Approximate Cluster-Based Sparse Document Retrieval with Segmented Maximum Term Weights

Anonymous ACL submission

Abstract

This paper revisits cluster-based sparse retrieval that partitions the inverted index and skips the index partially at cluster and document levels during inference. It proposes an approximate search scheme called ASC with two parameters to control pruning and provide a probabilistic guarantee on rank-safeness competitiveness. ASC uses cluster-level maximum weight segmentation to improve accuracy of bound estimation and threshold-based pruning. The experiments with MS MARCO and BEIR show that ASC delivers strong relevance with a low latency on a single-threaded CPU.

1 Introduction

There are two main categories of approaches for top-k text document retrieval. One is dense retrieval with dual encoders (e.g. (Karpukhin et al., 2020; Ren et al., 2021; Xiao et al., 2022; Wang et al., 2023)), relying on GPUs for fast computation. Approximation techniques for dense retrieval have been developed with a visible relevance drop (Johnson et al., 2019; Malkov and Yashunin, 2020; Kulkarni et al., 2023; Zhang et al., 2023). Another category is lexical sparse retrieval models, such as BM25, which take advantage of fast inverted index implementations on CPUs. The recent popularity of this method can be attributed to advances in learned sparse representations that derive token weights from a BERT-based neural model (Dai and Callan, 2020; Mallia et al., 2021a; Lin and Ma, 2021; Gao et al., 2021; Formal et al., 2021; Shen et al., 2023). 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 with BEIR datasets, learned sparse retrieval exhibits stronger relevance than BERT-based dense models. Additionally, while 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. GPUs are economically and environmentally less appealing for first-stage retrieval of a large-scale search engine which runs index partitions on a massive number of machines. Thus this paper studies online inference efficiency optimization for sparse retrieval on CPUs. Another motivation for this work is that fusion of sparse and dense retrieval (Li et al., 2022; Zhang et al., 2023) improves relevance, which calls for faster but effective sparse retrieval.

A traditional speed optimization for sparse retrieval is dynamic rank-safe index pruning, such as MaxScore (Turtle and Flood, 1995), WAND (Broder et al., 2003), BlockMax WAND (BMW) (Ding and Suel, 2011), and live block filtering (Dimopoulos et al., 2013; Mallia et al., 2021b) which accurately skips the evaluation of low-scoring documents that are unable to appear in the final top-k results. Early work on rank-unsafe pruning includes threshold over-estimation (Macdonald et al., 2012; Tonellotto et al., 2013; Crane et al., 2017) and early termination (Lin and Trotman, 2015). Anytime Ranking (Mackenzie et al., 2021), following the previous cluster-based retrieval studies, organizes posting lists as clusters with cluster-level pruning after dynamic cluster ordering, in addition to early termination optimization. The above rank-unsafe methods can be fast at the cost of a visible relevance drop, and there are no formal guarantees on their relevance safeness.

This paper revisits dynamic index pruning in both safe and unsafe settings for cluster-based retrieval. The contributions of this paper is an approximate search scheme called ASC with two parameters that control pruning with a probabilistic guarantee on rank-safeness competitiveness. ASC uses cluster-level maximum weight segmentation to improve accuracy of bound estimation and threshold-based pruning. This paper treats early termination as orthogonal optimization and will show ASC's compatibility.

Our evaluation shows that ASC delivers strong relevance in running SPLADE (Formal et al., 2021, 2022) and LexMAE (Shen et al., 2023), taking only tens of milliseconds on a single-threaded low-end CPU for MS MARCO passages with up-to 0.425 MRR@10 and 0.988 Recall@1K. It achieves about 0.5 nDCG@10 for BEIR datasets on average. The achieved relevance is much stronger than other approximation baselines while its CPU latency is reasonably fast for interactive query processing.

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-based 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. Live block filtering clusters document IDs within a range and estimates a range-based maximum score for pruning. 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.

Cluster-based retrieval. A *cluster skipping inverted index* (Can et al., 2004; Hafizoglu et al., 2017) arranges each posting list as "clusters" for selective retrieval. Anytime Ranking (Mackenzie et al., 2021) 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. Our work follows and extends the above work while increasing index-skipping opportunities through cluster-level maximum weight segmentation and a probabilistic rank-safeness assurance with a small impact to relevance. ASC improves cluster-level threshold-based pruning without considering early termination.

