Threshold-driven Pruning with Segmented Maximum Term Weights for Approximate Cluster-based Sparse Retrieval

Anonymous ACL submission

Abstract

This paper revisits dynamic pruning through rank score thresholding in cluster-based sparse retrieval to skip the index partially at cluster and document levels during inference. It proposes a two-parameter pruning control scheme called ASC with a probabilistic guarantee on rank-safeness competitiveness. ASC uses cluster-level maximum weight segmentation to improve accuracy of rank score bound estimation and threshold-driven pruning, and is targeted for speeding up retrieval applications requiring high relevance competitiveness. The experiments with MS MARCO and BEIR show that ASC improves the accuracy and safeness of pruning for better relevance while delivering a low latency on a single-threaded CPU.

1 Introduction

Fast and effective document retrieval is a critical component of large-scale search systems. This can also be important for retrieval-augmented generation systems which are gaining in popularity. Retrieval systems fall into two broad categories: dense (single or multi-vector) (Karpukhin et al., 2020; Ren et al., 2021; Xiao et al., 2022; Wang et al., 2023; Santhanam et al., 2022) and sparse (lexical or learned) (Dai and Callan, 2020; Mallia et al., 2021a; Lin and Ma, 2021; Gao et al., 2021; Formal et al., 2021; Shen et al., 2023). Efficient dense retrieval relies on approximation techniques with notable relevance drops (Johnson et al., 2019; Malkov and Yashunin, 2020; Kulkarni et al., 2023; Zhang et al., 2023), whereas sparse retrieval takes advantage of fast inverted index implementations on CPUs. Well-trained models from these two categories can achieve similar relevance numbers on the standard MS MARCO passage ranking task. However, for zero-shot out-of-domain search on the BEIR datasets, learned sparse retrieval exhibits stronger relevance than BERT-based dense models. Accordingly, this paper focuses on optimizing online inference efficiency for sparse retrieval. Another reason for this focus is that sparse retrieval does not require expensive GPUs, and thus can significantly lower the infrastructure cost for a large-scale retrieval system that hosts data partitions on a massive number of inexpensive CPU servers.

A traditional optimization for sparse retrieval is rank-safe threshold-driven pruning algorithms, such as MaxScore (Turtle and Flood, 1995), WAND (Broder et al., 2003), and BlockMax WAND (BMW) (Ding and Suel, 2011), which accurately skip the evaluation of low-scoring documents that are unable to appear in the final top-k results. Two key extensions of these pruning methods are cluster-based pruning and rank-unsafe threshold over-estimation. Cluster-based (or block-based) pruning extends rank-safe methods to skip the evaluation of groups of documents (Dimopoulos et al., 2013; Mallia et al., 2021b; Mackenzie et al., 2021). However, the cluster bounds estimated by current methods are often loose, which limits pruning opportunities. Threshold over-estimation (Macdonald et al., 2012; Tonellotto et al., 2013; Crane et al., 2017) relaxes the safeness, and allows some potentially relevant documents to be skipped, trading relevance for faster retrieval. However, there are no formal analysis or guarantee on the impact of rank-unsafeness on relevance and its speed gain can often come with a substantial relevance drop.

This paper revisits rank score threshold-driven pruning for cluster-based retrieval in both safe and unsafe settings. We introduce a two-parameter threshold control scheme called ASC, which addresses the above two limitations of current threshold-driven pruning methods. ASC uses cluster-level maximum weight segmentation to improve the accuracy of cluster bound estimation and offer a probabilistic guarantee on rank-safeness when used with threshold over-estimation. Consequently, ASC is targeted at speeding up retrieval in applications that desire high relevance.

Our evaluation shows that ASC makes sparse retrieval with SPLADE (Formal et al., 2022), uniCOIL (Lin and Ma, 2021), and LexMAE (Shen et al., 2023) much faster while effectively retaining their relevance. ASC takes only 9.7ms with $k = 10$ and 21ms with $k = 1000$ for LexMAE on a single-threaded consumer CPU to search MS MARCO passages with 0.4252 MRR. It takes only 5.59ms and 15.8ms respectively for SPLADE with over 0.3962 MRR. When prioritizing for a small MRR relevance loss, ASC can be an order of magnitude faster than other approximation baselines.

2 Background and Related Work

Problem definition. Sparse document retrieval identifies top-k ranked candidates that match a query. Each document in a data collection is modeled as a sparse vector with many zero entries. These candidates are ranked using a simple additive formula, and the rank score of each document d is defined as: $RankScore(d) = \sum_{t \in Q} w_{t,d}$, where Q is the set of search terms in the given query, $w_{t,d}$ is a weight contribution of term t in document d, possibly scaled by a corresponding query term weight. Term weights can be based on a lexical model such as BM25 (Jones et al., 2000) or are learned from a neural model. Terms are tokens in these neural models. For a sparse representation, a retrieval algorithm uses an *inverted index* with a set of terms, and a *document posting list* for each term. A posting record in this list contains a document ID and its weight for the corresponding term.

Threshold-driven skipping. During sparse retrieval, a pruning 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. WAND uses the maximum term weight of documents in a posting list for their score upper bound, while BMW and its variants (e.g. VBMW (Mallia et al., 2017)) use block-based maximum weights. MaxScore uses a similar skipping strategy with term partitioning. 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 over-estimation is a "rank-unsafe" skipping strategy that deliberately over-estimates the current top-k threshold by a factor (Macdonald et al., 2012; Tonellotto et al., 2013; Crane et al., 2017). There is no formal analysis of the above rank-safeness approximation, whereas our work generalizes and improves threshold over-estimation for better rank-safeness control in cluster-based retrieval with a formal guarantee.

Live block filtering and cluster-based retrieval. Live block filtering (Dimopoulos et al., 2013; Mallia et al., 2021b) clusters document IDs within a range and estimates a range-based maximum score for pruning. Anytime Ranking (Mackenzie et al., 2021) extends *cluster skipping inverted index* (Can et al., 2004; Hafizoglu et al., 2017) which arranges each posting list as "clusters" for selective retrieval, and searches top clusters under a time budget. Without early termination, Anytime Ranking is rank-safe and conceptually the same as live block filtering with an optimization that cluster visitation is ordered dynamically. Contemporary work in (Mallia et al., 2024) introduces several optimizations for live block filtering called BMP with block reordering and threshold overestimation and shows that a block-based (cluster-based, equivalently) retrieval still represents a state-of-the-art approach for safe pruning and for approximate search.

Our work can be effectively combined with the above work using maximum cluster-level score bounds and threshold over-estimation, and is focused on improving accuracy of cluster score bounds and threshold-driven pruning to increase index-skipping opportunities and introduce a probabilistic rank-safeness assurance.

Efficiency optimization for learned sparse retrieval. There are orthogonal techniques to speedup learned sparse retrieval. BM25-guided pruning skips documents during learned index traversal (Mallia et al., 2022; Qiao et al., 2023b). Static index pruning (Qiao et al., 2023a; Lassance et al., 2023) removes low-scoring term weights during index generation. An efficient version of SPLADE (Lassance and Clinchant, 2022) uses L1 regularization for query vectors, dual document and query encoders, and language model middle training. Term impact decomposition (Mackenzie et al., 2022a) partitions each posting list into two groups with high and low impact weights. Our work is complementary to the above techniques.

Approximation with score-at-a-time retrieval (SAAT). The above retrieval approaches often conduct document-at-a-time (DAAT) traversal over document-ordered indexes. The SAAT retrieval over impact-ordered indexes is an alternative method used together with earlier termination such as JASS (Lin and Trotman, 2015) and IOQP (Mackenzie et al., 2022b).

An experimental study (Mackenzie et al., 2023) compares DAAT and SAAT for a number of sparse models and indicates that while no single system dominates all scenarios, it confirms that DAAT Anytime code is a strong contender, especially for SPLADE when maintaining the small MRR@10 loss. Since IOQP has been shown to be highly competitive to an optimized version of JASS, the baselines in Section 4 includes Anytime and IOQP.

Big-ANN competition for sparse retrieval. NeurIPS 2023 Big-ANN competition sparse track (Big-ANN, 2024) uses 90% recall of safe search top 10 results as the relevance budget to select the fastest entry for MS MARCO dev set with SPLADE, and this metric drives a different optimization tradeoff compared to our paper. Our paper prioritizes MRR@10 competitiveness of approximate retrieval with a much tighter relevance loss budget before considering gains in latency reduction. Appendix E provides a comparison of ASC with two top winners of this competition. Reference (Bruch et al., 2024) is listed for the Pinecone entry with no open source code released, and it presents an approach to combine dense and sparse retrieval representations with random projection, which is orthogonal to our approach.

