University of California
Santa Barbara

Pruning Optimization for Efficient Top-k Document Retrieval with Learned Sparse Representations

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Yifan Qiao 乔逸凡

Committee in charge:

Professor Tao Yang, Chair
Professor Shiyu Chang
Professor Xifeng Yan

September 2024

The Dissertation of Yifan Qiao 乔逸凡 is approved.

Professor Shiyu Chang

Professor Xifeng Yan

Professor Tao Yang, Committee Chair

August 2024

Pruning Optimization for Efficient Top-k Document Retrieval with Learned Sparse Representations

by

Yifan Qiao 乔逸凡

Dedicated to

My parents, Lihui Chang and Dongsheng Qiao 常丽慧 乔冬生;

and my wife, Ruoyun Tan 檀若云.

Acknowledgements

In the first place, I would like to express my heartfelt gratitude to my Ph.D. advisor, Professor Tao Yang, for his continuous guidance and support from the very fist day I came to the United States. I am profoundly appreciative of his invaluable mentorship, his guidance of my research, and his excellent life advice. His kindness and support have not only shaped me into a better researcher but also into a better person.

I also extend my sincere thanks to my Ph.D. committee members, Professor Shiyu Chang and Professor Xifeng Yan. Their enlightening feedback and suggestions have greatly contributed to the improvement of this dissertation. I am deeply thankful for their support since my Major Area Exam.

Additionally, I want to thank all my research group members for their guidance and support over the years: Shiyu Ji, Jinjin Shao, Yingrui Yang, Shanxiu He, Parker Carlson, Karl Wang, and Wentai Xie. I am also grateful to Haixin Lin, Tianbo Xiong, and Xiyue Wang for their efforts in becoming acquainted with the PISA and JASS search systems.

Finally, I would like to express my profound gratitude to my parents, Lihui and Dongsheng, and my wife, Ruoyun, for their unwavering support throughout this journey.

This thesis is supported in part by NSF IIS-2225942 and IIS-2040146, a Google faculty research award, and has used computing resource of the XSEDE and ACCESS programs supported by NSF. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

Curriculum Vitæ

Yifan Qiao 乔逸凡

Education

2024	Ph.D. in Computer Science (Expected), University of California, Santa Barbara.
2019	B.S. in Computer Science, Tsinghua University, China.

Publications

Yifan Qiao, Parker Carlson, Shanxiu He, Yingrui Yang, Tao Yang. "Threshold-driven Pruning with Segmented Maximum Term Weights for Approximate Cluster-based Sparse Retrieval". In submission to *2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

Yingrui Yang, Parker Carlson, Shanxiu He, **Yifan Qiao**, Tao Yang. "Cluster-based Partial Dense Retrieval Fused with Sparse Text Retrieval". *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2024.

Yingrui Yang, **Yifan Qiao**, Shanxiu He, Tao Yang. "Weighted KL-Divergence for Document Ranking Model Refinement". *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2024.

Yifan Qiao, Otto Godwin, Hua Ouyang. "On-Device Query Auto-Completion for Email Search". *46th European Conference on Information Retrieval (ECIR)*, 2024.

Yifan Qiao, Yingrui Yang, Shanxiu He, Tao Yang. "Representation Sparsification with Hybrid Thresholding for Fast SPLADE-based Document Retrieval". *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2023.

Yingrui Yang, Shanxiu He, **Yifan Qiao**, Wentai Xie, Tao Yang. "Balanced Knowledge Distillation with Contrastive Learning for Document Re-ranking". *Proceedings of the 2023 ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR)*, 2023.

Yifan Qiao, Shiyu Ji, Changhai Wang, Jinjin Shao, Tao Yang. "Privacy-aware Document Retrieval with Two-level Inverted Indexing". *Information Retrieval Journal, Volume 26, Article number 12*, 2023.

Yifan Qiao, Yingrui Yang, Haixin Lin, Tao Yang. "Optimizing Guided Traversal for Fast Learned Sparse Retrieval". *Proceedings of the ACM Web Conference 2023 (WebConf)*, 2023.

Yifan Qiao, Yingrui Yang, Haixin Lin, Tianbo Xiong, Xiyue Wang, Tao Yang. "Dual Skipping Guidance for Document Retrieval with Learned Sparse Representations". *arXiv: 2204.11154*, 2022.

Yingrui Yang, **Yifan Qiao**, Tao Yang. "Compact Token Representations with Contextual Quantization for Efficient Document Re-ranking". *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

Yingrui Yang, **Yifan Qiao**, Jinjin Shao, Xifeng Yan, Tao Yang. "Lightweight Composite Re-Ranking for Efficient Keyword Search with BERT". *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM)*, 2022.

Jinjin Shao, **Yifan Qiao**, Shiyu Ji, Tao Yang. "Window Navigation with Adaptive Probing for Executing BlockMax WAND". *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval (SIGIR)*, 2021.

Jinjin Shao, Shiyu Ji, Alvin Oliver Glova, **Yifan Qiao**, Tao Yang, Tim Sherwood. "Index Obfuscation for Oblivious Document Retrieval in a Trusted Execution Environment". *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM)*, 2020.

Yifan Qiao, Chenyan Xiong, Zhenghao Liu, Zhiyuan Liu. "Understanding the Behaviors of BERT in Ranking". *arXiv:1904.07531*, 2019.

Work Experience

Summer 2023	Ph.D. Intern at Apple.
Summer 2022	Ph.D. Intern at Apple.
Summer 2021	Ph.D. Intern at Apple.

Abstract

Pruning Optimization for Efficient Top-k Document Retrieval with Learned Sparse Representations

by

Yifan Qiao 乔逸凡

Efficiently searching for relevant documents on a large dataset typically employs an initial retrieval stage to extract the most relevant candidates. This process often utilizes a sparse data structure known as an inverted index, coupled with a simple yet fast ranking method, to identify the top-k matches for a given query. Recent advancements in retrieval methodologies have seen the integration of learned sparse representations, leveraging transformer-based language models to expand and weigh document terms, thereby enhancing the semantic alignment between query and document vocabularies. However, the distribution of these neural model generated weights differs from traditional term-frequency-based BM25, posing challenges for dynamic pruning algorithms for inverted index traversal, and significantly impeding retrieval speed. Moreover, these techniques often exacerbate document representation sparsity, further slowing down retrieval algorithms. Addressing these challenges, this thesis focuses on accelerating document retrieval algorithms through representation sparsification, BM25-guided document pruning, and cluster-based sparse retrieval strategies.

Contents

Chapter 1

Introduction

Information retrieval system plays a vital role in people's digital lives. With the rapid growth of digital contents' size, it becomes more and more important to retrieve relevant documents efficiently and effectively. In terms of effectiveness, given a query, to retrieve the most relevant document(s) from a large corpus, we need a tool to measure the similarity between the given query to any document in the corpus. Then, for efficiency, the retrieval system should be able to do some pruning work along the way to search, to avoid calculation of similarities among all documents.

1.1 Search Based on Keyword Matching

In early days, search is based on the word matching between a query and documents, and search engines mainly use sparse retrieval signals to capture the similarity. The sparse retrieval signals usually comprises statistical information including term frequency (TF), inverse document frequency (IDF). These signals are then combined by ranking functions developed from probabilistic retrieval framework, to estimate the similarity between a query and a given document. The common ranking functions include TF-IDF, and

BM25 [1].

In order to search in an efficient manner, many researchers have contributed ideas in pruning. Statically, documents are organized in the structure of an inverted index. An inverted index comprises posting lists for each query word. The posting list of a given query word records all the document identifiers where the corresponding document contains the query word, and the term frequency information of this word. During search time, only those posting lists that are relevant to the search query need to be visited, meaning that those documents have no overlapping words with the query can be pruned before search. Dynamically, more pruning can be done during search time. Usually, a priority queue is used to collect top-k documents with the highest similarity score, where k is the number of documents that will be retrieved. If at any time, the maximum possible score of a document or a block of documents cannot exceed the lowest score (which is called threshold) in the priority queue, those documents can be pruned away, to save search time cost. Notable dynamic pruning work includes WAND [2], BlockMax-WAND [3], and MaxScore [4] algorithms.

1.2 Semantic Search

With the power of Large Language Models (LLM), machines can intelligently respond to natural language questions with natural language answers. Rather than memorizing all possible answers in LLMs, a more feasible and flexible approach is to search for the context first, and send all the relevant documents along with users' query as prompt to LLMs. This technique, called Retrieval Augmented Generation (RAG), becomes a hot topic recently in the information retrieval community. The natural language queries, or semantic search queries, differ from prior keyword matching queries, tend to be composed of a complete sentence, and it's possible that synonyms rather than exact words shown

up in the desired document are employed by the query. Supporting semantic search queries by the search system also benefits traditional search, where users don't have to remember or know the exact word used in the document they want to retrieve. However, since users may rephrase their queries in any ways, traditional term frequency based keyword matching algorithms are not sufficient anymore.

Instead of representing a word by a scalar, or a one-hot vector, word embeddings are introduced to use dense vectors to represent the semantic meaning of a word. These embeddings are usually unsupervised trained on large corpora using Continuous Bag of Words (CBOW). Words are semantically similar should have close distances (L2 distance or cos-similarity) in the embedding space. Algorithms are then developed to measure the similarity of a query and a document based on their individual word embeddings, including DRMM [5], KNRM [6], and Conv-KNRM [7].

BERT-based language models [8] utilize attention mechanisms to capture the interaction among tokens in the same document. MonoBERT [9] proposed to concatenate the query text with the document text, feed the concatenated text to BERT encoder, and apply a projection layer on top of the BERT encoded [CLS] token to predict the similarity between the query and the document. This approach takes advantage of the attention mechanism which calculates the interaction between query terms and document terms, and propagate and summarize the information to the final representation of the [CLS] token. This kind of structures, where queries and documents are concatenated before the language model, is called cross-encoders. Generally, it yields the highest quality of relevance measurements. However, due to the early concatenation, it requires one BERT inference for each document, which can be prohibitively expansive for retrieval, where there can be millions of candidates need to be evaluated. Practically, people usually apply a lightweight retriever to obtain a list of k most relevant documents, and use the cross-encoder as a "re-ranker" to only re-rank those k documents.

In order to push the interaction into later stages, to make room for pre-calculation of inference of documents during indexing time, bi-encoders are invented. In the setup of bi-encoder, queries and documents are fed separately into the languge model like BERT, and one single dense vector is used to represent the whole meaning of the query/document text. The document embeddings thus can be pre-calculated and indexed without any information from the query. During online retrieval, the query is passed through the language model one time, to generate the query embedding. Then, approximate nearest neighbor search (ANN) is applied to find the nearest document vectors in the document corpus to retrieve the relevant documents to that query. Famous bi-encoders include DPR [10], SimLM [11], and RetroMAE [12, 13]. ANN frameworks (e.g. FAISS [14]) taking advantage of GPU's parallel computing capabilities achieve reasonably good retrieval latency for retrieval on large corpora. This retrieval scheme is usually referred to as dense retrieval.

1.3 Learned Sparse Retrieval

Dense retrieval is able to support semantic search, and can be fast with GPU supports. However, infrastructure cost for constructing a large-scale retrieval system with GPUs can be expansive, supporting of efficient semantic search on CPU-only machines is still in demand. People then went back to the traditional sparse retrieval framework with inverted index, and fuse the semantic representation produced by language models to it, producing learned sparse retrievers.

In dense retrieval, usually, the relevance score is calculated by the dot product of the query representation and document representation, where "representation" means a dense vector with commonly seen sizes of 768 (BERT-based) or 1536 (OpenAI). The traditional term frequency based retrieval function can also be fit into this framework,

with the only difference being the representation of the queries and documents are sparse, and the dimension of the vectors is the size of the vocabulary. Each entry of the sparse vector stores the term frequency of the corresponding word in the query/document text. With this in mind, in learned sparse retrieval, the sparse vectors do not have to store statistical information like term frequency, instead, it can store the semantic similarity of the corresponding word to the whole query/document text. Learned sparse retrievers, for example, SPLADE [15, 16, 17], and LexMAE [18], produce such sparse vectors by projecting each token's language model-encoded output back to the sparse space by calculating its dot product to the embedding of each token in the vocabulary, and pooling them together into one sparse vector for the whole query/document. Regularization is applied on the pooled sparse vector for sparsity, which has an effect on the trade-off between relevance effectiveness and efficiency.

Since the documents are again represented as sparse vectors, just like traditional term frequency based scores, they can be offline indexed and stored into the inverted index format. During online search time, traditional dynamic pruning algorithms including WAND [2], BlockMax-WAND [3] and Maxscore [4] can be used for acceleration as well. Piratically, learned sparse retrievers often yield better efficiency than dense retrievers. Other advantage of learned sparse retrievers over dense retrievers include better interpretability, and better zero-shot performance on dataset with no or few training samples.

1.4 Efficiency Optimization and Pruning

To efficiently search on a large corpus, there is a large body of research on reducing the footprint of inverted index at the offline time or avoiding visiting the entire index to minimize retrieval latency, and this category of efficiency optimization is called pruning. Commonly, there are two types of pruning, static pruning and dynamic pruning.

Static pruning happens before the system obtains user queries. Such techniques usually remove the low-impact documents presented in each posting list to shrink size. Examples include document spam filtering, term-centric pruning [19, 20] and document-centric pruning [21] for posting lists. Hybrid thresholding (HT) discussed in Chapter 2 is a static pruning technique, where a learnable threshold controls the sparsity of the learned sparse representation, hence reduces the index size.

Dynamic pruning, on the other hand, happens during online search, when the user query is presented. Usually, documents are pre-organized into grouped structures including blocks, or clusters, and statistical information including maximum or average value, are pre-calculated and stored for each group to summarize the documents contained in this group. During online inference, if the algorithm finds out that it is unlikely that there exists a highly relevant document to the user query based on the statistical information of a group, all the documents within that group can be pruned without further search. For dense retrieval, documents are grouped into clusters, based on the similarities between dense vectors. Common pruning techniques for ANN search include IVF [22] and HNSW [23]. For sparse retrieval, documents are usually grouped into blocks based on pre-assigned document ids in each posting list. Each block can be of the same size (for BlockMax WAND, BMW [3]), or variable sizes (VBMW [24]). Similarly, the statistical information is pre-calculated for each block and stored as metadata. Pruning algorithms including BMW and MaxScore [4] can then take advantage of these information to dynamically prune documents or blocks away. Chapter 3 is a direct optimization for these algorithms. Chapter 4 further combines the clustering idea in dense retrieval, with the dynamic pruning techniques on the inverted index structure.

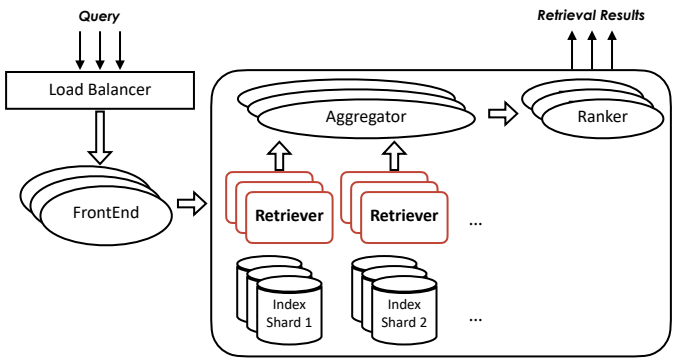


Figure 1.1: Web search

1.5 Applications of Efficient Sparse Retrieval

Sparse retrievers have been widely used for large scale web search in industry. Figure 1.1 shows a simplified framework of modern large scale search engine. Inside the back-end, documents are split into shards, and each retriever is responsible for retrieving the most relevant documents in the corresponding shard. In order to lower the cost of operation and decrease latency for end users, the core part, retrievers, need to be fast and effective. Learned sparse retrieval can fit rather easier into this framework, compared to dense retrieval, due to its similarity to the traditional sparse retrieval. Another issue to consider is that CPU-only machines are better than GPU machines for cost and feasibility when building the data center, where learned sparse retrievers have a huge advantage in speed than dense retrievers.

Another application for the learned sparse retrieval to shine is in the retrieval augmented generation (RAG) framework shown in Figure 1.2. Document retrieval is an important part, where relevant documents to the user query need to be fetched first. These documents are then used as the context for LLMs to generate answers to the query. Learned sparse retriever has better interpretability and generalizability than dense retrievers, which can potentially benefit the RAG framework.