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.

3 Cluster-based Retrieval with Approximation and Segmentation

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

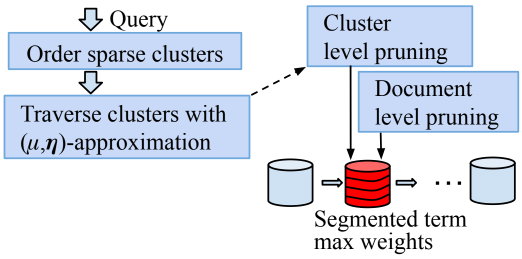


Figure 1: Flow of ASC: approximate retrieval with segmented cluster-level maximum term weights

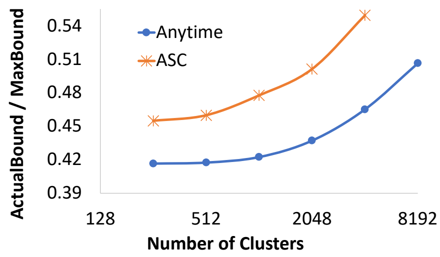


Figure 2: The average ratio of the actual and estimated cluster bounds with Formula (1) on MS MARCO

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.

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

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 bound sum 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 average actual and estimated bound ratio using Formula (1) for MS MARCO passage clusters, which is $\frac{1}{m}\sum_{i=1}^{m} \frac{\max_{d_j \in C_i} RankScore(d_j)}{BoundSum(C_i)}$, where m is the number of clusters. This ratio with value 1 means the bound estimation is accurate, and a small ratio value towards 0 means a loose estimation. This average ratio becomes bigger with a smaller error when m increases with a smaller average cluster size. This figure also plots the improved cluster upper bound computed in ASC described below.

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 condition is modified as $BoundSum(C_i) \leq \frac{\theta}{\mu}$ and $Bound(d) \leq \frac{\theta}{\mu}$. The introduction of threshold over-estimation with μ allows the skipping of more low-scoring documents when the bound estimation is too loose. However, thresholding is applied uniformly to all cases and can incorrectly prune many desired relevant documents when the bound estimation is already tight in some clusters.

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

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

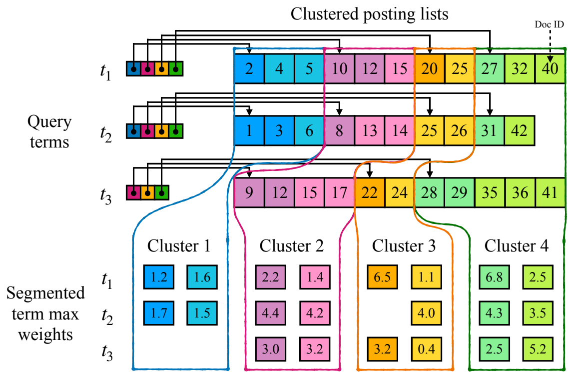
  and

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

- **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 $Bound(d) \le \frac{\theta}{\eta}$.

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.

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



(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

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

We call an algorithm (μ, η)-*approximate* if it is μ-approximate, and it satisfies that the expected average rank score of any top k' results produced by this algorithm, where $k' < k$, 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

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

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:*

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

From the above theorem, when μ is small and/or the gap between $MaxSBound(C_i)$ and $AvgSBound(C_i)$ is small, cluster-level pruning will not occur. This difference of the maximum and average segment bounds provides an approximate indication of the bound estimation tightness with $MaxSBound$, and Figure 4 gives an illustration as to why this difference is a meaningful indicator approximately. Figure 4 depicts the correlation between the average ratio of $AvgSBound(C_i)$ over $MaxSBound(C_i)$ for all clusters, and average ratio of the exact bound over the estimated bound $MaxSBound(C_i)$. The data is collected from the index of MS MARCO dataset with 4096 clusters and 8 segments per cluster. This figure shows that when $AvgSBound(C_i)$ is closer to $MaxSBound(C_i)$ on average, the gap between exact upper bound and $MaxSbound$ value becomes smaller, which means the bound estimation becomes tighter. Table 4 in Section 4 will further corroborate that the above smaller gap yields less cluster skipping opportunities in ASC for safer pruning, consistent with the result of Theorem 2.