3 Cluster-based Retrieval with Approximation and Segmentation

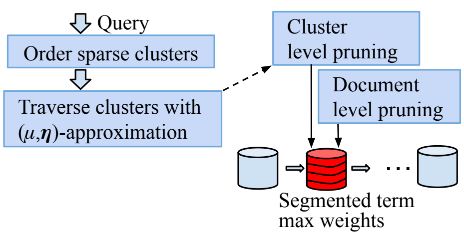


Figure 1: Flow of ASC with two-parameter pruning control and segmented cluster-level maximum term weights

The overall online inference flow of the proposed scheme during retrieval is shown in Figure 1. Initially, sparse clusters are sorted in a non-increasing order of their estimated cluster upper bounds. Then, search traverses the sorted clusters one-by-one to conduct approximate retrieval with two-level pruning with segmented term maximum weight.

We follow the notation in (Mackenzie et al., 2021). A document collection is divided into m clusters $\{C_1, \cdots, C_m\}$. Each posting list of an inverted index is structured using these clusters. Given query Q, the $BoundSum$ formula below estimates the maximum rank score of a document in a cluster. Anytime Ranking visits clusters in a non-increasing order of $BoundSum$ values.

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

The visitation to cluster C_i can be pruned if $BoundSum(C_i) \leq \theta$, where θ is the current top-k threshold. If this cluster is not pruned, then document-level index traversal and skipping can be conducted within each cluster following a standard retrieval algorithm. Any document within such a cluster may be skipped for evaluation if $Bound(d) \leq \theta$ where $Bound(d)$ is computed on the fly based on an underlying retrieval algorithm such as MaxScore and VBMW.

Design considerations. The cluster-level $BoundSum$ estimation in Formula (1) can be loose, especially when a cluster contains diverse document vectors, and this reduces the effectiveness of pruning. As an illustration, Figure 2 shows the bound tightness of Anytime for MS MARCO Passage clusters, calculated as the ratio between the average actual and estimated bound: $\frac{1}{m}\sum_{i=1}^{m} \frac{\max_{d_j \in C_i} RankScore(d_j)}{BoundSum(C_i)}$, where m is the number of clusters. A bound tightness near 1 means the bound estimate is accurate, whereas a value near 0 means a loose estimate. The average bound tightness increases with m because smaller clusters are more similar. ASC improves the tightness of the cluster bound estimation for all values of m.

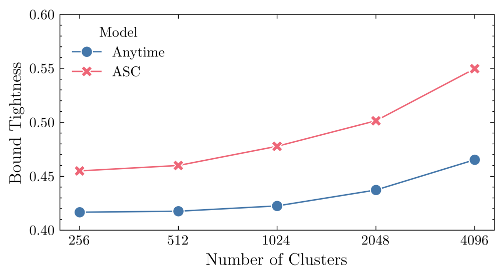


Figure 2: ASC predicts more accurate cluster bounds, which allows it to prune more aggressively. Cluster bound tightness is the average ratio of the actual and estimated cluster bounds, calculated with Formula (1).

Limited threshold over-estimation can be helpful to deal with a loose bound estimation. Specifically, over-estimation of the top-k threshold is applied by a factor of μ, where $0 < \mu \leq 1$, and the above pruning conditions are modified as $BoundSum(C_i) \leq \frac{\theta}{\mu}$ and $Bound(d) \leq \frac{\theta}{\mu}$. Threshold over-estimation with μ allows skipping more low-scoring documents when the bound estimation is too loose. However, thresholding is applied to all cases uniformly and can incorrectly prune many desired relevant documents when the bound estimation is already tight.

To improve the tightness of cluster-level bound estimation using Formula (1), one can decrease the size of each cluster. However, there is a significant overhead when increasing the number of clusters. One reason is that for each cluster, one needs to extract the maximum weights of query terms and estimate the cluster bound, which can become expensive for a large number of query terms. Another reason is that MaxScore identifies a list of essential query terms which are different from one cluster to another. Traversing more clusters yields more overhead for essential term derivation, in addition to the cluster bound computation.

3.1 ASC: (μ, η)-approximate retrieval with segmented cluster information

The proposed **ASC** method stands for (μ, η)-**A**pproximate retrieval with **S**egmented **C**luster-level maximum term weights. ASC segments cluster term maximum weights to improve the tightness of cluster bound estimation and guide cluster-level pruning. It employs two parameters, μ and η, satisfying $0 < \mu \leq \eta \leq 1$, to detect the cluster bound estimation tightness and improve pruning safeness. Details of our algorithm are described below.

Extension to the cluster-based skipping index. Each cluster C_i is subdivided into n segments $\{S_{i,1}, \cdots, S_{i,n}\}$ through random uniform partitioning during offline processing. The index for each cluster has an extra data structure which stores the maximum weight contribution of each term from each segment within this cluster. During retrieval, the maximum and average segment bounds of each cluster C_i are computed as shown below:

$$MaxSBound(C_i) = \max_{j=1}^{n} B_{i,j}, \tag{2}$$

$$AvgSBound(C_i) = \frac{1}{n} \sum_{j=1}^{n} B_{i,j}, \tag{3}$$

$$\text{and } B_{i,j} = \sum_{t \in Q} \max_{d \in S_{i,j}} w_{t,d}.$$

Two-level pruning conditions. Let θ be the current top-k threshold of retrieval in handling query Q.

- **Cluster-level:** Any cluster C_i is pruned when

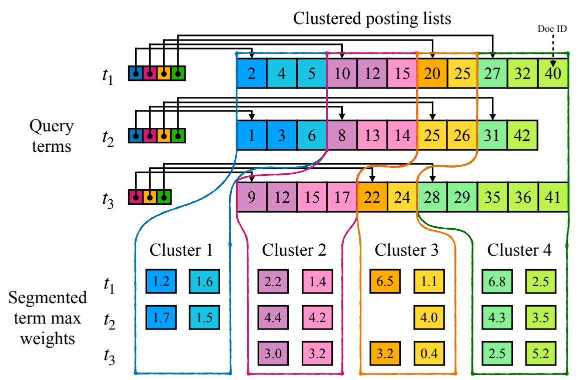
$$MaxSBound(C_i) \leq \frac{\theta}{\mu} \tag{4}$$

  and

$$AvgSBound(C_i) \leq \frac{\theta}{\eta}. \tag{5}$$

- **Document-level:** If a cluster is not pruned, then when visiting such a cluster with a MaxScore or another retrieval algorithm, a document d is pruned if $Bound(d) \leq \frac{\theta}{\eta}$.

Figure 3(a) illustrates a cluster skipping index of four clusters for handling query terms t_1, t_2, and t_3. This index is extended to include two maximum term weight segments per cluster for ASC and these weights are marked in a different color for different segments. Document term weights in posting records are not shown. Assume that the current top-k threshold θ is 9, Figure 3(b) lists the cluster-level pruning decision by Anytime Ranking without and with threshold overestimation and by ASC. The derived bound information used for making pruning decisions is also illustrated.



(a) Cluster skipping index with 2 weight segments per cluster

$\theta = 9$	Custer 1	Cluster 2	Cluster 3	Cluster 4
$BoundSum$	3.3	9.8	13.7	16.3
Anytime	Pruned	Kept	Kept	Kept
Anytime-μ=0.9	Pruned	Pruned	Kept	Kept
$MaxSBound$	3.1	9.6	9.7	13.6
$AvgSBound$	3.0	9.2	7.6	12.4
ASC μ=0.9, η=1	Pruned	Kept	Pruned	Kept

(b) Decisions of dynamic cluster-level pruning during retrieval

Figure 3: A cluster pruning example

Extra online space cost for segmented maximum weights. The extra space cost in ASC is to maintain non-zero maximum term weights for multiple segments at each cluster in a sparse format. For example, Figure 3 shows four non-zero maximum segment term weights at Cluster 1 are accessed for the given query. To save space, we use the quantized value. Our evaluation uses 1 byte for each weight, which is sufficiently accurate to guide pruning. For MS MARCO passages in our evaluation, the default configuration has 4096 clusters and 8 segments per cluster. This results in about 550MB extra space. With that, the total cluster-based inverted SPLADE index size increases from about 5.6GB for MaxScore without clustering to 6.2GB for ASC. This 9% space overhead is still acceptable in practice. The extra space overhead for Anytime Ranking is smaller because only cluster-level maximum term weights are needed.

