# Efficiency Implications of Term Weighting for Passage Retrieval

Joel Mackenzie
The University of Melbourne
joel.mackenzie@unimelb.edu.au

Zhuyun Dai
Carnegie Mellon University
zhuyund@cs.cmu.edu

Luke Gallagher
RMIT University
luke.gallagher@rmit.edu.au

Jamie Callan
Carnegie Mellon University
callan@cs.cmu.edu

## ABSTRACT

Language model pre-training has spurred a great deal of attention for tasks involving natural language understanding, and has been successfully applied to many downstream tasks with impressive results. Within information retrieval, many of these solutions are too costly to stand on their own, requiring multi-stage ranking architectures. Recent work has begun to consider how to "backport" salient aspects of these computationally expensive models to previous stages of the retrieval pipeline. One such instance is DeepCT, which uses BERT to re-weight term importance in a given context at the passage level. This process, which is computed offline, results in an augmented inverted index with re-weighted term frequency values. In this work, we conduct an investigation of query processing efficiency over DeepCT indexes. Using a number of candidate generation algorithms, we reveal how term re-weighting can impact query processing latency, and explore how DeepCT can be used as a static index pruning technique to accelerate query processing *without* harming search effectiveness.

**ACM Reference Format:**
Joel Mackenzie, Zhuyun Dai, Luke Gallagher, and Jamie Callan. 2020. Efficiency Implications of Term Weighting for Passage Retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3397271.3401263

## 1 INTRODUCTION

Neural models for document re-ranking have benefited from a surge in performance recently due to the success of unsupervised pre-training on large text corpora, resulting in models such as ELMo and BERT [7, 21]. These models are the foundation for many recent state-of-the-art results on natural language tasks. As such, there is growing interest in utilizing pre-trained models to enhance existing components of the retrieval pipeline beyond document re-ranking [5, 20, 26]. However, approaches that augment an inverted index to improve search effectiveness are yet to be scrutinized with respect to search efficiency beyond some simple measures such as inference throughput [20] or mean query latency [19].


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

ACM ISBN 978-1-4503-8016-4/20/07…$15.00
https://doi.org/10.1145/3397271.3401263

In this work, we conduct a detailed comparison of various inverted indexes that can be obtained via DeepCT-Index [5], a recently proposed contextual term re-weighting framework. After confirming the effectiveness of DeepCT-Index, we show that it does *not* add latency to the candidate generation stage of the search system, supporting the conjecture made by Dai and Callan [5]. Moreover, we show that DeepCT-Index can actually improve search efficiency via static index pruning, making it promising for use in production information retrieval systems.

## 2 BACKGROUND

### 2.1 Deep Contextualized Term Weighting

Many retrieval models use term frequency (*tf*) to estimate document-specific term weights. However, term frequency is independent of the specific linguistic context in which the term occurs, which is crucial to understanding a term's importance. Motivated by this observation, Dai and Callan [5] proposed a novel approach to estimate document-specific term weights using a *deep contextualized term weighting* framework (DeepCT). First, it generates contextualized term embeddings using BERT [7]. These term embeddings embody not only the token itself, but are also conditioned on its surrounding text. Next, DeepCT maps these embeddings into context-specific term weights. The predicted term weights are used to re-weight the original *tf* field in the inverted index. In some cases, DeepCT assigns term weights $\leq 0$, which can be viewed as a form of index pruning. DeepCT-Index was shown to significantly outperform classic *tf* retrieval methods in the context of both passage ranking [5] and web search [6]. Our experiments focus on how these re-weighted *tf* values impact search efficiency.

### 2.2 Efficient Query Processing

Large-scale search engines must effectively balance efficiency and effectiveness to provide a good user experience. Hence, these systems are often implemented as multi-stage cascades, where a set of increasingly sophisticated ranking models re-rank a decreasing number of documents, arriving at a final ranked results list [24]. The first stage, candidate generation, aims to rapidly retrieve a large set of documents that are likely to be relevant to the given query. Our work focuses on this stage of the retrieval pipeline.

