Efficient Sparse Retrieval with Lightweight Superblock Pruning

Parker Carlson
University of California, Santa Barbara
Santa Barbara, CA, USA
parker_carlson@ucsb.edu

Wentai Xie
University of California, Santa Barbara
Santa Barbara, CA, USA
wentaixie@ucsb.edu

Rohil Shah
University of California, Santa Barbara
Santa Barbara, CA, USA
rohildshah@ucsb.edu

Tao Yang
University of California, Santa Barbara
Santa Barbara, CA, USA
tyang@cs.ucsb.edu

Abstract

Learned sparse retrieval (LSR) is a popular method for first-stage retrieval because it combines the semantic matching of language models with efficient CPU-friendly algorithms. Previous work aggregates blocks into "superblocks" to quickly skip the visitation of blocks during query processing by using an advanced pruning heuristic. This paper proposes a simple and effective superblock pruning scheme that reduces the overhead of superblock score computation while preserving competitive relevance. It combines this scheme with a compact index structure and a robust zero-shot configuration that is effective across LSR models and multiple datasets. This paper provides an analytical justification and evaluation on the MS MARCO and BEIR datasets, demonstrating that the proposed scheme can be a strong alternative for efficient sparse retrieval.

CCS Concepts

• **Information systems** → **Information retrieval query processing**; **Search engine indexing**; *Search index compression*; **Retrieval efficiency**.

Keywords

Learned sparse retrieval, dynamic pruning, inverted index

ACM Reference Format:
Parker Carlson, Wentai Xie, Rohil Shah, and Tao Yang. 2026. Efficient Sparse Retrieval with Lightweight Superblock Pruning . In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '26), July 20–24, 2026, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3805712.3809739

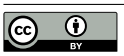
1 Introduction

Fast and effective retrieval is a critical component of large-scale search systems. It is also important for retrieval-augmented generation which is gaining in popularity [14, 24]. Learned sparse retrieval (LSR) [16, 21, 26, 32, 52, 61] can be used in the first-stage of the above systems as it can run fast on a low-cost CPU-only platform while delivering competitive relevance compared to dense retrieval which typically requires an expensive GPU server [50, 57, 58]. LSR can also be combined with dense retrieval to further boost relevance on a CPU platform [19, 25, 60].

Given a learned or traditional sparse index, a major optimization for fast online inference is to use dynamic index pruning that skips a significant portion of the index during query processing. The state-of-the-art LSR retrieval algorithms ASC, BMP, Seismic and SP all use block- or cluster-based pruning strategies [3, 7, 41, 45]. Block-based skipping [12, 38, 41] divides groups of documents into blocks to estimate the block-wise maximum rank score among its documents and prunes such a block when a block-wise maximum score is below a threshold. Similar pruning techniques are used in cluster-based skipping [31, 45]. Recently SP extends the work of BMP [41] and ASC [45] with a superblock-level pruning scheme that aggregates both superblock-level maximum and average rank scores of documents [7]. A superblock contains a set of document blocks, and superblock pruning in SP increases index-skipping opportunities while maintaining competitive relevance when threshold overestimation is used [9, 28, 54].

One weakness when using overestimation is that there exist erroneous cases where almost all documents are removed without producing enough top-k results. The pruning safeness protection added by average score computation in SP is not sufficient to prevent the pruning of almost all superblocks in such cases. Another weakness is that there is considerable overhead in maintaining and computing the average rank scores in SP. In addition, SP did not address index compression, and thus the impact of accessing compact data structures for two-level pruning is not well understood. To address these weaknesses, we propose a lightweight superblock pruning (LSP) scheme which guarantees visitation of a fixed number of superblocks while removing the average-bound-based safeguard to reduce overhead when using small block and superblock sizes. We also propose compact data structures that significantly reduce memory overhead with negligible effects to retrieval time.

We provide an analysis to justify the advantages of top superblock inclusion while obtaining competitive relevance. This also allows us to recommend simple parameter configurations of LSP without a grid search for a dataset size similar to or smaller than the training dataset used for this analysis. Parameter configuration without grid search is highly desirable for zero-shot retrieval. We have considered three versions of superblock pruning from a lightweight guarantee to more complex pruning schemes, and we evaluated these strategies with the MS MARCO and BEIR datasets to assess when the proposed methods can outperform baselines.



SIGIR '26, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-2599-9/2026/07
https://doi.org/10.1145/3805712.3809739

Our evaluation shows that the simple LSP version is 1.8-17x faster than SP, and 1.8-12x faster than BMP. It is 2x faster than SeismicWave for zero-shot retrieval with BEIR at various depths k. It is up-to 4.8x faster than SeismicWave for MS MARCO with $k = 1000$ while 30% slower for $k = 10$. When a grid search of all parameters is possible, it is up-to 78% faster than SeismicWave for $k = 1000$, while it is up-to 2x slower than SeismicWave for $k = 10$.

2 Background and Related Work

Problem definition. Given query q for searching a collection of D text documents: $\{d_i\}_{i=1}^{D}$, sparse retrieval computes the dot product similarity of a given query q and a document d as: $RankScore(d) = L(q) \cdot L(d_i)$ where $L(.)$ is a sparse vector of weighted term tokens for a document or a query. Term weights typically use the BM25 formula [49], or are learned through a BERT or LLM based model [16, 26, 33, 52, 61]. $L(.)$ may contain original and/or expanded terms. Online inference with sparse retrieval historically uses an inverted index to assist fast index traversal.

Given query Q, Candidates are ranked using the following formula simple formula, where the rank score of each document d is defined as: $RankScore(d) = \sum_{t \in Q} q_t \cdot w_{t,d}$, where the weight of term t in document d is $w_{t,d}$ and corresponding query term weight is q_t.

The block-based index approach [3, 31, 41, 46], where a document collection is divided into a sequence of N blocks $\{B_1, \cdots, B_N\}$. Like BMP, we assume that each block uniformly contains b documents. Like previous work [31, 41], blocks are visited in decreasing order of *BoundSum* values.

$$BoundSum(B_i) = \sum_{t \in Q} \max_{d \in B_i} q_t \cdot w_{t,d}. \quad (1)$$

The visitation to block B_i can be safely pruned if $BoundSum(B_i) \leq \theta$, where θ is the current top-k threshold. If this block is not pruned, then document-level index traversal can be conducted within each block following a standard retrieval algorithm.

Partial index traversal with threshold-driven pruning. Given a sparse index, a popular efficiency optimization to skip portions of the index during query processing is dynamic pruning. This strategy computes the upper bound rank score of a candidate document d, referred to as $Bound(d)$, satisfying $RankScore(d) \leq Bound(d)$. If $Bound(d) \leq \theta$, where θ is the rank score threshold to be in the top-k list, this document can be safely skipped. Examples of dynamic pruning methods include WAND [2], BMW [13], VBMW [35], and MaxScore [56]. A retrieval method is called *rank-safe* if it guarantees that the top-k documents returned are the k highest scoring documents. All of the above algorithms are rank-safe. Threshold overestimation is a "rank-unsafe" skipping strategy that deliberately over-estimates the current top-k threshold by a factor [9, 28, 54].

Block or cluster based sparse index pruning. The state-of-the-art fast sparse retrieval algorithms for learned sparse retrieval ASC, BMP, Seismic, and SP [3, 7, 41, 45] all use block- or cluster-based pruning strategies. There is a large body of studies on cluster-based document retrieval in traditional IR (e.g. [18, 27, 42]) for re-ranking an initially retrieved list by creating clusters of similar documents in this list. Then the clusters are ranked, and the cluster ranking may be further transformed to a document ranking. Block-based index skipping [12, 38, 41] divides documents into blocks to estimate the block-wise maximum rank score for pruning. Conceptually, cluster-based skipping follow a similar strategy [6, 17, 31]. BMP [41] optimizes execution with score quantization, SIMD *BoundSum* computation, partial block sorting, and query term pruning. A major optimization in Seismic [3] is aggressive static inverted index pruning while fully scoring documents with an unpruned forward index. Like BMP [41], Seismic also incorporates threshold overestimation, query term pruning, and dynamic cluster (block) maximum pruning. SeismicWave [4] extends Seismic to improve the relevance of approximate search by sorting blocks by their visitation heuristic and adding a post processing stage to select relevant documents by using an extra data structure with a document-level proximity graph. ASC [45] and SP [7] extend the above pruning studies and compute both maximum and average rank scores to assure probabilistic rank-safeness for competitive relevance. In terms of clustering or block assignment methods, the above studies employ a similarity-based method to group similar documents together, e.g. a bipartite partitioning algorithm [11, 30] or k-means [8].

Index compression. Earlier IR studies have proposed various index compression techniques, e.g. [1, 23, 43, 44, 55, 59, 62], and a study by Mallia et al. [37] comparing a number of these compression techniques found that SIMD-BP128 [23] provides a good trade-off between space reduction and decoding speed. These techniques are effective for traditional retrieval algorithms with inverted indices. To accommodate recently-developed cluster- or block-based algorithms, new investigations are desired to ensure a proper trade-off of space reduction and query processing latency when incorporating with two-level pruning. BMP [41] by Mallia et al. has devised a compact scheme that works well with its block filtering strategy, while Seismic, ASC, and SP do not address index compression. This paper assesses the use of BMP compression for LSP and designs an improved scheme that works effectively with the proposed method, based on SIMD-BP and an integer list packing method previously proposed for GPU encoding [39].

Other orthogonal efficiency techniques. There are orthogonal techniques to accelerate learned sparse retrieval through BM25-guided search, model training, and index re-organization, e.g. [20, 29, 34, 48]. Static index pruning for SPLADE is studied in [22, 47] and is employed extensively by Seismic [3]. Our work is complementary to these studies, and could benefit from these techniques.