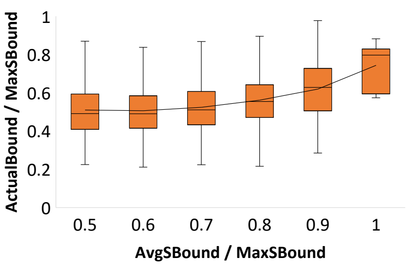


Figure 4: MS MARCO passage clusters. Correlation between bound estimation tightness and average $AvgSBound(C_i)/MaxSBound(C_i)$.

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 MS MARCO ranking dataset (Craswell et al., 2020) with 8.8 million passages in English. We report mean reciprocal rank (MRR@10) for Dev set which contains 6980 queries, and nDCG@10 for 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. We test two retrieval depths: $k = 10$ and $k = 1000$. The second collection is BEIR (Thakur et al., 2021) with 13 publicly available English datasets and a total of 24.6 million documents. The size of each dataset ranges from 3,633 to 5.4M.

Experimental setup. Documents are clustered by the k-means algorithm after comparing a few alternatives with their sparse or dense representations. Details are in Appendix B. We test ASC on a version of SPLADE, uniCOIL (Lin and Ma, 2021; Gao et al., 2021), and LexMAE. We primarily use SPLADE to assess ASC since LexMAE, following dense models such as SimLM (Xiao et al., 2022) and RetroMAE (Wang et al., 2023), uses title annotation on MS MARCO. This is considered to be non-standard in (Lassance and Clinchant, 2023). SPLADE does not use title annotation.

Table 1: A comparison with baselines using SPLADE on MS MARCO passages

Methods	C%	MS MARCO Dev MRR (Recall)	MS MARCO Dev MRT (P_{99})	DL'19 nDCG (Recall)	DL'19 MRT	DL'20 nDCG (Recall)	DL'20 MRT
Retrieval depth $k = 10$. No early termination							
Rank-safe							
MaxScore	-	0.3966 (.6824)	26.4 (116)	0.7398 (.1764)	26.3	0.7340 (.2462)	24.8
- Anytime Ranking	69.8%	0.3966 (.6824)	20.7 (89.3)	0.7398 (.1764)	18.4	0.7340 (.2462)	17.6
- ASC	49.1%	0.3966 (.6824)	15.2 (62.2)	0.7398 (.1764)	15.3	0.7340 (.2462)	14.8
μ vs. (μ, η)-approximate							
- Anytime-μ=0.9	62.7%	$0.3815^{\dagger}$ ($.6111^{\dagger}$)	15.3 (61.1)	0.7392 (.1775)	15.9	0.7126 (.2382)	15.2
- ASC-μ=0.9	7.99%	0.3964 (.6813)	**11.4 (55.9)**	0.7403 (.1764)	**11.6**	0.7338 (.2464)	**11.5**
Retrieval depth $k = 1000$. No early termination							
Rank-safe							
MaxScore	-	0.3966 (.9802)	65.8 (209)	0.7398 (.8207)	67.0	0.7340 (.8221)	63.2
- Anytime Ranking	93.0%	0.3966 (.9802)	50.1 (158)	0.7398 (.8207)	54.3	0.7340 (.8221)	51.1
- ASC	86.3%	0.3966 (.9802)	45.8 (148)	0.7398 (.8207)	49.9	0.7340 (.8221)	46.6
μ vs. (μ, η)-approximate							
- Anytime-$\mu = 0.9$	91.4%	0.3966 (.9801)	46.0 (149)	0.7398 (.8205)	45.1	0.7340 (.8206)	42.8
- ASC-μ=0.7	21.7%	0.3966 (.9799)	38.8 (135)	0.7398 (.8188)	40.5	0.7340 (.8218)	37.3
- Anytime-$\mu = 0.7$	88.9%	0.3963 ($.9696^{\dagger}$)	37.1 (127)	0.7398 (.7881)	37.9	0.7340 (.7937)	36.7
- ASC-μ=0.5	8.10%	0.3962 (.9739)	**21.8 (101)**	0.7398 (.7977)	**22.8**	0.7355 (.7989)	**21.7**

ASC implementation uses C++, extended from Anytime Ranking code release based on PISA retrieval package (Mallia et al., 2019a). Index is compressed with SIMD-BP128. The underlying retrieval method is MaxScore 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 ASC and Anytime Ranking code in extracting cluster-based term maximum weights when dealing with a large number of clusters. 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 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 the MaxScore retrieval at 95% confidence level.

Baseline comparison on MS MARCO. Table 1 lists the overall comparison of ASC with two baselines using SPLADE sparse passage representations on MS MARCO Dev and TREC DL'19/20 test sets. 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. Column marked "C%" is the percentage of clusters that are not pruned during retrieval. For rank-safe original 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. Then we extend it by adding rank-unsafe overestimation with $\mu = 0.9$ or 0.7. These are its best parameter choices for low latency and competitive relevance and a higher number clusters increases its latency significantly without relevance benefit. ASC always has $\eta = 1$. Rank-safe ASC uses 512 clusters with 16 segments and $\mu = 1$. Rank-unsafe ASC uses 4096 clusters and 8 segments with $\mu = 0.9$ for $k = 10$, and $\mu = 0.7$ or 0.5 for $k = 1000$.

Comparing the three rank-safe versions in Table 1, ASC is about 27% faster than Anytime for $k = 10$, and 8.6% faster for $k = 1000$, because segmentation offers a tighter cluster bound as shown in Theorem 1.

For approximate safe configurations when $k = 10$, ASC has 3.9% higher MRR@10, 11% higher recall, and is 25% faster than Anytime with $\mu = 0.9$. When $k = 1000$, ASC is about 1.2-1.7x 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. There is a small relevance degradation compared to the original retrieval, but ASC performs competitively while it is up-to 3.0x faster than the original MaxScore without using clusters.

ASC can skip more than 90% of 4098 clusters, but its latency does not decrease proportionally compared to the 512-cluster setting. This is because increased overhead for dealing with a large number of clusters reduces ASC's benefit.

Table 2: Other learned sparse retrieval models

Methods	uniCOIL MRR (Re)	uniCOIL T	LexMAE MRR (Re)	LexMAE T
Retrieval depth $k = 10$. No early termination				
Rank-safe				
MaxScore	0.352 (.617)	6.0	0.425 (.718)	47
- Anytime	0.352 (.617)	5.0	0.425 (.718)	27
- ASC	0.352 (.617)	4.1	0.425 (.718)	21
μ vs. (μ, η)-approximate				
- Anytime-μ=0.9	0.345† (.585†)	4.2	0.413† (.654†)	22
- ASC-μ=0.9	0.352 (.614)	**3.9**	0.425 (.718)	**16**
Retrieval depth $k = 1000$. No early termination				
Rank-safe				
MaxScore	0.352 (.958)	19	0.425 (.988)	94
- Anytime	0.352 (.958)	14	0.425 (.988)	67
- ASC	0.352 (.958)	13	0.425 (.988)	64
μ vs. (μ, η)-approximate				
- Anytime-μ=0.7	0.351 (.940†)	8.9	0.425 (.978)	46
- ASC-μ=0.5	0.351 (.946)	**6.4**	0.425 (.980)	**26**

Table 2 applies ASC to uniCOIL and LexMAE and shows MRR@10, Recall@10 or @1000 (shortened as "Re"), and latency time (shortened as T). The conclusions are similar as the ones obtained above for SPLADE.