**Index Traversal and Dynamic Pruning**. While various index organizations have been proposed, the most common technique stores a postings list for each term in the index, comprised of a list of document identifiers and corresponding term frequencies in

ascending order of the document identifier. During Document-at-a-Time query processing, the postings lists that correspond to the query terms are accessed and traversed at once, such that a single document is evaluated at a time. A min-heap is used to store the top-$k$ results that have been observed, and returns these results after processing terminates.

To increase the efficiency of index traversal, a number of *dynamic pruning* algorithms have been proposed. Assuming the scoring model is additive, these algorithms pre-compute and store the highest possible score of each term. During query processing, these upper-bound values are then used to determine whether a document could achieve a score that exceeds the current heap threshold — if so, the document will be scored. Otherwise, the document can be skipped, resulting in faster query processing. Both MaxScore [23], and WAND [2] accelerate query processing in this manner. Further enhancements can be achieved by storing a number of per-block upper-bound scores, allowing tighter estimations on the maximum possible score for each document. The state-of-the-art BMW [8] and VBMW [14] algorithms follow this approach. We refer the interested reader to the survey of Tonellotto et al. [22] and recent empirical comparisons [4, 16] for more details.

**Static Index Pruning**. Another way to reduce both latency and storage costs is to remove documents or postings at index time. Good candidates for document pruning are those likely to hinder a search system's effectiveness via the contribution of noise to the search results, such as spam documents, for example. A document prior may be used to remove these documents, reducing the index size, which has the effect of changing the underlying index statistics in a fairly course grained manner.

A primitive form of postings removal is the omission of stop words during indexing. Other term level approaches typically modify frequency statistics which are used to calculate a score for a given term and document, where the score is used to derive various term pruning strategies [3, 9]. Other term based methods modify the underlying statistics in a way that results in the terms being *re-weighted* rather than solely discounted [1].

## 3 EXPERIMENTS

**Datasets and Queries**. Our experiments are conducted on the MS MARCO passage collection, which contains 8.8 million passages extracted from 3.6 million documents [18]. The query set contains 6,980 natural language questions with associated relevance judgments from the *dev* set, allowing us to explore both efficiency and effectiveness. We use both an *unstopped* and a *stopped* version of this query set for our experiments. We build a number of different indexes across the collection, each using the same end-to-end tokenization and indexing pipeline to ensure a fair comparison. Since we are interested in examining how both the term re-weighting and the static pruning components of DeepCT-Index impact efficiency, we build four unique indexes: `Orig-U` represents the *default* inverted index containing the original *tf* values; `Orig-P` is the `Orig-U` index, but static pruning is applied using DeepCT-Index; `DeepCT-U` stores re-weighted *tf* values, but includes the original *tf* value where pruning occurs; and `DeepCT-P` stores re-weighted *tf* values, allowing pruning to take place. A pictorial representation is shown in Figure 1.

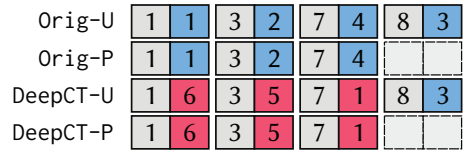


**Figure 1:** A pictorial example of a single postings list for each of the indexes, shown as docid/*tf* pairs. Blue cells represent original *tf* values from the corpus, red cells represent re-weighted *tf* values, and dashed lines denote cases where DeepCT-Index applied a weight $\leq 0$, resulting in the posting being pruned.

**Table 1:** Basic statistics for the `Unpruned` and `Pruned` representations of the MS MARCO passage corpus.

| Index | Documents | Terms | Postings |
|---|---|---|---|
| `Unpruned` | 8,841,823 | 1,515,955 | 265,718,705 |
| `Pruned` | 8,841,796 | 989,873 | 128,969,826 |