3.2 Formal Properties

With any integer $0 < k' \leq k$, we call a retrieval algorithm (μ, η)-*approximate* if 1) 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; and 2) the *expected* average rank score of any top k' results produced by this algorithm is competitive to that of rank-safe retrieval within a factor of η. When choosing $\eta = 1$, we call a (μ, η)-approximate retrieval algorithm to be *probabilistically safe*. ASC satisfies the above condition and Theorem 4 gives more details. The default setting of ASC uses $\eta = 1$ in Section 4. The theorems on properties of ASC are listed below and Appendix A lists the proofs. We show that Theorem 3 is also true for Anytime Ranking with threshold overestimation and without early termination and we denote it as Anytime-μ.

Theorem 1

$$BoundSum(C_i) \geq MaxSBound(C_i) \geq \max_{d \in C_i} RankScore(d).$$

The above result shows that Formula (2) provides a tighter upperbound estimation than Formula (1) as demonstrated by Figure 2.

In ASC, choosing a small μ value prunes clusters more aggressively, and having the extra safeness condition using the average segment bound with η counteracts such pruning decisions. Given the requirement $\mu \leq \eta$, we can choose η to be close to 1 or exactly 1 for being safer. When the average segment bound is close to their maximum bound in a cluster, this cluster may not be pruned by ASC. This is characterized by the following property.

Theorem 2 *Cluster-level pruning in ASC does not occur to cluster C_i when one of the two following conditions is true:*

- $MaxSBound(C_i) > \frac{\theta}{\mu}$
- $MaxSBound(C_i) - AvgSBound(C_i) \leq \left(\frac{1}{\mu} - \frac{1}{\eta}\right)\theta.$

The difference between the maximum and average segment bounds provides an approximate indication of the estimated bound tightness. The value of this heuristic is demonstrated in Figure 4, which shows the correlation between bound tightness and the ratio of $AvgSBound(C_i)$ to $MaxSBound(C_i)$ for all clusters. The data is from the MS MARCO Passage dataset with 4096 clusters and 8 segments per cluster. Figure 4 shows that when this ratio approaches 1, the average bound tightness increases and its variation decreases. By the above theorem, when the gap between $MaxSBound(C_i)$ and $AvgSBound(C_i)$ is small (and thus their ratio is near 1), cluster-level pruning will not occur. Therefore, ASC will not prune clusters that already have high-quality and tight bound estimates. Table 5 will further corroborate the results of Theorem 2: that ASC should not prune clusters when this gap is small.

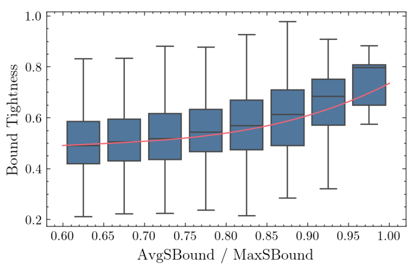


Figure 4: Correlation between the tightness of the estimated bound and the ratio of $AvgSBound$ and $MaxSBound$. As $AvgSBound$ approaches $MaxSBound$, the quality and tightness of the bound increases, and the probability of pruning decreases.

Define $Avg(x, A)$ as the average rank score of the top-x results by algorithm A. Let integer $k' \leq k$. The theorem below characterizes the approximate rank-safeness of pruning in ASC and Anytime-μ.

Theorem 3 *The average top-k' rank score of ASC and Anytime-μ without imposing a time budget is the same as any rank-safe retrieval algorithm R within a factor of μ. Namely $Avg(k', \text{ASC}) \geq \mu Avg(k', R)$, and $Avg(k', \text{Anytime-}\mu) \geq \mu Avg(k', R)$.*

The theorem below characterizes the extra probabilistic approximate rank-safeness of ASC.

Theorem 4 *The average top-k' rank score of ASC achieves the expected value of any rank-safe retrieval algorithm R within a factor of η. Namely $E[Avg(k', \text{ASC})] \geq \eta E[Avg(k', R)]$ where $E[]$ denotes the expected value.*

The probabilistic rank-safeness approximation of ASC relies upon a condition where each document having an equal chance to be in any segment within a cluster. That is true because our segmentation method is random uniform partitioning.

4 Evaluation

Datasets and metrics. We use the MS MARCO Passage ranking dataset (Craswell et al., 2020) with 8.8 million English passages. We report mean reciprocal rank (MRR@10) for the Dev set which contains 6980 queries, and nDCG@10 for the TREC deep learning (DL) 2019 and 2020 sets. We also report recall, which is the percentage of relevant-labeled results that appear in the final top-k results. Retrieval depth k tested is 10 or 1000. We also evaluate on BEIR (Thakur et al., 2021), a collection of 13 publicly available English datasets totaling 24.6 million documents. The size of each dataset ranges from 3,633 to 5.4M documents.

Experimental setup. Documents are clustered using k-means on dense vectors. Details, including a comparison between a few alternatives such as sparse vectors, are in Appendix B.

Sparse models tested include a version of SPLADE (Formal et al., 2021, 2022), uniCOIL (Lin and Ma, 2021; Gao et al., 2021), and LexMAE (Shen et al., 2023). We primarily use SPLADE to assess ASC because LexMAE, following dense models such as SimLM (Xiao et al., 2022) and RetroMAE (Wang et al., 2023), uses MS MARCO title annotations. This is considered as non-standard (Lassance and Clinchant, 2023). SPLADE does not use title annotations.

ASC's implementation uses C++, extended from Anytime Ranking code's release based on the PISA retrieval package (Mallia et al., 2019a). The index is compressed with SIMD-BP128. MaxScore is used to process queries because it is faster than VBMW for long queries (Mallia et al., 2019b; Qiao et al., 2023b) generated by SPLADE and LexMAE. We applied an efficiency optimization to both the ASC and Anytime Ranking code in extracting cluster-based term maximum weights when dealing with a large number of clusters. IOQP uses the authors' code release (Mackenzie et al., 2022b). A comparison to other recent methods in the NeurIPS Big-ANN Competition are presented in Appendix E. All timing results are collected by running as a single thread on a Linux server with Intel i7-1260P and 64GB memory. Before timing queries, all compressed posting lists and metadata for tested queries are pre-loaded into memory, following the common practice. Our code will be released under the Apache License 2.0 after publication.

For all of our experiments on MS MARCO Dev queries, we perform pairwise t-tests on the relevance between ASC and corresponding baselines. "†" is tagged when significant drop is observed from MaxScore retrieval at 95% confidence level.

Baseline comparison on MS MARCO. Table 1 lists the overall comparison of ASC with two baselines using SPLADE model on the MS MARCO Dev and TREC DL'19/20 test sets. Column "Loss" is the percent difference of MRR@10 compared to exact search. Recall@10 and Recall@1000 are reported for retrieval depth $k = 10$ and 1000, respectively. Retrieval mean response time (MRT) and 99th percentile latency (P_{99}) in parentheses are reported in milliseconds. The column marked "C%" is the percentage of clusters that are not pruned during retrieval. For the original rank-safe MaxScore without clustering, we have incorporated document reordering (Mackenzie et al., 2021) to optimize its index based on document similarity, which shortens its latency by about 10-15%.

Anytime Ranking is configured to use 512 clusters with no early termination. ASC is configured with 4096 clusters and 8 segments. Appendix C explains the above cluster configuration for Anytime and ASC to deliver low latency under competitive relevance. Rank-safe ASC uses $\mu = \eta = 1$ and rank-unsafe ASC uses $\eta = 1$ with $\mu = 0.9$ for $k = 10$ and $\mu = 0.5$ for $k = 1000$. As shown in Table 1, these choices yield a tiny MRR@10 loss ratio. For Anytime-μ with over-estimation, we choose the same or higher μ value as ASC to demonstrate ASC improves relevance while gaining the speedup under such a setting.