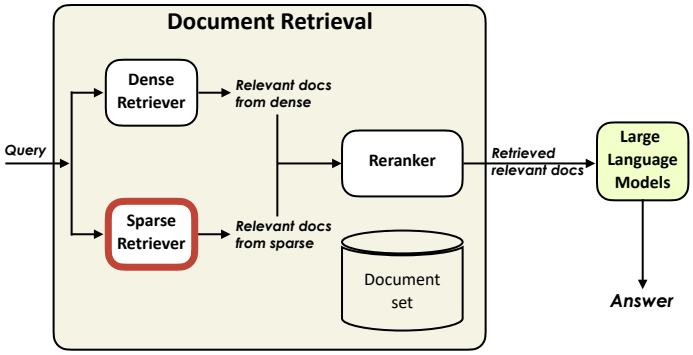


Figure 1.2: Web search

1.6 Summary of Contributions

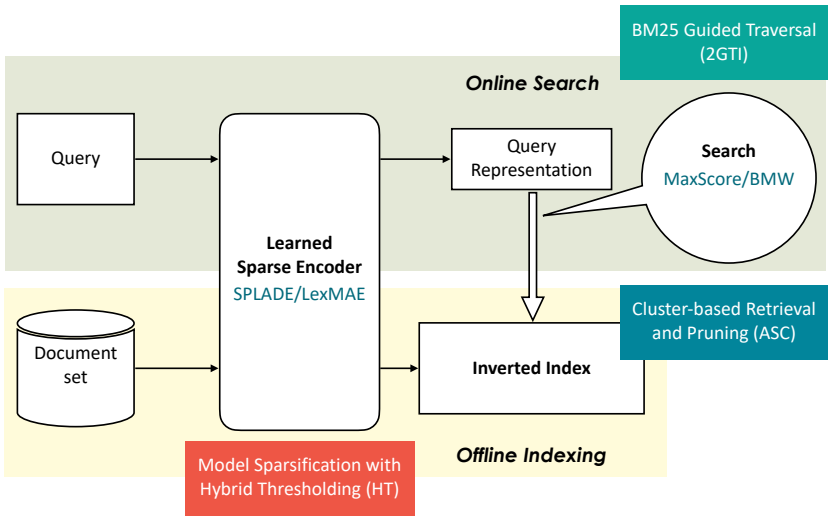


Figure 1.3: Workflow of learned sparse retrieval

Figure 1.3 shows the overall workflow of learned sparse retrieval systems. During offline indexing time, each document d in the corpus will be encoded into sparse vectors by language models like SPLADE, and stored into the inverted index structure. During online search, user query q will be encoded into sparse vectors by SPLADE, and corresponding posting lists of the non zero entries of the encoded vector stored in

the inverted index will be traversed and evaluated by dynamic pruning algorithms, for example, MaxScore.

In this retrieval framework, multiple components can be optimized for better efficiency. First of all, the sparsity of the sparse representation vectors plays a vital role on the efficiency of the retrieval system, because the more sparse the representation is, fewer tokens are present in the index, shorter the posting list will be. Chapter 2, as shown in the orange block in Figure 1.3, enhances the sparsity of the language model, by taking the sparsity objective into the retriever training procedure with a dynamic learnable threshold for static index pruning. Secondly, the dynamic pruning algorithms including MaxScore and BlockMax-WAND are designed for traditional statistical signals like term frequency or BM25, while the values in the learned sparse representations have significantly different distributions to the previous statistical signals, making dynamic pruning algorithms less effective. Chapter 3 thus improves these dynamic pruning algorithms by treating the term frequency information as a guidance of pruning and actually accumulating relevance scores based on learned representations, which is shown in green in Figure 1.3. Lastly, the inverted index could benefit from the clustering structure commonly used for dense vectors. Chapter 4 organizes documents into clusters based on the dense vector representations, and pre-compute statistical meta scores for cluster-level pruning during online search time to improve efficiency. It is displayed in the blue box Figure 1.3.

Chapter 2

Representation Sparsification with Hybrid Thresholding for Fast SPLADE-based Document Retrieval

2.1 Introduction

Recently learned sparse retrieval techniques [25, 26, 27, 28, 29, 15, 16, 30] have become attractive because such a representation can deliver a strong relevance by leveraging transformer-based models to expand document tokens with learned weights and can take advantage of traditional inverted index based retrieval techniques [24, 31]. Its query processing is cheaper than a dense representation which requires GPU support (e.g. [32, 33]) even with efficiency optimization through approximate nearest neighbor search [22, 34, 35].

This paper focuses on the SPLADE family of sparse representations [15, 16, 30] because it can deliver a high MRR@10 score for MS MARCO passage ranking [36] and a strong zero-shot performance for the BEIR datasets [37], which are well-recognized

IR benchmarks. The sparsification optimization in SPLADE has used L1 and FLOPS regularization to minimize non-zero weights during model learning, and our objective is to exploit additional opportunities to further increase the sparsity of inverted indices produced by SPLADE. Earlier static inverted index pruning research [19, 20, 21] for a lexical model has shown the usefulness of trimming a term posting list or a document by a limit. Yang et al. [38] conduct top token masking by limiting the top activated weight count uniformly per document and gradually reducing this weight count limit to a targeted constant during training. Motivated by these studies [19, 20, 21, 38] and since they have not addressed the learnability of a pruning limit through relevance-driven training, this paper exploits a learnable thresholding architecture to filter out unimportant neural weights produced by the SPLADE model through joint training.

The contribution of this paper is a learnable hybrid hard and soft thresholding scheme with an inverted index approximation to increase the sparsity of SPLADE-based document and query feature vectors for faster retrieval. In addition to experimental validation with MS MARCO and BEIR datasets, we provide an analysis of the impact of hybrid thresholding with joint training on index approximation errors and training update effectiveness.

2.2 Background

For a query q and a document d, after expansion and encoding, they can be represented by vector $\vec{w}(q)$ and $\vec{w}(d)$ with length $|V|$, where V is the vocabulary set. The rank score of q and d is computed as $R(q,d) = \vec{w}(q) \cdot \vec{w}(d) = \sum_{i=1}^{|V|} w_i^q \times w_i^d$. For sparse vectors with many zeros, retrieval can utilize a data structure called inverted index during online inference for fast score computation [24, 31]. The SPLADE model uses the BERT token space to predict the feature vector $\vec{w}$. In its latest SPLADE++ model, it first calculates

the importance of i-th input token in d for each j in V: $w_{ij}(\Theta) = \text{Transform}(\vec{h_i})^T \vec{E_j} + b_j$, where $\vec{h_i}$ is the BERT embedding of i-th token in d, $\vec{E_j}$ is the BERT input embedding for j-th token. Transform() is a linear layer with GeLU activation and LayerNorm. The weights in this linear layer, $\vec{E_j}$, and b_j are the SPLADE parameters updated during training and we call them set Θ. Then the j-th entry w_j of document d (or a query) is max-pooled as $w_j(\Theta) = \max_{i \in d}\{\log(1 + \text{ReLU}(w_{ij}(\Theta)))\}$. Notice that $w_j \geq 0$.

The loss function of SPLADE models [15, 16, 30] contains a per-query ranking loss L_R and sparsity regularization. The ranking loss has evolved from a log likelihood based function for maximizing positive document probability to margin MSE for knowledge distillation. This paper uses the loss of SPLADE with a combination that delivers the best result in our training process. L_R is the ranking loss with margin MSE for knowledge distillation [39]. The document token regularization L_D is computed on the training documents in each batch based on FLOPS regularization. The query token regularization L_Q is based on L1 norm. Let B be a set of training queries with N documents involved in a batch. $L_Q = \sum_{j \in V} \frac{1}{|B|} \sum_{q \in B} w_j^q$; $L_D = \sum_{j \in V} (\frac{1}{N} \sum_{d=1}^{N} w_j^d)^2$.

Related work. Other than SPLADE, sparse retrieval studies include SNRM [25], DeepCT [26], DeepImpact [27], and uniCOIL [28, 29]. The sparsity of a neural network is studied in the deep learning community. Soft thresholding in [40] adopts a learnable threshold with function $S(x,t) = ReLU(x-t)$ to make parameter x zero under threshold t. A hard thresholding function $H(x,t) = x$ when $x \geq t$ otherwise 0. Approximate hard thresholding [41] uses a Gauss error function to approximate $H(x,t)$ with smooth gradients. Dynamic sparse training [42] finds a dynamic threshold with marked layers. These works including the recent ones [43] are targeted for sparsification of parameter edges in a deep neural network. In our context, a token weight w_j is an output node in a network. The sparsification of output nodes is addressed in activation map compression [44] using ReLU as soft thresholding together with L1 regularization.

The work of [45] further boosts sparsity with the Hoyer regularization and a variant of ReLU. The above techniques have not been investigated in the context of sparse retrieval, and the impact of thresholding on relevance and query processing time with inverted indices, requires new design considerations and model structuring for document retrieval, even the previous work can be leveraged.

2.3 Hybrid Thresholding (HT)

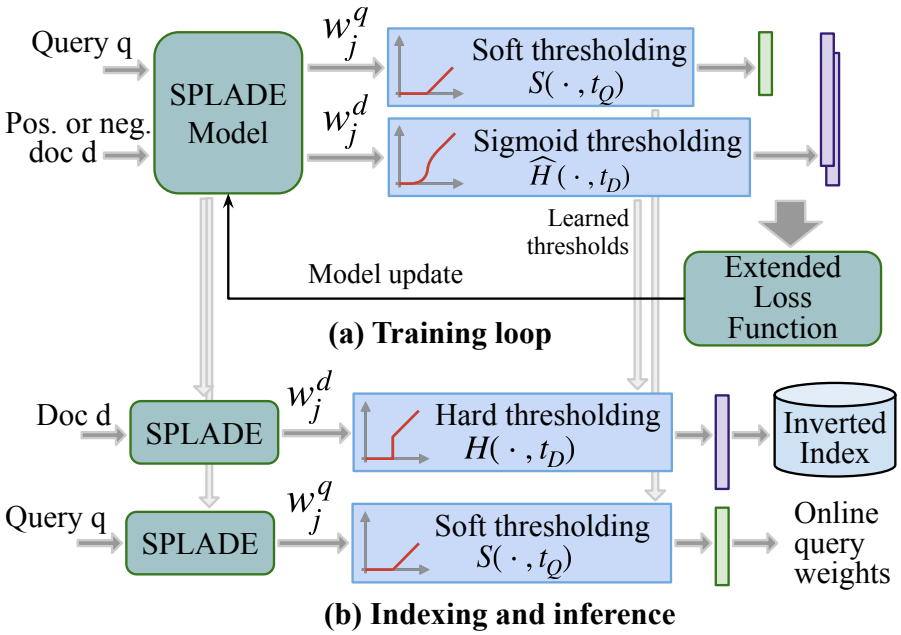


Figure 2.1: Hybrid thresholding with an index approximation

Design considerations. To zero out a token weight below a learnable threshold, there are two options: soft thresholding [40], and approximate hard thresholding [41]. For query token weights, we find that soft thresholding does not affect relevance significantly. For document token weights, our study finds that compared to soft thresholding, hard thresholding can retain relevance better since it does not change token weights when exceeding a threshold. Since the subgradient for hard thresholding with respect to

a threshold is always 0, an approximation needs to be carried out for training. For search index generation, an inverted index produced with the same approximate hard thresholding as training keeps many unnecessary non-zero document token weights, slowing down retrieval significantly. Thus we directly apply hard thresholding with a threshold learned from training, as shown in Figure 2.1. There is a gap between trained document token weights and actual weights used in our inverted index generation and online inference, and we intend to minimize this gap (called an index approximation error).

Thus our design takes a hybrid approach that applies soft thresholding to query token weights during training and inference and applies approximate hard thresholding to document token weights during training while using hard thresholding for documents during index generation. For approximate hard thresholding, we propose to use a logistic sigmoid-based function instead of a Gauss error function [41]. This sigmoid thresholding simplifies our analysis of the impact of its hyperparameter choice to index approximation errors, and to training stability.

2.3.1 Trainable and approximate thresholding

Training computes threshold parameters t_D, and t_Q for documents and queries, respectively. From the output of the SPLADE model, every token weight of a query is replaced with $S(w_j^q, t_Q)$, which is $ReLU(w_j^q - t_Q)$, and every document token weight is replaced with $\widehat{H}(w_j^q, t_D)$ before their dot product is computed during training as shown in Figure 2.1(a). Sigmoid thresholding $\widehat{H}$ is defined as:

$$\widehat{\mathrm{H}}(w_j^d, t_D) = w_j^d \sigma(K(w_j^d - t_D)) \text{ where } \sigma(x) = \frac{1}{1+e^{-x}}. \tag{2.1}$$

Here K is a hyperparameter to control the slope steepness of step approximation that jumps from 0 to 1 when exceeding a threshold.

The indexing process uses hard thresholding to replace all document weights that are below threshold t_D as 0 as depicted in Figure 2.1(b). The above post processing introduces an index approximation error $E = |\widehat{H}(w_j^d, t_D) - H(w_j^d, t_D)|$. We derive its upper bound as follows. Notice that $w_j \geq 0$, and for any $x \geq 0$, $1 + x \leq e^x$.

$$E = w_j^d \sigma(K(w_j^d - t_D)) = \frac{w_j^d}{1 + e^{K(t_D - w_j^d)}} \leq \frac{w_j^d}{2 + K(t_D - w_j^d)}.$$

When $w_j^d \geq t_D$, we can derive that

$$E = w_j^d(1 - \sigma(K(w_j^d - t_D))) = w_j^d \sigma(K(t_D - w_j^d))) \leq \frac{w_j^d}{2 + K(w_j^d - t_D)}.$$

Let σ^- denote $\sigma(K(w_i^d - t_D))$. $0 < \sigma^- < 1$. In both of the above cases, the error upper bound is minimized when K is large. This is consistent with the fact that error E is monotonically decreasing as K increases because $\frac{\partial E}{\partial K} = -w_j^d \ \sigma^-(1 - \sigma^-) \ |w_j^d - t_D| \leq 0$. When $|w_j^d - t_D|$ is big, the error is negligible and when $|w_j^d - t_D|$ is small, the error could become big with a small K value. But as shown later, an excessively large K value could cause a big parameter update during a training step, affecting joint training stability.

Let $Dlen$ and $Qlen$ be the non-zero token weight count of document d and query q, respectively. For our hybrid thresholding, $Dlen = \sum_j \mathbf{1}_{w_j^d \geq t_D}, Qlen = \sum_j \mathbf{1}_{w_j^q \geq t_Q}$. Here $\mathbf{1}_{x \geq y}$ is an indicator function as 1 if $x \geq y$ otherwise 0. When increasing t_D and t_Q, $Dlen$ and $Qlen$ decrease. Thus for a batch of training queries B, the original SPLADE loss is extended as: $L = (\frac{1}{|B|} \sum_{q \in B} L_R) + \lambda_Q L_Q + \lambda_D L_D + \lambda_T L_T$. The extra item added is $L_T = \log(1 + e^{-t_D}) + \log(1 + e^{-t_Q})$. We retain the original L_Q and L_D expressions because as w_j^q or w_j^d decreases, more weights can quickly be zeroed out.

2.3.2 Threshold and token weight updating

We study the change of t_D, t_Q, w_j^d, and w_j^q after each training step with a mini-batch gradient descent update. The analysis below uses the first-order Taylor polynomial approximation and follows the fact that sigmoid thresholding $\widehat{H}$ and soft thresholding function S are used independently for a query and a document in the loss function. Symbol α is the learning rate. Let "$d \triangleleft q$" mean d is a positive or negative document of query q.

$$\begin{aligned}\Delta t_D = t_D^{new} - t_D^{old} = -\alpha \frac{\partial L}{\partial t_D} = -\alpha \left(\frac{1}{|B|} \sum_{q \in B} \frac{\partial L_R}{\partial \widehat{H}} \frac{\partial \widehat{H}}{\partial t_D} + \lambda_T \frac{\partial L_T}{\partial t_D} \right) \\ = \alpha \left(\frac{1}{|B|} \sum_{q \in B} \left(K \frac{\partial L_R}{\partial \widehat{H}} \sum_{d \triangleleft q} \sum_i w_i^d (1-\sigma)^- \sigma^- \right) + \lambda_T \frac{e^{-t_D}}{1+e^{-t_D}} \right).\end{aligned}$$

$$\begin{aligned}\Delta t_Q = t_Q^{new} - t_Q^{old} = -\alpha \frac{\partial L}{\partial t_Q} = -\alpha \left(\frac{1}{|B|} \sum_{q \in B} \frac{\partial L_R}{\partial S} \frac{\partial S}{\partial t_Q} + \lambda_T \frac{\partial L_T}{\partial t_Q} \right) \\ = \alpha \left(\frac{1}{|B|} \sum_{q \in B} \left(\frac{\partial L_R}{\partial S} \sum_i \mathbf{1}_{w_i^q \geq t_Q} \right) + \lambda_T \frac{e^{-t_Q}}{1+e^{-t_Q}} \right).\end{aligned}$$

$$\begin{aligned}\Delta w_j^d = w_j^{d,new} - w_j^{d,old} \approx \sum_{\theta \in \Theta} \frac{\partial w_j^d}{\partial \theta} \Delta\theta = -\sum_{\theta \in \Theta} \frac{\partial w_j^d}{\partial \theta} \alpha \frac{\partial L}{\partial \theta} \\ = -\alpha \sum_{\theta \in \Theta} \frac{\partial w_j^d}{\partial \theta} \left(\frac{1}{|B|} \sum_{q \in B} \left(\frac{\partial L_R}{\partial \widehat{H}} \left(\sum_{d \triangleleft q} \sum_i \frac{\partial \widehat{H}}{\partial w_i^d} \frac{\partial w_i^d}{\partial \theta} \right) + \right. \right. \\ \left. \left. \frac{\partial L_R}{\partial S} \left(\sum_i \frac{\partial S}{\partial w_i^q} \frac{\partial w_i^q}{\partial \theta} \right) \right) + \lambda_D \frac{\partial L_D}{\partial \theta} + \lambda_Q \frac{\partial L_Q}{\partial \theta} \right).\end{aligned}$$

Notice that $\frac{\partial \widehat{H}}{\partial w_i^d} = \sigma^- + K w_i^d \sigma^- (1 - \sigma^-)$. The above results indicate:

- A significant number of terms in Δt_D and Δw_j^d involve linear coefficient K. This is verifiably true also for Δw_j^q. Although a large K value can minimize the index approximation error $|\widehat{H}(w_j^d, t_D) - H(w_j^d, t_D)|$, it can cause an aggressive change of

token weights and thresholds at a training iteration, making training overshoot and miss the global optimum. Thus K cannot be too large, and our evaluation further studies this.