Table 3: 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	50.8
FiQA	0.358	3.64	0.356	2.49	0.358	2.67
NQ	0.555	44.9	0.545	39.8	0.549	25.6
HotpotQA	0.682	323	0.674	270	0.680	260
NFCorpus	0.352	0.17	0.350	0.15	0.352	0.17
T-COVID	0.719	5.20	0.673	2.48	0.719	2.64
Touche-2020	0.307	4.73	0.281	2.27	0.307	2.00
ArguAna	0.432	9.07	0.411	9.17	0.432	9.02
C-FEVER	0.243	895	0.242	735	0.243	738
FEVER	0.786	694	0.782	587	0.786	557
Quora	0.806	5.16	0.795	2.05	0.806	1.73
SCIDOCS	0.151	2.53	0.150	2.17	0.151	2.13
SciFact	0.676	2.54	0.673	2.45	0.676	2.42
Average	0.501	-	0.490	1.43x	0.501	1.54x
Retrieval depth $k = 1000$						
Average	0.501	-	0.498	1.96x	0.499	3.12x

Zero-shot out-of-domain retrieval. Table 3 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 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 1.54x faster for $k = 10$ and 3.12x faster for $k = 1000$. Comparing with Anytime, ASC is 7.7% faster and has 2.2% higher nDCG@10 on average for $k = 10$, and it is 1.59x faster while maintaining similar relevance scores for $k = 1000$.

Table 4: 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†)	11.9	0.396 (.972)	20.7
0.4, 1	0.393 (.942†)	12.6	0.396 (.972)	20.8
0.5, 1	0.395 (.959†)	17.7	0.396 (.974)	21.8
0.6, 1	0.397 (.977)	29.0	0.397 (.979)	27.7
0.7, 1	0.397 (.980)	41.6	0.397 (.980)	38.7
1, 1	0.397 (.980)	69.1	0.397 (.980)	66.6

	$\frac{Actual}{MaxSBound}$	$\frac{MaxSbound - AvgSBound}{Actual}$
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 4 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 4 shows average ratio of actual cluster upper bound over estimated $MaxSBound$, and average difference of $MaxSBound$ and $AvgSBound$ scaled by the actual bound. Random uniform partitioning gives slightly better cluster upper 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.

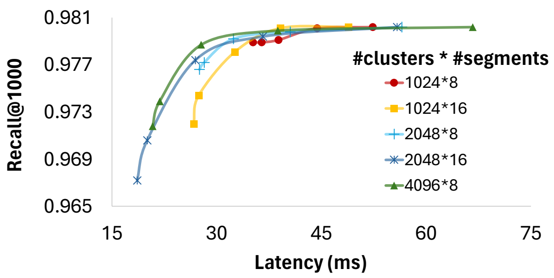


Figure 5: Latency and recall vs. μ for ASC (η=1)

Varying μ, #clusters, and #segments. Figure 5 examines the relation of Recall@1000 of ASC and its latency when varying μ with each curve in a distinct color representing a setting "$m * n$" as m clusters and n segments per cluster. $k = 1000$, and $\eta = 1$ for SPLADE on MS MARCO Dev. Each curve has 5 markers from left to right, denoting $\mu = 0.4$, 0.5, 0.6, 0.7, and 1, respectively. Having more clusters leads to better cluster bound estimation, and finer-grained decisions on pruning, but more cluster oriented overhead that affects latency, as discussed in Section 3. This figure shows that to pursue a shorter latency under a better or equal relevance constraint, ASC should choose 512 clusters for $\mu = 1$, and 4096 clusters for $\mu < 1$.

Table 5: Anytime vs. ASC (η=1) with time budgets

Model	Setup	MRR (Re)	MRT (P_{99})
Retrieval depth $k = 10$. Time budget 10ms			
SPLADE	Anytime-$\mu = 1$	$0.370^{\dagger}$ ($.632^{\dagger}$)	8.34 (10.3)
	ASC-$\mu = 1$	0.395 (.678)	7.31 (10.1)
	Anytime-$\mu = 0.9$	$0.360^{\dagger}$ ($.575^{\dagger}$)	7.70 (10.2)
	ASC-$\mu = 0.9$	0.395 (.678)	6.81 (10.0)
LexMAE	ASC-$\mu = 0.9$	0.421 (.710)	8.35 (10.3)
Retrieval depth $k = 1000$. Time budget 20ms			
SPLADE	Anytime-$\mu = 1$	$0.364^{\dagger}$ ($.865^{\dagger}$)	19.1 (20.4)
	ASC-$\mu = 1$	0.394 ($.966^{\dagger}$)	19.9 (20.1)
	Anytime-$\mu = 0.9$	$0.363^{\dagger}$ ($.864^{\dagger}$)	19.1 (20.3)
	ASC-$\mu = 0.7$	0.395 ($.970^{\dagger}$)	17.0 (20.0)
LexMAE	ASC-$\mu = 0.7$	0.421 ($.968^{\dagger}$)	17.2 (20.1)