3 Block/Superblock Pruning and Design Consideration

First, we describe block-based filtering in BMP and superblock pruning in SP. We assume that a document collection is divided into N blocks. Following [7, 41], these blocks are formed based on similarity, and each block uniformly contains b documents. During online retrieval, BMP visits these blocks in a decreasing order of their *BoundSum* values, where $BoundSum(B)$ is a maximum rank score bound of documents within block B. Let θ be the estimated [40] or currently known top-k rank score of documents scored so far by this retriever for a query. The visitation to block B can be pruned if $BoundSum(B) \leq \theta$. The block-level rank score bound is estimated using maximum term weights within each block. This bound can be very loose, especially when a cluster contains documents with

disparate rank scores. Because of loose bound estimation, blocks composed of low-scoring documents often do not satisfy the condition $BoundSum(B) \leq \theta$ and are not pruned. Threshold overestimation [9, 28, 54] with μ-approximation addresses that by changing the above pruning condition as $BoundSum(C_i) \leq \frac{\theta}{\mu}$ where $0 < \mu < 1$. However, this can incorrectly prune desired relevant documents when their cluster bound estimation is tight.

To identify low-scoring blocks earlier and address the above weakness, SP furthers uniformly aggregate a sequence of c consecutive document blocks into one superblock, and online inference traverses the index in a top-down manner. Namely, SP visits all superblocks first and then visits the blocks of unpruned superblocks. Finally, it visits and scores documents within unpruned blocks.

For each superblock X, SP computes the rank score bound ($SBMax(X)$) and average score bound ($\overline{SBMax}(X)$) of documents within superblock X as follows

$$SBMax(X) = \sum_{t \in Q} q_t \cdot W_{X,t};\ \overline{SBMax}(X) = \sum_{t \in Q} q_t \cdot \overline{W}_{X,t} \quad (2)$$

where $W_{X,t}$ and $\overline{W}_{X,t}$ are the maximum and average weight of term t among documents of superblock X, respectively. Then SP adds an extra pruning safeness guard by using the average score bound. Any superblock X is pruned when its maximum and average superblock bounds satisfy

$$SBMax(X) \leq \frac{\theta}{\mu} \quad (3)$$

and

$$\overline{SBMax}(X) \leq \frac{\theta}{\eta} \quad (4)$$

where parameters μ and η satisfy $0 < \mu \leq \eta \leq 1$. This two-level pruning of SP is similar to the (μ, η)-approximation in ASC [45], except SP incorporates additional efficiency optimizations. The use of Inequality (3) retains μ overestimation safeness guarantees, while Inequality (4) ensures probabilistic safeness.

Design Considerations. Our considerations to accelerate SP are:

- **Visit fewer superblocks.** To accelerate retrieval in a block- and superblock-based index structure, we seek more opportunities to prune superblocks. As superblock maximum bounds are generally loose, selecting a relatively smaller μ value exposes opportunities for filtering them out using Inequality (3), guarded by Inequality (4). Define the bound tightness of superblock X as $max_{d \in X} RankScore(d)$ divided by $SBMax(X)$. Figure 1 depicts the distribution of the average superblock bound tightness on the MS MARCO Dev queries with block size $b = 8$ and with $c = 16$ blocks per superblock. The tightness value varies from about 0.2 to 1. To prune most superblocks, μ would need to be set as low as possible, near 0.2.
- **Erroneous pruning with μ-overestimation**. There are cases when μ is small where Inequality (3) is true for any superblock X in a dataset while Inequality (4) is also true for most superblocks even when $\eta = 1$. Then, Inequalities (3) and (4) erroneously prune enough superblocks that there are not enough documents left to contribute k results. Figure 2 shows how often erroneous pruning occurs in processing MS MARCO Dev queries with SPLADE by SP with retrieval depth $k = 1000$ when $\mu \leq 0.5$ and $\eta = 1$. The y-axis shows the percentage of queries failed to produce any result in orange (big dashes), the percentage of queries producing results less than k in green (small dashes), and the percentage of queries producing exactly top k results in blue (solid), when μ varies from 0.1 to 0.5 on the x axis. When $\mu \geq 0.5$, the above erroneous pruning does not occur for these queries with SPLADE, but it happens with E-SPLADE as shown in Table 2 of Section 5.
- **Significant overhead to maintain and compute average rank score bounds**. The additional storage and computation required to find $\overline{SBMax}(X)$ for each superblock sometimes outweighs the benefit of additional safeness when there is a large number of superblocks.
- **Index compression.** The current SP implementation stores block- and superblock-bounds uncompressed. Because computing these bounds is latency-sensitive, it is desirable to design a proper compact data structure that works well with superblock pruning.

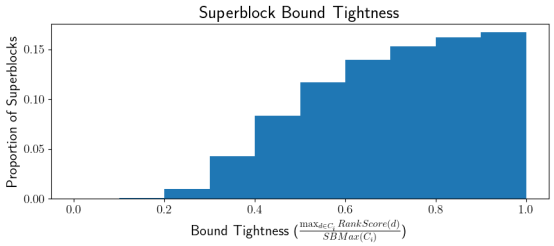


Figure 1: Distribution of bound tightness values of superblocks on MS MARCO Dev set with b=8, c=16

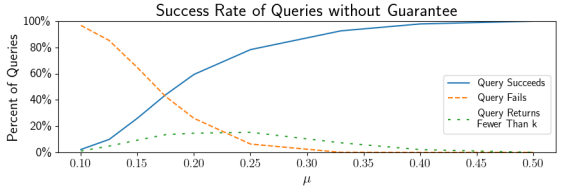


Figure 2: Successful, partially successful, and failed top-1000 query processing for MS MARCO Dev set when μ varies

Our design pursues the following strategies:

- To avoid the erroneous pruning, our idea is to impose an extra condition guaranteeing top-γ superblocks with safe pruning during index traversal. More details are in Section 4.1 with a design justification in Section 4.2.
- We view the average maximum bound condition as an extra guard to work in tandem with the μ-based pruning for improved safeness. Our finding is that with inclusion of top-γ superblocks, the average maximum bound check with Inequality (4) may not be necessary, as shown in Section 5, and it can be removed for improved time and space efficiency.
- We devise a compact data structure for superblocks and blocks, that adds reasonable access overhead when conducting superblock pruning. More details are in Section 4.3.

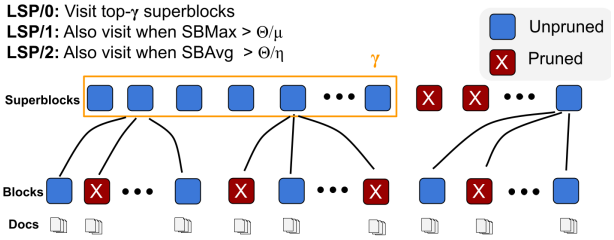


Figure 3: Traversal with superblock pruning in LSP

4 Lightweight Superblock Pruning

4.1 Pruning with top-γ superblock inclusion

Our idea to address erroneous pruning is to impose a simple search guarantee that retrieval always searches the top-γ superblocks with safe pruning. That comes from an observation that superblocks with the largest maximum bound values have a high chance to deliver the top-k relevant results, even though the estimation of maximum score bound can be fairly loose. There is no need to attempt to prune such superblocks. To justify the above design with a proper γ, Section 4.2 provides a statistical analysis.

Imposing the guaranteed inclusion of top-γ superblocks may sufficiently ensure that top-k documents will be found with a high confidence when γ is properly chosen. That means this safeness protection has a large overlap with the role of Inequality (3) and/or Inequality (4) in SP. We consider three versions to simplify SP while avoiding erroneous pruning, as shown in Figure 3:

Version LSP/0: Lightweight superblock pruning with guaranteed top-γ superblock inclusion.

- Identify the top-γ superblocks with the highest $SBMax(X)$ values as long as their superblock level rank bound satisfies $SBMax(X) \geq \theta$.
- Visit the blocks of the above selected superblocks and prune according to $\frac{\Theta}{\eta}$, following SP, and score the documents within these unpruned blocks.

This is a simple version of block-based retrieval with superblock-level pruning without using Inequalities (3) or (4). The block-level pruning and visitation behave the same as BMP and SP with threshold overestimation applied using η (typically 1.0). Compared to BMP, the main advantage is that superblock-level filtering avoids the visitation of large blocks of documents without accessing their block-level or document-level information.

Version LSP/1: Lightweight superblock pruning with guaranteed top-γ superblock inclusion and μ-overestimation.

- This is the same as LSP/0 except that it includes additional superblocks X satisfying $SBMax(X) > \frac{\theta}{\mu}$.

This version ensures that top-γ superblocks are searched, then adds the additional safeness of threshold overestimation. Thus, the average rank score of any top-k results produced by this algorithm is competitive to that of rank-safe retrieval within a factor of μ [45].

Version LSP/2: Superblock pruning with guaranteed γ-superblock inclusion and $(\mu - \eta)$-pruning.

- This algorithm first guarantees top γ superblocks X with safe pruning as long as $SBMax(X) \geq \theta$.
- Then it behaves the same as SP for other superblocks.

This version enhances the safeness of LSP/1 by adding probabilistic safeness with η as described in [7, 45]. Our finding is that this extra safeness is largely overlapped with top-γ superblock inclusion. It is less valuable when the superblock size $b \times c$ is relatively small in which the average superblock rank bound is not too far away from the maximum.

4.2 Choice of γ for top superblock inclusion

This subsection analyzes suitable γ values for the inclusion of top-γ superblocks. Given N superblocks $S_1, \cdots, S_N$ sorted by their rank score bound $SBMax$ values in a non-ascending order, our goal is to find a value γ so that with a high confidence superblock S_i $(\gamma + 1 \leq i \leq N)$ does not contain a document that appears in the top-k document list of a safe pruning algorithm.

For superblock S, we define an $SBMax$ ratio value as $\frac{SBMax(S)}{\max(SBMax(S))}$, representing its closeness to the top-1 superblock $SBMax$ value for a query. From a training dataset, we can approximately derive a distribution for the $SBMax$ ratio values that superblocks can take.