- If $\frac{\partial L_R}{\partial \widehat{H}} \geq 0$, $\Delta t_D \geq 0$, and the document threshold increases, decreasing $Dlen$. Otherwise document token threshold may decrease after a parameter update step during training, and the degree of decreasing is reduced by a positive value $\frac{e^{-t_D}}{1+e^{-t_D}}$. Based on the sign of $\frac{\partial L_R}{\partial S}$, we can draw a similar conclusion on Δt_Q.

2.4 Evaluation

Our evaluation uses MS MARCO passages [36] and BEIR datasets [37]. MS MARCO has 8.8M passages while BEIR has 13 different datasets of varying sizes up-to 5.4M. As a common practice, we report the relevance in terms of mean reciprocal rank MRR@10 for the MS MARCO passage Dev query set with 6980 queries, and the normalized discounted cumulative gain nDCG@10 [46] for its DL'19 and DL'20 sets, and also for BEIR. For retrieval with a SPLADE inverted index, we report the mean response time (MRT) and 99th percentile time (P_{99}) in milliseconds. The query encoding time is not included. For the SPLADE model, we warm up it following [30, 47], and train it with $\lambda_Q = 0.01$ and $\lambda_D = 0.008$, and hybrid thresholding. We use the PISA [48] search system to index documents and search queries using SIMD-BP128 compression [49] and MaxScore retrieval [50, 31]. Our evaluation runs as a single thread on a Linux CPU-only server with Intel i5-8259U 2.3GHz and 32GB memory. Similar retrieval latency results are observed on a 2.3GHz AMD EPYC 7742 processor. The checkpoints and related code will be released in https://github.com/Qiaoyf96/HT.

Overall results with MS MARCO. Table 2.1 is a comparison with the baselines on MS MARCO passage Dev set, DL'19, and DL'20. It lists the average $Dlen$ value, and

Table 2.1: Overall results on MS MARCO passages

Methods	**MRR** Dev	**MRT**(P_{99}) $k = 10$	**MRT**(P_{99}) $k = 1000$	**nDCG** DL'19	**nDCG** DL'20	**Dlen**
SPLADE	**0.3966**	48.3(228)	127(408)	**0.7398**	**0.7340**	351
/DT [41]	0.3922	102(457)	262(786)	0.7392	0.7319	444
/Top305 [38]	0.3962	42.4(202)	114(369)	0.7353	0.7288	277
/Top100 [38]	0.3908	21.8(106)	62.5(196)	0.7192	0.7119	99
/DCP50% [21]	0.3958	30.0(145)	83.9(271)	0.7385	0.7321	175
/DCP40% [21]	0.3933	25.9(124)	73.3(235)	0.7335	0.7280	140
/DCP30% [21]	0.3912	21.6(101)	61.8(193)	0.7287	0.7217	105
/Cut0.5	0.3924	21.9(104)	62.6(195)	0.7296	0.7212	144
/Cut0.8	0.3885	15.6(70.4)	43.8(128)	0.7207	0.7118	112
/HT_1	0.3955	22.8(108)	62.3(195)	0.7322	0.7210	140
/HT_3	0.3942	14.2(67.2)	40.6(123)	0.7327	0.7228	106
/HT_1-2GTI [51]	0.3959	10.0(49.1)	27.6(92.2)	0.7330	0.7210	140
/HT_3-2GTI [51]	0.3942	**6.9(33.9)**	**19.3(62.1)**	0.7320	0.7228	106

top-k retrieval time with depth $k = 10$ and 1000. Row 3 is for original SPLADE trained by ourselves with an MRR number higher than 0.38 reported in [30, 47]. Rows 12 and 13 list the result of our hybrid thresholding marked as HT_{λ_T} and $K = 25$. With $\lambda_T = 1$, SPLADE/HT_1 converges to a point where $t_Q = 0.4$ and $t_D = 0.5$, which is about 2x faster in retrieval. HT_3 with $\lambda_T = 3$ converges at $t_Q = 0.7$ and $t_D = 0.8$, resulting 3.1x speedup than SPLADE while having a slightly lower MRR@10 0.3942. No statistically significant degradation in relevance has been observed at the 95% confidence level for both HT_1 and HT_3. The inverted index size reduces from 6.4GB for original SPLADE to 2.8GB and 2.2GB for HT_1 and HT_3 respectively. When applying two-level guided traversal 2GTI [51] with its fast configuration, Rows 14 and 15 show a further latency reduction to 6.9ms or 19.3ms.

We discuss other baselines listed in this table. Row 4 named DT uses the thresholding scheme from [41]. Its training does not converge with its loss function, and its retrieval is much slower. Rows 5 and 6 follow joint training of top-k masking [38] with the top 305 tokens as suggested in [38] and with the top 100 tokens. Rows 7, 8 and 9 marked with

DCPx follow document centric pruning [21] that keeps x of top tokens per document where x=50%, 40%, and 30%. We did not list term centric pruning [19, 20] because [21] shows DCP is slightly better in relevance under the same latency constraint. Rows 10 and 11 with "/Cut0.5" and "/Cut0.8" apply a hard threshold with 0.5 and 0.8 in the output of original SPLADE without joint training. The index pruning options without learning from Rows 5 to 11 can either reduce the latency to the same level as HT, but their relevance score is visibly lower; or have a relevance similar to HT but with much slower latency. This illustrates the advantage of learned hybrid thresholding with joint training.

.

Table 2.2: Zero-shot performance on BEIR datasets

	SPLADE		SPLADE/HT_1		SPLADE/HT_3	
Dataset	**nDCG**	**MRT**	**nDCG**	**MRT**	**nDCG**	**MRT**
DBPedia	0.430	135	**0.435**	64.2	0.426	32.3
FiQA	**0.354**	6.5	0.345	4.0	0.336	3.2
NQ	**0.547**	81.8	0.545	45.9	0.539	28.6
HotpotQA	0.678	481	**0.680**	265	0.678	140
NFCorpus	0.351	0.5	**0.352**	0.3	0.346	0.2
T-COVID	0.719	16.0	**0.730**	10.1	0.695	7.5
Touche-2020	0.307	15.0	0.306	9.3	**0.313**	4.5
ArguAna	0.440	20.8	0.463	7.8	**0.500**	4.1
C-FEVER	**0.234**	1375	0.219	681	0.213	332
FEVER	**0.781**	1584	0.778	559	0.764	264
Quora	**0.806**	17.5	0.776	9.2	0.792	4.5
SCIDOCS	0.151	6.9	**0.155**	3.0	0.151	2.0
SciFact	0.676	5.7	**0.681**	2.4	0.672	1.4
Average	**0.498**	-	0.497	2.0x	0.494	3.6x

Table 4.4 lists the zero-shot performance of HT when $k = 1000$ by applying the SPLADE/HT model learned from MS MARCO to the BEIR datasets without any additional training. HT_1 has a similar nDCG@10 score as SPLADE without HT, while having a 2x MRT speedup on average. HT_3 is even faster with 3.6x speedup, and its nDCG@10 drops in some degree to 0.494.

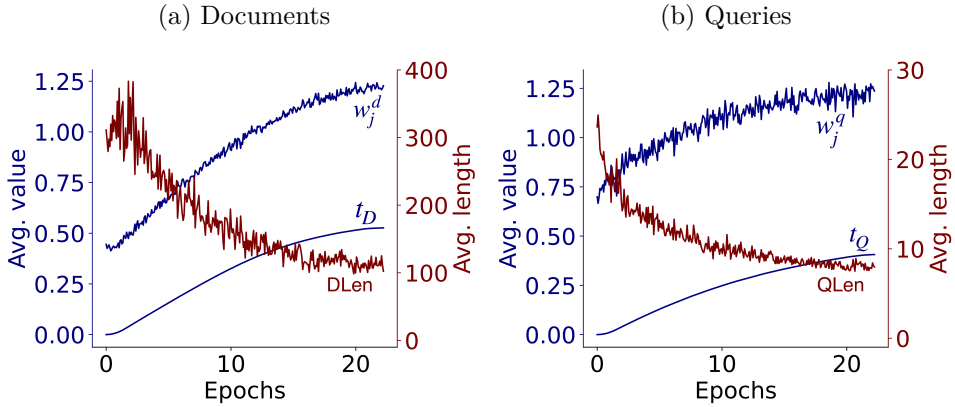


Figure 2.2: Weight/threshold/sparsity changes during training

Figure 2.2 depicts the average values of w_j^d, t_D, and $Dlen$ on the left and w_j^q, t_Q, and $Qlen$ on the right during MS MARCO training under HT_1. x-axis is the training epoch number. It shows that $Dlen$ and $Qlen$ decrease while t_D and t_Q increase as training makes progress and SPLADE/HT_1 converges after about 20 epochs.

Design options. Table 2.3 lists performance under 4 thresholding combinations from Row 3 to Row 7. $S[x]$ means soft thresholding function $S()$ is applied to x for both training and indexing where x can be documents (D) or queries (Q). $\widehat{H}$[x] means sigmoid thresholding $\widehat{H}$ is applied in both training and indexing. $\widehat{H}H[x]$ means $\widehat{H}$ is applied in training and H is applied in indexing. $\phi[x]$ means no thresholding is applied to x during training and indexing. When thresholding is not applied to queries, $\widehat{H}H[D]$ is 1.3x faster than $S[D]$ when $k = 10$ and $k = 1000$ while their relevance scores are similar. Shifting of document weight distribution by soft thresholding significantly affects retrieval time. Rows 6 and 7 fix $\widehat{H}H[D]$ setting, and show that soft thresholding is more effective in relevance than hard thresholding for query tokens. Shifting of query weight distribution has less effect on latency while gaining more relevance through model consistency between training and indexing.

Table 2.3: Impact of design options. MS MARCO passages.

HT Config. $\lambda_T = 1$	**MRR**	**MRT**(P_{99}) $k = 10$	**MRT**(P_{99}) $k = 1000$	**Qlen**	**Dlen**
Soft vs. hard thresholding in 4 combinations. Fix $K = 25$.					
$\phi[Q], S[D]$	0.3941	31.7(157)	91.5(315)	14.3	145
$\phi[Q], \widehat{H}H[D]$	0.3942	24.1(111)	70.7(219)	13.5	142
$S[Q], \widehat{H}H[D]$	**0.3955**	**22.8(108)**	**62.3**(195)	11.3	140
$\widehat{H}H[Q], \widehat{H}H[D]$	0.3904	24.9(106)	62.6(**182**)	9.0	142
Vary K. $\widehat{H}[D]$ vs. $\widehat{H}H[D]$. Fix $S[Q]$.					
$\widehat{H}H[D]$, $K = 2.5$	0.3947	22.8(110)	62.6(199)	11.5	149
$\widehat{H}[D]$, $K = 2.5$	**0.3963**	41.4(198)	112(358)	11.5	421
$\widehat{H}H[D]$, $K = 25$	0.3955	22.8(108)	62.3(195)	11.3	140
$\widehat{H}[D]$, $K = 25$	0.3961	28.7(136)	76.9(239)	11.3	208
$\widehat{H}H[D]$, $K = 250$	0.3946	**21.9(102)**	**60.5(189)**	11.2	135
$\widehat{H}[D]$, $K = 250$	0.3947	23.1(112)	63.9(203)	11.2	159
Usefulness of L_Q and L_D. Fix $S[Q], \widehat{H}H[D]$, and $K = 25$.					
w/o L_Q	0.3956	56.2(245)	166(502)	20.1	138
w/o L_Q, L_D	0.3954	99.4(434)	254(772)	25.9	421

Hyperparameter K in sigmoid thresholding $\widehat{H}$. Table 2.3 compares $\widehat{H}H[D]$ with $\widehat{H}[D]$ when varying K from Row 8 to Row 14. In these cases, training always uses $\widehat{H}$ while indexing uses $\widehat{H}$ or H. When K is small as 2.5, applying $\widehat{H}$ to both training and indexing yields good relevance, but retrieval is about 1.8x slower because much more non-zero weights are kept in the index. When K becomes large as 250, training does not converge to the global optimum due to large update sizes, resulting in an MRR score lower than K=25 even with no index approximation. $K = 25$ has a reasonable MRR while $\widehat{H}H[D]$ is up-to 26% faster than $\widehat{H}[D]$.

Retaining L_Q and L_D. Last three rows of Table 2.3 shows that the query length is higher when L_Q is removed from the loss function, and documents get longer when L_D is removed further. The result means L_Q and L_D are useful in sparsity control together with L_T.

2.5 Concluding Remarks

Our evaluation shows that learnable hybrid thresholding with index approximation can effectively increase the sparsity of inverted indices with 2-3x faster retrieval and competitive or slightly degraded relevance (0.28% - 0.6% MRR@10 drop). Its trainability allows relevance and sparsity guided threshold learning and it can outperform index pruning without such an optimization. Our scheme retains a non-uniform number of non-zero token weights per vector based on a trainable weight and threshold difference for flexibility in relevance optimization. Our analysis shows that hyperparameter K in sigmoid thresholding needs to be chosen judiciously for a small index approximation error without hurting training stability.

If a small relevance tradeoff is allowed, more retrieval time reduction is possible when applying other related orthogonal efficiency optimization techniques [47, 31, 52, 51, 53, 54]. Applying hybrid thresholding HT_3 to a checkpoint of a recent efficiency-driven SPLADE model [47] with 0.3799 MRR@10 on the MS MARCO passage Dev set, decreases the response time from 36.6ms to 21.7ms (1.7x faster) when k=1000 while having 0.3868 MRR@10. This latency can be further reduced to 14.2ms with the same MRR@10 number (0.3868) when 2GTI [51] is applied to the above index.

A future study is to investigate the use of the proposed hybrid thresholding scheme for other learned sparse models [27, 28, 29].

Chapter 3

Optimizing Guided Traversal for Fast Learned Sparse Retrieval

3.1 Introduction

Document retrieval for searching a large dataset often uses a sparse representation of document feature vectors implemented as an inverted index which associating each search term with a list of documents containing such a term. Recently learned sparse representations have been developed to compute term weights using a neural model such as transformer based retriever [26, 55, 15, 16, 56, 28] and deliver strong relevance results, together with document expansion (e.g. [57]). A downside is that top k document retrieval latency using a learned sparse representation is much large than using the BM25 model as discussed in [56, 58]. In the traditional BM25-based document retrieval with additive ranking, a dynamic index pruning strategy based on top k threshold is very effective by computing the rank score upper bound on the fly for each visited document during index traversal in order to skip low-scoring documents that are unable to appear in the final top k list. Well known traversal algorithms with such dynamic pruning strategies

include MaxScore [4] and WAND [2], and their block-based versions Block-Max WAND (BMW) [3] and Block-Max MaxScore (BMM) [59, 60].

Mallia et al. [31] propose to use BM25 to guide traversal, called GT, for fast learned sparse retrieval because the distribution of learned weights results in less pruning opportunities and they conducted an evaluation with retrieval model DeepImpact [56]. One variation they propose is to compute the final rank scoring as a linear combination of the learned weights and BM25 weights, denoted as GTI. GT is a special case of GTI and this paper treats GTI as the main baseline. Since the BM25 weight for a document term pair may not exist in a learned sparse index, zero filling is used in Mallia et al. [31] to align the BM25 and learned weight models. During our evaluation using GT for SPLADE v2 and its revision SPLADE++ [16, 30], we find that as retrieval depth k decreases, BM25 driven skipping becomes too aggressive in dropping documents desired by top k ranking based on learned term weights, which can cause a significant relevance degradation. In addition, there is still some room to further improve index alignment of GTI for more accurate BM25 driven pruning.

To address the above issues, we improve our earlier pruning study on dual guidance with combined BM25 and learned weights [52]. Our work generalizes GTI by constraining the pruning influence of BM25 and providing an alternative smoothing method to align the BM25 index with learned weights. In Section 3.4, we propose a two-level parameterized guidance scheme with index alignment, called 2GTI, to manage pruning decisions during MaxScore based traversal. We analyze some formal properties of 2GTI on its relevance behaviors and configuration conditions when 2GTI outperforms a two-stage top k search algorithm for a query in relevance.

Section 3.5 and Section 3.6 present an evaluation of 2GTI with SPLADE++ [15, 16, 30] and uniCOIL [28, 29] in addition to DeepImpact [56] when using MaxScore on the MS MARCO datasets. This evaluation shows that when retrieval depth k is small, or when

the BM25 index is not well aligned with the underlying learned sparse representation, 2GTI can outperform GTI and retain relevance more effectively. In some cases, there is a tradeoff that 2GTI based retrieval may be slower than that of GTI while 2GTI is still much faster than the original MaxScore method without BM25 guidance. 2GTI is also effective for the BEIR datasets in terms of the zero-shot relevance and retrieval latency. In Section 3.7, we have extended the use of 2GTI for a BMW-based algorithm such as VBMW [24]. We demonstrate that 2GTI with VBMW can be useful for a class of short queries and when k is small.

3.2 Background and Related Work

The top-k document retrieval problem identifies top ranked results in matching a query. A document representation uses a feature vector to capture the semantics of a document. If these vectors contain much more zeros than non-zero entries, then such a representation is considered sparse. For a large dataset, document retrieval often uses a simple additive formula as the first stage of search and it computes the rank score of each document d as:

$$\sum_{t \in Q} w_t \cdot w(t, d), \tag{3.1}$$

where Q is the set of all search terms, $w(t, d)$ is a weight contribution of term t in document d, and w_t is a document-independent or query-specific term weight. Assume that $w(t, d)$ can be statically or dynamically scaled, this paper views $w_t = 1$ for simplicity of presentation. An example of such formula is BM25 [1] which is widely used. For a sparse representation, a retrieval algorithm often uses an *inverted index* with a set of terms, and a *document posting list* of each term. A posting record in this list contains document ID and its weight for the corresponding term.

