ID (docID) of a document relevant to the term, and s_i is an *impact score*, quantized to a limited number of bits, that models the degree of relevance of d_i . Given an inverted index, the score of a document d with respect to a query q is defined as $s(q, d) = \sum_{t \in q} w(t, q) \cdot s(t, d)$ where $w(t, q)$ is a term weight for t in q , and $s(t, d)$ is the corresponding impact score stored in the inverted list for t – or 0 if there is no such posting for document d . Our goal is to return the top- k scoring documents for a query, say for $k = 10$ or $k = 1000$.

Once the learned sparse model has been encoded into an inverted index, algorithms such as MaxScore [31] and BlockMaxWand (BMW) [2, 5, 22], or score-at-a-time (SaaT) traversal algorithms [1],

can be used to perform efficient query processing, by performing dynamic pruning during index traversal. However, the efficiency of these algorithms on learned sparse models does not match that commonly reported on traditional similarity models [27], such as BM25 or query likelihood [29]. This has motivated several recent approaches that get better performance for index structures based on learned sparse models. This includes the Anytime approach from Mackenzie et al. [17], which allows existing algorithms to operate on confined segments of a topically-clustered inverted index, and Clipping [14], where MaxScore is adapted to enable efficient query processing for learned term importance schemes via a simple technique called term impact decomposition. Another set of approaches [21, 28] uses standard models such as BM25 for index traversal and then immediately reranks results with the learned sparse model.

This paper introduces BMP, a novel query processing strategy that is optimized for the indexes generated by learned sparse models. BMP employs a block filtering mechanism [4, 24] to prioritise clusters of documents based on their potential relevance. Promising subsets of documents that need to be processed are identified through an optimized computation of range-based upper bounds that are compared to a current threshold. When scoring a range of documents, BMP relies on a variant of a forward index, i.e., an array of documents stored as term-impact pairs.

Our experiments on three learned sparse models, SPLADE, ESPLADE, and uniCOIL, show that BMP is significantly faster than previous methods for safe retrieval of top results, ranging from $\sim 2\times$ to $\sim 60\times$, while achieving a much better trade-off between efficiency and effectiveness when used for approximate query processing as shown in Figure 1 for the MS MARCO dataset.

2 BLOCK-MAX PRUNING

We now describe our approach, called Block-Max Pruning (BMP), which consists of two major phases, block filtering, and evaluation of individual candidate blocks until some termination condition is satisfied.

Block Filtering. In this phase, we adapt the block filtering approach in [4, 24] to our scenario. Recall that each document is identified by a docID between 0 and $n - 1$ where n is the collection size. We divide this range into blocks of b consecutive docIDs, for some b to be chosen later. Now for each term in the collection, we have $\lceil n/b \rceil$ blocks, and for each such block we store the maximum impact score of any posting in the block. (This may be 0 if there are no postings for the term in the block.) Thus, for each term, we now have an array of $\lceil n/b \rceil$ block-max impact scores.

Given a query q , we can compute upper bounds for the possible scores of all documents by simply taking the arrays of block-max impact score for the query terms $t \in q$, and adding them up, weighted by term weights $w(t, q)$. As shown in [4, 24], this can be done in a highly efficient manner using vectorized instructions available in modern CPUs. This gives us a score upper bound, with respect to q , for each block of b documents in the collection.

Evaluating Candidate Blocks. Next, we evaluate individual blocks of b documents, in decreasing order of the block upper bounds, until some termination condition is met. To access blocks in decreasing order, we need to first sort the blocks by their upper bounds, or at least partially sort them.

To efficiently evaluate a block, we use a hybrid between inverted and forward index. That is, for each block of b documents, we store inverted lists of postings of the form (d, s) where d is a $\log_2(b)$ -bit number identifying a document within its block, and s is the associated impact score, plus a sorted list of all terms that occur in the block, each with a pointer to the start of the corresponding posting list. We evaluate a block by fetching the postings for the query terms, and aggregating the impact scores into an array of b accumulators. We use a heap to retain the k highest-scoring documents found.

We stop evaluating blocks once a termination condition is met.