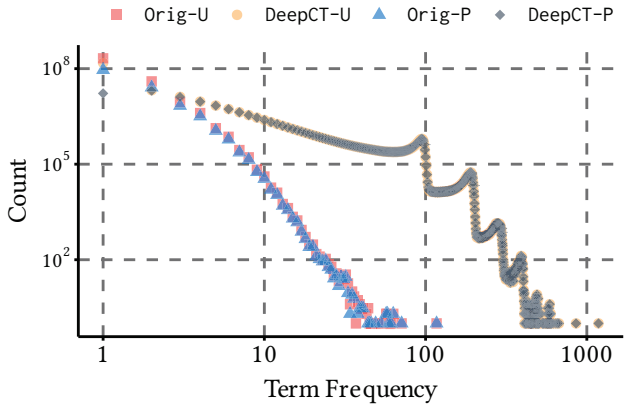


**Figure 2:** Number of times different term frequencies are observed across each of the indexes. Note the log-log scale.

Table 1 shows some basic statistics for the indexes. As expected, the `Pruned` indexes contain fewer documents, postings, and terms than their `Unpruned` counterparts. Figure 2 plots the number of times each term frequency is observed across each index. Clearly, the DeepCT indexes contain a much larger range of term frequencies, with a higher density of larger values. The wave-like artifact observed for DeepCT indexes results from the term weights being applied on the raw corpus before stemming, causing the frequencies of some terms to spike.

**Hardware and Software**. We use Anserini [25] to generate the inverted indexes for each of the collections. We then export the Anserini indexes using the *common index file format* [13], and load them into PISA [15]. BM25 was used for ranking, with the optimal parameters selected as reported by Dai and Callan [5]. Indexes were compressed in fixed-sized blocks of 128 elements with SIMD-BP128 [12], which was recently shown to give good time-space trade-offs during decoding [12, 16]. Experiments are conducted on a Linux machine with two Intel Xeon Gold 6144 CPUs (3.50GHz) and

**Table 2:** Effectiveness scores for each index, using both unstopped and stopped query logs. The ‡ symbol represents significant differences with respect to `Orig-U`.

| Index | *Unstopped* | | | *Stopped* | | |
|---|---|---|---|---|---|---|
| | AP | Recall | MRR | AP | Recall | MRR |
| `Orig-U` | 0.195 | 0.860 | 0.188 | 0.203 | 0.873 | 0.195 |
| `DeepCT-U` | 0.252‡ | 0.911‡ | 0.244‡ | 0.256‡ | 0.910‡ | 0.249‡ |
| `Orig-P` | 0.203‡ | 0.861 | 0.195‡ | 0.215‡ | 0.877 | 0.206‡ |
| `DeepCT-P` | 0.252‡ | 0.906‡ | 0.244‡ | 0.253‡ | 0.907‡ | 0.245‡ |

512 GiB of RAM. Experiments which collect timings use a single processing thread, and take the average of three independent runs. Our experiments are made available for reproducibility.[1]

## 3.1 Effectiveness Evaluation

Our first experiment aims to reproduce the early-stage effectiveness improvements that are achieved by DeepCT-Index [5]. We retrieve the top $k = 1000$ documents for each query, and report the values of average precision (AP) and recall at depth 1000, and mean reciprocal rank (MRR) at depth 10. We conduct Bonferroni corrected pairwise $t$-tests, and report significance with $p < 0.01$. Table 2 shows the results. As expected, we are able to closely replicate the effectiveness scores suggested by earlier work [5, 19], with minor differences arising due to the underlying BM25 computation [10]. In comparing the effectiveness between the four indexes, it is clear that the majority of the improvements arise due to the term frequency weighting provided by DeepCT-Index. However, significant gains in effectiveness for AP and MRR are observed when comparing `Orig-P` to `Orig-U`, suggesting that using DeepCT-Index for static index pruning alone can improve effectiveness.

## 3.2 Efficiency Analysis

Our next experiment aims to quantify the efficiency of top-$k$ retrieval across the four indexes. Along with a typical exhaustive ranked disjunction (RankedOR), we employ a number of efficient dynamic pruning algorithms, including MaxScore, WAND, and BMW with fixed-sized blocks of 40 elements.[2] Here, we report timings from the stopped log only due to space constraints. Figure 3 reports the time taken to retrieve the top 1000 documents in milliseconds as a 'Tukey' boxplot; the boxes bound the 25th to 75th percentiles, whiskers cover data within 1.5× the IQR, and outliers are plotted as points. For convenience, we also report some common efficiency metrics in Table 3. Finally, we report the index size and sequential decoding speed in Table 4.