Comparing the three rank-safe versions in Table 1, ASC is about 2.9x faster than Anytime for $k = 10$, and 1.5x faster for $k = 1000$, because segmentation offers a tighter cluster bound as shown in Theorem 1. ASC is 29x faster than IOQP with $k = 10$. Safe IOQP is substantially slower than

Table 1: A comparison with baselines using SPLADE on MS MARCO passages. No time budget

Methods	C%	MS MARCO Dev MRR (Loss)	Recall	MRT (P_{99})	Speedup	DL'19 nDCG (Recall)	DL'20 nDCG (Recall)
		Retrieval depth $k = 10$					
Exact Search							
IOQP	-	0.3966	0.6824	207 (461)	29x	0.7398 (.1764)	0.7340 (.2462)
MaxScore	-	0.3966	0.6824	26.4 (116)	3.7x	0.7398 (.1764)	0.7340 (.2462)
Anytime Ranking	69.8%	0.3966	0.6824	20.7 (89.3)	2.9x	0.7398 (.1764)	0.7340 (.2462)
ASC	49.1%	0.3966	0.6824	7.19 (26.7)	-	0.7398 (.1764)	0.7340 (.2462)
Approximate							
IOQP-10%	-	0.3782† (4.6%)	0.6541†	24.0 (52.2)	4.3x	0.7381 (.1781)	0.7047 (.2350)
Anytime-μ=0.9	62.7%	0.3815† (3.8%)	0.6111†	15.3 (61.1)	2.7x	0.7392 (.1775)	0.7126 (.2382)
ASC-μ=0.9, η=1	7.99%	0.3964 (0.05%)	0.6813	**5.59 (18.7)**	-	0.7403 (.1764)	0.7338 (.2464)
		Retrieval depth $k = 1000$					
Exact Search							
IOQP	-	0.3966	0.9802	214 (465)	6.4x	0.7398 (.8207)	0.7340 (.8221)
MaxScore	-	0.3966	0.9802	65.8 (209)	2.0x	0.7398 (.8207)	0.7340 (.8221)
Anytime Ranking	93.0%	0.3966	0.9802	50.1 (158)	1.5x	0.7398 (.8207)	0.7340 (.8221)
ASC	54.3%	0.3966	0.9802	33.5 (103)	-	0.7398 (.8207)	0.7340 (.8221)
Approximate							
IOQP-10%	-	0.3782† (4.6%)	0.9746	24.4 (53.1)	1.5x	0.7381 (.8124)	0.7047 (.8081)
Anytime-$\mu = 0.7$	88.9%	0.3963 (0.07%)	0.9696†	37.1 (127)	2.3x	0.7398 (.7881)	0.7340 (.7937)
ASC-μ=0.7, η=1	21.7%	0.3966 (0.0%)	0.9799	25.4 (78.8)	1.6x	0.7398 (.8188)	0.7340 (.8218)
ASC-μ=0.5, η=1	8.10%	0.3962 (0.1%)	0.9739	**15.8 (48.2)**	-	0.7398 (.7977)	0.7355 (.7989)

Anytime, which differs from the finding of (Mallia et al., 2024), possibly because of the difference in data clustering and SPLADE versions.

For approximate retrieval when $k = 10$, ASC has 3.9% higher MRR@10, 11% higher recall, and is 2.7x faster than Anytime with $\mu = 0.9$. When $k = 1000$, ASC is 2.3x faster than Anytime under similar relevance. Even with μ being as low as 0.5, ASC offers competitive relevance scores. This demonstrates the importance of Theorem 4. For this reason, ASC is configured to be probabilistically safe with $\eta = 1$ while choosing μ value modestly below 1 for efficiency. For $k = 10$, there is a very small MRR loss ($\leq 0.1\%$) compared to the original retrieval, but ASC performs competitively while it is up to 4.7x faster than the original MaxScore without using clusters. Approximate IOQP is configured to visit 10% of documents, which is a default choice in (Mackenzie et al., 2022b). ASC outperforms IOQP-10% with 4.8% higher MRR@10 and 3.7% higher recall while ASC is 4.3x faster.

Table 2: Performance at a fixed MRR@10 loss. $k = 10$

MRR Loss	10%	5%	2%	1%	0.5%
Anytime-μ	15ms (7.8x)	16 (5.9x)	17 (4.4x)	18 (3.9x)	19 (4.0x)
	Re: 0.5412	0.5921	0.6287	0.6570	0.6682
IOQP	12ms (6.3x)	22 (8.1x)	55 (14x)	90 (20x)	153 (33x)
	Re: 0.6271	0.6548	0.6741	0.6775	0.6782
ASC	1.9ms (–)	2.7 (–)	3.9 (–)	4.4 (–)	4.7 (–)
	Re: 0.5878	0.6315	0.6639	0.6707	0.6759

Table 2 compares latency in milliseconds and Recall@10 of approximate retrieval under a different and fixed MRR@10 loss compared to rank-safe retrieval with 0.3966 MRR@10 and 0.6824 Recall@10. Rows marked with "Re" list Recall@10 of approximate search. To meet the relevance budget under each fixed MRR loss ratio, we vary μ for ASC and Anytime, and the percent of documents visited for IOQP to minimize latency. The results show that when the MRR loss is controlled within 1-2%, ASC is about 4x faster than Anytime and is 13x to 33x faster than IOQP.

Table 3: Other learned sparse retrieval models

Methods	uniCOIL MRR (Re)	MRT	LexMAE MRR (Re)	MRT
	Retrieval depth $k = 10$. No time budget			
Exact Search				
IOQP	0.352 (.617)	81	0.425 (.718)	163
MaxScore	0.352 (.617)	6.0	0.425 (.718)	47
Anytime	0.352 (.617)	5.0	0.425 (.718)	27
ASC	0.352 (.617)	1.8	0.425 (.718)	12
Approximate				
IOQP-10%	0.320† (.568†)	11	0.405† (.693†)	18
Anytime-μ=0.9	0.345† (.585†)	4.2	0.413† (.654†)	22
ASC-μ=0.9, η=1	0.352 (.614)	**1.4**	0.425 (.718)	**9.7**
	Retrieval depth $k = 1000$. No time budget			
Exact Search				
IOQP	0.352 (.958)	82	0.425 (.988)	165
MaxScore	0.352 (.958)	19	0.425 (.988)	94
Anytime	0.352 (.958)	14	0.425 (.988)	67
ASC	0.352 (.958)	8.8	0.425 (.988)	49
Approximate				
IOQP-10%	0.320† (.937†)	12	0.405† (.985)	**20**
Anytime-μ=0.7	0.351 (.940†)	8.9	0.425 (.978)	46
ASC-μ=0.5, η=1	0.351 (.946)	**4.0**	0.425 (.980)	21

Table 3 applies ASC to uniCOIL and LexMAE and shows MRR@10, Recall@10 or @1000 (denoted as "Re"), and latency (denoted as MRT). The conclusions are similar as the ones obtained above for SPLADE.

Table 4: Zero-shot performance with SPLADE on BEIR

Dataset	MaxScore nDCG	MaxScore MRT	Anytime-μ =0.9 nDCG	Anytime-μ =0.9 MRT	ASC nDCG	ASC MRT
Retrieval depth $k = 10$						
DBPedia	0.443	81.2	0.431	58.1	0.442	40.7
FiQA	0.358	3.64	0.356	2.49	0.358	1.86
NQ	0.555	44.9	0.545	39.8	0.549	18.2
HotpotQA	0.682	323	0.674	270	0.680	158
NFCorpus	0.352	0.17	0.350	0.15	0.352	0.15
T-COVID	0.719	5.20	0.673	2.48	0.719	2.23
Touche-2020	0.307	4.73	0.281	2.27	0.307	1.83
ArguAna	0.432	9.07	0.411	9.17	0.432	8.27
C-FEVER	0.243	895	0.242	735	0.243	555
FEVER	0.786	694	0.782	587	0.786	372
Quora	0.806	5.16	0.795	2.05	0.806	1.53
SCIDOCS	0.151	2.53	0.150	2.17	0.151	1.96
SciFact	0.676	2.54	0.673	2.45	0.676	2.31
Average	0.501	1.91x	0.490	1.35x	0.501	-
Retrieval depth $k = 1000$						
Average	0.501	3.25x	0.498	1.95x	0.499	-

Zero-shot out-of-domain retrieval. Table 4 shows average nDCG@10 and latency in milliseconds for 13 BEIR datasets. SPLADE training is only based on MS MARCO passages. For smaller datasets, the number of clusters is proportionally reduced so that each cluster contains approximately 2000 documents, which is aligned with 4096 clusters setup for MS MARCO. The number of segments is kept at 8. ASC has $\eta = 1$, and its $\mu = 0.9$ for $k = 10$ and $\mu = 0.5$ for $k = 1000$. We use $\mu = 0.9$ for Anytime Ranking without early termination. LexMAE has slightly lower average nDCG@10 0.495, and is omitted due to the page limit.

ASC offers nDCG@10 similar as MaxScore while being 1.91x faster for $k = 10$ and 3.25x faster for $k = 1000$. Comparing with Anytime, ASC is 1.35x faster and has 2.2% higher nDCG@10 on average for $k = 10$, and it is 1.95x faster while maintaining similar relevance scores for $k = 1000$.

Table 5: K-means segmentation vs. random uniform

k=1000 μ, η	K-means MRR (Re)	K-means T	Random MRR (Re)	Random T
0.3, 1	0.393 (.939†)	9.92	0.396 (.972)	15.3
0.4, 1	0.393 (.942†)	10.5	0.396 (.972)	15.4
0.5, 1	0.395 (.959†)	13.8	0.396 (.974)	15.8
0.6, 1	0.397 (.977)	18.1	0.397 (.979)	17.2
0.7, 1	0.397 (.980)	24.4	0.397 (.980)	21.7
1, 1	0.397 (.980)	34.8	0.397 (.980)	33.5

	Bound Tightness	$\frac{MaxSbound - AvgSBound}{Actual}$
Random	0.55	0.49
K-means	0.53	0.69

Segmentation choices. ASC uses random even partitioning to segment term weights of each cluster and satisfy the probabilistic safeness condition that each document in a cluster has an equal chance to appear in any segment. Another approach is to use k-means sub-clustering based on document similarity. The top portion of Table 5 shows random uniform partitioning is more effective than k-means when running SPLADE on MS MARCO passages with 4098 clusters and 8 segments per cluster. Random uniform partitioning offers equal or better relevance in terms of MRR@10 and Recall@1000, especially when μ is small. As μ affects cluster-level pruning in ASC, random segmentation results in a better prevention of incorrect aggressive pruning, although this can result in less cluster-level pruning and a longer latency. To explain the above result, the lower portion of Table 5 shows the estimated bound tightness (ratio of actual bound to $MaxSBound$), and average difference of $MaxSBound$ and $AvgSBound$ scaled by the actual bound. Random uniform partitioning gives slightly better cluster bound estimation, while its average difference of $MaxSBound$ and $AvgSBound$ is much smaller than k-means sub-clustering. Then, when μ is small, there are more un-skipped clusters, following Theorem 2.

The above result also indicates cluster-level pruning in ASC becomes safer due to its adaptiveness to the gap between the maximum and average segment bounds, which is consistent with Theorem 2. The advantage of random uniform partitioning shown above corroborates with Theorem 4 and demonstrates the usefulness of possessing probabilistic approximate rank-safeness.

5 Concluding Remarks

ASC is an (μ, η)-approximate control scheme for dynamic threshold-driven pruning that aggressively skips clusters while being probabilistically safe. ASC can speed up retrieval applications that still desire high relevance effectiveness. For example, when MRR loss is constrained to under 1-2%, the mean latency of ASC is about 4x faster than Anytime Ranking and is 13x to 33x faster than IOQP for MS MARCO Passage Dev set with $k = 10$.

Our evaluations with the MS MARCO and BEIR datasets show that $\mu = 0.5$ for $k = 1000$, and $\mu = 0.9$ for $k = 10$ are good choices with $\eta = 1$ to retain high relevance effectiveness. Our findings recommend $\eta = 1$ for probabilistic safeness and varying μ from 1 to 0.5 for a tradeoff between efficiency and effectiveness.

6 Limitations

Space overhead. There is a manageable space overhead for storing cluster-wise segmented maximum weights. Increasing the number of clusters for a given dataset is useful to reduce ASC latency up to a point, but then the overhead of additional clusters leads to diminishing returns.

Dense retrieval baselines and GPUs. This paper does not compare ASC to dense retrieval baselines because dense models represent a different category of retrieval techniques. ASC achieves up to 0.4252 MRR@10 with LexMAE for MS MARCO Dev, which is close to the highest number 0.4258 obtained in state-of-the-art BERT-based dense retrievers (Xiao et al., 2022; Wang et al., 2023; Liu et al., 2023). The zero-shot performance of ASC with SPLADE on BEIR performs better than these dense models. The above dense model studies use expensive GPUs to reach their full relevance effectiveness. Approximate nearest neighbor search techniques of dense retrieval have been developed following IVF cluster search (Johnson et al., 2019) and graph navigation with HNSW (Malkov and Yashunin, 2020). But there is a significant MRR@10 drop using these approximation techniques.

Although GPUs are readily available, they are expensive and more energy-intensive than CPUs. For example, AWS EC2 charges one to two orders of magnitude more for an advanced GPU instance than a CPU instance with similar memory capacity. Like other sparse retrieval studies, our evaluation is conducted on CPU servers.

Code implementation choice and block-based pruning. Our evaluation uses MaxScore instead of VBMW because MaxScore was shown to be faster for relatively longer queries (Mallia et al., 2019b; Qiao et al., 2023b), which fits in the case of SPLADE and LexMAE under the tested retrieval depths. A previous study (Mallia et al., 2021b) confirms live block filtering with MaxScore called Range-MaxScore is a strong choice for such cases. It can be interesting to examine the use of different base retriever methods in different settings within each cluster for ASC in the future.

Instead of the live block filtering code, ASC implementation was extended from Anytime Ranking's code because of its features that support dynamic cluster ordering and early termination. ASC's techniques can be applied to the framework of contemporary BMP (Mallia et al., 2024) to improve block max estimation and add a probabilistic guarantee for its threshold-driven block pruning. Alternatively, the techniques introduced in BMP, such as partial block (cluster) sorting and hybrid cluster structure with a forward index could also improve our code implementation.

References

Big-ANN. 2024. Neurips'23 competition track: https://big-ann-benchmarks.com/neurips23.html.

Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proc. of the 12th ACM International Conference on Information and Knowledge Management*, pages 426–434.

Sebastian Bruch, Franco Maria Nardini, Amir Ingber, and Edo Liberty. 2024. Bridging dense and sparse maximum inner product search. *ACM Trans. Inf. Syst.*

Fazli Can, Ismail Altingövde, and Engin Demir. 2004. Efficiency and effectiveness of query processing in cluster-based retrieval. *Information Systems*, 29:697–717.

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*, WSDM '17, pages 201–210, New York, NY, USA. ACM.

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.

Zhuyun Dai and Jamie Callan. 2020. Context-aware term weighting for first stage passage retrieval. *SIGIR*.

Sayak Dey, Swagatam Das, and Rammohan Mallipeddi. 2020. The sparse minmax k-means algorithm for high-dimensional clustering. In *Proc. of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2103–2110.

Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. 2013. A candidate filtering mechanism for fast top-k query processing on modern cpus. In *Proc. of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 723–732.

Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In *Proc. of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 993–1002.

Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From distillation to hard negative sampling: Making sparse neural ir models more effective. *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse lexical and expansion model for first stage ranking. *SIGIR*.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. COIL: revisit exact lexical match in information retrieval with contextualized inverted list. *NAACL*.

Fatih Hafizoglu, Emre Can Kucukoglu, and İsmail Sengör Altıngövde. 2017. On the efficiency of selective search. In *ECIR 2017*, volume 10193, pages 705–712.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Trans. on Big Data*, 7(3):535–547.

Karen Spärck Jones, Steve Walker, and Stephen E. Robertson. 2000. A probabilistic model of information retrieval: development and comparative experiments. In *Information Processing and Management*, pages 779–840.

V. Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Yu Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. 2020. Dense passage retrieval for open-domain question answering. *EMNLP'2020*, ArXiv abs/2010.08191.

Yubin Kim, Jamie Callan, J. Shane Culpepper, and Alistair Moffat. 2017. Efficient distributed selective search. *Inf. Retr. J.*, 20(3):221–252.

Anagha Kulkarni and Jamie Callan. 2015. Selective search: Efficient and effective search of large textual collections. *ACM Trans. Inf. Syst.*, 33(4):17:1–17:33.

Hrishikesh Kulkarni, Sean MacAvaney, Nazli Goharian, and Ophir Frieder. 2023. Lexically-accelerated dense retrieval. In *Proc. of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 152–162, New York, NY, USA. Association for Computing Machinery.

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*, pages 2220–2226.

Carlos Lassance and Stephane Clinchant. 2023. The tale of two msmarco - and their unfair comparisons. In *Proceed. of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2431–2435, New York, NY, USA. ACM.

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*, SIGIR '23, page 1771–1775, New York, NY, USA. Association for Computing Machinery.

Jimmy Lin and Andrew Trotman. 2015. Anytime ranking for impact-ordered indexes. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, ICTIR '15, page 301–304, New York, NY, USA. Association for Computing Machinery.

Jimmy J. Lin and Xueguang Ma. 2021. A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques. *ArXiv*, abs/2106.14807.

Zheng Liu, Shitao Xiao, Yingxia Shao, and Zhao Cao. 2023. RetroMAE-2: Duplex masked auto-encoder for pre-training retrieval-oriented language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2635–2648, Toronto, Canada. Association for Computational Linguistics.

Stuart Lloyd. 1982. Least squares quantization in pcm. *IEEE Trans. on Information Theory*, 28(2):129–137.

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*, SIGIR '12, pages 1051–1052, New York, NY, USA. Association for Computing Machinery.

Joel Mackenzie, Antonio Mallia, Alistair Moffat, and Matthias Petri. 2022a. Accelerating learned sparse indexes via term impact decomposition. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2830–2842.

Joel Mackenzie, Matthias Petri, and Luke Gallaghera. 2022b. Ioqp: A simple impact-ordered query processor written in rust. In *Proceedings of DESIRES 2022*.

Joel Mackenzie, Matthias Petri, and Alistair Moffat. 2021. Anytime ranking on document-ordered indexes. *ACM Trans. Inf. Syst.*, 40(1).

Joel Mackenzie, Andrew Trotman, and Jimmy Lin. 2023. Efficient document-at-a-time and score-at-a-time query evaluation for learned sparse representations. *ACM Trans. Inf. Syst.*, 41(4).

Yu A. Malkov and D. A. Yashunin. 2020. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836.

Antonio Mallia, O. Khattab, Nicola Tonellotto, and Torsten Suel. 2021a. Learning passage impacts for inverted indexes. *SIGIR*.

Antonio Mallia, Joel Mackenzie, Torsten Suel, and Nicola Tonellotto. 2022. Faster learned sparse retrieval with guided traversal. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1901–1905.

Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. 2017. Faster blockmax wand with variable-sized blocks. In *Proc. of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 625–634.

Antonio Mallia, Michal Siedlaczek, Joel Mackenzie, and Torsten Suel. 2019a. PISA: Performant indexes and search for academia. *Proceedings of the Open-Source IR Replicability Challenge*.

Antonio Mallia, Michal Siedlaczek, and Torsten Suel. 2019b. An experimental study of index compression and DAAT query processing methods. In *Proc. of 41st European Conference on IR Research, ECIR' 2019*, pages 353–368.

Antonio Mallia, Michał Siedlaczek, and Torsten Suel. 2021b. Fast disjunctive candidate generation using live block filtering. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, page 671–679, New York, NY, USA. Association for Computing Machinery.

Antonio Mallia, Torten Suel, and Nicola Tonellotto. 2024. Faster learned sparse retrieval with block-max pruning. *Preprint*, SIGIR 2024 and arXiv:2405.01117.

Yifan Qiao, Yingrui Yang, Shanxiu He, and Tao Yang. 2023a. Representation sparsification with hybrid thresholding for fast SPLADE-based document retrieval. *ACM SIGIR'23*.

Yifan Qiao, Yingrui Yang, Haixin Lin, and Tao Yang. 2023b. Optimizing guided traversal for fast learned sparse retrieval. In *Proceedings of the ACM Web Conference 2023*, WWW '23, Austin, TX, USA. ACM.

Ruiyang Ren, Yingqi Qu, Jing Liu, Wayne Xin Zhao, QiaoQiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. 2021. RocketQAv2: A joint training method for dense passage retrieval and passage re-ranking. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2825–2835, Online and Punta Cana, Dominican Republic. ACM.

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.

Tao Shen, Xiubo Geng, Chongyang Tao, Can Xu, Xiaolong Huang, Binxing Jiao, Linjun Yang, and Daxin Jiang. 2023. LexMAE: Lexicon-bottlenecked pre-training for large-scale retrieval. In *The Eleventh International Conference on Learning Representations*.

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)*.

Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2013. Efficient and effective retrieval using selective pruning. In *Proc. of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 63–72. ACM.

Howard Turtle and James Flood. 1995. Query evaluation: Strategies and optimizations. *Information Processing & Management*, 31(6):831–850.

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*.

Shitao Xiao, Zheng Liu, Yingxia Shao, and Zhao Cao. 2022. RetroMAE: pre-training retrieval-oriented transformers via masked auto-encoder. *EMNLP*.

Peitian Zhang, Zheng Liu, Shitao Xiao, Zhicheng Dou, and Jing Yao. 2023. Hybrid inverted index is a robust accelerator for dense retrieval. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1877–1888, Singapore. Association for Computational Linguistics.

Zhiyue Zhang, Kenneth Lange, and Jason Xu. 2020. Simple and scalable sparse k-means clustering via feature ranking. In *NeurIPS 20220: Advances in Neural Information Processing Systems*, volume 33, pages 10148–10160.

A Proofs of Formal Properties

Proof of Theorem 1. Without loss of generality, assume in Cluster C_i, the maximum cluster bound $MaxSBound(C_i)$ is the same as the bound of Segment $S_{i,j}$. Then

$$\begin{aligned} MaxSBound(C_i) &= B_{i,j} = \sum_{t \in Q} \max_{d \in S_{i,j}} w_{t,d} \\ &\le \sum_{t \in Q} \max_{d \in C_i} w_{t,d} = BoundSum(C_i). \end{aligned}$$

For any document d, assume it appears in j-th segment of C_i, then

$$\begin{aligned} RankScore(d) &= \sum_{t \in Q} w_{t,d} \le \sum_{t \in Q} \max_{d \in S_{i,j}} w_{t,d} \\ &= B_{i,j} \le MaxSBound(C_i). \end{aligned}$$

$\blacksquare$

Proof of Theorem 2. When a cluster C_i is not pruned by ASC, that is because one of Inequalities

(4) and (5) is false. When Inequality (4) is true but Inequality (5) is false, we have

$$MaxSBound(C_i) \leq \frac{\theta}{\mu} \text{ and } -AvgSBound(C_i) \leq -\frac{\theta}{\eta}.$$

Add these two inequalities together, that proves this theorem.

■

Proof of Theorem 3. Let $L(x)$ be the top-k' list of Algorithm x. To prove $Avg(k', \text{ASC}) \geq \mu Avg(k', R)$, we first remove any document that appears in both $L(\text{ASC})$ and $L(R)$ in both side of the above inequality. Then, we only need to show:

$$\sum_{d \in L(\text{ASC}), d \notin L(R)} RankScore(d) \geq \mu \cdot \sum_{d \in L(R), d \notin L(\text{ASC})} RankScore(d).$$

For the right side of above inequality, if the rank score of every document d in $L(R)$ (but $d \notin L(\text{ASC})$) does not exceed the lowest score in $L(\text{ASC})$ divided by μ, then the above inequality is true. There are two cases to prove this condition.

- Case 1. If d is not pruned by ASC, then d is ranked below k'-th position in ASC.
- Case 2. Document d is pruned by ASC when the top-k threshold is θ_{ASC}. The final top-k threshold when ASC finishes is Θ_{ASC}. If this document d is pruned at the cluster level, then $RankScore(d) \leq \max_{j=1}^{n} B_{i,j} \leq \frac{\theta_{\text{ASC}}}{\mu} \leq \frac{\Theta_{\text{ASC}}}{\mu}$. If it is pruned at the document level, $RankScore(d) \leq \frac{\theta_{\text{ASC}}}{\eta} \leq \frac{\theta_{\text{ASC}}}{\mu} \leq \frac{\Theta_{\text{ASC}}}{\mu}$.

In both cases, $RankScore(d)$ does not exceed the lowest score in $L(\text{ASC})$ divided by μ.

Anytime-μ with no early termination behaves in the same way as ASC with $\mu = \eta$. Thus this theorem is also true for Anytime-μ.

■

Proof of Theorem 4: Define $Top(k', \text{ASC})$ as the score of top k'-th ranked document produced by ASC. $\Theta_{\text{ASC}} = Top(k, \text{ASC})$.