Representing Block-Max Impact Scores. Our approach uses fairly small blocks, in some cases as small as $b = 8$ or $b = 16$. Thus we have to store and aggregate fairly long arrays of 8-bit impact scores – for MS MARCO with 8.8M documents and $b = 16$ we get more than 500K blocks and scores per distinct term. However, many scores are zero, and thus we use a sparse representation of the non-zero values.

Document Ordering. The assignment of docIDs to documents can have a major impact on index size and processing speed in many scenarios, including ours here. We assign docIDs according to BP ordering [3], considered state of the art in the area. This leads to sparser and thus more compressible block-max impact scores, and to tighter upper bounds for blocks after aggregation, as documents with similar scores are grouped into one block.

Partial Sorting. After computing block upper bounds, we need to support block access in sorted order. Doing a full sort of the hundreds of thousands of upper bound scores would be very expensive. Instead, we use a simple top- k threshold estimator based on single-term quantiles [25], and sort only the block scores above the threshold using a simple counting sort-based approach.

Forward or Inverted Index. We experimented with several index organizations for evaluating a block, including a standard inverted index, a forward index, and the hybrid structure discussed above which performed best. The inverted index resulted in many expensive pointer movements where we jump from block to block (both forward and backward), while a forward index requires us to separately locate the relevant postings in each document. Thus, our approach does not require any standard inverted index at all.

Termination Conditions. If we process blocks until the top- k threshold in the heap is higher than the upper bound of the next block, we are guaranteed to retrieve the top- k highest scoring results. We can also choose more aggressive approximate policies where we only process a certain number of blocks, or terminate if the current threshold is within a certain factor of the upper bound of the next block. In our approach, the granularity of approximation can be precisely controlled through an adjustable parameter, α (ranging between 0 and 1), which determines the early stopping condition. Specifically, processing of blocks continues until the top- k threshold in the heap exceeds the α -adjusted upper bound of the next block. This mechanism ensures that we can retrieve the top- k highest scoring results with a tunable level of approximation.

In contrast to other approximate methods like SaaT where documents could be partially scored, our approach maintains the integrity of exact document scoring, enabling more accurate and reliable ranking of documents.

Query Term Pruning. Another approximation approach we explored is represented by query term pruning, which involves the removal of certain query terms when their weight is below a given threshold. For this, we make our system tunable using a β parameter to set the percentage of query terms we want to drop.

3 EXPERIMENTAL RESULTS

We assess the performance of the suggested approach by using the well-known MS MARCO Passage [26] dataset. The effectiveness and efficiency of query processing are evaluated by comparing all models against the MSMARCO Dev Queries. The experiments were carried out in memory, utilizing a single thread on a Linux system equipped with dual 2.8 GHz Intel Xeon CPUs and 512 GiB of RAM.

SPLADE¹ refers to the model named CoCondenser-EnsembleDistil in [6], the most effective SPLADE model, where the model has been initialized from a pretrained CoCondenser [8] checkpoint and fine-tuned using knowledge distillation and hard negatives.

ESPLADE² denoted as ESPLADE-V-large in [11], represents a more efficient iteration of SPLADE. Although effective on the MS MARCO dataset, it demonstrates limited capability in generalization compared to its predecessor, as on the BEIR dataset [6, 11].

uniCOIL [12, 34] assigns scalar weights to terms, instead of the vector weights used in the original COIL [9] formulation, and is further enhanced by incorporating a TILDE [35] expansion component.

Table 1: Index space consumption (in GB) for SPLADE.

Block size	8	16	32	64	128	256
Forward Index	7.1	6.0	5.1	4.3	3.7	3.3
BM Index						
Raw	30.0	15.0	7.4	3.7	1.9	0.9
Compressed	5.5	4.1	3.0	2.3	1.7	1.2 ³

We use Anserini [32] for creating the inverted indexes of the collections, export them into the common index file format (CIFF) [13], and reorder using BP [15, 18]. We run MaxScore [31] and BMW [5] using PISA [23]. BMW uses a block size of 40 postings. For Anytime⁴ [17], Clipping⁵ [14], and IOQP⁶ (an efficient implementation of SaaT) [16] we leverage the official reference implementations. Anytime uses MaxScore as its inner DaaT traversal algorithm and the index is split into 100 clusters following similar experimentation in [19]. We also tested Anytime with BMW, but do not report the results as they are worse than with Anytime coupled with MaxScore. We experimented with the guided traversal method presented

in [28] which can only perform approximate retrieval, but we decided not to include the results since the fastest version 2GTI-Fast resulted in longer running times than the slowest of our baseline methods in Table 3 (45.0 ms for SPLADE).

BMP is written in Rust, and was compiled with rustc 1.77 using -O3 optimization as per the default release profile. BMP_{raw} indicates that the BM-index is stored uncompressed, b refers to the block size used to generate the BM-index, and α and β control early stopping and query term pruning, respectively.

Index Size. Table 1 provides a summary of the space required (in GB) to store both the block-based forward index and the additional block-max index structure necessary for implementing cluster-based pruning with SPLADE, across various block sizes. As the block size increases, the size of the forward index consistently decreases. This reduction can be attributed to the fact that larger blocks facilitate more term overlap across documents within the same block, thus reducing redundancy. Regarding the block-max index, it is observed that smaller blocks lead to increased memory requirements. However, when compressed, the extra space needed is less than the forward index size.

We do not compare index sizes between our method and baseline approaches, as the size differences between the inverted and forward indexes are generally minor in the broadest cases (for instance, the uncompressed inverted index for SPLADE is approximately 7.9 GB). While both types of indexes can undergo similar compression techniques, such discussions are beyond the scope of this paper. Instead, our primary focus is on the additional space needed by our filtering structure. It is important to highlight that for many techniques, such as MaxScore, the extra space requirement is minimal. In contrast, BMW necessitates storing an upper bound score for each posting block, leading to higher space consumption.

Overall Comparison. Next, we compare all methods in terms of overall efficiency, for different retrieval depths, and summarize the results in Table 2. We observe that SPLADE is dramatically slower than ESPLADE and uniCOIL for all the strategies, in particular for BMW, MaxScore and Clipping. This is due to the fact that SPLADE strongly leverages query expansion, i.e. it uses queries with a large number of terms, which these algorithms are particularly sensitive to. Conversely, Anytime and IOQP exhibit greater robustness to the variability in query length. Anytime benefits from operating on specific clusters of the index with shorter posting lists, whereas IOQP adopts a more brute-force, term-at-a-time approach during safe searches, simplifying the retrieval process by eliminating the need for additional management of long queries. Among the methods evaluated, BMP stands out as the fastest on SPLADE, achieving speeds 2.9 to 7.5 times faster than its closest competitor across the three retrieval depths examined.

ESPLADE and uniCOIL, while not employing explicit query expansion, present their own challenges: (i) potentially longer queries due to Transformer-based tokenizers, (ii) extended posting lists from additional document expansion, and (iii) skewed score distributions from model fine-tuning. In this context, BMW manages to outperform MaxScore, yet it faces challenges when compared to the competitive performances of Anytime and Clipping. IOQP maintains nearly constant latency across different retrieval depths. BMP demonstrates similar efficiency for both ESPLADE and uniCOIL,

¹<https://huggingface.co/naver/splade-cocondenser-ensembledistil>

²<https://huggingface.co/naver/efficient-splade-v-large-doc>

³Compression introduces some overhead, which is well amortized for small blocks but becomes significant for larger ones.

⁴<https://github.com/JMMackenzie/anytime-daat>

⁵<https://github.com/JMMackenzie/postings-clipping>

⁶<https://github.com/JMMackenzie/IOQP>

Table 2: Query times (in ms) of different exact query processing strategies for $k = 10, 100, 1000$

Strategy	SPLADE						ESPLADE						uniCOIL					
	$k = 10$		$k = 100$		$k = 1000$		$k = 10$		$k = 100$		$k = 1000$		$k = 10$		$k = 100$		$k = 1000$	
MaxScore	120.6	(11.5x)	152.8	(9.6x)	193.8	(7.0x)	13.6	(5.9x)	19.2	(4.2x)	28.2	(2.6x)	14.5	(5.8x)	19.5	(4.1x)	28.8	(2.9)
BMW	614.2	(58.5x)	658.7	(41.4x)	686.7	(24.9x)	10.8	(4.7x)	16.0	(3.5x)	27.0	(2.5x)	12.4	(5.0x)	18.4	(3.8)	31.7	(3.2)
IOQP	79.1	(7.5x)	80.2	(5.0x)	80.8	(2.9x)	27.3	(11.9x)	26.9	(5.9x)	27.6	(2.5x)	34.8	(13.9x)	33.5	(7.0x)	35.5	(3.6x)
Anytime	80.6	(7.7x)	114.0	(7.2x)	163.1	(5.9x)	8.2	(3.6x)	12.6	(2.7x)	20.8	(1.9x)	8.3	(3.3x)	12.4	(2.6x)	19.4	(2.0x)
Clipping	245.9	(23.4)	358.8	(22.6x)	504.1	(18.3x)	9.8	(4.3x)	15.0	(3.3x)	24.8	(2.3x)	9.1	(3.6x)	14.5	(3.0x)	25.6	(2.6x)
BMP																		
$b = 32$	10.5		23.1	(1.5x)	66.9	(2.4x)	2.3		6.6	(1.4x)	26.4	(2.4x)	2.5		6.1	(1.3x)	21.1	(2.1x)
$b = 16$	11.0	(1.1x)	15.9		37.8	(1.4x)	2.6	(1.1x)	4.6		15.1	(1.4x)	3.2	(1.3x)	4.8		12.9	(1.3x)
$b = 8$	15.0	(x1.4)	16.9	(1.1x)	27.6		4.2	(1.8x)	5.1	(1.1x)	10.9		5.0	(2.0x)	5.7	(1.2x)	9.9	

consistently outperforming other strategies by at least a factor of two, particularly at the lowest k values.

Table 3: Query times (in ms) and RR@10 of different approximate query processing strategies for $k = 10$.

Strategy	SPLADE		ESPLADE		uniCOIL	
	MRT	RR@10	MRT	RR@10	MRT	RR@10
Exhaustive	–	38.11	–	38.81	–	35.03
IOQP _{1%}	1.4	23.96	1.2	27.62	1.3	24.30
IOQP _{5%}	3.1	32.22	3.0	34.62	2.8	31.66
IOQP _{10%}	5.3	34.61	5.4	36.63	4.9	33.60
Anytime ₅	13.6	17.08	1.4	21.48	1.3	21.31
Anytime ₁₀	21.9	23.24	2.2	28.96	2.1	28.22
Anytime ₃₀	43.4	33.30	4.4	36.74	4.3	34.08

BMP

$b = 256, \alpha = 0.60$	4.7	35.26	0.6	30.90	0.6	27.95
$b = 128, \alpha = 0.75$	7.6	37.78	1.1	36.66	1.1	33.14
$b = 64, \alpha = 0.85$	8.1	38.05	1.5	38.22	1.5	34.70

+ raw BM Index

$b = 64, \alpha = 0.65$	1.6	35.01	0.4	33.96	0.4	30.29
$b = 64, \alpha = 0.75$	2.7	37.45	0.6	36.52	0.5	33.13
$b = 32, \alpha = 0.85$	3.2	37.97	0.7	38.11	0.7	34.63

Approximate Retrieval. Our next experiment involves approximate retrieval. IOQP can be turned into an approximate method by setting the maximum number of postings to process per query; Mackenzie et al. [16] propose to do that by finding that number as a fraction of the total number of documents in the collection. For this experiment we use 1%, 5%, 10% as fractions. Anytime can be tuned by setting the maximum number of clusters that are accessed per query or by setting a query latency budget. For the maximum number of cluster, we use 5, 10 and 30 out of the 100 total clusters in the index. Although we experimented with a latency budget by aligning it with one of the execution times from BMP tests, we found these results less compelling for the comparison.

Table 3 shows the latency (in ms) for retrieving the top-10 documents in an unsafe scenario. IOQP is the fastest method for SPLADE, but it consistently reduces the effectiveness across the different models w.r.t. the exhaustive, i.e., safe, scenario. Similarly, Anytime negatively impact the effectiveness, while being similarly efficient

as IOQP for ESPLADE and uniCOIL, resulting in the worst trade-off. On the other side, BMP is able to obtain the highest effectiveness with the smallest mean response time across all models. An additional efficiency gain, ranging from 1.5× to 2.5× relative to BMP, is realized when utilizing the raw BM index with BMP. Notably, with our proposed approach, both ESPLADE and uniCOIL achieve sub-millisecond average response times, with a negligible loss of no more than 1% in RR@10 compared to the exhaustive scenario.

Table 4: The impact of varying query term pruning ratio for efficiency and effectiveness w.r.t. SPLADE.

β	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
MRT	0.5	0.9	1.2	1.5	1.8	2.2	2.5	2.8	3.0	3.2
RR@10	30.38	35.81	37.31	37.78	38.13	38.12	38.03	38.04	38.06	37.97

Another pruning mechanism can be applied to query terms based on their term weights. Table 4 illustrates the impact of query term pruning when executed by BMP on SPLADE. BMP achieves sub-millisecond retrieval times with only a slight decrease in precision.

4 CONCLUSIONS AND FUTURE WORK

Learned sparse models offer significant improvements in retrieval quality while narrowing the efficiency gap between neural retrieval and quicker traditional similarity models. In this paper, we introduced BMP, a query processing strategy designed to address the efficiency challenges inherent in learned sparse retrieval. Future research will aim to investigate how BMP compares to graph-based approximate nearest neighbor (ANN) algorithms when applied to sparse retrieval, offering potential insights into further optimizing retrieval efficiency.

Acknowledgments

This work is supported, in part, by the Spoke “FutureHPC & Big-Data” of the ICSC – Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing, the Spoke “Human-centered AI” of the M4C2 - Investimento 1.3, Partenariato Esteso PE00000013 - “FAIR - Future Artificial Intelligence Research”, funded by European Union – NextGenerationEU, the FoReLab project (Departments of Excellence), and the NEREO PRIN project (2022AEF-HAZ) funded by the Italian Ministry of Education and Research.

REFERENCES

- [1] Vo Ngoc Anh, Owen de Kretser, and Alistair Moffat. 2001. Vector-space ranking with effective early termination. In *Proc. SIGIR*. 35–42.
- [2] Kaushik Chakrabarti, Surajit Chaudhuri, and Venkatesh Ganti. 2011. Interval-based pruning for top-k processing over compressed lists. In *Proc. ICDE*. 709–720.
- [3] Laxman Dhulipala, Igor Kabiljo, Brian Karrer, Giuseppe Ottaviano, Sergey Pupyrev, and Alon Shalita. 2016. Compressing graphs and indexes with recursive graph bisection. In *Proc. SIGKDD*. 1535–1544.
- [4] Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. 2013. A candidate filtering mechanism for fast top-k query processing on modern cpus. In *Proc. SIGIR*. 723–732.
- [5] Shuai Ding and Torsten Suel. 2011. Faster Top-k Document Retrieval Using Block-max Indexes. In *Proc. SIGIR*. 993–1002.
- [6] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From distillation to hard negative sampling: Making sparse neural ir models more effective. In *Proc. SIGIR*. 2353–2359.
- [7] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *Proc. SIGIR*. 2288–2292.
- [8] Luyu Gao and Jamie Callan. 2022. Unsupervised Corpus Aware Language Model Pre-training for Dense Passage Retrieval. In *Proc. ACL*. 2843–2853.
- [9] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List. In *Proc. NAACL-HLT*. 3030–3042.
- [10] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proc. SIGIR*. 39–48.
- [11] Carlos Lassance and Stéphane Clinchant. 2022. An efficiency study for SPLADE models. In *Proc. SIGIR*. 2220–2226.
- [12] Jimmy Lin and Xueguang Ma. 2021. A Few Brief Notes on DeepImpact, COIL, and a Conceptual Framework for Information Retrieval Techniques. *Preprint: arXiv:2106.14807* (2021).
- [13] Jimmy Lin, Joel Mackenzie, Chris Kamphuis, Craig Macdonald, Antonio Mallia, Michał Siedlaczek, Andrew Trotman, and Arjen de Vries. 2020. Supporting interoperability between open-source search engines with the common index file format. In *Proc. SIGIR on Research and Development in Information Retrieval*. 2149–2152.
- [14] Joel Mackenzie, Antonio Mallia, Alistair Moffat, and Matthias Petri. 2022. Accelerating learned sparse indexes via term impact decomposition. In *Proc. EMNLP*. 2830–2842.
- [15] Joel Mackenzie, Antonio Mallia, Matthias Petri, J Shane Culpepper, and Torsten Suel. 2019. Compressing inverted indexes with recursive graph bisection: A reproducibility study. In *Proc. ECIR*. Springer, 339–352.
- [16] J. Mackenzie, M. Petri, and L. Gallagher. 2022. IOQP: A simple Impact-Ordered Query Processor written in Rust. In *Proc. DESIRES*. 22–34.
- [17] Joel Mackenzie, Matthias Petri, and Alistair Moffat. 2021. Anytime ranking on document-ordered indexes. *ACM TOIS* 40, 1 (2021), 1–32.
- [18] Joel Mackenzie, Matthias Petri, and Alistair Moffat. 2021. Faster index reordering with bipartite graph partitioning. In *Proc. SIGIR*. 1910–1914.
- [19] Joel Mackenzie, Andrew Trotman, and Jimmy Lin. 2023. Efficient document-at-a-time and score-at-a-time query evaluation for learned sparse representations. *ACM TOIS* 41, 4 (2023), 1–28.
- [20] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonello. 2021. Learning Passage Impacts for Inverted Indexes. In *Proc. SIGIR*. 1723–1727.
- [21] Antonio Mallia, Joel Mackenzie, Torsten Suel, and Nicola Tonello. 2022. Faster Learned Sparse Retrieval with Guided Traversal. In *Proc. SIGIR*. 1901–1905.
- [22] Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonello, and Rossano Venturini. 2017. Faster BlockMax WAND with Variable-sized Blocks. In *Proc. SIGIR*. 625–634.
- [23] Antonio Mallia, Michał Siedlaczek, Joel Mackenzie, and Torsten Suel. 2019. PISA: Performant indexes and search for academia. In *Proc. OSIRRC@SIGIR*.
- [24] Antonio Mallia, Michał Siedlaczek, and Torsten Suel. 2021. Fast disjunctive candidate generation using live block filtering. In *Proc. WSDM*. 671–679.
- [25] Antonio Mallia, Michał Siedlaczek, Mengyang Sun, and Torsten Suel. 2020. A comparison of top-k threshold estimation techniques for disjunctive query processing. In *Proc. CIKM*. 2141–2144.
- [26] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human Generated Machine Reading Comprehension Dataset. In *Proc. CoCo@NIPS*.
- [27] Matthias Petri, J. Shane Culpepper, and Alistair Moffat. 2013. Exploring the Magic of WAND. In *Proc. ADCS*. 58–65.
- [28] Yifan Qiao, Yingrui Yang, Haixin Lin, and Tao Yang. 2023. Optimizing Guided Traversal for Fast Learned Sparse Retrieval. In *Proc. WWW*. 3375–3385.
- [29] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (2009), 333–389.
- [30] Nicola Tonello, Craig Macdonald, and Iadh Ounis. 2018. Efficient Query Processing for Scalable Web Search. *Found. Trends in Inf. Retr.* 12, 4–5 (2018), 319–492.
- [31] Howard Turtle and James Flood. 1995. Query evaluation: strategies and optimizations. *Information Processing & Management* 31, 6 (1995), 831–850.
- [32] Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the use of Lucene for information retrieval research. In *Proc. SIGIR*. 1253–1256.
- [33] Jingtao Zhan, Jiabin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. Optimizing Dense Retrieval Model Training with Hard Negatives. In *Proc. SIGIR*. 1503–1512.
- [34] Shengyao Zhuang and Guido Zuccon. 2021. Fast passage re-ranking with contextualized exact term matching and efficient passage expansion. *arXiv preprint arXiv:2108.08513* (2021).
- [35] Shengyao Zhuang and Guido Zuccon. 2021. TILDE: Term Independent Likelihood Model for Passage Re-Ranking. In *Proc. SIGIR*. 1483–1492.