**Effect of Pruning**. Figure 3 shows that, irrespective of the value of $k$ or the query processing algorithm, retrieval over the `Pruned` indexes is much more efficient than retrieval over the `Unpruned` indexes. This benefit comes from the given retrieval algorithm scoring fewer documents and postings during traversal. Pruning is very effective in reducing tail latency, with improvements in $P_{99}$ of up to 2.5× for RankedOR, 1.6× for MaxScore, 2.1× for WAND, and 2.0× for BMW. Interestingly, the dynamic pruning algorithms generally evaluate fewer documents on the `Pruned` indexes, indicating that both static and dynamic pruning approaches may be able to interact to achieve *additive* efficiency gains. A similar effect has been noted in prior work on *selective search* [11].

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

[2] We also experimented with VBMW, but noted a very similar efficiency to the fixed-block BMW, so report only the latter for clarity.

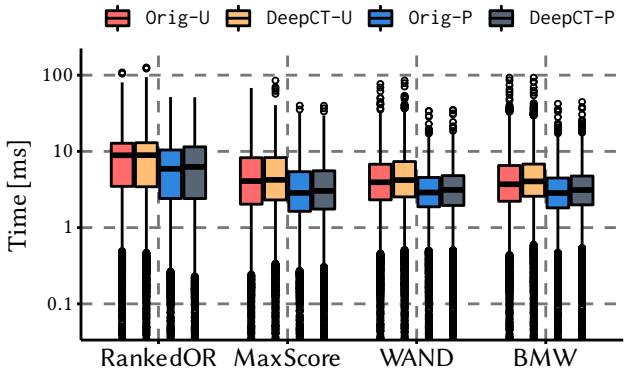


**Figure 3:** Query latency in milliseconds for retrieving the top 1000 passages across each index for each query processing algorithm. Note the log scale on the $y$-axis.

**Table 3:** Median latency ($P_{50}$), 99th percentile latency ($P_{99}$), and the *mean* number of documents scored in thousands (*Docs*) for retrieval across all algorithms and indexes for $k = 1000$.

| Algorithm | Measure | `Orig-U` | `DeepCT-U` | `Orig-P` | `DeepCT-P` |
|---|---|---|---|---|---|
| RankedOR | $P_{50}$ | 8.8 | 8.9 | 5.9 | 6.3 |
| | $P_{99}$ | 42.3 | 48.2 | 18.9 | 19.4 |
| | *Docs* | 447.9 | 447.9 | 241.5 | 241.5 |
| MaxScore | $P_{50}$ | 4.1 | 4.3 | 2.9 | 3.0 |
| | $P_{99}$ | 24.0 | 25.9 | 15.9 | 16.5 |
| | *Docs* | 152.6 | 163.4 | 99.7 | 101.5 |
| WAND | $P_{50}$ | 3.9 | 4.2 | 2.9 | 3.1 |
| | $P_{99}$ | 24.5 | 28.1 | 13.5 | 13.2 |
| | *Docs* | 58.7 | 75.9 | 46.9 | 53.0 |
| BMW | $P_{50}$ | 3.7 | 4.0 | 2.8 | 3.1 |
| | $P_{99}$ | 25.6 | 25.5 | 13.0 | 14.0 |
| | *Docs* | 27.2 | 35.3 | 24.5 | 29.6 |

**Table 4:** Total index size (MiB), compression rate (bits per integer), and sequential decoding time (nanoseconds per integer) for the four inverted indexes, broken down by document identifiers (Docs) and term frequencies (Freqs).

| Index | Index Size | | | Decoding Time | |
|---|---|---|---|---|---|
| | Total | Docs | Freqs | Docs | Freqs |
| `Orig-U` | 433 | 11.1 | 2.4 | 0.59 | 0.56 |
| `DeepCT-U` | 534 | 11.1 | 5.6 | 0.59 | 0.63 |
| `Orig-P` | 240 | 12.8 | 2.8 | 0.80 | 0.78 |
| `DeepCT-P` | 312 | 12.8 | 7.4 | 0.81 | 0.87 |

Examining Table 4, we observe that the Pruned indexes are slower to decode than their Unpruned equivalents. This is because the SIMD-BP128 compression codec cannot compress blocks with fewer than 128 values, and reverts to using binary interpolative coding [17] (BIC) for those blocks. Examining the indexes shows that 5% of the postings in the Unpruned indexes are compressed with BIC, compared to 8% in the Pruned indexes. Furthermore, since the postings lists are shorter in the Pruned indexes, on average, the instruction throughput is lower, resulting in less efficient decompression and processing. While we also note a slight reduction in compression rate for the Pruned indexes, these reductions are far outweighed by the *absolute* time and space improvements achieved from pruning.

**Effect of Term Weighting**. Indexes which employ DeepCT term frequencies exhibit slightly slower retrieval than those which use the default *tf* values. This trend can be confirmed by comparing the columns of Table 3, and is most evident for the 99th percentile latency. Interestingly, this trend occurs even for the exhaustive RankedOR algorithm, which processes the same volume of documents and postings regardless of the internal term-frequency values. To further understand this phenomenon, we refer to Table 4. Evidently, the DeepCT-Index term frequencies are much harder to compress, with a 2.3× and 2.6× increase on bits per integer for Unpruned and Pruned, respectively, due to the increased size and variance of the *tf* values in the DeepCT indexes (see Figure 2). As such, a larger amount of memory must be accessed to decode the frequencies of the DeepCT-U and DeepCT-P indexes, resulting in increased decoding time. Thus, an unexpected effect of term re-weighting is that it can negatively impact the compression of the postings lists, resulting in slight performance penalties.

### 3.3 Summary

Based on our analysis, we highlight a number of key observations:

- Effectiveness improvements from DeepCT-Index are mostly due to the term re-weighting component of the algorithm, though the static pruning component also contributes to effectiveness improvements.
- From an efficiency perspective, using DeepCT-Index as a context-aware static pruning algorithm is very promising.
- Static and dynamic pruning methods may achieve *additive* efficiency gains, warranting further exploration.
- Term re-weighting can decrease the compression rate of the underlying *tf* data, which results in a slight performance penalty with respect to the original *tf* values.

## 4 CONCLUSION AND FUTURE WORK

With continuing improvements in neural language modeling, it is important to understand how these frameworks interact with traditional search systems. In this work, we take a first step to understanding the efficiency implications that such models have on the efficacy of candidate generation for multi-stage retrieval systems. In particular, we explore how term re-weighting and static pruning from the recently proposed DeepCT-Index impacts latency. Our findings show that, while re-weighted term frequencies add slight costs to both storage and decompression, these costs are easily won back by the static pruning conducted by DeepCT-Index.

In future work, we would like to compare the static pruning capabilities of DeepCT-Index to other static pruning baselines [9], and investigate the impact of term pruning and re-weighting for larger collections such as those used for web search tasks.

**Acknowledgments**. We thank the anonymous reviewers for improving this paper. This work was supported by the Australian Research Council (ARC) Discovery Grant DP170102231, an Australian Government Research Training Program Scholarship, and the National Science Foundation (NSF) grant IIS-1815528.

## REFERENCES

[1] R. Blanco and Á Barreiro. 2010. Probabilistic Static Pruning of Inverted Files. *ACM Trans. Information Systems* 28, 1 (2010), 1:1–1:33.
[2] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. 2003. Efficient Query Evaluation Using a Two-level Retrieval Process. In *Proc. CIKM*. 426–434.
[3] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. S. Maarek, and A. Soffer. 2001. Static Index Pruning for Information Retrieval Systems. In *Proc. SIGIR*. 43–50.
[4] M. Crane, J. S. Culpepper, J. Lin, J. Mackenzie, and A. Trotman. 2017. A Comparison of Document-at-a-Time and Score-at-a-Time Query Evaluation. In *Proc. WSDM*. 201–210.
[5] Z. Dai and J. Callan. 2019. Context-Aware Sentence/Passage Term Importance Estimation for First Stage Retrieval. *arXiv preprint arXiv:1910.10687* (2019).
[6] Z. Dai. and J. Callan. 2020. Context-Aware Document Term Weighting for Ad-Hoc Search. In *Proc. WWW*. 1897–1907.
[7] J. Devlin, M-W. Chang, K. Lee, and K. Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. NAACL-HLT*. 4171–4186.
[8] S. Ding and T. Suel. 2011. Faster top-$k$ document retrieval using block-max indexes. In *Proc. SIGIR*. 993–1002.
[9] W. Jiang, J. Rodriguez, and T. Suel. 2017. Improved methods for static index pruning. In *Proc. IEEE BigData*. 686–695.
[10] C. Kamphuis, A. de Vries, L. Boytsov, and J. Lin. 2020. Which BM25 Do You Mean? A Large-Scale Reproducibility Study of Scoring Variants. In *Proc. ECIR*. 28–34.
[11] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. 2016. Does Selective Search Benefit from WAND Optimization?. In *Proc. ECIR*. 145–158.
[12] D. Lemire and L. Boytsov. 2015. Decoding billions of integers per second through vectorization. *Soft. Prac. & Exp.* 41, 1 (2015), 1–29.
[13] J. Lin, J. Mackenzie, C. Kamphuis, C. Macdonald, A. Mallia, M. Siedlaczek, A. Trotman, and A. de Vries. 2020. Supporting Interoperability Between Open-Source Search Engines with the Common Index File Format. In *Proc. SIGIR*. To Appear.
[14] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini. 2017. Faster BlockMax WAND with Variable-sized Blocks. In *Proc. SIGIR*. 625–634.
[15] A. Mallia, M. Siedlaczek, J. Mackenzie, and T. Suel. 2019. PISA: Performant Indexes and Search for Academia. In *Proc. of the Open-Source IR Replicability Challenge (OSIRRC) at SIGIR 2019*. 50–56.
[16] A. Mallia, M. Siedlaczek, and T. Suel. 2019. An Experimental Study of Index Compression and DAAT Query Processing Methods. In *Proc. ECIR*. 353–368.
[17] A. Moffat and L. Stuiver. 2000. Binary Interpolative Coding for Effective Index Compression. *Information Retrieval* 3, 1 (2000), 25–47.
[18] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In *Proc. NeurIPS Workshop on Cognitive Computation*. 96–105.
[19] R. Nogueira and J. Lin. 2019. From doc2query to docTTTTTquery. *Online preprint* (2019).
[20] R. Nogueira, W. Yang, J. Lin, and K. Cho. 2019. Document Expansion by Query Prediction. *arXiv preprint arXiv:1904.08375* (2019).
[21] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proc. NAACL-HLT*. 2227–2237.
[22] N. Tonellotto, C. Macdonald, and I. Ounis. 2018. Efficient Query Processing for Scalable Web Search. *Found. Trends Inf. Ret.* 12, 4-5 (2018), 319–500.
[23] H. R. Turtle and J. Flood. 1995. Query Evaluation: Strategies and Optimizations. *Inf. Proc. & Man.* 31, 6 (1995), 831–850.
[24] L. Wang, J. Lin, and D. Metzler. 2011. A Cascade Ranking Model for Efficient Ranked Retrieval. In *Proc. SIGIR*. 105–114.
[25] P. Yang, H. Fang, and J. Lin. 2018. Anserini: Reproducible Ranking Baselines Using Lucene. *J. Data Inf. Qual.* 10, 4 (2018), 16.1–16.20.
[26] H. Zamani, M. Dehghani, W. B. Croft, E. Learned-Miller, and J. Kamps. 2018. From Neural Re-Ranking to Neural Ranking: Learning a Sparse Representation for Inverted Indexing. In *Proc. CIKM*. 497–506.