Threshold-based skipping. During the traversal of posting lists in document retrieval, the previous studies have advocated dynamic pruning strategies to skip low-scoring documents, which cannot appear on the final top-k list [2, 61]. To skip the scoring of a document, a pruning strategy computes the upper bound rank score of a candidate document d, referred to as $Bound(d)$.

If $Bound(d) \leq \theta$ where θ is the rank score threshold in the top final k list, this document can be skipped. For example, WAND [2] uses the maximum term weights of documents of each posting list to determine the rank score upper bound of a pivot document while BMW [3] and its variants (e.g. [24]) optimize WAND use block-based maximum weights to compute the score upper bounds. MaxScore [4] uses term partitioning and the top-k threshold to skip unnecessary index visitation and scoring computation. Previous work has also pursued a "rank-unsafe" skipping strategy by deliberately over-estimating the current top-k threshold by a factor [2, 62, 63, 64].

Learned sparse representations. Earlier sparse representation studies are conducted in [25], DeepCT [26], and SparTerm [55]. Recent work on this subject includes SPLADE [15, 16, 30], which learns token importance for document expansion with sparsity control. DeepImpact [56] learns neural term weights on documents expanded by DocT5Query [57]. Similarly, uniCOIL [28] extends the work of COIL [29] for contextualized term weights. Document retrieval with term weights learned from a transformer has been found slow in [31, 58]. Mallia et al. [31] state that the MaxScore retrieval algorithm does not efficiently exploit the DeepImpact scores. Mackenzie et al. [58] view that the learned sparse term weights are "wacky" as they affect document skipping during retrieval thus they advocate ranking approximation with score-at-a-time traversal.

Our scheme uses a hybrid combination of BM25 and learned term weights, motivated by the previous work in composing lexical and neural ranking [65, 66, 67, 68, 69]. GTI adopts that for final ranking. A key difference in our work is that hybrid scoring is used

for two-level pruning control and its formula can be different from final ranking. The multi-level hybrid scoring difference provides an opportunity for additional pruning and its quality control. Thus the outcome of 2GTI is not a simple linear ranking combination of BM25 and learned weights and two-level guided pruning yields a non-linear ensemble effect to improve time efficiency while retaining relevance. Our evaluation will include a relevance and efficiency comparison with MaxScore using a simple linear combination.

This paper mainly focuses on MaxScore because it has been shown more effective for relatively longer queries [50]. We also consider VBMW [24] because it is generally acknowledged to represent the state of the art [58] for many cases, especially when k is small and the query length is short [50].

3.3 Design Considerations

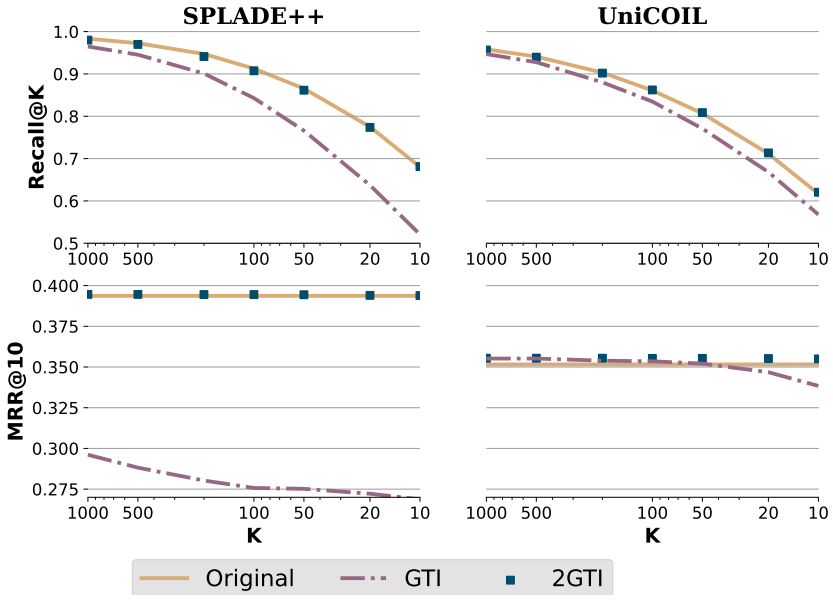


Figure 3.1: Recall@k and MRR@10 when k varies.

Figure 3.1 shows the performance of the original MaxScore retrieval algorithm with-

out BM25 guidance, GTI, and the proposed 2GTI scheme in terms of MRR@10 and recall@k when varying top k in searching MS MARCO passages on Dev query set. Here k is the targeted number of top documents to retrieve and it is also called retrieval depth sometime in the literature. Section 3.5 has more detailed dataset and index information. For both SPLADE++ and uniCOIL, we build the BM25 model following [31] to expand passages first using DocT5Query, and then use the BERT's Word Piece tokenizer to tokenize the text, and align the token choices of BM25 with these learned models. From Figure 3.1, there are significant recall and MRR drops with GTI when k varies from 1,000 to 10. There are two reasons contributing to the relevance drops.

1. When the number of top documents k is relatively small, the relevance drops significantly. As k is small, dynamically-updated top k score threshold becomes closer to the maximum rank score of the best document. Fewer documents fall into top k positions and more documents below the updated top k score threshold would be removed earlier. Then the accuracy of skipping becomes more sensitive. The discrepancy of BM25 scoring and learned weight scoring can cause good candidates to be removed inappropriately by BM25 guided pruning, which can lead to a significant relevance drop for small k.

2. The relevance drop for SPLADE++ with BM25 guided pruning is noticeably much more significant than uniCOIL. That can be related to the fact that SPLADE++ expands tokens of each document tokens differently and much more aggressively than uniCOIL. As a result, 98.6% of term document pairs in SPLADE++ index does not exist in the BM25 index even after docT5Query document expansion while this number is 1.4% for uniCOIL. Thus, BM25 guidance can become less accurate and improperly skip more good documents.

With the above consideration, our objective is to control the influence of BM25

weights in a constrained manner for safeguarding relevance prudently, and to develop better weight alignment when the BM25 index is not well aligned with the learned sparse index. In Figure 3.1, the recall@k number of 2GTI marked with blue squares is similar to that of the original method without BM25 guidance. Their MRR@10 numbers overlapped with each other, forming a nearly-flat lines, which indicates their MRR@10 numbers are similar even k decreases. The following two sections present our solutions in addressing the above two issues respectively.

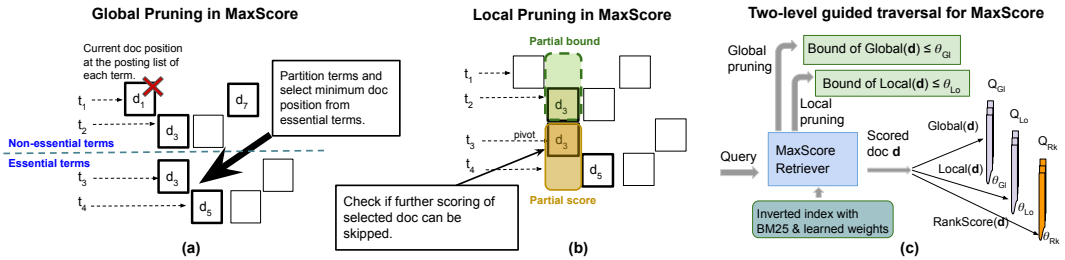


Figure 3.2: (a) and (b): Example of two-level pruning in MaxScore. (c) Two-level guided traversal for MaxScore.

3.4 Two-level Guided Traversal

3.4.1 Two-level guidance for MaxScore

We assume the posting list of each term is sorted in an increasing order of document IDs in the list. The MaxScore algorithm [4] can be viewed to conduct a sequence of traversal steps and at each traversal step, it conducts term partitioning and then examines if scoring of a selected document should be skipped. We differentiate pruning-oriented actions in two levels as follows.

- **Global level.** MaxScore uses the maximum scores (upper bounds) of each term and the current known top k threshold to partition terms into two lists at each

index traversal step: the essential list and non-essential list. The documents that do not contain essential terms are impossible to appear in top k results and thus can be eliminated. In the next step of index traversal, it will start with the minimum unvisited document ID only from the posting lists of essential terms. Thus index visitation is driven by moving such a minimum document ID pointer from the essential list.

We consider this level of pruning as global because it guides skipping of multiple documents and explores inter-document relationship implied by maximum term weights. Figure 3.2(a) depicts an example of global pruning flow in MaxScore with 4 terms and each posting list maintains a pointer to the current document being visited at a traversal step. The term partitioning identifies two essential terms t_3 and t_4. The minimum document ID among the current document pointers in these essential terms is d_3, and any document ID smaller than d_3 is skipped from further consideration during this traversal step. The current visitation pointer of the posting list of non-essential lists also moves to the smallest document ID equal to or bigger than d_3.

- **Local level.** Once a document is selected for possible full evaluation, the ranking score upper bound of this document can be estimated and gradually tightened using maximum weight contribution or the actual weight of each query term for this document. This incrementally refined score upper bound is compared against the dynamically updated top k threshold, which provides another opportunity to fully or partially skip the evaluation of this document. We differentiate this level of skipping decision as local because this pruning is localized towards a specific document selected. Figure 3.2(b) illustrates an example of local pruning in MaxScore. d_3 is the document selected after term partitioning and the maximum or actual weights

contributed from all posting lists for document d_3 are utilized for the local pruning decision.

Instead of directly using BM25 to guide pruning at the global and local levels, we propose to use a linear combination of BM25 weights and learned weights to guide skipping at each level as follows, which allows a parameterizable control of their influence.

- We incrementally maintain three accumulated scores for each document $Global(d)$, $Local(d)$, and $RankScore(d)$. $Global(d)$ is for global pruning, $Local(d)$ is for local pruning, and $RankScore(d)$ is for final ranking.

$$\begin{aligned} Global(d) &= \alpha RankScore_B(d) + (1-\alpha) RankScore_L(d) \\ Local(d) &= \beta RankScore_B(d) + (1-\beta) RankScore_L(d) \\ RankScore(d) &= \gamma RankScore_B(d) + (1-\gamma) RankScore_L(d) \end{aligned}$$

  where $0 \leq \alpha, \beta, \gamma \leq 1$, $RankScore_B(d)$ follows Expression 3.1 using BM25 weights, and $RankScore_L(d)$ follows Expression 3.1 using learned weights. The RankScore formula follows the GTI setting in [31], and 2GTI with $\alpha = \beta = 1$ behaves like GTI. 2GTI with $\alpha = \beta = \gamma$ is the same as MaxScore retrieval and it uses learned neural weights only when $\gamma = 0$.

- With the above three scores for each evaluated document, we maintain three separate queues: Q_{Gl}, Q_{Lo}, Q_{Rk} for documents with the k largest scores in terms of $Global(d)$, $Local(d)$, and $RankScore(d)$ respectively. The lowest-scoring document in each queue is removed separately without inter-queue coordination. These queues are maintained for different purposes: the first two queues regulate global and local pruning while the last queue is to produce the final top k results. When a

document based on local pruning is eliminated for further consideration, this document is not added to global and local queues Q_{Gl} and Q_{Lo}. But this document may have some partial score accumulated for its $RankScore(d)$, and it is still added to Q_{Rk} in case this document with the partial score may qualify in the top k results based on the latest $RankScore(d)$ value.

These three queues yield three dynamic top-k thresholds θ_{Gl}, θ_{Lo}, and θ_{Rk}. They can be used for a pruning decision to avoid any further scoring effort to obtain or refine $RankScore(d)$.

Revised MaxScore pruning control flow: Figure 3.2(c) illustrates the extra control flow added for the revised MaxScore algorithm. Let N be the number of query terms. We define:

- Given N posting lists corresponding to N query terms, each i-th posting list contains a sequence of posting records and each record contains document ID d, its BM25 weight $w_B(i, d)$ and learned weight $w_L(i, d)$. Posting records are sorted in an increasing order of their document IDs.

- An array σ_L of N where $\sigma_L[i]$ is the maximum contribution of the learned weight to any document for i-th term.

- An array σ_B of N where $\sigma_B[i]$ is the maximum contribution of the BM25 weight to any document for i-th term.

- N search terms are presorted so that $\alpha\sigma_B[i] + (1-\alpha)\sigma_L[i] \leq \alpha\sigma_B[i+1] + (1-\alpha)\sigma_L[i+1]$ where $1 \leq i \leq N-1$.

Global pruning with term partitioning. For each query term $1 \leq i \leq N$, we find the largest integer *pivot* from 1 to N so that $\sum_{j=1}^{pivot-1}(\alpha\sigma_B[j] + (1-\alpha)\sigma_L[j]) \leq \theta_{Gl}$.

All terms from *pivot* to N are considered as *essential.* If a document d does not contain any essential term, the upper bound of $Global(d) \leq \sum_{j=1}^{pivot-1} \alpha\sigma_B[j] + (1-\alpha)\sigma_L[j] \leq \theta_{Gl}$. This document cannot appear in the final top k list based on the global score. Then this document is skipped without appearing in any of the three queues.

Once the essential term list above the *pivot* position is determined, let the next minimum document ID among the current position pointers in the posting lists of all essential terms be document d. We also call it the *pivot* document.

Local pruning. Next we check if the detailed scoring of the selected pivot document d can be avoided fully or partially. Following an implementation in [70], we describe this procedure with a modification to use hybrid scoring as follows and it repeats the following three steps with the initial value of term position x as the *pivot* position and x decreases by 1 at each loop iteration.

- Let $PartialScore_{Local}(d)$ be the sum of all term weights of document d in the posting lists from position x to N after linear combination. Namely $PartialScore_{Local}(d) = \sum_{i=x}^{N-1} \beta w_B(i,d) + (1-\beta) w_L(i,d)$ when i-th posting list contains d, and otherwise this value is 0.

  As x decreases, the term weight of pivot document d is extracted from the posting list of x-th term if available.

- Let $PartialBound_{Local}(d)$ be the bound for partial local score of document d in the posting lists of the first to x-th query terms.

$$PartialBound_{Local}(d) = \sum_{j=1}^{x} \beta\sigma_B[j] + (1-\beta)\sigma_L[j].$$

- At any time during the above calculation, if

$$PartialBound_{Local}(d) + PartialScore_{Local}(d) \leq \theta_{Lo},$$

  further rank scoring for *pivot* document d is skipped and this document will not appear in any of the three queues. Figure 3.2(b) depicts that the partial bound and partial score of $Local(d_3)$ for pivot document d_3 are computed to assist a pruning decision.

Complexity. 2GTI' s complexity is the same as MaxScore and GTI. The in-memory space cost includes the space to host the inverted index involved for this query and the three queues. The time complexity is proportional to the total number of posting records involved for a query multiplied by $\log k$ for queue updating.

A posting list may be divided and compressed in a block-wise manner and Block MaxScore can use 2GT similarly while a previous study [50] shows Block-Max MaxScore is actually slower than MaxScore under several compression schemes. We will discuss the use of 2GT in block-based BMW in Appendix 3.7.

3.4.2 Relevance properties of 2GTI

2GTI ensembles BM25 and learned weights for pruning in addition to rank score composition, producing a top k ranked list which can be different than additive ranking with learned weights or their linear combination of BM25 weights. Thus 2GTI is not rank-safe compared to any of such baselines. Two-level pruning is driven by different combination coefficients α, β, and γ configured in 2GTI and their value gap provides an opportunity for additional pruning while 2GTI tries to retain relevance effectiveness. Is there a relevance guarantee 2GTI can offer in case such pruning skips relevant documents erroneously sometimes? To address this question analytically, this subsection presents

three properties regarding the relevance outcome and competitiveness of the 2GTI based retrieval.

Our analysis will use the following terms. Given query Q, let R_x be a ranked list of all documents of the given dataset sorted in a descend order of their rank scores based on a linear combination of their BM25 weights and learned weights with coefficient x, namely $\sum_{t \in Q} x * w_B(t,d) + (1-x) w_L(t,d)$ for document d. Specifically, there are three ranked lists: R_α, R_β, and R_γ. 2GTI maintains 3 queues Q_{Gl}, Q_{Lo}, and Q_{Rk} with 3 dynamically updated top k thresholds, θ_{Gl}, θ_{Lo}, θ_{Rk}. Let Θ_{Gl}, Θ_{Lo}, Θ_{Rk} be the final top k threshold of these 3 queues at the end of 2GTI. Namely it is the rank score of k-th document in the corresponding queue. The following fact is true:

$$\theta_{Gl} \leq \Theta_{Gl},\ \theta_{Lo} \leq \Theta_{Lo},\ \text{ and } \theta_{Rk} \leq \Theta_{Rk}.$$

Theorem 1 *Assume the subset of top k documents in each of R_α,R_β, and R_γ is unique after arbitrarily swapping rank positions of documents with the same score. Then any document that appears in top-k positions of R_α, R_β, and R_γ is in the top-k outcome of 2GTI.*

Proof: For any document d that appears in the top k positions of all three ranked lists, $Global(d) \geq \Theta_{Gl} \geq \theta_{Gl}$, $Local(d) \geq \Theta_{Lo} \geq \theta_{Lo}$ and $RankScore(d) \geq \Theta_{Rk} \geq \theta_{Rk}$.

If document d is eliminated by global pruning during 2GTI retrieval, $Global(d) = \Theta_{Gl} = \theta_{Gl}$ and the R_α-based rank score of both document d and $(k+1)$-th document in ranked list R_α has to be Θ_{Gl}. Then the subset of top k documents in R_α is not unique after arbitrarily swapping rank positions of documents with the same score, which is a contradiction.

With the same reason, we can argue that document d cannot be eliminated by local pruning or rejected by θ_{Rk} when being added to Q_{Rk} during 2GTI retrieval. Then this

document has to appear in the final outcome of 2GTI. ■

The following two propositions analyze when 2GTI performs better in relevance than a two-stage search algorithm called $R2_{\alpha,\gamma}$ which fetches top k results from list R_α, and then re-ranks using the scoring formula of R_γ.

Theorem 2 *Assume the subset of top k documents in each of R_α,R_β, and R_γ is unique after arbitrarily swapping rank positions of documents with the same score. If 2GTI is configured with $\alpha = \beta$ or $\beta = \gamma$, the average R_γ-based rank score of the top k documents produced by 2GTI is no less than that of two-stage algorithm $R2_{\alpha,\gamma}$.*

Proof: We let $R2[k]$ denote the top k document subset in the outcome of $R2_{\alpha,\gamma}$. To prove this proposition, we compare the average R_γ-based rank score of documents in $R2[k]$ and that in Q_{Rk} at the end of 2GTI. Notice that for any document d satisfying $d \in R2[k]$, it is in the top k results of ranked list R_α and this top k subset is deterministic based on the assumption of this proposition. Then d cannot be eliminated by global pruning in 2GTI.

Given any document d satisfying $d \in R2[k]$ and $d \notin Q_{Rk}$ at the end of 2GTI, it is either eliminated by local pruning with threshold Θ_{Lo} or by top k thresholding of Queue Q_{Rk} with threshold θ_{Rk}. In the later case, $RankScore(d) \leq \theta_{Rk} \leq \Theta_{Rk}$. When d is eliminated by local pruning, global pruning has to use a different formula because d is not eliminated by global pruning, and then 2GTI has to be configured with $\beta = \gamma$ instead of $\alpha = \beta$. In that case local pruning is identical to elimination with top k threshold of Q_{Rk}. Then $RankScore(d) \leq \theta_{Rk} \leq \Theta_{Rk}$.

Since the size of both $R2[k]$ and Q_{Rk} is k, $|R2[k] - R2[k] \cap Q_{Rk}| = |Q_{Rk} - R2[k] \cap Q_{Rk}|$.

We can derive:

$$\begin{aligned}
&\sum_{d \in R2[k]} RankScore(d) \\
&= \sum_{d \in R2[k] \cap Q_{Rk}} RankScore(d) + \sum_{d \in R2[k], d \notin Q_{Rk}} RankScore(d) \\
&\leq \sum_{d \in R2[k] \cap Q_{Rk}} RankScore(d) + \sum_{d \in R2[k], d \notin Q_{Rk}} \Theta_{Rk} \\
&= \sum_{d \in R2[k] \cap Q_{Rk}} RankScore(d) + \sum_{d \notin R2[k], d \in Q_{Rk}} \Theta_{Rk} \\
&\leq \sum_{d \in R2[k] \cap Q_{Rk}} RankScore(d) + \sum_{d \notin R2[k], d \in Q_{Rk}} RankScore(d) \\
&= \sum_{d \in Q_{Rk}} RankScore(d).
\end{aligned}$$

Thus

$$\frac{1}{k} \sum_{d \in R2[k]} RankScore(d) \leq \frac{1}{k} \sum_{d \in Q_{Rk}} RankScore(d).$$

■

Definition 1. For a dataset in which documents are only labeled relevant or irrelevant for any test query, we call ranked list R_x *outmatches* R_y if whenever R_y orders a pair of relevant and irrelevant documents correctly for a query, R_x also orders them correctly.

Theorem 3 *Assume documents in a dataset are only labeled as relevant or irrelevant for a test query. Given a query, when R_γ outmatches R_β, which outmatches R_α, 2GTI retrieves equal or more relevant documents in top-k positions than two-stage algorithm $R2_{\alpha,\gamma}$.*

Proof: When 2GTI completes its retrieval for a query, we count the number of relevant documents in top k positions of list R_α, queue Q_{Lo}, and queue Q_{Rk} as c_α, c_β, and c_γ, respectively. To show $c_\alpha \leq c_\beta$, we initialize them as 0 first and run the following

loop to compute c_α and c_β iteratively. The loop index variable i varies from k, $k-1$, until 1, and at each iteration we look at document x at Position i of R_α, and document y at Position i of Q_{Lo}. Let L_x and L_y be their binary label by which value 1 means relevant and 0 means irrelevant.

- If $L_x = L_y$, we add L_x to both c_α and c_β. Continue this loop.
- Now $L_x \neq L_y$. If $L_x = 0$, $L_y = 1$, we add 1 to c_β, and continue the loop. If $L_x = 1$, $L_y = 0$, there are two cases:
  - If x is within top i positions of current Q_{Lo}, we add 1 to both c_α and c_β. Swap the positions of documents x and y in Q_{Lo}. Continue the loop.
  - If x is not within top i positions of Q_{Lo}, since x is in the top k of R_α, it cannot be globally pruned and it will be evaluated by 2GTI for a possibility of entering Q_{Lo}. If x is ranked before y in list R_α, and since R_β outmatches R_α, x has to be ranked before y in both R_β and Q_{Lo}. That is a contradiction. If x is ranked after y in R_α, we swap the positions of x and y in R_α. Continue the loop.

The above process repeats and moves to a higher position until $i = 1$. When $i = 1$, with top-1 document x in R_α and top-1 y in Q_{Lo}, the only possible cases are $L_x = L_y$ or $L_x = 0$ and $L_y = 1$. Therefore, at the end of the above process, $c_\beta \geq c_\alpha$.

Similarly, we can verify that $c_\gamma \geq c_\beta$ since R_γ outmatches R_β. Therefore $c_\gamma \geq c_\beta \geq c_\alpha$. The number of relevant documents up to position k retrieved for 2GTI is c_γ while the number of relevant documents up to position k retrieved for $R2_{\alpha,\gamma}$ is c_α. Thus this proposition is true. ■

The above analysis indicates that the top documents agreed by three rankings R_α, R_β, and R_γ are always kept on the top by 2GTI, and a properly configured 2GTI al-

gorithm could outperform a two-stage retrieval and re-ranking algorithm in relevance, especially when ranking R_γ outmatches R_β and R_β outmatches R_α for a query. Since two-stage search with neural re-ranking conducted after BM25 retrieval is well adopted in the literature, this analysis provides useful insight into the "worst-case" relevance competitiveness of 2GTI with two-level pruning. GTI can be considered as a special case of 2GTI with $\alpha = \beta = 1$ when the same index is used, and the above three propositions are true for GTI. 2GTI provides more flexibility in pruning with quality control than GTI and Section 3.5 further evaluates their relevance difference.

3.4.3 Alignment of tokens and weights

The BM25 model is usually built on word-level tokenization on the original or expanded document sets and the popular expansion method uses DocT5Query with the same tokenization method. When a learned representation uses a different tokenization method such as BERT's WordPiece based on subwords from BERT vocabulary, we need to align it with BM25 for a consistent term reference. For example, when using BM25 to guide the traversal of SPLADE index, the WordPiece tokenizer is used for a document expanded with DocT5Query before BM25 weighting is applied to each token. Once tokens are aligned, from the index point of view, the same token has two different posting lists based on BM25 weights and based on SPLADE. To merge them when postings do not align one-to-one, the missing weight is set to zero as proposed in [31]. We call this zero-filling alignment. As alternatives, we propose two more methods to fill missing weights with better weight smoothness.

- **One-filling alignment**. We assign 1 as term frequency for a missing token in the BM25 model while this token appears in the learned token list of a document. The justification is that a zero weight is to be too abrupt when such a term is

considered to be useful for a document based on a learned neural model. Having term frequency one means that this token is present in the document, even with the lowest value.

- **Scaled alignment**. This alternative replaces the missing weights in the BM25 model based on a scaled learned score by using the ratio of mean values of non-zero weights in both models. For document ID d that contains term t, let its BM25 weight be $w_B(t,d)$ and its learned weight be $w_L(t,d)$. Let $w_B^*(t,d)$ be an adjusted BM25 weight. Set P_B contains all posting records with nonzero BM25 weights. Set P_L contains posting records with non-zero learned weights. Then $w_B^*(t,d)$ is defined as:

$$w_B^*(t,d) = \begin{cases} w_B(t,d), & w_B(t,d) \neq 0, \\ \frac{\sum_{(t',d')\in P_B} w_B(t',d')/|P_B|}{\sum_{(t',d')\in P_L} w_L(t',d')/|P_L|} w_L(t,d), & w_B(t,d) = 0. \end{cases}$$

3.5 Evaluations

Table 3.1: Model characteristics with MS MARCO Dev set

Index	Avg. Q Length	#Postings	Size	Merged
MS MARCO passages				
BM25-T5	4.5 (4.5)	644M	1.2G	-
DeepImpact	4.2 (4.2)	644M	2.6G	2.6G
BM25-T5-B	6.6 (6.6)	699M	1.2G	-
UniCOIL	6.6 (686.3)	592M	1.5G	1.7G
SPLADE++	23.3 (867.6)	2.62B	5.6G	8.3G
MS MARCO documents				
BM25-T5-B	6.8 (7.0)	3.39B	5.4G	-
UniCOIL	6.6 (685.0)	3.04B	7.0G	8.3G

Datasets and settings. Our evaluation uses the MS MARCO document and passage collections [36, 71], and 13 publicly available BEIR datasets [37]. The results for the

BEIR datasets are described in Appendix 3.6. For MS MARCO, the contents in the document collections are segmented during indexing and re-grouped after retrieval using "max-passage" strategy following [72]. There are 8.8M passages with an average length of 55 words, and 3.2M documents with an average length of 1131 words before segmentation. The Dev query set for passage and document ranking has 6980 and 5193 queries respectively with about one judgment label per query. Each of the passage/document ranking task of TREC Deep Learning (DL) 2019 and 2020 tracks provides 43 and 54 queries respectively with many judgment labels per query.

In producing an inverted index, all words use lower case letters. Following GT, we packed the learned score and the term frequency in the same integer. For DeepImpact, we adopt GT's index[1] directly. The BM25-T5's index is dumped from the DeepImpact index. Both BM25-T5 and DeepImpact are using natural words tokenization.

SPLADE and uniCOIL use the BERT's Word Piece tokenizer. In order to align with them, the BM25-T5-B index reported in the following tables uses the same tokenizer as well. The impact scores of uniCOIL is obtained from Pyserini [72] [2]. For SPLADE, in order to achieve the best performance, we retrained the model following the setup in SPLADE++ [30]. We start from the pretrained model coCondenser [73] and distill using the sentenceBERT hard negatives [3] from a cross-encoder teacher [74] with Margin-MSE loss. For FLOP regularization, we use 0.01 and 0.008 for query and documents respectively. We construct the inverted indexes, convert them to the PISA format, and compress them using SIMD-BP128 [49] following [50, 31].

Table 3.1 shows the dataset and index characteristics of the different weighting models on the MS MARCO Dev dataset. Following [58], we assume that a query can be pre-processed with a "pseudo-document" trick that assigns custom weights to query terms in

[1]https://github.com/DI4IR/dual-score
[2]https://github.com/castorini/pyserini/blob/master/docs/experiments-unicoil.md
[3]https://huggingface.co/datasets/sentence-transformers/msmarco-hard-negatives

uniCOIL and SPLADE. Therefore, there may be token repetition in each query to reflect token weighting. Column 1 is the mean query length in tokens without or with counting duplicates. Column 3 is the inverted index size while the last column is the size after merging BM25 and learned weights in the index.

The C++ implementation of 2GTI with the modified MaxScore and VBMW algorithms are embedded in PISA [48], and the code will be released in `https://github.com/Qiaoyf96/2GTI`. Our evaluation using this implementation runs as a single thread on a Linux server with Intel i5-8259U 2.3GHz and 32GB memory. Weights are chosen by sampling queries from the MS MARCO training dataset.

Metrics. For MS MARCO Dev set, we report the relevance in terms of mean reciprocal rank (MRR@10 on passages and MRR@100 on documents), following the official leader-board standard. We also report the recall@k ratio which is the percentage of relevant-labeled results appeared in the final top-k results. For TREC DL test sets, we report normalized discounted cumulative gain (nDCG@10) [46]. The above reporting follows the common practice of the previous work (e.g. [56, 29, 67, 16]).

Before timing queries, all compressed posting lists and metadata for tested queries are pre-loaded into memory, following the same assumption in [75, 24]. Retrieval mean response times (MRT) are reported in milliseconds. The 99th percentile time (P_{99}) is reported within parentheses in the tables below, corresponding to the time occurring in the 99th percentile denoted as tail latency in [76].

Statistical significance. For the reported numbers on MS MARCO passage and document Dev sets in the rest of this section, we have performed a pairwise t-test on relevance difference between 2GTI and a GTI baseline, and between 2GTI and the original learned sparse retrieval without BM25 guidance. No statistically significant degradation has been observed at the 95% confidence level. We have also performed a pairwise t-test comparing the reported relevance numbers of 2GTI and GTI and mark '†' in the

evaluation tables if there is a statistically significant improvement by 2GTI over GTI at the 95% confidence level. We do not perform a t-test on DL'19 and DL'20 query sets as the number of queries in these sets is small.

Table 3.2: A Comparison of 2GTI, GTI and the original method with no BM25 guidance for MaxScore ($k = 10$)

	MS MARCO Dev		DL'19		DL'20	
Method	**MRR (Recall)**	**MRT (P_{99})**	**nDCG (Recall)**	**MRT**	**nDCG (Recall)**	**MRT**
SPLADE++, Passages.						
BM25-T5-B	0.2611 (0.5179)	1.7 (8.7)	0.5931 (0.1556)	0.8	0.5981 (0.2034)	1.0
SPLADE++-Org	0.3937 (0.6801)	121 (483)	0.7304 (0.1776)	135	0.7290 (0.2437)	138
-GT	0.2720 (0.5246)	121 (438)	0.6379 (0.1677)	130	0.6106 (0.2192)	139
-GTI	0.2687 (0.5209)	118 (440)	0.6352 (0.1669)	131	0.6083 (0.2190)	139
-2GTI-Accurate	**0.3939**† **(0.6812)**	31.1 (171)	**0.7401 (0.1846)**	31.9	**0.7278 (0.2480)**	36.8
-2GTI-Fast	0.3934† (0.6792)	**22.7 (116)**	0.7380 (0.1837)	**23.5**	**0.7278 (0.2480)**	**26.2**
UniCOIL, Passages.						
BM25-T5-B	0.2611 (0.5179)	1.7 (8.7)	0.5931 (0.1556)	0.8	0.5981 (0.2034)	1.0
UniCOIL-Org	0.3516 (0.6168)	10.5 (102)	0.7027 (0.1761)	10.4	0.6746 (0.2346)	14.2
-GT	0.3347 (0.5639)	**2.1 (11.2)**	0.6990 (0.1770)	1.7	0.6769 (0.2444)	**2.4**
-GTI	0.3384 (0.5678)	**2.1 (11.2)**	0.6959 (0.1733)	**1.6**	0.6739 (0.2422)	**2.4**
-2GTI-Accurate	**0.3550**† **(0.6205)**	3.3 (19.2)	**0.7135 (0.1769)**	2.4	**0.6891 (0.2451)**	3.4
-2GTI-Fast	0.3548† (0.6193)	2.6 (14.3)	**0.7135 (0.1769)**	1.9	**0.6891 (0.2451)**	2.8
UniCOIL, Documents.						
BM25-T5-B	0.2716 (0.4749)	2.7 (13.9)	0.4246 (0.0741)	3.6	0.4463 (0.1693)	3.1
UniCOIL-Org	0.3313 (0.5638)	26.0 (252)	0.5477 (0.0880)	28.1	0.4996 (0.1920)	27.3
-GT	0.3280 (0.5455)	**6.8 (36.2)**	0.5199 (0.0779)	6.2	0.4903 (0.1871)	**7.1**
-GTI	0.3334 (0.5531)	6.9 (36.6)	0.5223 (0.0781)	**6.1**	0.4905 (0.1857)	7.2
-2GTI-Accurate	**0.3423**† **(0.5710)**	10.2 (56.4)	**0.5486 (0.0852)**	8.9	**0.4998 (0.1886)**	10.5
-2GTI-Fast	0.3418† (0.5663)	7.4 (40.4)	0.5453 (0.0847)	6.5	0.4997 (0.1845)	8.2
DeepImpact, Passages.						
BM25-T5	0.2723 (0.5319)	0.7 (4.7)	0.6283 (0.1611)	0.3	0.6321 (0.2218)	0.5
DeepImpact-Org	0.3276 (0.5844)	9.4 (73.3)	0.6964 (0.1698)	4.9	0.6524 (0.2035)	7.8
-GT	0.3276 (0.5805)	**0.7 (4.7)**	0.6997 (0.1698)	**0.3**	0.6682 (0.2194)	**0.5**
-GTI	0.3375 (0.5866)	**0.7 (4.7)**	0.6953(0.1690)	**0.3**	0.6846 (**0.2372**)	**0.5**
-2GTI-Accurate	**0.3405**† **(0.5987)**	1.1 (8.1)	**0.7065 (0.1703)**	0.7	0.6850 (0.2361)	1.2
-2GTI-Fast	0.3395† (0.5934)	**0.7** (5.1)	0.7045 (0.1693)	0.4	**0.6882** (0.2371)	0.6

Overall results with MS MACRO. Table 3.2 and 3.3 lists a comparison of 2GTI with the baseline using three sparse representations for retrieval on MS MARCO and TREC DL datasets. 2GTI uses scaled filling alignment as default while GTI uses zero filling as specified in [31]. The γ value is chosen the same for GTI and 2GTI for each representation, which is the best for most of cases. The "accurate" configuration denotes the one that reaches the highest relevance score. The "fast" configuration denotes the

Table 3.3: A Comparison of 2GTI, GTI and the original method with no BM25 guidance for MaxScore ($k = 1000$)

	MS MARCO Dev		DL'19		DL'20	
Method	**MRR (Recall)**	**MRT (P_{99})**	**nDCG (Recall)**	**MRT**	**nDCG (Recall)**	**MRT**
SPLADE++, Passages.						
BM25-T5-B	0.2611 (0.9361)	9.2 (28.4)	0.5931 (0.7608)	6.4	0.5981 (0.7641)	7.4
SPLADE++-Org	0.3937 (0.9832)	278 (819)	0.7304 (0.8286)	317	0.7290 (0.8287)	307
-GT	0.2973 (0.9648)	336 (1048)	0.6636 (0.8030)	330	0.6605 (0.8072)	344
-GTI	0.2961 (0.9648)	332 (1059)	0.6595 (0.8025)	318	0.6587 (0.8066)	333
-2GTI-Accurate	**0.3946**† (0.9799)	109 (478)	**0.7394** (0.8209)	123	0.7297 (**0.8339**)	132
-2GTI-Fast	0.3937† (0.9662)	**43.1 (144)**	**0.7394** (0.8218)	**42.1**	**0.7306** (0.8205)	**45.0**
UniCOIL, Passages.						
BM25-T5-B	0.2611 (0.9361)	9.2 (28.4)	0.5931 (0.7608)	6.4	0.5981 (0.7641)	7.4
UniCOIL-Org	0.3516 (0.9582)	35.3 (197)	0.7027 (0.7822)	38.6	0.6746 (0.7758)	42.8
-GT	0.3514 (0.9458)	**10.6** (33.9)	0.7028 (0.7857)	**10.3**	0.6746 (0.7741)	**10.8**
-GTI	0.3552 (0.9468)	**10.6 (33.2)**	**0.7130 (0.7917)**	**10.3**	**0.6899 (0.7857)**	**10.8**
-2GTI-Accurate	**0.3554** (0.9566)	16.9 (68.4)	**0.7130** (0.7904)	15.5	**0.6899** (0.7823)	16.5
-2GTI-Fast	0.3552 (0.9468)	**10.6 (33.2)**	0.7129 (**0.7917**)	**10.3**	**0.6899 (0.7857)**	**10.8**
UniCOIL, Documents.						
BM25-T5-B	0.2950 (0.9197)	12.4 (50.0)	0.5594 (0.5690)	15.0	0.5742 (0.7148)	15.7
UniCOIL-Org	0.3530 (0.9426)	71.0 (447)	0.6415 (0.5864)	79.2	0.6059 (0.7502)	86.0
-GT	0.3530 (0.9361)	20.9 (73.8)	0.6445 (0.5842)	**21.6**	0.6059 (0.7444)	22.7
-GTI	0.3639 (0.9368)	**20.6 (72.9)**	**0.6581** (0.5920)	21.7	**0.6156** (0.7445)	22.6
-2GTI-Accurate	**0.3644 (0.9422)**	33.2 (149)	**0.6581 (0.5932)**	32.8	**0.6156 (0.7477)**	37.4
-2GTI-Fast	0.3639 (0.9368)	**20.6** (73.0)	**0.6581** (0.5920)	21.7	**0.6156** (0.7445)	**22.5**
DeepImpact, Passages.						
BM25-T5	0.2723 (0.9348)	4.9 (21.3)	0.6283 (0.7704)	4.6	0.6321 (0.7598)	5.6
DeepImpact-Org	0.3276 (0.9474)	23.8 (97.0)	0.6964 (0.7623)	25.5	0.6524 (0.7534)	38.9
-GT	0.3276 (0.9454)	**5.1 (21.1)**	0.6964 (0.7745)	4.9	0.6527 (0.7574)	5.8
-GTI	0.3413 (0.9455)	5.2 (21.7)	**0.7072** (0.7871)	**4.7**	**0.6854** (0.7745)	**5.7**
-2GTI-Accurate	**0.3414** (0.9469)	7.4 (33.0)	**0.7072 (0.7875)**	7.8	**0.6854 (0.7778)**	9.1
-2GTI-Fast	0.3413 (0.9455)	5.2 (21.7)	**0.7072** (0.7871)	**4.7**	**0.6854** (0.7745)	**5.7**

one that reaches a relevance score within 1% of the accurate configuration while being much more faster.

2GTI vs. GTI in SPLADE++. Table 3.2 and 3.3 shows 2GTI with default scaled filling significantly outperforms GTI with default zero filling for SPLADE++, where BM25 index is not well aligned. "SPLADE++-Org" denotes the original MaxScore retrieval performance using SPLADE++ model trained by ourselves and its MRR@10 number is higher than what has been reported in [30]. When $k = 1,000$, GT is slightly better than GTI, and with the fast configuration, MRR@10 of 2GTI is 32.4% higher than that of GT while 2GTI is 7.8x faster than GT for the Dev set. The significant increase in nDCG@10 and decrease in the MRT are also observed in DL'19 and DL'20. When $k = 10$, there is also a large relevance increase and time reduction from GTI or GT to 2GTI for all three test sets. For example, the relevance is 46.4% or 44.6% higher and the mean latency is 5.2x or 5.3x faster for the Dev set.

Compared to the original MaxScore method, 2GTI has about the same relevance score for both $k = 10$ and $k = 1,000$ while having much smaller latency. For example, 6.5x reduction (278ms vs. 43.1ms) for the Dev passage set when $k = 1,000$ and 5.3x reduction when $k = 10$ (121ms vs 22.7ms) with the 2GTI-fast configuration.

2GTI vs. GTI in DeepImpact and uniCOIL. As shown in Table 3.2 and 3.3, GTI (or GT) performs very well for $k = 1,000$ in both DeepImpact and uniCOIL in speeding up retrieval while maintaining a relevance similar as the original retrieval. The two-level differentiation for dynamic index pruning does not improve relevance or shorten retrieval time. This can be explained as BM25-T5 index is well aligned with the DeepImpact index and with the uniCOIL index. Also because of this reason, filling to address index alignment is not needed with no improvement in these two cases.

When k decreases from 1,000 to 10, as shown in Figure 3.1 discussed in Section 3.3, the recall ratio starts to drop, and relevance effectiveness degrades. When $k = 10$ as

shown in Table 3.2 and 3.3, DeepImpact-2GTI-fast can increase MRR@10 from 0.3375 by GTI to 0.3395 for the Dev set and deliver slightly higher MRR@10 or nDCG@10 scores than GTI in DL'19 and DL'19 sets. For uniCOIL, 2GTI-fast increases MRR@10 from 0.3384 by GTI to 0.3548 for the Dev set and increases nDCG@10 from 0.6959 to 0.7135 for DL'19. There is also a modest relevance increase for DL'20 passages with $k = 10$ and a similar trend is observed for the document retrieval task. The price paid for 2GTI is its retrieval latency increase while its latency is still much smaller than the original retrieval time.

Design options with weight alignment and threshold over-estimation. Table 3.4 examines the impact of weight alignment and a design alternative based on threshold over-estimation for MS MARCO passage Dev set using SPLADE++ when $k = 10$. In the top portion of this table, threshold over-estimation by a factor of F (1.1, 1.3, and 1.5) is used in the original retrieval algorithm without BM25 guidance, and these factor choices are similar as ones in [62, 63, 64]. That essentially sets $\alpha = 0$, $\beta = 0$, and $\gamma = 0$ while multiplying θ_{Gl} and θ_{Lo} by the above factor in 2GTI. The result shows that even threshold over-estimation can reduce the retrieval time, relevance reduction is significant, meaning that the aggressive threshold used causes incorrect dropping of some desired documents.

The second portion of Table 3.4 examines the impact of different weight filling methods described in Section 3.4.3 for alignment when they are applied to GTI and 2GTI, respectively. In both cases, scaled filling marked as "/s" is most effective while one-filling marked as "/1" outperforms zero-filling marked as "/0" also. The MRT of 2GTI/s becomes 10.5x smaller than 2GTI/0 while there is no negative impact to its MRR@10. The MRT of GTI/s is about 13.0x smaller than GTI/0 while there is a large MRR@10 number increase.

A validation on 2GTI's properties. To corroborate the competitiveness analysis

Table 3.4: Impact of design options on MS MARCO passages

SPLADE++. $k = 10$	**MRR@10**	**Recall@10**	**MRT**	P_{99}
Threshold over-estimation				
Original	0.3937	0.6801	121	483
- $F = 1.1$	0.3690	0.5707	107	457
- $F = 1.3$	0.3210	0.4393	95.0	420
- $F = 1.5$	0.2825	0.3670	88.2	393
Weight alignment for GTI ($\alpha = 1, \beta = 1, \gamma = 0.05$)				
GTI/0	0.2687	0.5209	118	440
GTI/1	0.3036	0.5544	26.7	114
GTI/s	0.3468	0.5774	9.1	36.1
Weight alignment for 2GTI-Accurate ($\alpha = 1, \beta = 0, \gamma = 0.05$)				
2GTI/0	0.3933	0.6799	328	1262
2GTI/1	0.3933	0.6818	89.3	393
2GTI/s	0.3939	0.6812	31.1	171

in Section 4.3.2, Table 3.5 gives MRR@10 scores and retrieval times in milliseconds of the algorithms with different configurations on the Dev set of MS MARCO passages with $k = 10$ and SPLADE++ weights. The result shows that the listed configurations of 2GTI have a higher MRR@10 number than 2-stage search $R2_{\alpha,\gamma}$, and 2GTI with $\alpha = \beta = 1$ that behaves as GTI. MRR@10 of ranking with a simple linear combination of BM25 and learned weights is only slightly higher than 2GTI, but it is much slower.

Table 3.5: A validation on 2GTI's properties. $k = 10$

	MRR@10	**Recall@10**	**MRT**	P_{99}
$R2_{\alpha,\gamma}$ (BM25 retri. SPLADE++ rerank)	0.3461	0.5179	-	-
GTI/s ($\alpha = \beta = 1$, $\gamma = 0.05$)	0.3468	0.5774	9.1	36.1
2GTI/s ($\alpha = 1$, $\beta = \gamma = 0.05$)	0.3939	0.6812	29.8	165
2GTI/s-Accurate (α=1, β=0, γ=0.05)	0.3939	0.6812	31.1	171
2GTI/s-Fast ($\alpha = 1$, $\beta = 0.3, \gamma = 0.05$)	0.3934	0.6792	22.7	116
Linear comb. ($\alpha = \beta = \gamma = 0.05$)	0.3946	0.6805	120	477

Sensitivity on weight distribution. We have distorted the SPLADE++ weight distribution in several ways to examine the sensitivity of 2GTI and found that 2GTI is still effective. For example, we apply a square root function to the neural weight of every

Table 3.6: Use of 2GTI with a new SPLADE model [47]. $k = 10$

BT-SPLADE-L	**MRR@10**	**Recall@10**	**MRT**	P_{99}
Original MaxScore	0.3799	0.6626	17.4	59.4
2GTI/s (α=1, β=0.3, γ=0.05)	0.3772	0.6584	8.0	27.5
GTI/s ($\alpha = \beta = 1$, $\gamma = 0.05$)	0.3284	0.5520	6.6	24.9

token in MS MARCO passages, the relevance score of both original retrieval and 2GTI drops to 0.356 MRR@10 due to weight distortion, while 2GTI is 5.0x faster than the original MaxScore when $k = 10$.

Efficient SPLADE model. Table 3.6 shows the application of 2GTI in a recently published efficient SPLADE model [47] which has made several improvements in retrieval speed. We have used the released checkpoint of this efficient model called BT-SPLADE-L, which has a weaker MRR@10 score, but significantly faster than our trained SPLADE baseline reported in Table 3.2 and 3.3. When used with this new SPLADE model, 2GTI/s-Fast version results in a 2.2x retrieval time speedup over MaxScore. Its MRR@10 is higher than GTI/s and has less than 1% degradation compared to the original MaxScore.

3.6 Additional Evaluation Results

Impact of α and β adjustment on 2GTI. Figure 3.3 examines the impact of adjusting parameters α and β on global and local pruning for the MS MARCO Dev passage test set when $k = 10$ in controlling the influence of BM25 weights for SPLADE++ (left) and uniCOIL (right). The x axis corresponds to the latency increase while y axis corresponds to the MRR@10 or nDCG@10 increase. The results for MS MARCO DL'19, and DL'20 are similar.

The red curve connected with dots fixes $\beta = 1$ and varies α from 1 at the left end to 0 at the right end. As α decreases from 1 to 0, the latency increases because BM25

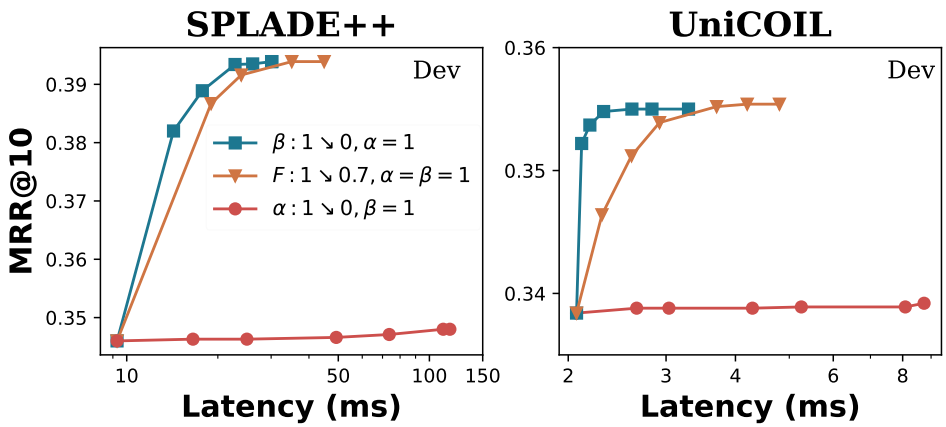


Figure 3.3: Controlling influence of BM25 on pruning

influences diminish at the global pruning level and fewer documents are skipped. The relevance for this curve is relatively flat in general and lower than that of the blue curve, representing the global level BM25 guidance reduces time significantly, while having less impact on the relevance.

The blue curve connected with squares fixes $\alpha = 1$ at the global level and varies β from 1 at the left bottom end to 0 at the right top end. Decreasing β value is positive in general for relevance towards some point as BM25 influence decreases gradually at the local level and after such a point, the relevance gain becomes much smaller or negative. For example, after β in the blue curve in SPLADE++ becomes 0.3 for the Dev set, its additional decrease does not lift MRR@10 visibly anymore while the latency continues to increase, which indicates the relevance benefit has reached the peak at that point. Our experience with the tested datasets is that the parameter setting for 2GTI can reach a relevance peak typically when α is close to 1 and β varies between 0.3 and 1.

Note that even the above result advocates that α is close to 1, α and β still have different values to be more effective for the tested data, reflecting the usefulness of two-level pruning control.

Threshold under-estimation. In Figure 3.3, the brown curve connected with triangles fixes $\alpha = \beta = 1$ and under-estimates the skipping threshold by a factor of F at the local and global levels. That behaves like GTI coupled with scaled weight filling as a special case of 2GTI. F varies from 1 at the left bottom end to 0.7 at the right top end of this brown curve. As F decreases, the skipping threshold becomes very loose and there is less chance that desired documents are skipped. Then retrieval relevance can improve while retrieval time can increase substantially. Comparing with the blue curve that adjusts β, retrieval takes a much longer time in the brown curve to reach the peak relevance, as shown in this figure, and the brown curve is generally placed on the right side of the blue curve. For example on the Dev set with uniCOIL, the brown curve with threshold under-estimation reaches the best relevance at mean latency 3.7ms while the blue curve with β adjustment reaches the same peak at mean latency 2.3ms, which is 1.6x faster.

Zero-shot performance on the BEIR datasets. We evaluate the zero-shot ranking effectiveness and response time of 2GTI using the 13 search and semantic relatedness datasets from the BEIR collection. Our training of SPLADE++ model is only based on MS MARCO data without using any BEIR data. Table 4.4 lists the nDCG@10 scores of original MaxScore on SPLADE++, 2GTI/s-Fast (α=1, β=0.3, γ=0.05) and GTI (α=β=1, γ=0.05). The retrieval depth is $k = 10$ and $k = 1000$. This table also reports mean response time of retrieval in milliseconds. The SPLADE++ model trained by ourself has an average nDCG@10 score 0.500 close to 0.507 reported in the SPLADE++ paper [30]. The original MaxScore's nDCG@10 score does not change when $k = 10$ and $k = 1000$.

When $k = 10$, 2GTI has almost identical nDCG@10 scores as the original MaxScore while 2GTI is on average 2.0x faster than MaxScore for these BEIR datasets. When GTI runs on the same index data, its average nDCG@10 score is 0.43 MRR@10 and it

.

Table 3.7: Zero-shot relevance in NDCG@10 and retrieval latency in milliseconds on BEIR datasets with SPLADE++

	Original MaxScore		2GTI/s-Fast		GTI/s	
Dataset	**nDCG**	**MRT**	**nDCG**	**MRT**	**nDCG**	**MRT**
k=**10**						
DBPedia	0.447	99.0	**0.449**	34.5	0.306	10.6
FiQA	**0.355**	5.1	0.354	3.4	0.256	0.8
NQ	**0.551**	72.9	**0.551**	28.6	0.524	7.3
HotpotQA	**0.681**	453	**0.681**	191	0.549	46.8
NFCorpus	**0.351**	0.3	0.347	0.2	0.327	0.1
T-COVID	0.705	15.9	**0.707**	9.9	0.569	2.6
Touche-2020	**0.291**	8.7	**0.291**	3.2	0.237	1.3
ArguAna	0.446	8.8	0.448	4.0	**0.454**	4.0
C-FEVER	**0.234**	635	0.231	355	0.196	241
FEVER	**0.781**	1028	0.771	655	0.590	160
Quora	**0.817**	21.5	**0.817**	6.7	0.763	1.7
SCIDOCS	**0.155**	3.7	**0.155**	2.1	0.140	1.2
SciFact	**0.682**	3.2	0.680	2.9	0.680	1.7
Average	**0.500**	-	0.499	2.0x	0.430	6.1x
k=**1000**						
Average	0.500	-	**0.501**	2.5x	0.496	2.7x

is faster than 2GTI with an average 6.1x speedup over the original MaxScore for these datasets. Two-level pruning in 2GTI can preserve relevance better than GTI and this is consistent with what we have observed for searching MS MARCO passages.

When $k = 1000$, the guided traversal algorithms have a better chance to retain relevance. 2GTI has a slightly higher average relevance of 0.501 MRR@10 than that with $k = 10$ and it is about 2.5x faster on average than the original MaxScore. For GTI running on the same index with the same alignment, the average MRR@10 is 0.496 whil average speedup 2.7x over MaxScore. Its relevance score is close to that of 2GTI as BM25-driven pruning under a large k value can still keep a good recall ratio.

3.7 Two-level guidance for BMW

Two-level guidance can be adopted to control index traversal of a BMW based algorithm such as VBMW as well because we can also view that such an algorithm conducts a sequence of index traversal steps, and can differentiate its index pruning of each traversal step at the global inter-document and local intra-document levels. We use the same symbol notations as in the previous subsection, assuming the posting lists are sorted by an increasing order of their document IDs. We still keep a position pointer in each posting list of search terms to track the current document ID d_{t_i} being handled for each term t_i, incrementally accumulate three scores $Global(d)$, $Local(d)$, and $RankScore(d)$ for each document d visited, and maintain three separate score-sorted queues Q_{Gl}, Q_{Lo}, and Q_{Rk}.

- **Pruning at the global inter-document level with pivot identification.** BMW [3] keeps a sorted search term list in each traversal step so that $d_{t_i} \leq d_{t_{i+1}}$ with $1 \leq i \leq N-1$. The pivot position that partitions these current document pointers is the smallest integer called $pivot$ such that $\sum_{i=1}^{pivot} \alpha\sigma_B[i] + (1-\alpha)\sigma_L[i] >$

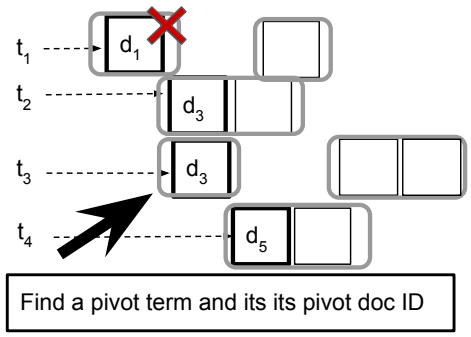


Figure 3.4: Global pruning in BMW

θ_{Gl}. This inequality means that any document ID d where $d_{t_0} \leq d < d_{t_{pivot}}$ does not qualify for being in the final top k list based on score $Global(d)$. Then with the above pivot detection, for $1 \leq i < pivot$, the current visitation pointer of the i-th posting list moves to the closest block that contains a document ID equal to or bigger than $d_{t_{pivot}}$.

Figure 3.4 illustrates an example of global pruning in BMW with 4 terms and each posting list maintains a pointer to the current document being visited at a traversal step. Documents in each posting list are covered by a curved rectangle, representing these lists are stored and compressed in a block-wise manner. In the figure, the pivot identification at one traversal step locates document d_3, and document IDs smaller than d_3 are skipped for any further consideration in this traversal step.

- **Local pruning.** Let d be the corresponding pivot document in pivot term t_{pivot}. In Figure 3.4, pivot term $t_{pivot} = t_3$ and $d = d_3$. A traversal procedure is executed to check if detailed scoring of document d can be avoided fully or partially and this procedure can be similar as the one in the revised MaxScore algorithm described earlier. As each posting list is packaged in a block manner in BMW, let $\Delta_B[x]$ and $\Delta_L[x]$ be the BM25 and learned block maximum weights of the block in the

Table 3.8: Guided VBMW and MaxScore with uniCOIL on MS MARCO passages

Dataset	Method	$k = 10$	$k = 20$	$k = 100$
Dev	MaxScore-2GTI	0.355†, 2.6 (14.3)	0.355†, 3.4 (18.4)	0.355, 5.5 (26.0)
	VBMW-2GTI	0.353†, 4.3 (30.6)	0.354†, 5.2 (35.6)	0.355, 8.6 (51.6)
	VBMW-GTI	0.339, 2.4 (14.2)	0.347, 3.0 (17.2)	0.353, 5.4 (27.1)
DL'19	MaxScore-2GTI	0.714, 1.9 (12.9)	0.713, 2.3 (14.2)	0.713, 4.3 (18.6)
	VBMW-2GTI	0.708, 2.0 (20.0)	0.708, 3.7 (23.2)	0.710, 6.6 (33.1)
	VBMW-GTI	0.694, 1.7 (9.0)	0.700, 2.2 (11.7)	0.710, 4.3 (17.9)
DL'20	MaxScore-2GTI	0.689, 2.8 (12.1)	0.689, 3.3 (13.0)	0.689, 5.3 (22.2)
	VBMW-2GTI	0.683, 3.9 (18.4)	0.686, 4.9 (22.6)	0.686, 8.4 (46.8)
	VBMW-GTI	0.676, 2.3 (10.5)	0.680, 2.9 (13.7)	0.685, 5.3 (24.8)

x-th posting list that contains d, respectively, and they are 0 if no such a block exists in this list. The upper bound of $Local(d)$ can be tightened using the block-wise maximum weight instead of the list-wise maximum weight contributed by each term as: $\sum_{i=1}^{N} \beta\Delta_B[i] + (1-\beta)\Delta_L[i]$. When decompressing the needed block of a posting list, the block-max contribution from the corresponding term in the above expression can be replaced by the actual BM25 and learned weights for document d. Then the upper bound of $Local(d)$ is further tightened, which can be directly compared with θ_{Lo} after every downward adjustment.

Evaluations on effectiveness of 2GTI on VBMW. We choose uniCOIL to study the usefulness of VBMW-2GTI in searching the MS MARCO Dev set. SPLADE++ is not chosen because the test queries are long on average and MaxScore is faster than VBMW for such queries. Table 3.8 reports the performance for VBMW-2GTI, VBMW-GTI, and MaxScore-2GTI for passage retrieval with uniCOIL when varying k. Each entry has a report format of $x, y(z)$ where x is MRR@10 for Dev or NDCG@10 for DL'19 and DL'20. y is the MRT in ms, and z is the P_{99} latency in ms. 2GTI uses the fast setting with $\alpha = 1$, $\beta = 0.3$. For both 2GTI and GTI, $\gamma = 0.1$. The result shows 2GTI provides a positive boost in relevance for VBMW compared to GTI when k is 10 and 20. For $k = 100$, the relevance difference is negligible. MaxScore-2GTI is still faster than

Table 3.9: Performance under different query classes with $k = 10$, uniCOIL, and MS MARCO passage Dev set

QLength	≤ 3	4-5	6-7	≥ 8
# Q w/ SW	113	1720	2175	2030
MaxScore-2GTI	0.286, 1.6 (12.2)	0.376, 1.7 (8.9)	0.347, 2.4 (11.5)	0.315, 4.3 (20.9)
VBMW-2GTI	0.289, 2.1 (15.2)	0.373, 2.2 (12.5)	0.346, 3.4 (16.1)	0.313, 8.5 (50.8)
VBMW-GTI	0.257, 1.1 (6.1)	0.346, 1.4 (6.8)	0.334, 2.6 (12.2)	0.307, 7.4 (43.9)
# Q w/o SW	327	445	130	40
MaxScore-2GTI	0.397, 1.3 (9.0)	0.430, 1.9 (9.4)	0.439, 3.7 (12.1)	0.558, 5.4 (15.9)
VBMW-2GTI	0.399, 0.9 (4.0)	0.430, 1.6 (6.3)	0.438, 3.6 (12.7)	0.565, 5.6 (23.3)
VBMW-GTI	0.385, 0.7 (3.4)	0.420, 1.3 (5.1)	0.433, 2.6 (9.9)	0.560, 4.1 (12.9)

VBMW-2GTI on average for all tested queries while their relevance difference is small. we examine below if VBMW-2GTI can be useful for a subset of queries.

Table 3.9 reports the relevance and time of these three algorithms in the passage Dev set for queries subdivided based on their lengths and if a query contains a stop word or not. That is for uniCOIL with $k = 10$, $\alpha = 1$, $\beta = 0.3$, and $\gamma = 0.1$. Each entry has the same report format as in Table 3.8. The result shows that VBMW-2GTI is much faster than MaxScore-2GTI for short queries ($k \leq 5$) that do not contain stop words and VBMW-2GTI has an edge in relevance over VBMW-GTI while being very close to MaxScore-2GTI for this class of queries. The above result suggests that a fusion method can do well by switching the algorithm choice based on query characteristics and VBMW-2GTI can be used for a class of queries.

3.8 Concluding Remarks

The contribution of this paper is a two-level parameterized guidance scheme with index alignment to optimize retrieval traversal with a learned sparse representation. Our formal analysis shows that a properly configured 2GTI algorithm including GTI can outperform a two-stage retrieval and re-ranking algorithm in relevance.

Our evaluation shows that the proposed 2GTI scheme can make the BM25 pruning guidance more accurate to retain the relevance. For MaxScore with SPLADE++ on MS MARCO passages, 2GTI can lift relevance by up-to 32.4% and is 7.8x faster than GTI when $k = 1,000$, and by up-to 46.4% more accurate and 5.2x faster when $k = 10$. In all evaluated cases, 2GTI is much faster than the original retrieval without BM25 guidance. For example, up-to 6.5x faster than MaxScore on SPLADE++ when $k = 10$. We have also observed similar performance patterns on BEIR datasets when comparing 2GTI with GTI and the original MaxScore using SPLADE++ learned weights. Compared to other options such as threshold underestimation to reduce the influence of BM25 weights, the two-level control is more accurate in maintaining the strong relevance with a much lower time cost. While our study is mainly centered with MaxScore-based retrieval, 2GTI can be used for VBMW and our evaluation shows that VBMW-2GTI can be a preferred choice for a class of short queries without stop words when k is small.

Chapter 4

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

4.1 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) [10, 77, 12, 11, 33] and sparse (lexical or learned) [26, 27, 28, 29, 15, 18]. Efficient dense retrieval relies on approximation techniques with notable relevance drops [22, 23, 78, 79], 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 [4], WAND [2], and BlockMax WAND (BMW) [3], 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 [80, 81, 82]. However, the cluster bounds estimated by current methods are often loose, which limits pruning opportunities. Threshold over-estimation [62, 63, 64] 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 [30], uni-

COIL [28], and LexMAE [18] 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.

4.2 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 [1] 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 [24]) 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 [62, 63, 64]. 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 [80, 81] clusters document IDs within a range and estimates a range-based maximum score for pruning. Anytime Ranking [82] extends *cluster skipping inverted index* [83, 84] 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 [85] 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 [31, 51]. Static index pruning [86, 87] removes low-scoring term weights during index generation. An efficient version of SPLADE [47] uses L1

regularization for query vectors, dual document and query encoders, and language model middle training. Term impact decomposition [88] 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 [53] and IOQP [89].

An experimental study [90] 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 4.4 includes Anytime and IOQP.

Big-ANN competition for sparse retrieval. NeurIPS 2023 Big-ANN competition sparse track [91] 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. Section 4.9 provides a comparison of ASC with two top winners of this competition. Reference [92] 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.

4.3 Cluster-based Retrieval with Approximation and Segmentation

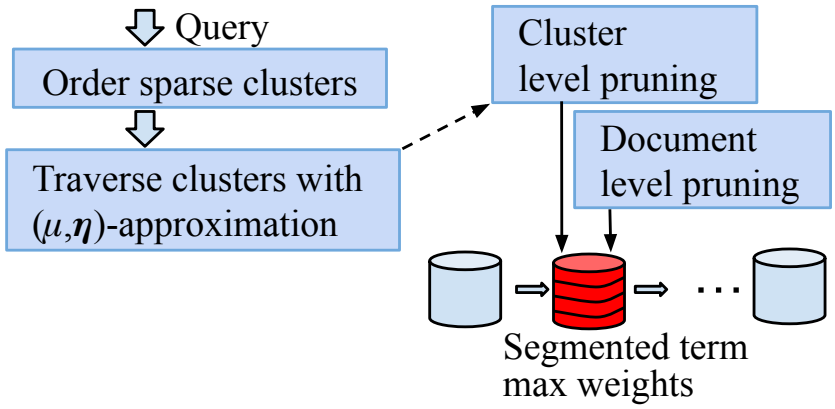


Figure 4.1: Flow of ASC with two-parameter pruning control and segmented cluster-level maximum term weights

The overall online inference flow of the proposed scheme during retrieval is shown in Figure 4.1. 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 [82]. 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.

$$BoundSum(C_i) = \sum_{t \in Q} \max_{d \in C_i} w_{t,d}. \tag{4.1}$$

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 (4.1) can be loose, especially when a cluster contains diverse document vectors, and this reduces the effectiveness of pruning. As an illustration, Figure 4.2 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.

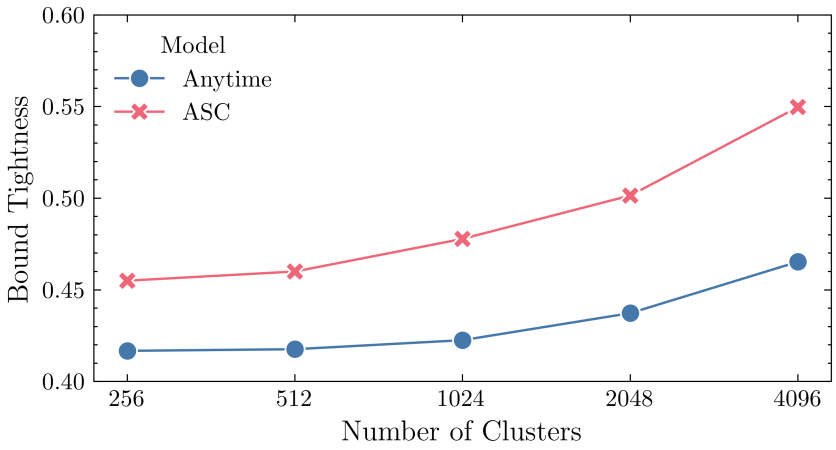


Figure 4.2: ASC predicts more accurate cluster bounds, which allows it to prune more aggressively. Cluster bound tightness is the average ratio of the actual and estimated cluster bounds, calculated with Formula (4.1).

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 (4.1), 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.

4.3.1 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:

$$MaxSBound(C_i) = \max_{j=1}^{n} B_{i,j}, \tag{4.2}$$

$$AvgSBound(C_i) = \frac{1}{n}\sum_{j=1}^{n} B_{i,j}, \tag{4.3}$$

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

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

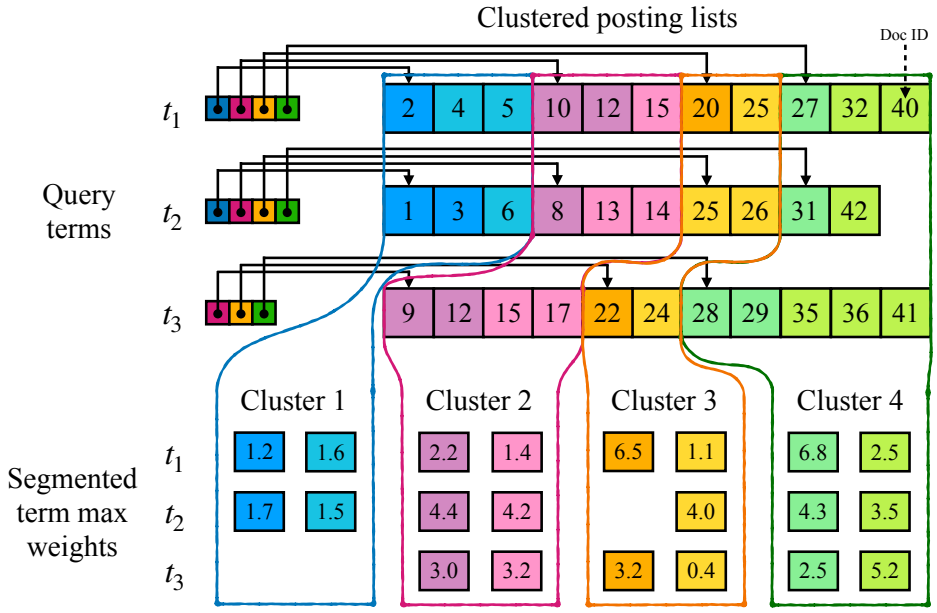
$$MaxSBound(C_i) \leq \frac{\theta}{\mu} \tag{4.4}$$

and

$$AvgSBound(C_i) \leq \frac{\theta}{\eta}. \tag{4.5}$$

- **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 4.3(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 4.3(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 4.3: 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 4.3 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.

4.3.2 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 7 gives more details. The default setting of ASC uses $\eta = 1$ in Section 4.4. The theorems on properties of ASC are listed below and Section 4.5 lists the proofs. We show that Theorem 6 is also true for Anytime Ranking with threshold overestimation and without early termination and we denote it as Anytime-μ.

Theorem 4

$$BoundSum(C_i) \geq MaxSBound(C_i) \geq \max_{d \in C_i} RankScore(d). \tag{4.6}$$

The above result shows that Formula (4.2) provides a tighter upperbound estimation than Formula (4.1) as demonstrated by Figure 4.2.

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 5 *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 4.4, 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 4.4 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 4.5 will further corroborate the results of Theorem 5: that ASC should not prune clusters when this gap is small.

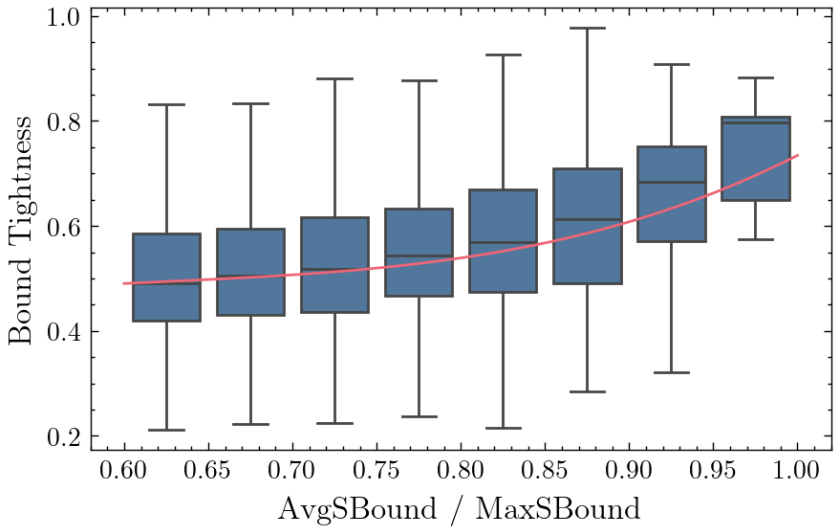


Figure 4.4: Correlation between the tightness of the estimated bound and the ratio of $AvgSBound$ and $MaxSBound$. As $AvgSBound$ approaches $MaxSBound$, the quality and tightness of the bound increases, and the probability of pruning decreases.

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 6 *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 $Avg(k', Anytime\text{-}\mu) \geq \mu Avg(k', R)$.*

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

Theorem 7 *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.

4.4 Evaluation

Table 4.1: A comparison with baselines using SPLADE on MS MARCO passages. No time budget

		MS MARCO Dev				DL'19	DL'20
Methods	**C%**	**MRR (Loss)**	**Recall**	**MRT (P_{99})**	**Speedup**	**nDCG (Recall)**	**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)

Datasets and metrics. We use the MS MARCO Passage ranking dataset [36] 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 [37], 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 Section 4.6.

Sparse models tested include a version of SPLADE [15, 30], uniCOIL [28, 29], and LexMAE [18]. We primarily use SPLADE to assess ASC because LexMAE, following dense models such as SimLM [12] and RetroMAE [11], uses MS MARCO title annotations. This is considered as non-standard [93]. SPLADE does not use title annotations.

ASC's implementation uses C++, extended from Anytime Ranking code's release based on the PISA retrieval package [48]. The index is compressed with SIMD-BP128. MaxScore is used to process queries because it is faster than VBMW for long queries [50, 51] 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 [89]. A comparison to other recent methods in the NeurIPS Big-ANN Competition are presented in Section 4.9. 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 4.1 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 [82] 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. Section 4.7 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 4.1, 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 4.1, 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 4. ASC is 29x faster than IOQP with $k = 10$. Safe IOQP is substantially slower than Anytime, which differs from the finding of [85], 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 7. 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 [89]. ASC outperforms IOQP-10% with 4.8% higher MRR@10 and 3.7% higher recall while ASC is 4.3x faster.

Table 4.2: 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 4.2 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 4.3 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.

Zero-shot out-of-domain retrieval. Table 4.4 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.

Table 4.3: Other learned sparse retrieval models

Methods	uniCOIL MRR (Re)	uniCOIL MRT	LexMAE MRR (Re)	LexMAE 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%	$0.320^{\dagger}$ ($.568^{\dagger}$)	11	$0.405^{\dagger}$ ($.693^{\dagger}$)	18
Anytime-μ=0.9	$0.345^{\dagger}$ ($.585^{\dagger}$)	4.2	$0.413^{\dagger}$ ($.654^{\dagger}$)	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%	$0.320^{\dagger}$ ($.937^{\dagger}$)	12	$0.405^{\dagger}$ (.985)	**20**
Anytime-μ=0.7	0.351 ($.940^{\dagger}$)	8.9	0.425 (.978)	46
ASC-μ=0.5, η=1	0.351 (.946)	**4.0**	0.425 (.980)	21

Table 4.4: Zero-shot performance with SPLADE on BEIR

	MaxScore		Anytime-μ =0.9		ASC	
Dataset	**nDCG**	**MRT**	**nDCG**	**MRT**	**nDCG**	**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	-

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 $k = 1000$.

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 4.5 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 segmen-

Table 4.5: K-means segmentation vs. random uniform

k=1000	K-means		Random	
μ, η	**MRR (Re)**	**T**	**MRR (Re)**	**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

tation 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 4.5 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 5.

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 5. The advantage of random uniform partitioning shown above corroborates with Theorem 7 and demonstrates the usefulness of possessing probabilistic approximate rank-safeness.

4.5 Proofs of Formal Properties

Proof of Theorem 4. 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 5. When a cluster C_i is not pruned by ASC, that is because one of Inequalities (4.4) and (4.5) is false. When Inequality (4.4) is true but Inequality (4.5) 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 6. 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 7: 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 $E[Avg(k', \text{ASC})] \geq \eta E[Avg(k', R)]$.

■

4.6 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 [15, 30] and LexMAE [18] 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 [82], we follow the approach of [94, 95] to apply the Lloyd's k-means clustering [96]. 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 [97, 98]. 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 [11], a state-of-the-art dense retrieval model.

Table 4.6: 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%

Table 4.6 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 [22] 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.

4.7 Impact of varying #clusters for Anytime Ranking and ASC

Figure 4.5 shows the latency of Anytime and ASC for $k = 10$ with safe pruning and a similar trend is seen for $k = 1000$. Table 4.7 shows their performance with threshold overestimation ($\mu = 0.9$). 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 4.4. 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.

Table 4.7: Performance of Anytime Ranking vs. ASC when varying #clusters for threshold overestimation. $k = 10$.

Cluster	Anytime Ranking $\mu = 0.9$			ASC $\mu = 0.9$, $\eta = 1$	
Count	**MRR (Re)**	**Orig.**	**Opt.**	**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**

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

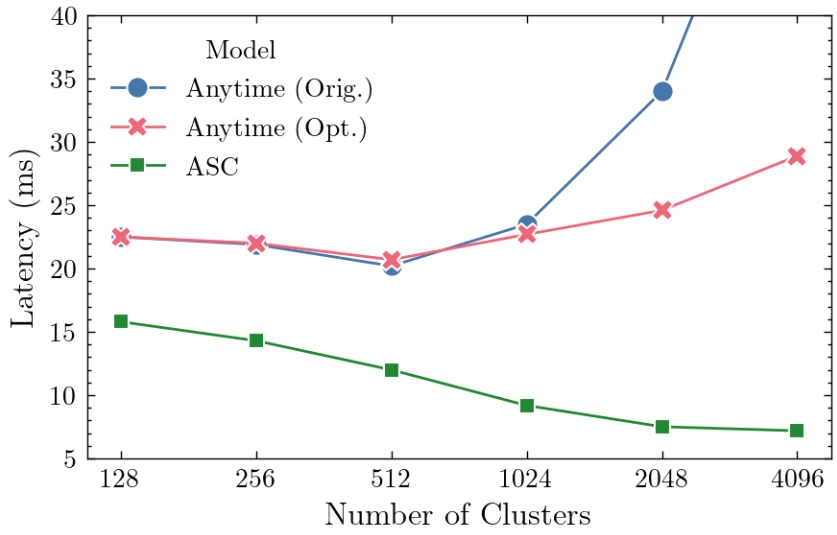


Figure 4.5: The effect of the number of clusters on latency. For Anytime (Orig.) and Anytime (Opt.), latency grows significantly with clusters. ASC is the fastest method for all clusters and exhibits the slowest growth in latency of all methods.

of clusters because that increases the space overhead for ASC. The finding is similar for different choices of μ and for $k = 1000$. Figure 4.6 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 4.3. This figure shows that the best configuration of ASC is 4096 clusters and 8 segments per cluster for all values of μ.

4.8 Compatibility with other speedup techniques

Table 4.8 lists MRR@10 and Recall@1000 of combining ASC with early termination technique of Anytime Ranking [82] under a time budget on MS MARCO Dev set for

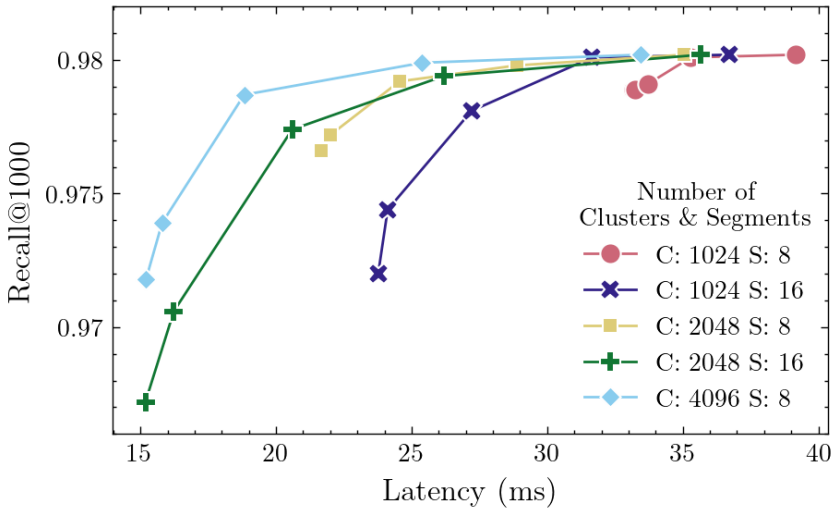


Figure 4.6: Recall vs. latency of ASC (η=1) for varying values of μ at retrieval depth $k = 1000$. For each fixed number of clusters and segments, μ varies from 0.4, 0.5, 0.6, 0.7, to 1.

Table 4.8: Anytime vs. ASC (η=1) with time budgets

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

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 4.1, 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 [86] for a version of SPLADE used in Big-ANN competition as discussed in Section 4.9 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 [88] 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.

4.9 Comparison to NeurIPS '23 Big-ANN Methods

The sparse track of NeurIPS 2023 competition for fast approximate nearest neighbor search (Bi-ANN) [91] 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 4.9 reports reciprocal

rank (MRR@10), Recall@10, and single-thread latency (MRT) in milliseconds on our machine. Table 4.9 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 4.4.

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 4.9 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.

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 4.1 of Section 4.4 are within a 0.1% MRR@10 loss budget for

Table 4.9: 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**

Dev set. Thus while ASC makes a cluster-based retriever more competitive in the BigANN tradeoff setting, ASC is designed to speed up retrieval applications that desire high relevance effectiveness.

4.10 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.

4.11 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 [12, 11, 13]. 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 [22] and graph navigation with HNSW [23]. 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 [50, 51], which fits in the case of SPLADE and LexMAE under the tested retrieval depths. A previous study [81] 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 [85] 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.

Chapter 5

Conclusion and Future Work

With the help of neural language models, retrieval systems support searching with semantic queries. However, due to the nature of more complex calculation and document representation, the retrieval systems are not as efficient as before with only traditional statistical ranking signal. In this thesis, optimization across different components of the retrieval systems for semantic search, especially learned sparse retrieval, has been explored.

- In Section 2, we studied to maximize the sparsity of the representation of the sparse vectors, by taking consideration of the threshold-based pruning in the design of rank loss, to enable a learnable threshold for static pruning. Experiments show that the proposed algorithm, HT, can effectively increase sparsity and shorten the retrieval latency without hurting retrieval performance too much.

- In Section 3, we modified the traditional dynamic pruning algorithms including BlockMax-WAND and MaxScore, so that they can efficiently prune documents for learned sparse representations with the guidance of traditional statistical signals. By compressing the BM25 scores inside the posting records along with the learned

scores, the proposed framework 2GT successfully decreased the latency with small storage overhead.

- In Section 4, inspired by the clustering method commonly used by dense retrievers, ASC groups documents into clusters, designed indicators for better cluster level pruning, and provided theoretical analysis and safeness guarantee of the cluster level unsafe pruning. Experimental results show that more than 90% clusters can be dynamically pruned under probabilistic safeness guarantee.

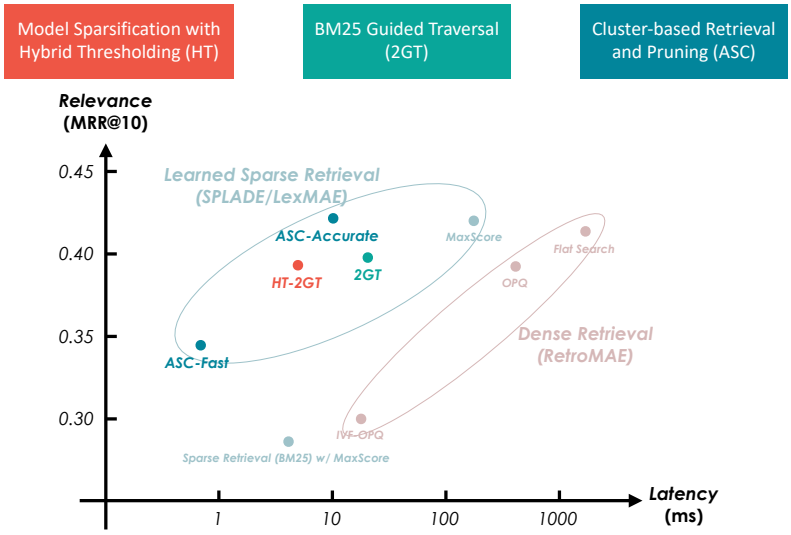


Figure 5.1: Relevance and latency trade-off among different retrieval models

As an example, Figure 5.1 demonstrates the effectiveness of these algorithms on the graph of relevance vs. lantecy trade-off. The learned sparse retrievers have relatively shorter latency compared to dense retrievers on the single threaded CPU setup, while the relevance is of the same level as the state-of-the-art dense models. The accurate configuration of ASC can achieve state of the art relevance, but being 20x faster than the original max score which cost around 200ms. The fast configuration of ASC takes

less than 1ms, but still has reasonably good relevance.

Overall speaking, the original learned sparse retrieval with around 200ms latency is reduced to 10ms with minimal relevance loss, by the combination of all the aforementioned optimizations. This fast and effective sparse retrieval model can benefit semantic search and modern RAG frameworks better.

Although much progress has been made for efficiency of learned sparse retrieval, there is still plenty of headroom for improvement. Some of potential topics are presented as follows. Firstly, recent research on learned sparse representations mainly focuses on semantic search query sets, including MS MARCO and BEIR. Experimenting newly developed techniques on more diverse datasets, especially on large scale web data including ClueWeb [99] and its recent update [100], could be interesting. Secondly, learned sparse retrieval shows some out-of-domain advantage over dense retrieval. It could be interesting to test these sparse algorithms in the setting of RAG and validate the impact on the RAG system's overall performance. Thirdly, sparse retrieval can be better integrated with dense retrieval, where some progress has already been made in [101]. Lastly, with the help of unsafe pruning, the room for latency reduction is greatly extended. More studies can be done on the theoretical guarantee of unsafe pruning.

Bibliography

[1] K. S. Jones, S. Walker, and S. E. Robertson, *A probabilistic model of information retrieval: development and comparative experiments*, in *Information Processing and Management*, pp. 779–840, 2000.

[2] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien, *Efficient query evaluation using a two-level retrieval process*, in *Proc. of the 12th ACM International Conference on Information and Knowledge Management*, pp. 426–434, 2003.

[3] S. Ding and T. Suel, *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*, pp. 993–1002, 2011.

[4] H. Turtle and J. Flood, *Query evaluation: Strategies and optimizations*, *Information Processing & Management* **31** (1995), no. 6 831–850.

[5] J. Guo, Y. Fan, Q. Ai, and W. B. Croft, *A deep relevance matching model for ad-hoc retrieval*, in *Proceedings of the 25th ACM international on conference on information and knowledge management*, pp. 55–64, 2016.

[6] C. Xiong, Z. Dai, J. Callan, Z. Liu, and R. Power, *End-to-end neural ad-hoc ranking with kernel pooling*, in *Proceedings of the 40th International ACM SIGIR conference on research and development in information retrieval*, pp. 55–64, 2017.

[7] Z. Dai, C. Xiong, J. Callan, and Z. Liu, *Convolutional neural networks for soft-matching n-grams in ad-hoc search*, in *Proceedings of the eleventh ACM international conference on web search and data mining*, pp. 126–134, 2018.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: pre-training of deep bidirectional transformers for language understanding*, in *NAACL-HLT*, 2019.

[9] R. Nogueira, W. Yang, K. Cho, and J. Lin, *Multi-stage document ranking with bert*, *ArXiv* **abs/1910.14424** (2019).

[10] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Y. Wu, S. Edunov, D. