Accelerating Learned Sparse Indexes Via Term Impact Decomposition

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

Antonio Mallia
Amazon Alexa, Italy
malliaam@amazon.com

Alistair Moffat
University of Melbourne, Australia
ammoffat@unimelb.edu.au

Matthias Petri
Amazon Alexa, USA
mkp@amazon.com

Abstract

Novel inverted index-based *learned sparse ranking models* provide more effective, but less efficient, retrieval performance compared to traditional ranking models like BM25. In this paper, we introduce a technique we call *postings clipping* to improve the query efficiency of learned representations. Our technique amplifies the benefit of dynamic pruning query processing techniques by accounting for changes in term importance distributions of learned ranking models. The new clipping mechanism accelerates top-k retrieval by up to $9.6\times$ without any loss in effectiveness.

1 Introduction

Sparse term importance representations such as DeepImpact (Mallia et al., 2021) and uniCOIL (Gao et al., 2021; Lin and Ma, 2021) have enabled the use of effective transformer-based text representations that can match the effectiveness of recent dense text representations (Karpukhin et al., 2020; Qu et al., 2021) while still being supported by *inverted indexes* and their query operations. This is of importance as inverted indexes have been optimized to provide search functionality in distributed settings at web-scale through 40 years of research, providing a variety of time, space and retrieval quality tradeoffs; while also supporting efficient updates, advanced querying modes such as phrase matching or filtering, and good scalability, all of which are crucial in real-world settings (Risvik et al., 2013; Tonellotto et al., 2018).

One of the key techniques that enables efficient top-k query processing in inverted indexes is storing additional metadata about index term importance scores (also referred to as impacts), seeking to facilitate the bypassing of the majority of the matching documents, and thus allowing faster retrieval than would be possible via exhaustive disjunctive processing. For example, dynamic pruning algorithms such as MaxScore (Turtle and Flood, 1995) and WAND (Broder et al., 2003) store the index-wide maximum impact of each term; at query-time, these impacts can be used to rapidly estimate document scores, allowing documents that have no prospect of entering the current min-heap of k results to be bypassed.

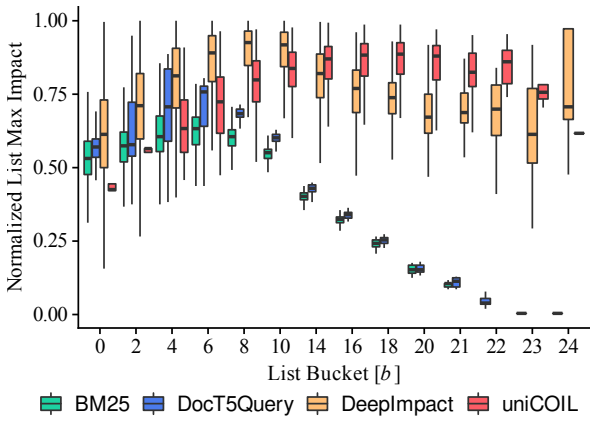

Figure 1: Normalized maximum list impact distribution stratified by list length buckets $b \in [2^b, 2^{b+1})$. Classical schemes such as BM25 and DocT5Query exhibit small maximum impacts for long lists, whereas DeepImpact and uniCOIL assign high importance to terms regardless of their document frequency. Note the irregular scale on the horizontal axis.

Traditional similarity models such as BM25 guarantee that frequent terms have low importance scores, a symbiotic relationship that allows fast query processing. On the other hand, recent transformer-based learned term importance techniques such as DeepImpact are not constrained by term occurrence frequency when assigning importance scores to terms in documents. For example, consider the term "does" in an English corpus. Traditional models such as BM25 would assign a low impact for this term in all documents, since it occurs so frequently; a direct effect of *inverse document frequency* (IDF). On the other hand, learned

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 2830 - 2842
December 7-11, 2022

Impact	MSMARCO-v1 Passage
0.91	**Does** are the females in the deer family of mammals, individually called a doe (pronounced doe as in toe). The plural of doe can also be doe. **Does** is also the word meaning the present tense of the verb do. Pronounced duz as opposed to the pronunciation for the female deer.
0.71	You spelled it right in the question: **does**. The word **does** (performs action) is a verb, and the plural of the noun doe (female deer). Many people mix up two similar words: dose and **does**. Dose is a noun and is the amount of medication prescribed. **Does** is a verb, a form of to do.
0.59	Job Seekers The District of Columbia Department of Employment Services (**DOES**) was created to develop Jobs for People and People for Jobs. **DOES** provides job seekers with a number of employment opportunities through its American Job Centers.
0.50	Take your medication exactly as prescribed. Taking higher **does** of Benzedrine may cause a change in a person's sex drive, allergic reactions, chills, depression, irritability or mood swings, and problems with the digestive system.
0.02	when my dog passes wind its the worst smell ever. **does** anyone know how to stop it smelling so bad. Add your answer.

Table 1: Sample MSMARCO-v1 passages and the normalized impact assigned by DeepImpact to the word "does", which occurs in 61% of all passages. High impact scores can be assigned to common terms legitimately (based on the context of the term), or may be caused by misspellings, acronyms, or homonyms.

models are trained to exploit the *contextual information* of a passage to assign term impacts, resulting in high importance scores for even the most common terms. Table 1 demonstrates this behavior for DeepImpact, where passages extracted from MSMARCO-v1 contain normalized impact scores of widely varying magnitudes. While DeepImpact assigns impacts from 0.91 (very important) all the way down to 0.02 (not important) for the term "does", the equivalent *maximum impact* observed over our two BM25-based indexes (see Section 4) are 0.21 (BM25) and 0.004 (DocT5Query).

Figure 1 further highlights the pervasive nature of this issue. Learned representations such as DeepImpact or uniCOIL assign high term importance to even very frequent terms (such as "does"), whereas BM25 always assigns low importance to such terms. This divergent behavior substantially reduces the ability of MaxScore and WAND to bypass low-impact documents during querying, with both techniques relying on maximum list-wise impact scores to prune the search space.

Contribution We adapt the MaxScore and WAND dynamic pruning mechanisms to enable efficient query processing for learned term importance schemes such as DeepImpact via a simple technique we call *term impact decomposition*. We describe partitioning schemes that separate the postings for each term into two groups – those that are *high-impact*, and are likely to result in documents being scored; and those that are *low-impact*, and more likely to be associated with bypassed documents; and present a new form of impact decomposition that we call *postings clipping*. When integrated into the retrieval engine, impact decomposition allows almost $10\times$ faster top-k term-based querying, with negligible increases to index storage costs, and no effect on result quality.

2 Background

Term-Based Similarity Many retrieval similarity formulations can be expressed as a sum over per-document query-term impacts, computed for document d and query Q as

$$S(Q, d) = C(d) + \sum_{t \in Q} w_{t,d} \qquad (1)$$

where $C(d)$ is a static score component associated with document d; and $w_{t,d}$ is the importance, or *impact*, of term t in document d (see, for example, Zobel and Moffat, 2006). The values of $w_{t,d}$ might be pre-computed at indexing time and stored in the inverted index in quantized form; or might be computed via a function $F()$ from raw index statistics such as term frequency and document length.

Learned Sparse Models The recent development of pre-trained contextualized language models (LMs) has resulted in impressive benefits in search effectiveness, albeit with higher retrieval cost than traditional lexical models (MacAvaney et al., 2019; Pradeep et al., 2021; Khattab and Zaharia, 2020). This has motivated recent work on making transformer-based ranking more efficient (Cohen et al., 2022; Karpukhin et al., 2020; Zhan et al., 2021).

Different solutions have been proposed to address this performance bottleneck, including the

application of approximate nearest neighbor search on dense representations (see, for example, Izacard et al., 2020; Zhan et al., 2022; Yamada et al., 2021). Another approach is to apply LMs to improve the effectiveness of inverted index-based "sparse" representations.

Document expansion is one such innovation. It uses LMs to predict expansion terms to add to each document in an effort to address the vocabulary mismatch problem (Zhao, 2012), while still applying traditional scoring regimes like BM25. Currently, DocT5Query (Nogueira and Lin, 2019) and TILDE (Zhuang and Zuccon, 2021b,a) are the most effective expansion methods.

LMs can also be used to *learn* term importance directly. Early approaches such as DeepCT (Dai and Callan, 2019) learned an updated term frequency value which could be plugged into the existing ranking model. Other more effective approaches such as DeepImpact (Mallia et al., 2021), uniCOIL (Lin and Ma, 2021; Ma et al., 2022), TILDE (Zhuang and Zuccon, 2021a),[1] and SPLADE (Formal et al., 2021b,a) have been devised which predict the impact of each term within each document (that is, they learn the value of $w_{t,d}$ in Eqn. 1). These models are all tuned to optimize the downstream retrieval task, but differ in their vocabulary structures, document expansion techniques, and query expansion strategies. For example, DeepImpact first expands the documents in the collection via DocT5Query, and then directly estimates a single impact for each token in each document. The model is trained by directly optimizing the sum of the query term impacts to maximize the score difference between relevant and non-relevant documents for a query. Similar in spirit, uniCOIL also performs weighting on the query terms, such that document ranking becomes a *weighted* sum over term impacts.

We focus on DeepImpact and uniCOIL as effective learned representations, but our methods are also applicable to other learned sparse techniques.

Indexing An *inverted index* stores one *postings list* $\mathcal{I}_t$ for each distinct term t in the given text collection, with each postings list containing a sequence of *postings* of the form $\mathcal{I}_{t,i} = \langle d_{t,i}, w_{t,i} \rangle$, where $d_{t,i}$ is the document number of the i th document containing t, and $w_{t,i}$ can be taken to be the corresponding *impact* score (see Eqn. 1). These lists are normally stored in increasing document order, and compressed using integer compression techniques, see Zobel and Moffat (2006) and Pibiri and Venturini (2021) for examples and further explanation.

[1] Note that TILDE can be used for both document expansion and for term importance estimation

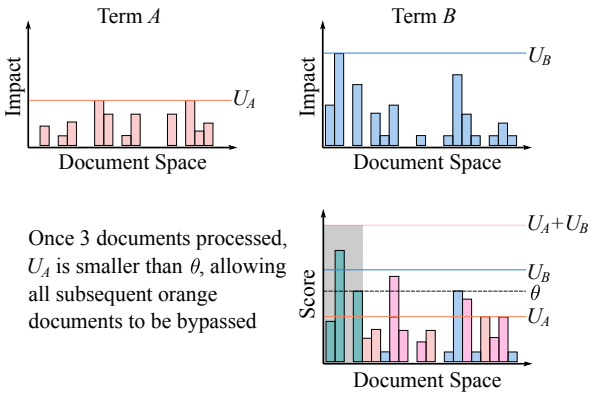

Figure 2: Dynamic pruning on a two-term query (term A, top left, and term B, top right) for top $k = 2$ retrieval. At the start of processing, the heap threshold θ is $-\infty$. After processing the three documents shown in green (bottom) we have $\theta > U_A$, and documents that contain only term A can be bypassed from now on.

Querying To retrieve the top-k highest scoring documents for a bag-of-terms query Q consisting of $q = |Q|$ query terms, a *document-at-a-time* processing regime is often used (Tonellotto et al., 2018). All q postings lists are open concurrently, each with a local cursor to step through the postings. Each document d that is encountered is fully scored at that time, and a min-heap maintained of the k highest-scoring documents encountered so far. Once all q postings lists are exhausted, the k documents in the heap can either be directly presented to the user or passed to another processing phase for a more sophisticated similarity computation that re-ranks that initial answer set.

MaxScore Algorithm 1 provides details of document-at-a-time querying processing, and also introduces the MaxScore dynamic pruning mechanism of Turtle and Flood (1995), structured in a manner that allows the development we propose in Section 3. In this description the U_t per-term upper bounds are a static attribute of the collection, established at indexing time, and $\mathcal{I}_{t,c[t]}$ is the "current" posting for term t, indicated by the cursor $c[t]$, with each index list $\mathcal{I}_t$ ordered by increasing document number, denoted $\mathcal{I}_{t,c[t]}.d$.

Figure 2 helps explain the pseudo-code. In the diagram a two-term query is being processed, con-

Algorithm 1 Standard MaxScore. Input is a set of q postings lists $\mathcal{I}_t$, with $\mathcal{I}_{t,i} = \langle d, w \rangle$ the docnum and impact score of the i th posting for the t th term; and a vector $U_t = \max_i\{\mathcal{I}_{t,i}.w\}$, the maximum impact for the t th term.

```
 1: active ← {0 ... q − 1}                    // active terms
 2: passive ← { }                             // passive terms
 3: sum_pass ← 0                              // sum of passive U_t's
 4: heap ← { }                                // heap of "best so far"
 5: c[t] ← 0 for 0 ≤ t < q                    // cursors
 6: θ ← −∞                                    // heap threshold
 7: while active postings remain do
 8:     // select next document, match all cursors
 9:     d ← min{I_{t,c[t]}.d | t ∈ active}
10:     for t ∈ passive do
11:         c[t] ← SeekGEQ(I_t, d)
12:     // score document
13:     score_d ← Σ{I_{t,c[t]}.w | I_{t,c[t]}.d = d}
14:     // advance cursors
15:     for t ∈ active do
16:         if I_{t,c[t]}.d = d then
17:             c[t] ← c[t] + 1
18:     // check against heap, update if needed
19:     if score_d > θ then
20:         heap ← heap ∪ {⟨d, score_d⟩}
21:         if |heap| > k then
22:             eject the least weight ⟨d, score_d⟩
23:                 heap item and update θ
24:     // try to expand passive set
25:     y ← argmax_t{|I_t| | t ∈ active}
26:     if sum_pass + U_y < θ then
27:         // toggle term y from active to passive
28:         active ← active − {y}
29:         passive ← passive ∪ {y}
30:         sum_pass ← sum_pass + U_y
```

sisting of postings for term A (top left) and term B (top right), and seeking the highest-scoring $k = 2$ documents. The index also records U_A and U_B, the maximum impact contributions of A and B across the collection. Once the first three documents in the union set of A and B have been scored, the k th largest-known document score – denoted by θ – is greater than U_A. After that point no further documents that contain term A alone need be considered; all candidates for scoring must contain B. In terms of Algorithm 1, term A is thus permanently moved from the *active* set to the *passive* set (steps 25–30) to record this change of status.

Algorithm 1 includes a number of subtleties. The ordering assumed at step 25 is constant, and computed once upon query commencement, rather at each loop iteration. As well, steps 25 to 30, shown as executing after every document has been scored, can be carried out infrequently without affecting the correctness of the top-k result set. For example, they might trigger only every 100 or 1000 iterations of the main while loop at step 7.

The key invariant in Algorithm 1 is that contributions from passive terms alone cannot yield a document score large enough to make it into the current top k answer set. That means that postings that appear only in passive postings lists can be bypassed, achieved at step 11 by function SeekGEQ$(\mathcal{I}_t, d)$, which advances the cursor $c[t]$ until a document number $\geq d$ is found in $\mathcal{I}_t$. Processing terminates when all of the postings associated with the active terms have been consumed. At that time, the required top-k documents are all in the heap.

WAND The WAND dynamic pruning mechanism (Broder et al., 2003) makes use of similar logic. But instead of labeling entire terms as being active or passive, it constantly rearranges the list cursors according to their next documents, in effect treating individual postings as being passive or active. That means that it can be more flexible in determining which postings combinations might yield scores greater than θ, and hence is more discerning in terms of which documents need scoring. Those gains must be offset against the additional cost of maintaining the list cursors in sorted order. Petri et al. (2013) give pseudo-code for WAND pruning.

Block-Max WAND Even more precise control over which documents need to be fully scored is achieved if localized upper bounds are used (denoted $U_{t,b}$) as well as whole-of-list U_t values. In the BlockMax-WAND (BMW) and Variable BlockMax-WAND (VBMW) approaches there are multiple $U_{t,b}$ values stored for each postings list, each of which provides a localized maximum impact bound for a *block* of contiguous postings (Ding and Suel, 2011; Mallia et al., 2017; Mallia and Porciani, 2019). During querying, global U_t values are used to select a candidate document, and the $U_{t,b}$ values are then used to refine the score estimate before checking whether the document should be scored or bypassed. Thus, storing these additional bounds allows more documents to be bypassed, albeit with increased processing required to handle the complex decision logic that arises, and the additional

space costs required to store localized bounds.

High-Impact List Segments and Priming Several authors have proposed explicitly or implicitly splitting postings lists into two (or more) parts, a high-impact segment $\mathcal{H}(t)$ and a low-impact segment $\mathcal{L}(t)$ to facilitate efficient processing; see, for example, Strohman and Croft (2007), Ding and Suel (2011), Daoud et al. (2016), Daoud et al. (2017), Kane and Tompa (2018) and Mackenzie et al. (2022a).

Another technique known as *priming* (Kane and Tompa, 2018; Petri et al., 2019) improves query performance by estimating lower bounds on the final heap threshold θ: if the value of the k th highest impact (or a value for the $k' > k$ th highest impact) for any of the q query terms is known, then the heap threshold θ can be initialized to the largest of those (up to) q values – it is certain that there will be k or more documents in the collection that score more highly than that value, even in the absence of any term overlaps. Moreover, if those k' high impact postings are maintained as a separate postings list, then the q high-impact list segments can be resolved against each other before any low-impact postings are considered, and might further lift the value of θ used when the q low-impact postings lists are employed to finalize the query.

3 Impact Decomposition

This section introduces the notion of *postings list splitting*, and shows how it can be combined with both MaxScore and with WAND variants. We then introduce a new technique, *postings clipping* that replicates the high-impact postings, rather than separating them from the low-impact postings. It has the benefit of allowing more precise score estimations, and hence faster pruned querying.

List Splitting Ding and Suel (2011), and later Daoud et al. (2016) and Kane and Tompa (2018), note that each postings list $\mathcal{I}_t$ can be split into two parts, denoted here as $\mathcal{H}(t)$ and $\mathcal{L}(t)$, with $\mathcal{H}(t)$ containing the postings with the highest impacts for t, and $\mathcal{L}(t)$ containing all the remaining ones. Since $\mathcal{H}(t)$ and $\mathcal{L}(t)$ are disjoint, query processing algorithms can treat them as independent terms.

The top part of Figure 3 illustrates list splitting. The complete set of postings for some term t (left) is reduced from (in the example) 21 postings to 17 posting to form $\mathcal{L}(t)$, with the other 4 postings assigned to $\mathcal{H}(t)$. Each of $\mathcal{L}(t)$ and $\mathcal{H}(t)$ then

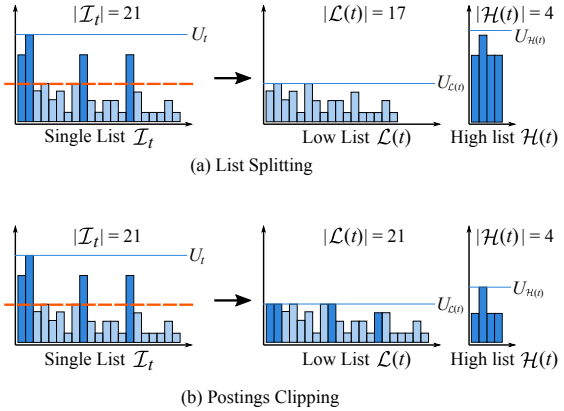

Figure 3: Two types of impact decomposition. *List splitting* involves moving high-impact postings into a separate postings list, $\mathcal{H}(t)$; whereas *postings clipping* involves trimming the impact scores in the low-impact list, and creating new postings in a separate list $\mathcal{H}(t)$ to account for the remainder.

receives its own upper bound (middle and right, $U_{\mathcal{L}(t)}$ and $U_{\mathcal{H}(t)}$ respectively), with $U_{\mathcal{H}(t)} > U_{\mathcal{L}(t)}$.

A number of splitting rules can be considered. For example, a fixed fraction of the original list might be taken; or the split could be based on local or global threshold scores. In this work, we take a fixed fraction, set at $1/64$ (based on preliminary experimentation) and respecting quantized impact levels, so that $|\mathcal{H}(t)|$ is maximized subject to $|\mathcal{H}(t)| \leq |\mathcal{I}_t|/64$, and also subject to the smallest impact in $\mathcal{H}(t)$ being greater than $U_{\mathcal{L}(t)}$. We also only apply splitting to lists with more than 256 postings, as short lists are always handled quickly.

Where the impact score distribution of the postings is skewed and has a long tail, splitting results in reduced variance inside each part. The maximum term importance U_t stored for any list $\mathcal{I}_t$ is intended to approximate the distribution of the impacts of the postings in that list; and hence storing two upper bounds, one for $\mathcal{H}(t)$ and one for $\mathcal{L}(t)$, allows a better approximation to the underling distribution. Note also that list splitting is performed at indexing time and results in only a modest increase in index size. At query time, each term is mapped to (one or) two postings lists, with at most twice as many cursors to maintain, but the same total number of postings to be processed.

MaxScore, WAND, and BMW Our first observation is simply that MaxScore should be implemented so that the static ordering over terms assumed at step 25 of Algorithm 1 is by decreasing list length, rather than by the more usual increas-

ing U_t, respecting the separation of these concepts that was noted above (that is, IDF is not obeyed by learned sparse models). The MaxScore pseudo-code presented earlier already shows this adaptation.

There are then a number of ways of proceeding when list splitting is considered. The simplest option is to ignore any knowledge of the list pairings, and allow a q-term query to be processed in the standard document-at-a-time manner over as many as $2q$ postings lists (Kane and Tompa, 2018). In terms of MaxScore, any combination of low- and high-impact lists might be in *passive*, with the remainder in *active*. However the use of the U_t limits to decide if a document that is in an active list should be scored remains valid – no document that might generate a similarity score greater than θ and thus should get scored will get bypassed. On the other hand, when the $\mathcal{L}(t)$ list for one of the terms is in *passive* (and because it is longer, it will enter earlier), only the postings in $\mathcal{H}(t)$ can now trigger a document scoring caused by term t, and hence there is a very real capacity for additional documents to be bypassed. Similar considerations arise with WAND and BMW: in all three processing modes the mere act of splitting the lists introduces the possibility of accelerated query processing, without risking any loss in terms of answer set correctness.

As an orthogonal enhancement, *priming* can be applied whenever any high-impact list contains k or more postings, $|\mathcal{H}(t)| \geq k$. If that holds, then

$$\theta_0 = \max\{U_{\mathcal{L}(t)} \mid t \in Q \wedge |\mathcal{H}(t)| \geq k\} \qquad (2)$$

can be used as a priming value for the heap bound, without risking the integrity of the top-k answers.

Next, if additional bookkeeping operations can be tolerated, it is also possible to compute what we denote as *smart bounds*. When the low-impact list for some term t first joins *passive*, the variable *sum_pass* is correctly increased by $U_{\mathcal{L}(t)}$. But if and when the partner term $\mathcal{H}(t)$ also joins *passive*, increasing *sum_pass* by $U_{\mathcal{H}(t)}$ is needlessly pessimistic, since no document can appear in both $\mathcal{L}(t)$ and $\mathcal{H}(t)$. Hence, the correct second increment associated with term t is by $U_{\mathcal{H}(t)} - U_{\mathcal{L}(t)}$. In the case of MaxScore, the corresponding smart bounds are easily computed, and are required only occasionally – when a postings list is moved from *active* to *passive*. However, for WAND and BMW the estimations must be modified much more frequently, and while smart bounds can certainly be computed, their benefit is less clear. One key part of the experimentation in Section 4 is to quantify the relationship between document scoring and bounds manipulation. Ding and Suel (2011) and Kane and Tompa (2018) also noted the idea of smart bounds estimation in their descriptions of list splitting, but they did not consider MaxScore-based processing.

Postings Clipping Our additional proposal – denoted *postings clipping* – is illustrated in the bottom half of Figure 3. Rather than partitioning the set of postings in $\mathcal{I}_t$ across $\mathcal{L}(t)$ and $\mathcal{H}(t)$, every posting remains in $\mathcal{L}(t)$, and we "clip" the high-impact postings by slicing them into two parts, and forming a *posting pair*. The base part remains in $\mathcal{L}(t)$ as a posting with an impact equal to $U_{\mathcal{L}(t)}$, the maximum score contribution permitted in $\mathcal{L}(t)$; and the second component of the pair becomes a new posting in $\mathcal{H}(t)$, to account for the "trimmed" part of the original impact value, and retain the same total.

This arrangement has the singular advantage of no longer requiring any smart bounds management, or equivalent run-time manipulation of score estimates. Smart bounds are needed in the list splitting approach of Kane and Tompa (2018) to adjust for the constraint that no document can appear in both $\mathcal{L}(t)$ and $\mathcal{H}(t)$, and hence that $U_{\mathcal{L}(t)} + U_{\mathcal{H}(t)}$ is an over-estimate (by an addend of $U_{\mathcal{L}(t)}$) of t's true upper bound U_t. But with postings clipping, $U_{\mathcal{H}(t)}$ is instead set to the maximum *residual* amount across all of t's postings, and hence we have $U_t = U_{\mathcal{L}(t)} + U_{\mathcal{H}(t)}$. In turn, that means that when queries are being processed the lists $\mathcal{L}(t)$ and $\mathcal{H}(t)$ can be treated as if they were derived from completely independent terms, with all interactions between them handled by the underlying processing logic, be that MaxScore, WAND, or BMW.

That is, while there are more total postings to be stored and processed, the change from list splitting with smart bounds to postings clipping substantially simplifies the query-time processing logic. Indeed, with the exception of priming – which can still be applied on the basis that is noted in Eqn. 2 – a MaxScore-based postings clipping implementation remains exactly as is shown by the logic provided in Algorithm 1. The result is that – as we demonstrate in Section 4 – quite dramatic reductions in query processing times for the learned sparse retrieval models can be achieved.

Figure 4 crystallizes the difference between list splitting and postings clipping. In the left pane (list splitting) the $U_{\mathcal{H}(t)}$ values rise as $U_{\mathcal{L}(t)}$ increases,

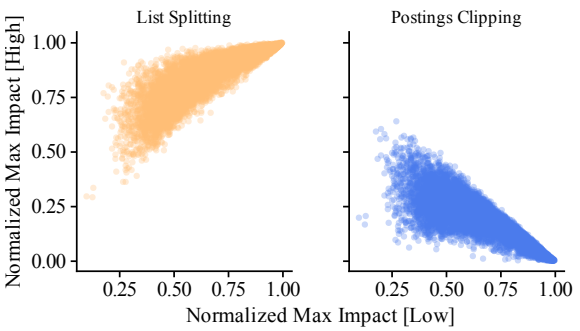

Figure 4: Bounding scores for list splitting (left) and postings clipping (right) using DeepImpact, with $U_{\mathcal{H}(t)}$ plotted as a function of $U_{\mathcal{L}(t)}$ for the unique terms occurring in the MSMARCO-v1 queries.

plotted over the set of MSMARCO-v1 query terms; whereas in the right pane (postings clipping) $U_{\mathcal{H}(t)}$ becomes increasingly constrained as $U_{\mathcal{L}(t)}$ grows. The difference affects the pruning bounds estimation, and while it can be partially ameliorated by smart bounds adjustments, the postings clipping mechanism is more precise.

4 Experiments

We now describe experiments that quantify the benefits arising from the postings clipping approach. Our experiments make use of both MSMARCO-v1 (8.8 million passages) and MSMARCO-v2 (138.4 million passages) collections, four representative ranking algorithms, and the PISA query processing system which was recently shown to outperform the commonly used Anserini system for document-at-a-time retrieval over learned sparse indexes (Mackenzie et al., 2021). Full details of the experimental setup are provided in Appendix A.

Index Size Table 2 reports the space consumption of each index/model combination for both the default index, and the index with postings clipping. Since clipping is applied only to postings with more than 256 elements, and even then only adds $1/64$ as many new postings, the space overhead compared to the default index is negligible. For instance, the largest overhead of 600 MiB to the $\approx$ 33 GiB index for the uniCOIL model on MSMARCO-v2 represents an increase of only 1.8%.

Query Speed Table 3 presents query processing times recorded for the MSMARCO-v1 collection and DeepImpact retrieval, with response latency measured as average milliseconds per query, and with

Collection	Model	Default	Clipping
MSMARCO-v1	BM25	0.8	0.8
	DocT5Query	1.2	1.2
	DeepImpact	1.6	1.6
	uniCOIL	2.1	2.2
MSMARCO-v2	BM25	20.3	20.6
	DocT5Query	27.7	27.9
	DeepImpact	24.7	25.0
	uniCOIL	32.7	33.3

Table 2: Index space requirement, in GiB, for default inverted indexes, and those with postings clipping. Results are shown for both collections and all four ranking models.

Method	$k = 10$	$k = 1000$
MaxScore baseline	8.1	18.8
+ length-based ordering	6.3	18.0
+ 1/64 list splitting	2.0	7.9
+ 1/64 priming	1.9	6.3
+ smart bounds	2.1	7.0
or postings clipping	1.6	5.9
WAND baseline	14.9	34.0
+ 1/64 list splitting	3.5	13.8
+ 1/64 priming	3.2	11.3
+ smart bounds	3.0	11.1
or postings clipping	2.7	10.8
VBMW baseline	4.2	12.2
+ 1/64 list splitting	3.0	11.7
+ 1/64 priming	2.9	9.8
+ smart bounds	2.8	10.0
or postings clipping	3.3	9.7

Table 3: Query processing times, all in average milliseconds per query, for the MSMARCO-v1 collection and DeepImpact retrieval model. Algorithmic enhancements are cumulative stepping down each of the three blocks in the table, except for *postings clipping*, which is an independent enhancement relative to *smart bounds*. Similar relativities were also observed in regard to median query times, and 90% and 99% tail latency query times.

the three blocks of values corresponding to three dynamic query pruning approaches. Within each block, we systematically add heuristics. First to be added in the MaxScore block is static term ordering based on length rather than on maximum

Method	BM25		DocT5Query		DeepImpact		uniCOIL	
	$k = 10$	1000	$k = 10$	1000	$k = 10$	1000	$k = 10$	1000
MaxScore baseline	11.0	38.7	8.8	28.2	828.0	1170.4	164.9	267.9
+ postings clipping	10.5	30.8	8.7	26.2	50.6	108.2	46.5	114.6
WAND baseline	15.8	61.3	17.4	60.3	1972.9	2592.7	213.4	510.2
+ postings clipping	10.4	36.1	11.5	41.0	166.2	449.0	54.4	169.6
VBMW baseline	11.3	37.5	12.0	44.7	488.2	719.2	128.2	219.6
+ postings clipping	13.4	39.4	13.7	45.4	167.8	293.3	164.7	235.4
× Speedup on best bl.	1.06	1.22	1.01	1.08	9.65	6.65	2.76	1.92

Table 4: Query processing times, all in average milliseconds per query, for the MSMARCO-v2 collection, four retrieval models, and three dynamic pruning approaches. The fastest time in each column is highlighted in blue, and the best of the three baseline approaches in each column is shown in black. The speedups in the last row are the ratio between the black and blue values in that column.

impact score; then the list splitting mechanism is added, with $1/64$ of the postings in each list longer than 256 extracted and placed in the high-impact list $\mathcal{H}(t)$; then the application (where possible) of Eqn. 2 to set an initial heap threshold; then the further addition of smart bounds. Finally, the last row in each block shows the combination of postings clipping, again with $1/64$ postings taken into $\mathcal{H}(t)$, in conjunction with priming (and length-based ordering for MaxScore). Both WAND and BMW apply the smart bounds adjustments during the pivoting step, and have no equivalent to the MaxScore static sorting step. The fastest query time in each of the six sections is highlighted in blue.

As can be seen, for DeepImpact retrieval, the fastest approach in five of the six table sections is achieved by MaxScore pruning with postings clipping. That combination takes less than half the time of standard MaxScore processing. The gains from posting clipping are less for WAND and VBMW, in part because both algorithms exhibit greater sensitivity to doubling the number of query terms.

Table 4 then applies all four retrieval models to the large MSMARCO-v2 collection. The six rows correspond to the first and last rows in each block in Table 3, with the first row in each pair showing "standard" retrieval, applying an inverted index and a dynamic pruning method; and then the second row compares that baseline against what can be achieved by postings clipping, priming using the same $U_{\mathcal{L}(t)}$ information that arises from the clipping, and (in the case of MaxScore), the matched static sorting. The best baseline in each column is shown in black, and the best overall time in each column in blue. Compared to a standard computation, postings clipping creates speedups of between two and nearly ten in connection with the two most expensive models, with MaxScore plus postings clipping being the best overall method for both $k = 10$ shallow retrieval and $k = 1000$ deep retrieval.

Retrieval Quality All of the enhancements investigated above result in rank-safe effectiveness. That is, the changes to the indexing structures and query processing regimes shown in Tables 3 and 4 do not degrade the quality of results compared to the unmodified algorithms, making the speedups even more attractive to search practitioners. Detailed effectiveness results for the four retrieval models are presented in Appendix B.

5 Conclusion and Future Work

To keep up with increasingly large volumes of data, search practitioners require sophisticated structures and processing algorithms, so that response times can remain plausible. In this paper, we have demonstrated the speed benefits that arise through the use of a new technique we call postings clipping. We have established new benchmarks for querying speed, with minimal costs overheads, for both shallow $k = 10$ and deep $k = 1000$ retrieval. Our techniques can also be embedded as part of a multi-phase processing stack, and are applicable to both normal term-based search and also to retrieval via enhanced learned sparse approaches.

Limitations

This paper modifies existing inverted index-based storage and query processing schemes to handle the different impact distributions produced by learned index representations. We have not explored how adjusting the training objective of models such as DeepImpact could produce better impact distributions directly targeting efficient query processing algorithms that exploit list upper bounds. Such approaches, if they were fruitful, would potentially mitigate the need for the techniques proposed in this paper.

Table 1 indicates that some of the latency problems arise from learned representations distinguishing between different semantic meaning of words, correctly assigning high importance to terms based on context. We have not explored incorporating these pre-index construction insights into the proposed splitting and subsequent query processing schemes, and instead have relied solely on numeric impact values. It is possible that making splitting decisions in conjunction with the learning process might lead to even better outcomes.

Resource constraints have meant that we have restricted our investigation to the DeepImpact- and uniCOIL-based learned sparse representations. While we believe our techniques will provide similar benefits to other learned sparse retrieval techniques such as TILDE (Zhuang and Zuccon, 2021a) and SPLADE (Formal et al., 2021b,a), we have not explored those approaches as part of this work.

Finally, our investigation explored how split lists can be used to prime the initial heap threshold θ. Recent work has shown that more accurate predictions can further accelerate querying on traditional ranking models (Petri et al., 2019; Mallia et al., 2020). To determine whether these approaches translate to learned sparse models, we applied idealized "oracle" thresholds to our experimental framework (see Appendix B for details). While the results are promising (up to a $2.1\times$ speedup over the best results in Table 4), it remains unclear if existing threshold estimators can be applied to learned sparse models, or if more sophisticated estimators are necessary.

Ethics Statement

The authors have no external conflicts of interest to declare, and have not been required to seek any ethics clearances in order to undertake this work.

If widely adopted, the techniques we propose will lead to fewer computational resources being required for querying tasks carried out via learned sparse models, and hence reduced electrical consumption and greenhouse emissions.

Acknowledgements

This work was supported by the Australian Research Council (project DP200103136). We thank the referees for their helpful suggestions.

References

V. N. Anh, O. de Kretser, and A. Moffat. 2001. Vector-space ranking with effective early termination. In *Proc. SIGIR*, pages 35–42.

N. Arabzadeh, A. Vtyurina, X. Yan, and C. L. A. Clarke. 2021. Shallow pooling for sparse labels. *arXiv:2109.00062v2*.

P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen, M. Rosenberg, X. Song, A. Stoica, S. Tiwary, and T. Wang. 2018. MS MARCO: A human generated MAchine Reading COmprehension dataset. *arXiv:1611.09268v3*.

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*, pages 426–434.

N. Cohen, A. Portnoy, B. Fetahu, and A. Ingber. 2022. SDR: efficient neural re-ranking using succinct document representation. In *Proc. ACL*, pages 6624–6637.

Z. Dai and J. Callan. 2019. Context-aware sentence/passage term importance estimation for first stage retrieval. *arXiv:1910.10687*.

C. M. Daoud, E. S. de Moura, A. L. Carvalho, A. S. da Silva, D. Fernandes, and C. Rossi. 2016. Fast top-k preserving query processing using two-tier indexes. *Inf. Proc. & Man.*, 52(5):855–872.

C. M. Daoud, E. S. de Moura, D. Fernandes, A. S. da Silva, C. Rossi, and A. Carvalho. 2017. Waves: A fast multi-tier top-k query processing algorithm. *Inf. Retr.*, 20(3):292–316.

L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita. 2016. Compressing graphs and indexes with recursive graph bisection. In *Proc. KDD*, pages 1535–1544.

S. Ding and T. Suel. 2011. Faster top-k document retrieval using block-max indexes. In *Proc. SIGIR*, pages 993–1002.

T. Formal, C. Lassance, B. Piwowarski, and S. Clinchant. 2021a. SPLADE v2: Sparse lexical and expansion model for information retrieval. *arXiv:2109.10086*.

T. Formal, B. Piwowarski, and S. Clinchant. 2021b. SPLADE: Sparse lexical and expansion model for first stage ranking. In *Proc. SIGIR*, pages 2288–2292.

L. Gao, Z. Dai, and J. Callan. 2021. COIL: Revisit exact lexical match in information retrieval with contextualized inverted list. In *Proc. NAACL*, pages 3030–3042.

G. Izacard, F. Petroni, L. Hosseini, N. De Cao, S. Riedel, and E. Grave. 2020. A memory efficient baseline for open domain question answering. *arXiv:2012.15156*.

A. Kane and F. W. Tompa. 2018. Split-lists and initial thresholds for WAND-based search. In *Proc. SIGIR*, pages 877–880.

V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proc. EMNLP*, pages 6769–6781.

O. Khattab and M. Zaharia. 2020. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proc. SIGIR*, pages 39–48.

D. Lemire and L. Boytsov. 2015. Decoding billions of integers per second through vectorization. *Soft. Prac. & Exp.*, 41(1):1–29.

J. Lin and X. Ma. 2021. A few brief notes on DeepImpact, COIL, and a conceptual framework for information retrieval techniques. *arXiv:2106.14807*.

J. Lin, X. Ma, S.-C. Lin, J.-H. Yang, R. Pradeep, and R. Nogueira. 2021. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proc. SIGIR*, pages 2356–2362.

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*, pages 2149–2152.

X. Ma, R. Pradeep, R. Nogueira, and J. Lin. 2022. Document expansions and learned sparse lexical representations for MSMARCO V1 and V2. In *Proc. SIGIR*.