The first part of this proof shows that for any document d such that $d \in L(R)$ and $d \notin L(\text{ASC})$, the following inequality is true:

$$E[RankScore(d)] \leq \frac{Top(k', \text{ASC})}{\eta}.$$

There are two cases that $d \notin L(\text{ASC})$:

- Case 1. If d is not pruned by ASC, then d is ranked below k'-th position in ASC. $RankScore(d) \leq Top(k', \text{ASC})$.
- Case 2. If document d is pruned at the document level by ASC when the top k-th rank score is θ_{ASC},

$$RankScore(d) \leq \frac{\theta_{\text{ASC}}}{\eta} \leq \frac{Top(k, \text{ASC})}{\eta} \leq \frac{Top(k', \text{ASC})}{\eta}.$$

If document d is pruned at the cluster level, notice that ASC uses random uniform partitioning, and thus this document has an equal chance being in any segment within its cluster.

$$E[RankScore(d)] \leq \frac{\sum_{j=1}^{n} B_{i,j}}{n} \leq \frac{\theta_{\text{ASC}}}{\eta} \leq \frac{Top(k, \text{ASC})}{\eta} \leq \frac{Top(k', \text{ASC})}{\eta}.$$

The second part of this proof shows the probabilistic rank-safeness approximation inequality based on the expected average top-k' rank score. Notice that list size $|L(R)| = |L(\text{ASC})| = k'$, and $|L(R) - L(S) \cap L(\text{ASC})| = |L(\text{ASC}) - L(R) \cap L(\text{ASC})|$ where minus notation '$-$' denotes the set subtraction. Using the result of the first part, the following inequality sequence is true:

$$\begin{aligned}
&E[\sum_{d \in L(R)} RankScore(d)] \\
=&E[\sum_{d \in L(R) \cap L(\text{ASC})} RankScore(d)] + E[\sum_{d \in L(R), d \notin L(\text{ASC})} RankScore(d)] \\
\leq &E[\sum_{d \in L(R) \cap L(\text{ASC})} RankScore(d)] + E[\sum_{d \in L(R), d \notin L(\text{ASC})} \frac{Top(k', \text{ASC})}{\eta}] \\
\leq &E[\sum_{d \in L(R) \cap L(\text{ASC})} RankScore(d)] + E[\sum_{d \in L(\text{ASC}), d \notin L(R)} \frac{RankScore(d)}{\eta}] \\
\leq &E[\sum_{d \in L(\text{ASC})} RankScore(d)] \frac{1}{\eta}.
\end{aligned}$$

Thus $E[Avg(k', \text{ASC})] \geq \eta E[Avg(k', R)]$.

■

B Clustering Choices

In this section, we provide a comparison between different clustering methods for ASC. We assume that a learned sparse representation is produced from a trained transformer encoder T. For example, SPLADE (Formal et al., 2021, 2022) and LexMAE (Shen et al., 2023) provide a trained BERT transformer to encode a document and a query. There are two approaches to represent documents for clustering:

- **K-means clustering of sparse vectors.** Encoder T is applied to each document in a data collection to produce a sparse weighted vector. Similar as Anytime Ranking (Mackenzie et al., 2021), we follow the approach of (Kulkarni and Callan, 2015; Kim et al., 2017) to apply the Lloyd's k-means clustering (Lloyd, 1982). Naively applying the k-means algorithm to the clustering of learned sparse vectors presents a challenge owing to their high dimensionality and a large number of sparse vectors as the dataset size scales. For example, each sparse SPLADE document vector is of dimension 30,522 although most elements are zero. Despite its efficacy and widespread use, the k-means algorithm is known to deteriorate when the dimensionality grows. Previous work on sparse k-means has addressed that with feature selection and dimension reduction (Zhang et al., 2020; Dey et al., 2020). These studies explored dataset sizes much smaller than our context and with different applications. Thus our retrieval application demands new considerations. Another difficulty is a lack of efficient implementations for sparse k-means in dealing with large datasets. We address the above challenge below by taking advantage of the dense vector representation produced by the transformer encoder as counterparts corresponding to their sparse vectors, with a much smaller dimensionality.

- **K-means clustering of dense vector counterparts.** Assuming this trained transformer T is BERT, we apply T to each document and produce a token embedding set $\{t_1, t_2, \cdots, t_L\}$ and a CLS token vector. Here t_i is the BERT output embedding of i-th token in this document and L is the total number of tokens of this document. Then, we have three ways to produce a dense vector of each document for clustering.
  - The CLS token vector.
  - The element-wise maximum pooling of all output token vectors. The i-th entry of this dense vector is $\max_{j=1}^{L} t_{i,j}$ where $t_{i,j}$ is the i-th entry of j-th token embedding.
  - The element-wise mean pooling of all output token vectors. The i-th entry of this dense vector is $\frac{1}{L}\sum_{j=1}^{L} t_{i,j}$ where $t_{i,j}$ is the i-th entry of j-th token embedding.

  In addition to the above options, we have compared the use of a dense representation based on SimLM (Wang et al., 2023), a state-of-the-art dense retrieval model.

Table 6 compares the performance of these five vector representations for k-means clustering for ASC. Results are shown with and without segmentation in a safe mode ($\mu = \eta = 1$) for SPLADE-based sparse retrieval on MS MARCO with 4096 clusters and 8 segments per cluster. The column marked "%C" shows the percentage of clusters that are not pruned during ASC retrieval, and MRT is the mean response time in milliseconds. All vectors are clustered using the FAISS library (Johnson et al., 2019) which provides an efficient k-means clustering implementation. Sparse vectors are clustered based on a sample of 100,000 documents because of their high dimensionality. Our results show that maximum pooling of SPLADE-based dense token vectors and direct clustering of the sparse SPLADE vectors have a similar latency and outperform the other three options. Considering the accuracy and implementation challenge in clustering high-dimension sparse vectors, our evaluation chooses max-pooled dense vectors derived from the corresponding transformer model.

Table 6: K-means clustering of MS MARCO passages for safe ASC ($\mu = \eta = 1$) with SPLADE sparse model

	w/o segmt.		w/ segmt.	
Passage representation	**MRT**	**%C**	**MRT**	**%C**
Sparse-SPLADE	55.9	67%	35.6	53%
Dense-SPLADE-CLS	68.2	80%	41.6	64%
Dense-SPLADE-Avg	56.3	76%	37.3	58%
Dense-SPLADE-Max	54.1	68%	33.5	54%
Dense-SimLM-CLS	63.3	78%	40.1	60%

C Impact of varying #clusters for Anytime Ranking and ASC

Figure 5 shows the latency of Anytime and ASC for $k = 10$ with safe pruning and a similar trend is seen for $k = 1000$. Table 7 shows their performance with threshold over-estimation ($\mu = 0.9$). We present latency results for two versions of Anytime Ranking. The original Anytime, with its latency denoted as "Orig.", becomes significantly slower as the number of clusters increase. Therefore, we added an optimization (denoted as "Opt.") in extracting cluster maximum weights as noted in Section 4. The fastest configuration for Anytime Ranking is with 512 clusters. Lowering the number of clusters to a smaller number such as 256 or 128 increases Anytime's latency because the maximum cluster bound estimation becomes less accurate.

The above result shows that ASC performs better with 4096 clusters when varying the number of clusters from 128 to 4096 when $k = 10$. We do not use a larger number of clusters because that

Table 7: Performance of Anytime Ranking vs. ASC when varying #clusters for threshold overestimation. $k = 10$.

Cluster Count	Anytime Ranking $\mu = 0.9$ MRR (Re)	Orig.	Opt.	ASC $\mu = 0.9$, $\eta = 1$ MRR (Re)	MRT
128	0.381 (0.604)	16.8	16.0	0.397 (0.682)	14.0
256	0.380 (0.607)	16.5	15.6	0.397 (0.682)	13.5
512	0.382 (0.611)	16.3	15.3	0.397 (0.682)	10.5
1024	0.380 (0.611)	20.0	17.8	0.396 (0.681)	7.41
2048	0.384 (0.615)	29.1	20.4	0.396 (0.681)	6.05
4096	0.381 (0.611)	53.2	**24.1**	0.396 (0.681)	**5.59**

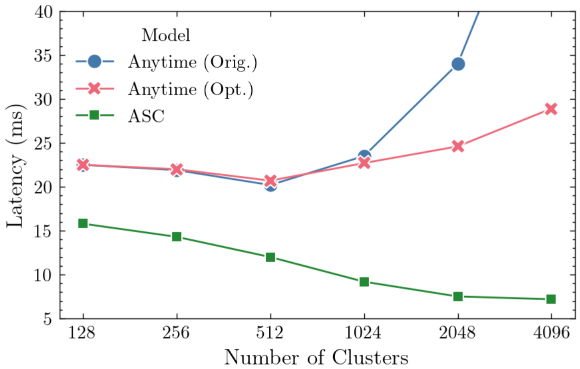


Figure 5: The effect of the number of clusters on latency. For Anytime (Orig.) and Anytime (Opt.), latency grows significantly with clusters. ASC is the fastest method for all clusters and exhibits the slowest growth in latency of all methods.

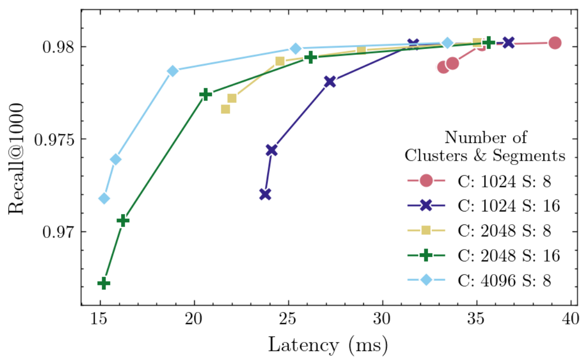


Figure 6: Recall vs. latency of ASC (η=1) for varying values of μ at retrieval depth $k = 1000$. For each fixed number of clusters and segments, μ varies from 0.4, 0.5, 0.6, 0.7, to 1.

Table 8: Anytime vs. ASC (η=1) with time budgets

Model	Setup	MRR (Re)	MRT (P_{99})
Retrieval depth $k = 10$. Time budget 10ms			
SPLADE	Anytime-$\mu = 1$	0.370† (.632†)	8.34 (10.3)
	ASC-$\mu = 1$	0.395 (.679)	5.14 (10.1)
	Anytime-$\mu = 0.9$	0.360† (.575†)	7.70 (10.2)
	ASC-$\mu = 0.9$	0.395 (.678)	4.21 (10.0)
LexMAE	ASC-$\mu = 0.9$	0.423 (.713)	5.14 (10.2)
Retrieval depth $k = 1000$. Time budget 20ms			
SPLADE	Anytime-$\mu = 1$	0.364† (.865†)	19.1 (20.4)
	ASC-$\mu = 1$	0.395 (.973)	18.2 (20.1)
	Anytime-$\mu = 0.9$	0.363† (.864†)	19.1 (20.3)
	ASC-$\mu = 0.7$	0.395 (.973)	15.2 (20.0)
LexMAE	ASC-$\mu = 0.7$	0.423 (.974†)	16.9 (20.1)

increases the space overhead for ASC. The finding is similar for different choices of μ and for $k = 1000$. Figure 6 examines the relation of Recall@1000 and latency for ASC when varying μ under different numbers of clusters and segments. Each curve represents a distinct number of clusters and number of segments per cluster. Each curve has 5 markers from left to right, denoting $\mu = 0.4$, 0.5, 0.6, 0.7, and 1, respectively. A greater number of clusters improves cluster bound estimation and allows finer-grained pruning decisions, however it also introduces additional overhead for visiting each cluster, as discussed in Section 3. This figure shows that the best configuration of ASC is 4096 clusters and 8 segments per cluster for all values of μ.

D Compatibility with other speedup techniques

Table 8 lists MRR@10 and Recall@1000 of combining ASC with early termination technique of Anytime Ranking (Mackenzie et al., 2021) under a time budget on MS MARCO Dev set for SPLADE mainly. Last row lists ASC performance with LexMAE for each k value. 512 clusters are configured for Anytime Ranking, and "4096 clusters*8 segments" are for ASC. Comparing to Table 1, there is a small relevance degradation for ASC with time budgets, but the 99th percentile time is improved substantially by this combination. Under the same time budget, this ASC/Anytime combination has higher MRR@10 and Recall@1000 than Anytime Ranking alone in both retrieval depths.

We also apply ASC with static index pruning (Qiao et al., 2023a) for a version of SPLADE used in Big-ANN competition as discussed in Appendix E below. The exact search with safe Anytime Ranking delivers 0.383 MRR@10 with 20.2ms with $k = 10$. ASC takes 3.8ms with 0.5%

MRR loss, and it only takes 0.81ms when following the Big-ANN relevance budget (90.5% recall to top-10 exact search results).

Term impact decomposition (Mackenzie et al., 2022a) is an orthogonal optimization on posting lists. Our preliminary test shows that it does not work well with SPLADE as its posting clipping and list splitting increase original SPLADE latency from 66ms to 95ms and 110ms, respectively. Thus our evaluation didn't include this optimization.

E Comparison to NeurIPS '23 Big-ANN Methods

The sparse track of NeurIPS 2023 competition for fast approximate nearest neighbor search (Bi-ANN) (Big-ANN, 2024) uses 90% recall of top 10 result of the exact search baseline as the relevance budget to select the fastest entry for MS MARCO dev set. The SPLADE version used in the Big-ANN competition has 0.383 MRR@10, which is different than our version with 0.3966. Top entries in Big-ANN can use any range of techniques, including unpublished optimizations or specialization. On the other hand, this paper is focused on a single optimization topic solved with general techniques, namely improving threshold-driven pruning based on cluster rank score bounds. Thus the purpose of this evaluation study is to demonstrate how ASC can make cluster-based retrieval competitive for the Big-ANN setting.

We compare ASC with the two best open-source submissions: PyANNS and SHNSW. The Sparse track measures relative recall against top 10 exact search and throughput with eight simultaneous threads. To follow a common practice, Table 9 reports reciprocal rank (MRR@10), Recall@10, and single-thread latency (MRT) in milliseconds on our machine. Table 9 also reports the recall to top-10 exact search as "R2Exact". The exact search baseline is rank-safe Anytime Ranking with 512 clusters, the same configuration as Section 4.

The Big-ANN competition prioritizes efficiency under a relatively loose approximation loss budget, whereas ASC is designed to preserve pruning safeness while reducing the latency. Thus we configure all models to minimize latency for meeting the following two loss budget settings.

- *Preserve 90% of top-10 exact search*. The best performing parameters were selected from the submitted configurations. For PyANNS $qdrop = 0.1$ and $ef = 60$ and for SHNSW $ef = 52$. For ASC, we use 512 clusters with 16 segments each, $\mu = 0.85, \eta = 1$ after applying static pruning.
- *Preserve 99% of top-10 exact search*. We select the best performing configuration for PyANNS with $qdrop = 0.0$ and $ef = 2000$ and for SHNSW with $ef = 2000$. For ASC, we use 4096 clusters with 8 segments each, $\mu = 0.9, \eta = 1$.

Table 9 shows that ASC is 4.1x to 5.2x faster than PyANNS and SHNSW respectively for the 99% setting while having better MRR@10. Noticeably PyANNS suffers 68% MRR@10 loss. For the 90% setting, ASC is 7% faster and has 0.9% higher MRR@10 than SHNSW. Even though PyANNS is faster than ASC, its MRR@10 loss is over 71%, which is huge.

Table 9: A comparison with BigANN methods using SPLADE on MS MARCO Passage Ranking

Methods	MRR(Recall)	R2Exact	MRT
Preserve 90% of top-10 exact search			
Exact search	0.383 (0.670)	100%	20.2
SHNSW	0.339 (0.601)	90.0%	0.87
PyANNS	0.110 (0.603)	90.3%	**0.48**
ASC	0.342 (0.604)	90.5%	0.81
Preserve 99% of top-10 exact search			
Exact search	0.383 (0.670)	100%	20.2
SHNSW	0.379 (0.665)	99.1%	19.9
PyANNS	0.122 (0.665)	99.1%	15.6
ASC	0.381 (0.667)	99.5%	**3.80**

The above result shows that the competition metric for Big-ANN drives a different optimization tradeoff compared to our paper. This is because our paper prioritizes MRR@10 competitiveness of approximate retrieval with a much tighter relevance loss budget before considering latency reduction gains. Configurations of ASC with unsafe pruning listed in Table 1 of Section 4 are within a 0.1% MRR@10 loss budget for Dev set. Thus while ASC makes a cluster-based retriever more competitive in the Big-ANN tradeoff setting, ASC is designed to speed up retrieval applications that desire high relevance effectiveness.