Compatibility with other efficiency optimization techniques. Table 5 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 for ASC with $\mu = 1$. "4096 clusters*8 segments" are for ASC with $\mu = 0.7$. 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 to a fast version of SPLADE with static index pruning called HT3 (Qiao et al., 2023a). HT3 has 0.3942 MRR@10 on MS MARCO Dev set with a retrieval latency of 24.7ms for retrieval depth $k = 1000$. ASC configured with "4096*8" and $\mu = 0.5/\eta = 1$ reduces the retrieval latency by 3.3x to 7.43ms, while the relevance slightly degrades to 0.3933 MRR@10.

5 Concluding Remarks

This paper has proposed an approximate sparse retrieval scheme to skip more clusters while being probabilistically competitive in safeness. The (μ, η)-approximation provides more flexible pruning control with a probabilistic guarantee. Our evaluation shows that ASC can be 25% faster for $k = 10$ and 41% faster for k=1000 than Anytime Ranking using SPLADE and MS MARCO Dev while ASC offers similar or even higher relevance scores than Anytime with threshold overestimation. ASC is up-to 3x faster than the original MaxScore algorithm.

Instead of the live block filtering code, ASC implementation was extended from Anytime Ranking's code because of its features to support dynamic cluster ordering and early termination.

ASC is compatible with early termination of Anytime Ranking and has not been tested with other such schemes such as JASS (Lin and Trotman, 2015) and IOQP (Mackenzie et al., 2022b) because Anytime Ranking (Mackenzie et al., 2021) has shown its advantages and competitiveness to other anytime schemes, and early termination optimization is orthogonal. ASC could apply other early termination methods within each cluster.

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.

6 Limitations

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, because more clusters leads to more overhead.

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.

References

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.

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.

Hang Li, Shuai Wang, Shengyao Zhuang, Ahmed Mourad, Xueguang Ma, Jimmy Lin, and G. Zuccon. 2022. To interpolate or not to interpolate: Prf, dense and sparse retrievers. *SIGIR*.

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.

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).

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.

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.

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*.

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

$$MaxSBound(C_i) = B_{i,j} = \sum_{t \in Q} \max_{d \in S_{i,j}} w_{t,d}$$
$$\leq \sum_{t \in Q} \max_{d \in C_i} w_{t,d} = BoundSum(C_i).$$

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

$$RankScore(d) = \sum_{t \in Q} w_{t,d} \leq \sum_{t \in Q} \max_{d \in S_{i,j}} w_{t,d}$$
$$= B_{i,j} \leq MaxSBound(C_i).$$

■

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.

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

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

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: 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	91.6	67%	70.3	53%
Dense-SPLADE-CLS	115	80%	82.7	64%
Dense-SPLADE-Avg	95.3	76%	74.2	58%
Dense-SPLADE-Max	90.8	68%	71.8	54%
Dense-SimLM-CLS	105	78%	78.5	60%

BERT vectors are of dimension 768, and we leverage the FAISS library (Johnson et al., 2019) for dense vector clustering with quantization support, which can compress vectors and further reduce the dimensionality.

Table 6 lists the performance of ASC with and without segmentation in a safe mode ($\mu = \eta = 1$) for SPLADE-based sparse retrieval. It compares the above five different vector representation options to apply k-means clustering. There are 4096 clusters and 8 random segments per cluster. MRT is the mean retrieval time in milliseconds. Column marked with "%C" shows the percentage of clusters that are not pruned during ASC retrieval. For sparse vectors, we leverage FAISS dense k-means implementation with sampling, which is still expensive. Table 6 shows that the maximum pooling of SPLADE-based dense token vectors has a similar latency as the sparse vector representation. These two options are better than 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.