Now let X be a random variable that takes a value between 0 and 1, following the above $SBMax$ ratio distribution. We consider a random sample of size N from this distribution, denoted by $X_1, X_2, \cdots, X_N$. A maximum order statistic is the result of arranging these random variables following our retrieval algorithm in a non-increasing order. Namely $X_{(1)} \geq X_{(2)} \geq \cdots \geq X_{(N)}$.

The standard formula for cumulative probability $P(X_{(i)} \leq x)$ CDF of the i-th largest order statistic is:

$$P(X_{(i)} \leq x) = \sum_{j=N-i+1}^{N} \binom{N}{j} [P(X \leq x)]^j [1 - P(X \leq x)]^{N-j}.$$

$P(X \leq x)$ is obtained using a training dataset by computing the distribution of each possible $SBMax$ value over all superblocks for different training queries. The formula of $P(X_{(i)} < x)$ is similar.

We also divide all possible $SBMax$ values into a set of bins B_j. Let R be an event that a superblock contains a top-k document. Let I be an event that a superblock does not contain a top-k document. Let $P_\gamma(I)$ be the confidence in probability that the γ-th superblock S_γ does not contain a top-k document. Then,

$$P_\gamma(I) = 1 - P_\gamma(R) = 1 - \int_0^1 P(R|x) P_\gamma(x) dx \approx 1 - \sum_j P(R|B_j) \cdot P_\gamma(B_j).$$

Here $P_\gamma(B_j)$ is the probability of a specific bin that the value of $X_{(\gamma)}$ can belong to, and it is determined by the difference between the cumulative probabilities at its right and left boundaries:

$$P_\gamma(B_j) = P(X_{(r)} \leq x_{j,r}) - P(X_{(r)} < x_{j,l})$$

where:

- $x_{j,r}$ is the right boundary SBMax ratio value of this bin B_j;
- $x_{j,l}$ is the left boundary SBMax ratio value of this bin.

For example with two bins: $[0, 0.5)$ and $[0.5, 1]$, $P_\gamma(B_2) = P(X_{(\gamma)} \leq 1) - P(X_{(\gamma)} < 0.5)$.

Figure 4 shows the $P_\gamma(R)$ value distribution when γ varies from 1 to 3,000 for $k = 10$ and $k = 1000$, derived from 10,000 MS MARCO

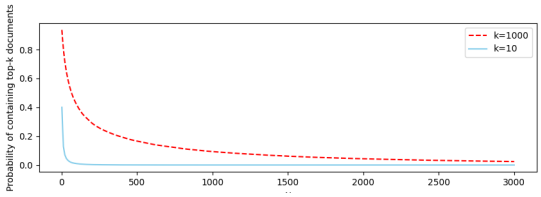


Figure 4: Probability of containing top-k documents at different sorted top superblock positions with $1 \leq \gamma \leq 3000$

V1 Passage training queries, and each superblock contains 128 documents (e.g. $b = 8$ and $c = 16$). The curves for $b = 4$ and 16 are very similar. Table 1 shows $P_\gamma(R)$ at selected values of γ, k, and b. The main takeaways are:

- $P_\gamma(R) > P_{\gamma+1}(R)$. The confidence to find a top-k document in superblock S_γ decreases monotonically as γ becomes large.
- With $k = 10$, confidences among different block/superblock size are about the same for fixed γ in Table 1. Our evaluation chooses $\gamma = 250$ and 500 with high 99.6+% confidence.
- With $k = 1000$ and same γ value, confidences among different block/superblock sizes differ by about 1% when $\gamma \geq 1000$. Our evaluation chooses $\gamma = 1000$ and 2000 with above 90% and 95% confidence. We choose these confidence levels because the above analysis is a conservative estimation of γ as it does not consider the impact of similarity-based clustering in block construction. Once a superblock with a high *SBMax* value contains one top-k document, there is a good chance that it contains more than one such documents for large values of k.

Table 1: Confidence $P_\gamma(I)$ based on MS MARCO training data

γ	100	200	300	1000	2000	3000
$k = 10, b \times c = 64$	98.9%	99.6%	99.8%	~ 100%	~100%	~100%
$k = 10, b \times c = 128$	99.0%	99.6%	99.8%	~100%	~100%	~100%
$k = 10, b \times c = 256$	99.0%	99.6%	99.8%	~100%	~100%	~100%
$k = 1,000, b \times c = 64$	56.4%	69.4%	74.6%	89.4%	95.0%	97.1%
$k = 1,000, b \times c = 128$	59.6%	71.4%	77.3%	90.8%	95.8%	97.6%
$k = 1,000, b \times c = 256$	62.2%	73.7%	79.4%	91.9%	96.5%	98.1%

4.3 Index compression

Compact data structure for block and superblock maximum weights. BMP supports the option of storing their block bounds in an uncompressed dense format and in a sparse format (denoted BMP-Dense and BMP-Sparse). The sparse format yields 5.5GB for MS MARCO with b=8 while the dense format yields 30GB. However, the use of BMP-Sparse for LSP is not effective for random access required by superblock pruning. For example, for safe search using BMP-Sparse is up to 76x slower as shown in Section 5.2.

We seek to adopt SIMDBP for superblock and block level maximum weights because SIMDBP-128 is found to be highly effective for traditional inverted indices [23, 37]. SIMDBP-128 organizes a list of integers to store in the following format $\boxed{S_0}\ \boxed{G_{0,0}, \cdots, G_{0,127}}\ \boxed{S_1}\ \boxed{G_{1,0} \cdots G_{1,127}}, \cdots$. Each group G_i contains 128 integers using at most w bits where $w \leq 32$ and these integers are stored in

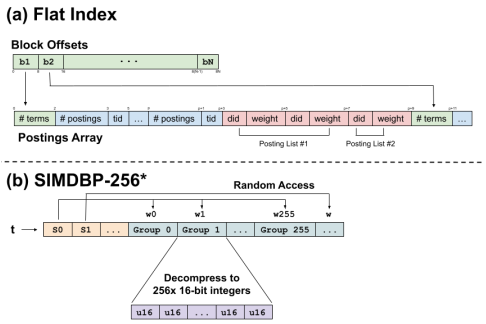


Figure 5: (a) Flat inverted index structure for documents within a block. (b) SIMDBP-256* data structure layout for storing maximum term weights for each block or superblock

128^*w bits with a compact form. When decoding, these integers are recovered into 32-bit integers. Group S_i is called a selector group which contains 128 numbers representing the maximum width for each of the following groups: $G_{i,0}, \cdots, G_{i,127}$.

We design a customized version of SIMDBP with 256 integers per group optimized for storing term maximum weights of a superblock or a block, as shown in Figure 5(b).

- We let each group store 256 block-level or superblock-level term maximum weights consecutively for the same level in an index forest structure. Notice that a 16-bit integer is sufficient to match the width limit of *BoundSum* and *SBMax* accumulation registers, and thus we uncompress a group of data to 256 16-bit integers instead of 128 32-bit integers used in SIMDBP-128. The use of 16-bit integers doubles the number of integers processed per SIMD instruction for faster decoding.
- For every 256 groups, the original SIMDBP method stores a group of "selectors" specifying the bit-width of the following 256 groups. This is effective for decoding of these integers with a sequential access. Instead, we store the selectors at the start of each list of maximum term weights which allows us to efficiently decompress any individual group in a random position.

The second idea is similar to a layout by Mallia et al. [39] for GPU decoding. The main advantage in our context is that moving selectors in advance allows the quick access of any block at a random position. Because of superblock-level pruning, some blocks may be accessed in a non-consecutive manner, causing random access. We denote the above customized design as SIMDBP-256*, and our evaluations show that it is up to 1.5x faster than SIMDBP-256 by enabling random access to block maximum weights.

Quantization for block/superblock maximum weights. BMP uses 8-bit quantization to store the block maximum weights of terms. We find that the use of 4-bit quantization is sufficient for both block and superblock maximum weights. The benefit of 4-bit quantization is a large space reduction of block and superblock maximum weights. It also accelerates decompression and thus maximum rank score computation while retaining competitive relevance, as shown in our evaluation. It should be noted that we still follow BMP to use 8-bit quantization for document-level term weights.

Document index. We consider four methods to store documents. We consider each method as a compact data structure, and do not evaluate applying additional lossless compression techniques.

1. BMP-Inv. BMP uses the terminology block forward index for its document index, meaning it stores an inverted index data structure within each block. We can use this methodology as an uncompressed document index (which we denote BMP-Inv for clarity).

This index is implemented using a Rust nested vector structure to store the term weights of each document. Like C++, each vector in Rust stores a pointer, a length, and a capacity; each vector requires a significant overhead of 24 bytes on a 64-bit system. For 8.8M MS MACRO passages with b=8, this means the vectors themselves require 15.5 GB when loading data to memory, and only about 3 GB is used to store the postings. Notice that when data is stored on disk, pointers and capacity per vector are not needed. BMP and SP both report the size of their indexes as the size on-disk.

2. Compact-Inv. We consider an optimized version of BMP-Inv. We limit block sizes to 256 documents per block, and thus can use a single byte to represent the length. We limit the index to 65k unique terms and store term-ids as two bytes. Additionally, at query time when the index is fixed, the length and capacity of all vectors are identical and we remove this redundancy.

3. Flat-Inv. We reorganize the data of Compact-Inv to a single array instead of using nested Rust vector representations, as shown in Figure 5(a). Additionally, we store a second list containing offsets to each block in the consolidated postings array. We have also tried applying SIMDBP-256* and Simple8b [1] to the postings array after using delta encoding on term IDs and block offsets, following findings of [37]. However, neither method significantly reduced the size of our flat data structure. Therefore we opt to keep element values of the compact postings array uncompressed.

4. Fwd. We also adopt a simple forward index design from Seismic [3, 4], which stores each document independently. Also following Seismic, we score documents using the entire query and only use the pruned query for candidate generation. We apply this technique to all four document scoring methodologies.

5 Experimental Studies

Datasets, metrics, and implementation details. We use the MS MARCO Passage ranking dataset [10] with 8.8 million English passages. We evaluate our results on the development (Dev) query set containing 6980 queries, as well as the TREC deep learning (DL) 2019 and 2020 sets with 43 and 54 queries respectively. We use the standard metrics of mean reciprocal rank (MRR@10) for the Dev set, and normalized discounted cumulative gain (nDCG@10) for DL19 and DL20. We report recall at k (R@k) for all query sets. For zero-shot retrieval performance, the BEIR [53] datasets are used, and the size of each dataset varies from 3,633 to 5.4M documents. Following SP [7], preserved recall ratio (recall budget) is the recall percentage of approximate search divided by that of safe search.

Our implementation extends SP code [7] which is based on BMP using Rust, and compiled with -O3 optimization.[1] All timing results are collected after multiple runs with a single thread on a Linux server using Intel i7-1260P with AVX2 SIMD support and 64GB memory. Before timing queries, all compressed posting lists and metadata for tested queries are pre-loaded into memory, following the common practice.

[1] https://github.com/thefxperson/hierarchical_pruning

Setup. We test our algorithm using two popular learned sparse retrieval algorithms, SPLADE++ [15, 16] and Efficient-SPLADE [20]. The state-of-the-art baselines to be compared include SP [7], BMP [41], and SeismicWave [4], an extended version of Seismic [3]. We do not compare versus ASC [45] because SP is shown to outperform ASC [7] and ASC does not support compression.

5.1 Effectiveness on MS MARCO and BEIR

In-domain search with MS MARCO passages on SPLADE. Table 2 compares LSP/0 with other options based on two fixed configurations. Config. 1 targets roughly 99% relevance, while Config. 2 intends to be close to safe retrieval. We use Config. 2 when searching the DL19/DL20 query sets. Each index and config. are based on recommended parameters for each method:

MaxScore. We use safe search implemented in PISA [36, 56].

BMP. Config. 1 uses a query pruning value of $\beta = 0.8$ (keep highest scoring 80% of query terms) and Config. 2 is safe search. Its indexes are built using $b = 32, 8$ for $k = 10, 1000$.

SP. Config. 1 uses $\mu = 0.5, \eta = 0.8$, and Config. 2 uses $\mu = 0.5, \eta = 1.0$. Indexes are built using $c = 64$ and $b = 16, 8$ for $k = 10, 1000$.

LSP/0. For $k = 10$, Configs. 1 and 2 use $\gamma = 250, 500$ and $k = 1000$ uses $\gamma = 1000, 2000$. For both k, we use a query pruning rate of $\beta = 0.33, 0.5$ for each config. respectively. Indexes use $c = 16$, and $b = 16, 4$ for $k = 10, 1000$.

SeismicWave. We use the best reported settings of `recall_98` and `recall_99` [51], respectively for Configs. 1 and 2. Both settings have a unique index. Config. 1 uses 4 query terms, a heap-factor of 0.8, and unsorted blocks. Config. 2 uses 6 query terms, a heap-factor of 0.8, and sorted blocks. Note that these configurations were selected after an exhaustive grid search on the Dev set, while all other methods could achieve better performance with additional tuning, explored further in Table 3. For $k = 1000$, SeismicWave does not provide any configurations and thus we use the indexing parameters of `recall_99` and the same query processing parameters as $k = 10$ while doubling the posting length limit to 8000. That is because the index used for $k = 10$ could only achieve 97.6 recall.

Table 2 shows that without index compression, LSP/0 is generally the best method, and it is up to 8.1x faster than BMP, 4.8x faster than SP, and up to 3.2x faster than SeismicWave. Excluding SeismicWave for k=10, LSP/0 is always the fastest uncompressed method. With compression and 4-bit block maximum weight quantization, LSP/0 is fastest in all cases except k=10. SeismicWave is fastest for this setting (by 17-40%), which is likely because the recommended parameters come directly from a detailed grid search on the Dev set. LSP/0 with 4-bit quantization does lose some quality compared to LSP/0 with 8-bit maximum weights. However, it is still Pareto-optimal. For example, 4-bit quantization loses 0.05 points of recall for SPLADE $k = 1000$ Config. 1 compared to 8-bit quantization. But, Config. 2 of 4-bit quantization gains 0.18 recall over 8-bit LSP/0 (Config. 1) in about the same time. LSP/0 has a similar or better 99th-percentile latency (P_{99}) as other methods, with about a 2.6x increase over MRT for $k = 10$ and about a 1.5x increase for $k = 1000$, where SeismicWave's P_{99} latencies are about 2.7x and 2.0x MRT respectively. For DL19/DL20, LSP/0 obtains 99% of safe recall. 4-bit quantization again slightly affects retrieval quality, and its recall is slightly better than 8-bit for $k = 1000$, while obtaining 99.9% of safe recall for $k = 10$. The space for indexes for $k = 10$ and

Table 2: MS MARCO passages: mean response time (*ms*) and relevance of LSP/0 and the baslines with two fixed configurations

	SPLADE (In-Domain Parameters)													**E-SPLADE** (Zero-Shot Parameters)						
Config	**Config. 1**				**Config. 2**				**DL19**		**DL20**		**Size**	**Config. 1**			**Config. 2**			**Size**
	MRR	R@k	MRT	P_{99}	MRR	R@k	MRT	P_{99}	nDCG	R@k	nDCG	R@k	**(GB)**	MRR	R@k	MRT	MRR	R@k	MRT	**(GB)**
Method	k=10													k=10						
	Uncompressed													**Uncompressed**						
BMP	38.20	**66.97**	2.53	13.2	38.11	**66.99**	3.18	18.0	73.16	17.25	71.97	24.54	19.2	38.69	**66.98**	0.629	38.81	67.44	0.744	24.5
SP	37.28	64.02	1.40	3.49	38.08	66.92	2.09	7.59	73.16	17.25	71.97	24.54	30.8	Fail due to erroneous pruning						37.0
Seismic-Wave	**38.25**	66.70	**0.297**	**0.801**	38.27	66.89	**0.403**	**1.12**	**73.28**	**17.34**	**71.99**	24.47	7.79	38.78	67.42	0.766	38.79	67.40	0.809	11.2
LSP/0	38.14	66.42	0.425	1.42	**38.28**	66.88	0.649	2.41	73.13	17.25	71.84	24.46	18.8	**38.91**	67.35	0.402	**38.92**	**67.59**	0.506	20.8
	Compressed													**Compressed**						
MaxScore	–	–	–	–	38.11	**66.99**	75.7	333	73.16	17.25	71.97	**24.54**	**1.74**	–	–	–	38.81	67.44	8.06	**2.13**
BMP	38.17	**66.97**	4.20	15.4	38.11	**66.99**	5.41	21.4	73.16	17.25	71.97	**24.54**	17.4	38.67	66.93	1.11	38.81	67.44	1.28	24.5
LSP/0	38.14	66.44	0.501	1.53	**38.28**	66.89	0.878	2.67	73.13	17.25	71.84	24.46	8.99	**38.91**	67.35	0.455	**38.92**	**67.59**	0.570	12.1
+ 4-bit Quant.	38.08	66.36	0.347	0.898	38.23	66.84	0.562	1.48	73.13	17.25	71.93	24.50	5.52	38.86	67.10	**0.327**	38.89	67.35	**0.424**	7.65
	k=1000													k=1000						
	Uncompressed													**Uncompressed**						
BMP	38.20	**98.33**	11.2	29.0	38.11	**98.36**	13.9	42.4	**73.16**	82.91	71.97	**83.91**	47.6	38.69	**97.94**	14.3	38.81	**98.03**	15.5	58.2
SP	38.09	98.24	5.70	24.7	38.09	98.32	11.1	35.4	**73.16**	82.95	71.97	**83.91**	49.0	34.52	96.48	25.8	34.52	96.64	30.1	58.3
Seismic-Wave	**38.30**	98.17	4.42	8.74	**38.30**	98.26	4.97	10.1	73.14	**82.97**	**71.99**	83.81	11.3	38.80	97.85	8.09	38.80	98.00	8.57	16.4
LSP/0	38.29	98.10	1.62	2.75	38.29	98.34	3.08	5.14	73.12	82.32	31.86	83.09	65.7	**38.91**	97.90	2.01	**38.91**	**98.03**	2.35	69.1
	Compressed													**Compressed**						
MaxScore	–	–	–	–	38.11	**98.36**	124	394	**73.16**	82.91	71.97	**83.91**	**1.74**	–	–	–	38.81	**98.03**	13.2	**2.13**
BMP	38.18	98.32	11.9	29.7	38.11	**98.36**	14.8	41.6	**73.16**	82.91	71.97	**83.91**	29.0	38.68	97.92	7.64	38.81	**98.03**	8.86	41.2
LSP/0	38.29	98.11	2.35	3.45	38.29	98.34	3.53	7.03	73.12	82.32	71.86	83.09	17.0	**38.91**	97.89	2.32	**39.91**	**98.03**	3.09	22.8
+ 4-bit Quant.	38.29	98.06	**1.38**	**2.04**	38.29	98.29	**2.30**	**3.73**	73.11	82.62	71.86	83.12	8.44	**38.91**	97.89	**1.44**	**38.91**	**98.03**	**1.80**	11.5

1000 will differ because the best indexing parameters were selected for each value of k.

Zero-shot parameter robustness with LSR index and model variations. We examine if retrieval performance with the same parameters is robust to small variations of the index, such as content updates and small model refinements. For instance, a news index updated hourly can have large variations in the index's distribution, such as posting list length. Here, it is not viable to run a full parameter grid search for each incremental change in index. Optimal performance may not be required during such a change, but retrieval performance needs to remain competitive.

We use E-SPLADE to study the parameter robustness of LSP and baselines in adapting to such a change as it is from the same family as SPLADE but has a variation in index size (i.e. different lengths of posting lists). We keep the same configurations derived from SPLADE on MS MARCO and test them on E-SPLADE in a zero-shot setting. For E-SPLADE with $k = 10$ and $\mu = 0.5$, SP failed to retrieve any documents for some queries because of the erroneous pruning discussed in Section 3. While SeismicWave was faster than LSP/0 for $k = 10$ on SPLADE, when those same configurations are used on E-SPLADE, compressed LSP/0 with quantization is up to 2.3x faster for $k = 10$ while delivering similar relevance, and up to 5.6x faster for $k = 1000$. Both BMP and LSP/0 achieve similar retrieval time and competitive relevance on both SPLADE and E-SPLADE using the same configurations. Certainly, a model could achieve better performance with a grid search of parameters, but this demonstrates that LSP/0's and BMP's parameter choices are robust to model variations within the SPLADE family, while SP fails and SeismicWave has degraded latency.

Best configuration search with LSP/0, LSP/1, and LSP/2 vs. SeismicWave on MS MARCO. Table 3 shows the performance of LSP/0, LSP/1, and LSP/2 when a grid search of the best configuration is allowed for MS MARCO. We use $b = 16, 4$ for $k = 10, 1000$ respectively, $c = 16$, and SIMDBP-256* with 4-bit quantization for all indexes. For each column showing a fixed recall budget, LSP/1 generally achieves the best results for $k = 1000$, but shows roughly similar performance to LSP/0 for $k = 10$. Both LSP/0 and LSP/1 are always better than LSP/2.

Table 3 also shows a comparison of LSP to SeismicWave. For each recall budget, the row labeled "Seismic-W." presents the recall of their best publicly provided configuration (e.g. `recall_98`). However, the best presented configurations never uses their KNN graph. We run a grid-search to find the best configuration including the KNN graph for both $k = 10$ and 1000. These results show that without a KNN graph, SeismicWave is slower than LSP/0 and 1 for $k = 10$ and recall above 97% of safe and faster from 93-97%. With a KNN graph, SeismicWave with a grid search is faster than LSP for $k = 10$. For $k = 1000$, LSP/0 and 1 are 38-87% faster than SeismicWave, and the KNN graph hurts Seismic's performance. Seismic outperforms LSP/2 at all recall budgets except 99% at $k = 1000$.

Table 3: Mean response time (μs) at a fixed recall budget using a grid search of parameters on MS MARCO Dev with SPLADE. All versions of LSP use a Fwd. document index.

Recall	**93%**	**95%**	**97%**	**98%**	**99%**
Preserved	MRT	MRT	MRT	MRT	MRT
	k=10				
LSP/0	195	214	275	281	320
LSP/1	191	215	248	285	315
LSP/2	474	477	527	527	716
Seismic-W.	159	179	221	323	402
$+knn$	**79**	**90**	**99**	**115**	**132**
	k=1000				
LSP/0	571	571	656	**765**	1030
LSP/1	**516**	**562**	**647**	780	**961**
LSP/2	846	937	1173	1280	1710
Seismic-W.	713	876	955	1080	1800
$+knn$	2330	2330	2330	2360	2440

Zero-shot retrieval on the BEIR datasets. We investigate applying the recommended $\gamma = 250, \beta = 0.33$ configuration (with 4-bit SIMDBP-256* compression) for LSP/0 to the BEIR datasets to see

Table 4: Zero-shot performance on 13 BEIR datasets. BMP uses $\beta = 0.8, b = 4$, SP uses $\mu = 0.5$, $\eta = 1.0, c = 16, b = 4$, LSP/0 uses $\gamma = 250, \beta = 0.33, c = 16, b = 4$, SIMDBP-256* with 4-bit weights, and Fwd. document index, Seismic uses `recall_99` configuration.

	Corpus	Safe	BMP			SP			SeismicWave			LSP/0		
Dataset	**Size**	nDCG	nDCG	MRT	GB	nDCG	MRT	GB	nDCG	MRT	GB	nDCG	MRT	GB
Arguana	8.7K	**52.0**	51.1	0.414	0.070	48.7	0.704	0.096	49.5	**0.089**	0.188	50.3	0.222	**0.023**
C-FEVER	5.4M	23.0	24.3	9.45	15.3	**24.5**	10.5	58.3	24.4	**0.453**	7.47	23.6	0.623	**6.14**
DBPedia	4.6M	**43.7**	43.3	6.34	19.8	43.6	2.02	48.3	43.2	0.381	5.63	42.9	**0.330**	**4.79**
FEVER	5.4M	78.8	79.9	7.38	15.2	79.5	4.36	58.3	**80.0**	0.501	7.47	79.2	**0.461**	**6.13**
FiQA	57K	34.7	35.5	0.639	0.405	35.5	0.598	0.666	**35.6**	0.841	0.713	35.1	**0.186**	**0.093**
Hotpot	5.2M	**68.7**	68.6	9.62	21.4	68.5	3.73	54.0	68.3	0.442	5.73	67.1	**0.422**	**5.04**
NFCorpus	3.6K	34.7	35.3	0.104	0.030	34.9	0.123	0.035	**35.6**	**0.045**	0.080	35.2	0.084	**0.011**
NQ	2.7M	53.8	53.8	4.61	14.8	**53.9**	2.11	30.3	53.8	0.363	7.04	53.4	**0.294**	**3.57**
Quora	520K	83.4	**83.5**	0.945	1.27	83.2	0.521	4.51	83.4	0.305	0.693	**83.5**	**0.193**	**0.294**
SciDocs	25K	15.9	15.8	0.444	0.213	15.9	0.499	0.306	15.8	0.239	0.413	**16.2**	**0.175**	**0.052**
SciFact	5K	70.4	70.7	0.350	0.046	70.5	0.469	0.055	70.8	**0.090**	0.115	70.8	0.129	**0.014**
T-COVID	171K	72.7	72.8	1.57	1.10	72.1	1.18	1.92	72.8	1.94	1.98	**72.9**	**0.241**	**0.227**
Touche	380K	24.7	**27.3**	1.77	2.61	**27.3**	0.793	4.56	27.2	0.861	4.32	**27.3**	**0.215**	**0.575**
Average	1.9M	50.5	**50.9**	3.22	7.10	50.6	2.12	20.1	50.8	0.504	3.22	50.6	**0.275**	**2.07**
vs. LSP/0	–	-0.7%	+0.6%	9.3x	3.7x	+0.2%	5.7x	8.0x	+0.4%	2.0x	5.0x	–	–	–
$k = 100$	1.9M	50.6	**51.0**	6.13	7.10	50.7	3.47	20.1	50.9	1.06	3.22	50.9	**0.490**	**2.07**
$k = 1000$	1.9M	50.6	50.9	13.9	7.10	50.7	8.75	20.1	50.9	2.52	3.22	**51.0**	**1.31**	**2.07**

Table 5: Effect of b and γ on latency (ms) and recall (R@1k) of LSP/0 on MS MARCO Dev, SPLADE, $c = 16$, Flat-Inv.

	$\gamma = 250$		$\gamma = 500$		$\gamma = 1000$		$\gamma = 2000$	
	MRT	R@1k	MRT	R@1k	MRT	R@1k	MRT	R@1k
b	k=1000							
4	**2.12**	**95.9**	**2.59**	97.4	**3.28**	98.1	**4.31**	**98.3**
8	2.55	95.5	3.31	97.3	4.40	98.0	5.60	98.2
16	3.94	95.0	5.53	97.0	7.35	97.9	9.48	98.2
32	6.68	94.0	9.87	**97.8**	13.8	**98.2**	17.8	98.2
64	11.3	92.9	21.2	95.8	25.7	97.7	33.6	98.2

if a simple configuration can achieve strong performance out-of-domain. While Seismic-Wave can outperform LSP/0 for k=10, it requires an exhaustive grid-search of parameters to be effective (as shown on E-SPLADE in Table 2), and a cumbersome grid-search over indexing parameters which affect final retrieval performance. Moreover, the proposed optimizations of SeismicWave are not universally better: KNN only benefits retrieval with $k = 10$, and sorting the blocks before evaluation is only used for some recall budgets. To compare SeismicWave in a true zero-shot manner, without a grid search of index or query time parameters, we select their `recall_99` configuration and index and retrieve according to parameters selected on MS MARCO. We also compare versus SP and BMP (with compression). We use the same indexing configuration for all datasets, and choose $b = 4$ because the largest datasets are about half the size of MS MARCO. For $k = 100$ and 1000, we increase γ for LSP/0 to 500 and 1000 respectively, following our results in Section 4.2, but do not adjust our query-pruning rate β. The other methods use a fixed configuration for all value of k.

Table 4 shows the performance of BMP, SP, SeismicWave, and LSP/0 on the 13 BEIR datasets. The row "vs. LSP/0" compares their performance to LSP/0 by calculating the average of the performance ratio between each method, not the ratio of the average. These results show that all methods maintained strong retrieval quality, and that for all datasets except four, LSP/0 is fastest while maintaining competitive relevance. On average, LSP/0 is twice as fast as SeismicWave, 5.7x faster than SP, and 9.3x faster than BMP. It always has the smallest index size, and it maintains its relative latency advantage for $k = 100$ and 1000. This demonstrates that LSP/0's fixed configuration can effectively generalize to new datasets while maintaining competitive relevance and latency.

Choice of block sizes. Table 5 shows the top-1000 retrieval latency of LSP/0 in milliseconds and recall when varying the block size from 4 to 64. As γ increases from 250 to 2000, bigger block sizes proportionally increase the retrieval latency. Small block sizes tend to perform well because the superblock bound estimation is relatively more accurate. Thus $b = 8$ or 4 is a preferred block size for $k = 1000$. For $k = 10$, the overhead of additional blocks outweigh the better bound estimation, and $b = 16$ is best.

A comparison of LSP/0, LSP/1, and LSP/2 with different parameters. Now we show that sometimes LSP/1 can be slightly better than LSP/0 even though Table 2 uses LSP/0 with pre-determined configurations for simplicity. Table 6 compares top-1000 retrieval performance of LSP/0, LSP/1, and LSP/2 under block size b=8 on MS MARCO Passages with 4-bit quantization for block and superblock maximum weights. For LSP/2, we set $\eta = 1$. Notice that safe search achieves recall 98.36. Preserving that by 99% means reaching recall ≥ 97.37. With that as a target, LSP/1 with $\gamma = 500$ and $\mu = 0.20$ gives the shortest latency 3.61ms with recall 97.5. Preserving safe recall by 99.5% means reaching recall 97.87. With that as a target, LSP/0 with $\gamma = 1000$ gives the shortest latency 4.37ms with recall 98. If we simply deploy LSP/0 with $\gamma = 1000$, it can reach a fairly high relevance budget while its speed is competitive to other options.

Scalability on MS MARCO V2 To test the scalability of our method to larger datasets, we evaluate LSP on MS MARCO Passage V2, which contains 138 million passages, approximately 15.7× larger than MS MARCO V1. The dataset size with 16-bit term ids and 8-bit

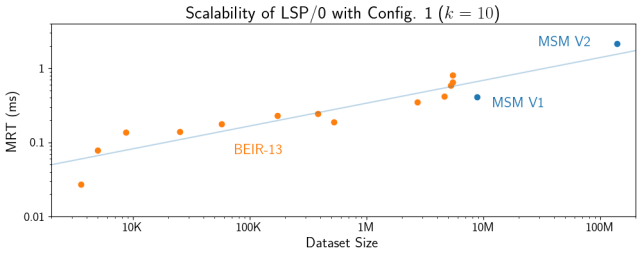


Figure 7: Relation between dataset size and mean query latency on 15 datasets with LSP/0.

Table 6: Comparison of LSP variants with SPLADE. b=8, c=16, 4-bit SIMDBP-256* compression, and Flat-Inv

	γ = 250		γ = 500		γ = 1000		γ = 2000	
	MRT	R@1k	MRT	R@1k	MRT	R@1k	MRT	R@1k
Method (μ)	k=1000							
LSP/0	2.55	95.5	3.37	97.2	4.37	98.0	5.66	98.2
LSP/1 (0.20)	3.06	96.2	3.61	97.5	4.40	98.0	5.55	98.2
LSP/2 (0.20)	3.71	96.2	4.39	97.5	5.11	98.0	6.06	98.2
LSP/1 (0.25)	3.78	96.9	4.13	97.7	4.69	98.1	5.60	98.2
LSP/2 (0.25)	4.35	96.9	4.81	97.7	5.91	98.1	6.26	98.2
LSP/1 (0.33)	4.95	97.7	5.11	98.0	5.35	98.1	5.83	98.2
LSP/2 (0.33)	5.51	97.7	6.00	98.0	6.18	98.1	6.52	98.2
LSP/1 (0.50)	7.11	98.3	7.07	98.3	7.12	98.3	7.19	98.3
LSP/2 (0.50)	7.80	98.3	7.56	98.3	8.01	98.3	7.63	98.3
LSP/1 (0.75)	8.82	98.3	8.86	98.3	8.85	98.3	8.95	98.3
LSP/2 (0.75)	9.04	98.3	9.23	98.3	8.93	98.3	9.13	98.3

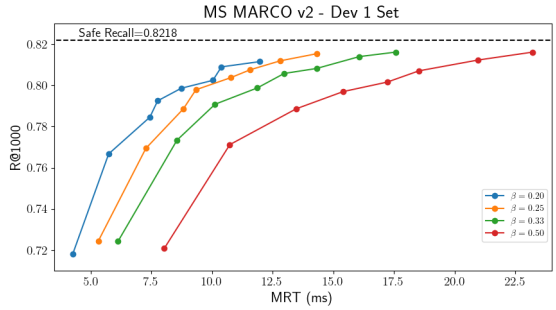


Figure 6: Recall–latency trade-off of LSP/0 on MS MARCO V2 Dev 1 (k = 1000, 2× memory budget). Each curve fixes β and varies γ; the dashed line marks safe recall (R@1000 = 82.18%).

term weights is ≈50 GB. Following [5], we evaluate on the Dev 1 query set (3,903 queries) with SPLADE++ and compare under a memory budget of 2× the dataset size (index size of ≤100 GB). We run experiments single-threaded on a server with two AMD EPYC 7742 processors with 64 cores each and 256GB of DDR4 RAM. We use LSP/0 with 4-bit SIMDBP-256* compression with $c = 16$ and $b = 16$, which yields an index of size 90 GB. We present a recall–latency trade-off for $k = 1000$ in Figure 6, where each curve represents query pruning rates of $\beta \in \{0.20, 0.25, 0.33, 0.50\}$, and each point from left to right shows a different value of $\gamma \in \{500, 1000, 1500, 2000, 2500, 3000, 4000, 5000\}$.

Safe search on Dev 1 achieves R@1000 of 82.18. Smaller β yields lower latency at the cost of a lower recall ceiling: at $\gamma = 5000$, $\beta = 0.20$ reaches 98.8% of safe recall in 11.97 ms, while $\beta = 0.25$ achieves 99.2% in 14.32 ms. On MS MARCO V1, Configs. 1 and 2 target ≈99% and ≈100% of safe recall. At the scale of MS MARCO V2, the same configurations achieve only 94.11% and 96.98% of safe recall for $k = 1000$; $\gamma \geq 4000$ is required to reach 99% preserved recall. For $k = 10$, they achieve 96.1% and 98.8% of safe recall respectively. This is consistent with the expected behavior of the superblock pruning threshold: a larger corpus with fixed indexing parameters yields more superblocks and a larger γ is needed to provide sufficient coverage. With $b = 16$ and $c = 16$ on MS MARCO V2, there are ∼ 540K superblocks, whereas our analysis in Table 1 only considered ∼ 34K-138K superblocks when $b \times c$ =256-64 respectively.

In Figure 7, we present an analysis on the relationship between MRT and dataset size on the 13 BEIR datasets, MS MARCO V1, and MS MARCO V2. We fit a power law using least-squares linear regression with an R^2 of 0.876. The best fitting function takes the form $MRT \approx 0.0048 * |D|^{0.31}$ scaling approximately with the cube root of the dataset size $|D|$. Doubling the size of the dataset will increase the latency by about 24%.

5.2 Compression evaluation

Table 7: Comparison of compression methods for LSP: MS MARCO in-memory index size (GB) with SPLADE. c=16

Block Size	2	4	8	16	32	64	128	256
Document index								
BMP-Inv	28	23	19	15	12	9.6	7.6	6.1
Compact-Inv	22	18	14	12	9.5	7.7	6.3	5.1
Flat-Inv	4.5	4.0	3.6	3.2	**2.9**	**2.6**	**2.4**	**2.4**
Fwd.	**3.1**	**3.1**	**3.1**	**3.1**	3.1	3.1	3.1	3.1
Superblock/block maximum weights								
BMP-Dense	125	63	32	16	8.0	4.0	2.0	1.0
BMP-Sparse	15	9.8	7.6	5.6	4.2	3.0	2.2	1.6
SIMDBP-256*	21	14	9.4	6.0	3.7	2.3	1.3	0.75
+ 4-bit Quant.	**7.4**	**5.3**	**3.5**	**2.4**	**1.5**	**0.95**	**0.59**	**0.33**

Compression Options and Index Size. Table 7 lists the in-memory storage size of document-level index when LSP uses original BMP-Inv, its optimized Compact-Inv, the proposed Flat-Inv, and a forward index (Fwd.). Separately, it lists the total storage for block and superblock maximum weights when LSP uses four options: uncompressed (BMP-Dense), BMP-Sparse, and SIMDBP-256* with 8-bit maximum weights, and SIMDBP-256* with 4-bit maximum weights. The total amount of in-memory storage required is the sum of space for block/superblock maximum weights, and for document-level index. For example, with $b = 8$, Flat-Inv with SIMDBP-256* and 4-bit block/superblock quantization would require 3.6 + 3.5 ≈ 7.1 GB in-memory.

For the document-level index within each block, the proposed flat compact structure constantly delivers the lowest storage size of all inverted indexes with up to 6.2x and 5.3x space reduction compared to the BMP method when $b = 2$ and $b = 8$, respectively. Seismic's forward index uses less storage when $b \leq 16$, and offers up to a 9x space reduction compared to BMP-Inv when $b = 2$. For block/superblock level weights, SIMDBP-256* with 4-bit quantization has the smallest index size for all configurations.

Compression ablation study. Table 8 examines retrieval latency and overall space of combining key compression techniques presented in Section 4.3 under two block sizes and three recall budgets.

In a row-wise top-down manner, we start with an uncompressed index (i.e. BMP-Dense), and then gradually apply selected compression methods. For superblock/block maximum weight compression, we first apply BMP-Sparse or SIMDBP-256*. For SIMDBP-256*, we further apply 4-bit quantization. After selecting SIMDBP-256* with 4-bit quantization, we additionally apply compression to the document index using Compact-Inv, Flat-Inv, or Seismic-Fwd. Times reported for 4-bit quantization are not equivalent to 8-bit safe and are distinguished with asterisks. These methods achieve a recall of about 99.8% of 8-bit safe. The results show that a forward index with SIMDBP-256* and 4-bit maximum weight quantization has a reasonably fast speed while the smallest space (6.8GB) with $b = 8$. For $b = 128$, using Compact-Inv for document inverted index is faster than both Flat-Inv and Seismic-Fwd but costs much more space. Flat-Inv uses slightly less space than Seismic-Fwd for $b = 128$.

Table 8: MS MARCO search time and index size with SPLADE when adding different compression techniques for LSP/1.

	Latency (ms), b=8			Latency (ms), b=128			Space (GB)	
Recall Budget	97%	99%	Safe	97%	99%	Safe	b=8	128
Uncompressed	**0.51**	0.84	**3.3**	12.0	13.3	14.2	51	9.6
Add superblock/block maximum weight compression								
BMP-Sparse	2.7	18	250	73	106	240	26	9.8
SIMDBP-256*	0.83	1.7	6.1	12.2	13.5	14.6	28	8.9
+4-bit Quant.	0.52	0.82	4.7*	7.8	8.3	8.9*	22	8.2
Add document index compression								
Compact-Inv	0.59	0.95	5.4*	**7.5**	**7.9**	**8.6***	18	6.9
Flat-Inv	0.56	0.85	4.7*	9.4	9.9	10.7*	7.3	**3.2**
Fwd.	0.52	**0.77**	4.8*	9.9	10.6	11.8*	**6.8**	3.7

Inverted vs. forward index per document block. Table 9 compares the latency of Flat-Inv and Fwd. across different block sizes for both safe and approximate retrieval. The forward index achieves lower latency at small block sizes ($b \leq 64$), but similar or higher latency than Flat-Inv at larger block sizes ($b \geq 96$). Fwd. stores separate arrays for term ids and weights, requiring only two random memory accesses per block. However, it must fetch all terms in each document, regardless of which terms appear in the query. The inverted index requires one random memory access per query term (~43 for SPLADE++ on the Dev set), but only fetches postings for terms present in the query. At small block sizes, the random memory accesses of the inverted index cause it to be slower, while at larger block sizes the total memory fetched per block becomes a bottleneck, favoring the inverted index.

Table 9: Latency comparison (ms) of Flat-Inv vs. Fwd. document indexes on MS MARCO with SPLADE for LSP/0 under different block sizes. c=16, k=1000, SIMDBP-256*.

	Block Size (b)					
Index Type	8	16	32	64	96	128
99% recall budget						
Flat-Inv	4.29	9.00	17.0	31.3	44.0	**55.6**
Fwd.	**2.43**	**5.30**	**11.4**	**25.8**	**43.1**	60.5
Safe retrieval						
Flat-Inv	11.6	17.0	31.3	56.4	76.5	**91.6**
Fwd.	**9.65**	**11.5**	**22.1**	**48.0**	**76.3**	103

6 Conclusion

We introduced the safe inclusion of top-γ superblocks in superblock-based retrieval to avoid erroneous index skipping and to simplify pruning with less overhead. The main takeaways are:

- **Out-of-domain zero-shot on BEIR.** LSP/0 with a simple configuration is 2x faster than SeismicWave, 5.7x faster than SP, and 9.3x faster than BMP with negligible quality differences, while requiring 5x, 8x and 4x less memory respectively. Thus for out-of-domain retrieval, the main advantage of LSP/0 over SeismicWave is its simplicity without a grid search of parameters and avoiding the need of a KNN graph which is complex to maintain.
- **In-domain search with MS MARCO.** LSP/0 is 1.8x to 12x faster than BMP while using 3.3x less memory, and up-to 17x faster than SP while using 1/6th the memory. LSP/0 with a simple configuration is up-to 3.2x faster than SeismicWave for $k = 1000$ on SPLADE, but it is about 30% slower for $k = 10$ on SPLADE. When a grid search of all parameters is possible with SPLADE, for $k = 1000$, LSP still outperforms SeismicWave by 55% to 78%. For $k = 10$, LSP/0 is about 20% faster than SeismicWave without a KNN graph above 98% preserved recall, while about 20% slower for 93-97% preserved recall. With a KNN graph, SeismicWave is about 2x as fast as LSP for $k = 10$.
- **Parameter adaptivity.** LSP/0 with our proposed configuration derived from SPLADE is able to be applied to E-SPLADE on MS MARCO successfully. Under the same setting, BMP succeeded, SP failed from erroneous pruning, and SeismicWave's latency roughly doubled. LSP/0 was the fastest of the three on E-SPLADE.
- **LSP/0 vs LSP/1 vs LSP/2.** LSP/1 can outperform LSP/0 in some cases when using a parameter grid search, however LSP/0 is easy-to-use with highly competitive performance both with and without compression. Both LSP/0 and LSP/1 outperform LSP/2, which shows that superblock average weights are not needed when using a large number of blocks and a top-γ guarantee.
- **Recommended LSP configuration.** LSP/0 with $\gamma = 250$ or 500 is recommended when $k = 10$, and $\gamma = 1000$ or 2000 when $k = 1000$, with query pruning set to $\beta \approx 0.33$ for datasets around $9M$ documents like MS MARCO V1 or fewer. For MS MARCO V2, which is ~ 16x larger than V1, γ needs to approximately double to achieve 99% of safe recall. We recommend SIMDBP-256* compression is used with 4-bit quantization, and a forward document index for small values of b. LSP performs best when the block size is small (e.g. $4 \leq b \leq 16$) and the superblock size is within the tested range: $64 \leq b \times c \leq 256$. More work is needed to assess different superblock sizes for larger datasets. These configurations provides competitive performance, albeit not the maximum, and generalize well across multiple datasets and models, which simplifies their use for zero-shot out-of-domain search settings.

Overall speaking, LSP is a strong alternative for efficient retrieval. Our future work is to explore generalizing LSP's pruning structure to more layers, and to investigate additional index compression strategies.

Acknowledgments. We thank anonymous referees for their valuable comments. This work is supported in part by U.S. NSF IIS-2225942 and has used the computing resource of the NSF's ACCESS program. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the U.S. NSF.

References

[1] Vo Ngoc Anh and Alistair Moffat. 2010. Index compression using 64-bit words. *Software: Practice and Experience* 40, 2 (2010), 131–147.

[2] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management* (New Orleans, LA, USA) *(CIKM '03)*. Association for Computing Machinery, New York, NY, USA, 426–434. doi:10.1145/956863.956944

[3] Sebastian Bruch, Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. 2024. Efficient Inverted Indexes for Approximate Retrieval over Learned Sparse Representations. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 152–162. doi:10.1145/3626772.3657769

[4] Sebastian Bruch, Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. 2024. Pairing Clustered Inverted Indexes with k-NN Graphs for Fast Approximate Retrieval over Learned Sparse Representations. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management* (Boise, ID, USA) *(CIKM '24)*. Association for Computing Machinery, New York, NY, USA, 3642–3646. doi:10.1145/3627673.3679977

[5] Sebastian Bruch, Franco Maria Nardini, Cosimo Rulli, Rossano Venturini, and Leonardo Venuta. 2025. Investigating the Scalability of Approximate Sparse Retrieval Algorithms to Massive Datasets. In *Advances in Information Retrieval: 47th European Conference on Information Retrieval (ECIR 2025) (ECIR '25)*. Springer Nature Switzerland, Cham, 437–445. doi:10.1007/978-3-031-88714-7_43

[6] Fazli Can, Ismail Sengör Altingövde, and Engin Demir. 2004. Efficiency and effectiveness of query processing in cluster-based retrieval. *Information Systems* 29, 8 (2004), 697–717. doi:10.1016/S0306-4379(03)00062-0

[7] Parker Carlson, Wentai Xie, Shanxiu He, and Tao Yang. 2025. Dynamic Superblock Pruning for Fast Learned Sparse Retrieval. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Padua, Italy) *(SIGIR '25)*. Association for Computing Machinery, New York, NY, USA, 3004–3009. doi:10.1145/3726302.3730183

[8] Flavio Chierichetti, Alessandro Panconesi, Prabhakar Raghavan, Mauro Sozio, Alessandro Tiberi, and Eli Upfal. 2007. Finding near neighbors through cluster pruning. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Beijing, China) *(PODS '07)*. Association for Computing Machinery, New York, NY, USA, 103–112. doi:10.1145/1265530.1265545

[9] Matt Crane, J. Shane Culpepper, Jimmy Lin, Joel Mackenzie, and Andrew Trotman. 2017. A Comparison of Document-at-a-Time and Score-at-a-Time Query Evaluation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining* (Cambridge, United Kingdom) *(WSDM '17)*. ACM, New York, NY, USA, 201–210.

[10] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Fernando Campos, and Ellen M. Voorhees. 2020. Overview of the TREC 2020 Deep Learning Track. *ArXiv* abs/2102.07662 (2020).

[11] Laxman Dhulipala, Igor Kabiljo, Brian Karrer, Giuseppe Ottaviano, Sergey Pupyrev, and Alon Shalita. 2016. Compressing Graphs and Indexes with Recursive Graph Bisection. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1535–1544. doi:10.1145/2939672.2939862

[12] Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. 2013. A Candidate Filtering Mechanism for Fast Top-k Query Processing on Modern CPUs. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Dublin, Ireland) *(SIGIR '13)*. Association for Computing Machinery, New York, NY, USA, 723–732. doi:10.1145/2484028.2484087

[13] Shuai Ding and Torsten Suel. 2011. Faster Top-k Document Retrieval Using Block-Max Indexes. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Beijing, China) *(SIGIR '11)*. Association for Computing Machinery, New York, NY, USA, 993–1002. doi:10.1145/2009916.2010048

[14] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) *(KDD '24)*. Association for Computing Machinery, New York, NY, USA, 6491–6501. doi:10.1145/3637528.3671470

[15] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From Distillation to Hard Negative Sampling: Making Sparse Neural IR Models More Effective. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Madrid, Spain) *(SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 2353–2359. doi:10.1145/3477495.3531857

[16] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada) *(SIGIR '21)*. Association for Computing Machinery, New York, NY, USA, 2288–2292. doi:10.1145/3404835.3463098

[17] Fatih Hafizoglu, Emre Can Kucukoglu, and Ismail Sengor Altingovde. 2017. On the Efficiency of Selective Search. In *Advances in Information Retrieval - 39th European Conference on IR Research, ECIR (Lecture Notes in Computer Science, Vol. 10193)*. Aberdeen, Scotland, UK, 705–712. doi:10.1007/978-3-319-56608-5_69

[18] Oren Kurland. 2008. The opposite of smoothing: A language model approach to ranking query-specific document clusters. In *Proce. of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. 171–178.

[19] Saar Kuzi, Mingyang Zhang, Cheng Li, Michael Bendersky, and Marc Najork. 2020. Leveraging semantic and lexical matching to improve the recall of document retrieval systems: A hybrid approach. *arXiv preprint arXiv:2010.01195* (2020).

[20] Carlos Lassance and Stéphane Clinchant. 2022. An Efficiency Study for SPLADE Models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Madrid, Spain) *(SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 2220–2226. doi:10.1145/3477495.3531833

[21] Carlos Lassance, Hervé Déjean, Thibault Formal, and Stéphane Clinchant. 2024. SPLADE-v3: New baselines for SPLADE. arXiv:2403.06789 [cs.IR]

[22] Carlos Lassance, Simon Lupart, Hervé Déjean, Stéphane Clinchant, and Nicola Tonellotto. 2023. A Static Pruning Study on Sparse Neural Retrievers. In *Proc. of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 1771–1775.

[23] Daniel Lemire and Leonid Boytsov. 2015. Decoding billions of integers per second through vectorization. *Softw. Pract. Exp.* 45, 1 (2015), 1–29.

[24] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 793, 16 pages.

[25] Hang Li, Shuai Wang, Shengyao Zhuang, Ahmed Mourad, Xueguang Ma, Jimmy Lin, and Guido Zuccon. 2022. To Interpolate or not to Interpolate: PRF, Dense and Sparse Retrievers. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Madrid, Spain) *(SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 2495–2500. doi:10.1145/3477495.3531884

[26] Jimmy Lin and Xueguang Ma. 2021. A Few Brief Notes on DeepImpact, COIL, and a Conceptual Framework for Information Retrieval Techniques. *CoRR* abs/2106.14807 (2021). arXiv:2106.14807 https://arxiv.org/abs/2106.14807

[27] Xiaoyong Liu and W Bruce Croft. 2004. Cluster-based retrieval using language models. In *Proc. of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. 186–193.

[28] Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2012. Effect of Dynamic Pruning Safety on Learning to Rank Effectiveness. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Portland, Oregon, USA) *(SIGIR '12)*. Association for Computing Machinery, New York, NY, USA, 1051–1052.

[29] Joel Mackenzie, Antonio Mallia, Alistair Moffat, and Matthias Petri. 2022. Accelerating Learned Sparse Indexes Via Term Impact Decomposition. In *Findings of the Association for Computational Linguistics: Empirical Methods in Natural Language Processing 2022*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 2830–2842. doi:10.18653/v1/2022.findings-emnlp.205

[30] Joel Mackenzie, Antonio Mallia, Matthias Petri, J Shane Culpepper, and Torsten Suel. 2019. Compressing inverted indexes with recursive graph bisection: A reproducibility study. In *Advances in Information Retrieval: 41st European Conference on IR Research, ECIR 2019, Cologne, Germany, April 14–18, 2019, Proceedings, Part I 41*. Springer, 339–352.

[31] Joel Mackenzie, Matthias Petri, and Alistair Moffat. 2021. Anytime Ranking on Document-Ordered Indexes. *ACM Transactions on Information Systems (TOIS)* 40, 1, Article 13 (2021), 32 pages.

[32] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. Learning passage impacts for inverted indexes. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1723–1727.

[33] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. Learning Passage Impacts for Inverted Indexes. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada) *(SIGIR '21)*. Association for Computing Machinery, New York, NY, USA, 1723–1727. doi:10.1145/3404835.3463030

[34] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. Learning Passage Impacts for Inverted Indexes. In *Proceedings of the 44th International*

ACM SIGIR Conference on Research and Development in Information Retrieval (Virtual Event, Canada) *(SIGIR '21)*. Association for Computing Machinery, New York, NY, USA, 1723–1727. doi:10.1145/3404835.3463030

[35] Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. 2017. Faster BlockMax WAND with Variable-sized Blocks. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Shinjuku, Tokyo, Japan) *(SIGIR '17)*. Association for Computing Machinery, New York, NY, USA, 625–634. doi:10.1145/3077136.3080780

[36] Antonio Mallia, Michal Siedlaczek, Joel Mackenzie, and Torsten Suel. 2019. PISA: Performant Indexes and Search for Academia. In *Proceedings of the Open-Source IR Replicability Challenge co-located with 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, OSIRRC@SIGIR 2019, Paris, France, July 25, 2019*. 50–56. http://ceur-ws.org/Vol-2409/docker08.pdf

[37] Antonio Mallia, Michał Siedlaczek, and Torsten Suel. 2019. An experimental study of index compression and DAAT query processing methods. In *European Conference on Information Retrieval*. Springer, 353–368.

[38] Antonio Mallia, Michał Siedlaczek, and Torsten Suel. 2021. Fast Disjunctive Candidate Generation Using Live Block Filtering. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining* (Virtual Event, Israel) *(WSDM '21)*. Association for Computing Machinery, New York, NY, USA, 671–679. doi:10.1145/3437963.3441813

[39] Antonio Mallia, Michał Siedlaczek, Torsten Suel, and Mohamed Zahran. 2019. GPU-Accelerated Decoding of Integer Lists. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 2193–2196. doi:10.1145/3357384.3358067

[40] Antonio Mallia, Michal Siedlaczek, Mengyang Sun, and Torsten Suel. 2020. A Comparison of Top-k Threshold Estimation Techniques for Disjunctive Query Processing. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 2141–2144. doi:10.1145/3340531.3412080

[41] 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* (Washington DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 2411–2415. doi:10.1145/3626772.3657906

[42] Egor Markovskiy, Fiana Raiber, Shoham Sabach, and Oren Kurland. 2022. From Cluster Ranking to Document Ranking. In *Proc. of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Madrid, Spain) *(SIGIR '22)*. 2137–2141.

[43] Alistair Moffat and Matthias Petri. 2018. Index compression using byte-aligned ANS coding and two-dimensional contexts. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 405–413.

[44] Giuseppe Ottaviano and Rossano Venturini. 2014. Partitioned elias-fano indexes. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. 273–282.

[45] Yifan Qiao, Parker Carlson, Shanxiu He, Yingrui Yang, and Tao Yang. 2024. Threshold-driven Pruning with Segmented Maximum Term Weights for Approximate Cluster-based Sparse Retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 19742–19757. https://aclanthology.org/2024.emnlp-main.1101

[46] Yifan Qiao, Parker Carlson, Shanxiu He, Yingrui Yang, and Tao Yang. 2024. Threshold-driven Pruning with Segmented Maximum Term Weights for Approximate Cluster-based Sparse Retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 19742–19757. https://aclanthology.org/2024.emnlp-main.1101

[47] Yifan Qiao, Yingrui Yang, Shanxiu He, and Tao Yang. 2023. Representation Sparsification with Hybrid Thresholding for Fast SPLADE-based Document Retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 2329–2333. doi:10.1145/3539618.3592051

[48] Yifan Qiao, Yingrui Yang, Haixin Lin, and Tao Yang. 2023. Optimizing Guided Traversal for Fast Learned Sparse Retrieval. In *Proceedings of the ACM Web Conference 2023* (Austin, TX, USA) *(WWW '23)*. Association for Computing Machinery, New York, NY, USA, 3375–3385. doi:10.1145/3543507.3583497

[49] Stephen E. Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3 (2009), 333–389.

[50] Keshav Santhanam, O. Khattab, Jon Saad-Falcon, Christopher Potts, and Matei A. Zaharia. 2022. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. *NAACL'22* ArXiv abs/2112.01488 (2022).

[51] SeismicWave. 2025. https://github.com/TusKANNy/seismic/blob/main/experiments/best_configs/msmarco-v1/splade-cocondenser/mem_budget_2/recall_99.toml. (2025).

[52] Tao Shen, Xiubo Geng, Chongyang Tao, Can Xu, Xiaolong Huang, Binxing Jiao, Linjun Yang, and Daxin Jiang. 2023. LexMAE: Lexicon-Bottlenecked Pretraining for Large-Scale Retrieval. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=PfpEtB3-csK

[53] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://openreview.net/forum?id=wCu6T5xFjeJ

[54] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2013. Efficient and effective retrieval using selective pruning. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining* (Rome, Italy) *(WSDM '13)*. Association for Computing Machinery, New York, NY, USA, 63–72. doi:10.1145/2433396.2433407

[55] Andrew Trotman and Jimmy Lin. 2016. In vacuo and in situ evaluation of SIMD codecs. In *Proceedings of the 21st Australasian Document Computing Symposium*. 1–8.

[56] Howard Turtle and James Flood. 1995. Query Evaluation: Strategies and Optimizations. *Information Processing & Management* 31, 6 (1995), 831–850.

[57] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2023. SimLM: Pre-training with Representation Bottleneck for Dense Passage Retrieval. *ACL* (2023).

[58] Shitao Xiao, Zheng Liu, Yingxia Shao, and Zhao Cao. 2022. RetroMAE: Pre-training Retrieval-oriented Transformers via Masked Auto-Encoder. *EMNLP* (2022).

[59] Hao Yan, Shuai Ding, and Torsten Suel. 2009. Inverted index compression and query processing with optimized document ordering. In *Proceedings of the 18th international conference on World wide web*. 401–410.

[60] Yingrui Yang, Parker Carlson, Shanxiu He, Yifan Qiao, and Tao Yang. 2024. Cluster-based Partial Dense Retrieval Fused with Sparse Text Retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 2327–2331. doi:10.1145/3626772.3657972

[61] Hansi Zeng, Julian Killingback, and Hamed Zamani. 2025. Scaling Sparse and Dense Retrieval in Decoder-Only LLMs. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Padua, Italy) *(SIGIR '25)*. Association for Computing Machinery, New York, NY, USA, 2679–2684. doi:10.1145/3726302.3730225

[62] Jiangong Zhang, Xiaohui Long, and Torsten Suel. 2008. Performance of compressed inverted list caching in search engines. In *Proceedings of the 17th international conference on World Wide Web*. 387–396.