S. MacAvaney, A. Yates, A. Cohan, and N. Goharian. 2019. CEDR: Contextualized embeddings for document ranking. In *Proc. SIGIR*, pages 1101–1104.

J. Mackenzie, M. Petri, and A. Moffat. 2022a. Anytime ranking on document-ordered indexes. *ACM Trans. Inf. Sys.*, 40(1):13:1–13:32.

J. Mackenzie, M. Petri, and A. Moffat. 2022b. Tradeoff options for bipartite graph partitioning. *IEEE Trans. Know. & Data Eng.* To appear.

J. Mackenzie, A. Trotman, and J. Lin. 2021. Wacky weights in learned sparse representations and the revenge of score-at-a-time query evaluation. *arXiv:2110.11540*.

A. Mallia, O. Khattab, N. Tonellotto, and T. Suel. 2021. Learning passage impacts for inverted indexes. In *Proc. SIGIR*, pages 1723–1727.

A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini. 2017. Faster BlockMax WAND with variable-sized blocks. In *Proc. SIGIR*, pages 625–634.

A. Mallia and E. Porciani. 2019. Faster BlockMax WAND with longer skipping. In *Proc. ECIR*, pages 771–778.

A. Mallia, M. Siedlaczek, J. Mackenzie, and T. Suel. 2019. PISA: Performant indexes and search for academia. In *Proc. OSIRRC at SIGIR 2019*, pages 50–56.

A. Mallia, M. Siedlaczek, M. Sun, and T. Suel. 2020. A comparison of top-k threshold estimation techniques for disjunctive query processing. In *Proc. CIKM*, pages 2141–2144.

R. Nogueira and J. Lin. 2019. From doc2query to docTTTTTquery. Unpublished report, David R. Cheriton School of Computer Science, University of Waterloo, Canada.

M. Petri, J. S. Culpepper, and A. Moffat. 2013. Exploring the magic of WAND. In *Proc. Aust. Doc. Comp. Symp.*, pages 58–65.

M. Petri, A. Moffat, J. Mackenzie, J. S. Culpepper, and D. Beck. 2019. Accelerated query processing via similarity score prediction. In *Proc. SIGIR*, pages 485–494.

G. E. Pibiri and R. Venturini. 2021. Techniques for inverted index compression. *ACM Comp. Surv.*, 53(6):125.1–125.36.

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

Y. Qu, Y. Ding, J. Liu, K. Liu, R. Ren, W. X. Zhao, D. Dong, H. Wu, and H. Wang. 2021. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In *Proc. NAACL*, pages 5835–5847.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

K. M. Risvik, T. Chilimbi, H. Tan, K. Kalyanaraman, and C. Anderson. 2013. Maguro, a system for indexing and searching over very large text collections. In *Proc. WSDM*, pages 727–736.

S. E. Robertson and H. Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Found. Trnd. Inf. Retr.*, 3:333–389.

T. Strohman and W. B. Croft. 2007. Efficient document retrieval in main memory. In *Proc. SIGIR*, pages 175–182.

N. Tonellotto, C. Macdonald, and I. Ounis. 2018. Efficient query processing for scalable web search. *Found. Trnd. Inf. Retr.*, 12(4-5):319–500.

H. R. Turtle and J. Flood. 1995. Query evaluation: Strategies and optimizations. *Inf. Proc. & Man.*, 31(6):831–850.

I. Yamada, A. Asai, and H. Hajishirzi. 2021. Efficient passage retrieval with hashing for open-domain question answering. In *Proc. ACL*, pages 979–986.

P. Yang, H. Fang, and J. Lin. 2018. Anserini: Reproducible ranking baselines using lucene. *J. Data Inf. Qual.*, 10(4):1–20.

J. Zhan, J. Mao, Y. Liu, J. Guo, M. Zhang, and S. Ma. 2021. Jointly optimizing query encoder and product quantization to improve retrieval performance. In *Proc. CIKM*, pages 2487–2496.

J. Zhan, J. Mao, Y. Liu, J. Guo, M. Zhang, and S. Ma. 2022. Learning discrete representations via constrained clustering for effective and efficient dense retrieval. In *Proc. WSDM*, pages 1328–1336.

L. Zhao. 2012. Modeling and solving term mismatch for full-text retrieval. *SIGIR Forum*, 46(2):117–118.

S. Zhuang and G. Zuccon. 2021a. Fast passage re-ranking with contextualized exact term matching and efficient passage expansion. *arXiv:2108.08513*.

S. Zhuang and G. Zuccon. 2021b. TILDE: Term independent likelihood moDEl for passage re-ranking. In *Proc. SIGIR*, pages 1483–1492.

J. Zobel and A. Moffat. 2006. Inverted files for text search engines. *ACM Comp. Surv.*, 38(2):6:1–6:56.

A Experimental Setup

Hardware and Latency Measurement All of our experiments are performed entirely in memory on a Linux server with two Intel Xeon Gold 6144 CPUs (3.5GHz) and 512 GiB of memory. Latency measurements are taken as the average of three independent runs, where each run utilizes 16 processing cores to process the query stream in parallel in a one-thread-per-query manner.

Collections and Metrics The MSMARCO-v1 passage collection contains around 8.8 million passages and a total of 6,980 dev queries (Bajaj et al., 2018). We measured effectiveness using the official RR@10 metric (see Arabzadeh et al. (2021) for additional discussion of this). The much larger MSMARCO-v2 collection contains around 138.4 million passages and 8,184 queries (after combining both the dev and dev2 query sets), with effectiveness measured using the official RR@100 metric. In this second collection, the short text passages are augmented with ancillary fields, specifically URLs, titles, and headings, distributed as additional resources (Ma et al., 2022). We also measured effectiveness using the deeper Recall@1000 metric, to validate the quality of our generated runs.

Indexing and Query Processing Indexes were constructed using Anserini (Yang et al., 2018) and converted into PISA indexes (Mallia et al., 2019) via the common index file format (Lin et al., 2020), allowing pre-built indexes to be used for improved reproducibility (Ma et al., 2022). Before time or space efficiency was measured, the indexes were also reordered via the recursive graph bisection algorithm to reduce their space consumption and improve locality of access (Dhulipala et al., 2016; Mackenzie et al., 2022b). All remaining experimentation was conducted with the PISA engine. Indexes were compressed with the SIMD-BP128 bitpacking codec (Lemire and Boytsov, 2015). The VBMW algorithm used an average block size of 40 ± 0.5 following the results of Mallia et al. (2017).

Ranking Models Our experiments made use of two traditional ranking models:

- BM25 (Robertson and Zaragoza, 2009) is the traditional BM25 lexical ranking model. The exact formulation of the BM25 version we employed is outlined in the PISA overview (Mallia et al., 2019). We applied BM25 to MSMARCO-v1 with $k_1 = 0.82$ and $b = 0.68$ (Lin et al., 2021), and used the default parameters $k_1 = 0.9$ and $b = 0.4$ for MSMARCO-v2.

- DocT5Query (Nogueira and Lin, 2019) applies the BM25 ranking model over an expanded version of the document corpus using the T5 sequence-to-sequence model (Raffel et al., 2020). The same BM25 formulation and parameters are used as above; DocT5Query can be thought of as a "neurally augmented" corpus with a traditional ranking model.

To those we added two representative learned sparse retrieval models:

- DeepImpact (Mallia et al., 2021) employs the DocT5Query model to expand the corpus, and then learns a per-term impact score for each passage.

Method	BM25		DocT5Query		DeepImpact		uniCOIL	
	$k = 10$	1000	$k = 10$	1000	$k = 10$	1000	$k = 10$	1000
MaxScore baseline	1.7	5.5	0.8	3.7	8.1	18.8	13.8	27.5
+ postings clipping	1.5	5.0	0.8	3.4	1.6	5.9	5.8	15.7
WAND baseline	2.3	7.4	1.3	7.0	14.9	34.0	23.3	56.5
+ postings clipping	1.7	5.6	1.0	5.5	2.7	10.8	8.1	27.5
VBMW baseline	2.0	7.3	1.3	7.0	4.2	12.2	11.5	28.5
+ postings clipping	2.4	7.4	1.3	6.7	3.3	9.7	13.6	27.2
× Speedup on best bl.	1.13	1.10	1.00	1.09	2.63	2.10	1.98	1.75

Table 5: Query processing times, all in average milliseconds per query, for the MSMARCO-v1 collection, four retrieval models, and three dynamic pruning approaches, with the same structure and interpretation as Table 4. The fastest time in each column is highlighted in blue, and the best of the three baseline approaches in each column is shown in black. The speedups in the last row are the ratio between the black and blue values in that column.

- uniCOIL (Lin and Ma, 2021) employs TILDE (Zhuang and Zuccon, 2021b,a) document expansion, and learns per-term weights according to a simplified (1-dimensional) COIL model (Gao et al., 2021). Unlike DeepImpact, uniCOIL also applies *term weighting* at query-time, transforming bag-of-words queries into weighted queries (and resulting in weighted bag-of-words ranking; see Section 2).

The learned sparse models work with pre-quantized scores, and so we also pre-computed and quantized the BM25 and DocT5Query indexes into integer impact scores in the range $[0, 255]$ using uniform quantization (Anh et al., 2001). All experimentation then involved computing document scores as (weighted) sums of impacts. Note also that DocT5Query, DeepImpact, and uniCOIL were all fine-tuned on the MSMARCO-v1 training data; those same models are then applied in a zero-shot manner to MSMARCO-v2.

Setting Hyperparameters In order to decide the split points for use in our experimentation, we ran a preliminary experiment where we tried splits of $1/p$ for $p \in \{8, 16, 32, 64, 128, 256\}$ using the DeepImpact ranker and the MSMARCO-v1 dev queries. While all split values resulted in large efficiency improvement, $p = 64$ was the best choice. We then fixed $p = 64$ for all remaining collections and experiments, and did not further tune this value.

Reproducibility Our list splitting, clipping, and priming contributions were all implemented inside the C++ PISA framework; this modified version of PISA is available at `https://github.com/jmmackenzie/postings-clipping`. Scripts for downloading and pre-processing data, computing split points, building indexes, and running the experiments are also available in that repository to facilitate reproducibility. Our experimentation is all based on widely-available datasets (Bajaj et al., 2018; Ma et al., 2022).

B Additional Measurements and Results

Query Speed Table 5 provides a set of timings for the MSMARCO-v1 collection, in the same format as was employed in Table 4. A similar pattern of behavior arises, demonstrating that our findings apply to both the smaller MSMARCO-v1 and the large MSMARCO-v2 collections. Unsurprisingly, the observed speedup ratios for MSMARCO-v1 are typically less than those measured for MSMARCO-v2.

Effectiveness Table 6 presents effectiveness scores of the four retrieval models, as measured within our experimental framework. While the emphasis in this paper is on efficiency rather than effectiveness, it is interesting to note the strong improvements that the neural augmented DocT5Query method and the two learned sparse methods obtain relative to the standard BM25 approach. Those substantial gains arise because of a combination of document-level term expansion, and the non-linear context-based relationships that are uncovered be-

Model	RR@d	Rec.@1000
MSMARCO-v1 ($d = 10$)		
BM25	0.187	0.858
DocT5Query	0.267	0.945
DeepImpact	0.327	0.948
uniCOIL	0.350	0.965
MSMARCO-v2 ($d = 100$)		
BM25	0.086	0.696
DocT5Query	0.110	0.762
DeepImpact	0.132	0.736
uniCOIL	0.153	0.772

Table 6: Effectiveness of the different models on both collections, using the official metrics associated with each, and runs of length $k = 1000$.

tween term frequency and term impact.

Our implementations achieve similar effectiveness scores to those previously reported for these three recent techniques – see, for example, Mackenzie et al. (2021) and Ma et al. (2022).

Idealized Initial Thresholds Threshold estimation is a technique that improves the efficiency of query processing (Mallia et al., 2020; Petri et al., 2019). Like *priming*, it enables better skipping over unimportant documents during index traversal by providing an initial minimum threshold score a document needs to obtain to be considered during ranking; unlike priming, however, various unsafe alternatives can be used for predicting initial thresholds. Table 7 demonstrates the potential speedup if the initial heap threshold for each query could (in an omniscient manner) be set at exactly the final score of the k th most similar document; that is, if the priming process could be clairvoyant. The substantial difference in execution times achievable, up to $2.1\times$ relative to the clipping runs shown in Table 4, indicates that more accurate initial threshold prediction mechanisms are a promising direction for further accelerating learned sparse retrieval mechanisms.

Method	$k = 10$	1000
MaxScore baseline	828.0	1170.4
+ postings splitting	50.6	108.2
+ oracle thresholds	35.4	66.8
WAND baseline	1972.9	2592.7
+ postings splitting	166.2	449.0
+ oracle thresholds	77.7	244.4
VBMW baseline	488.2	719.2
+ postings splitting	167.8	293.3
+ oracle thresholds	149.5	188.5

Table 7: Demonstrating the potential of accurate threshold estimation on the MSMARCO-v2 collection and the DeepImpact model, assuming clairvoyant pre-knowledge for each query. If the final heap threshold could be predicted accurately, further speedups are possible.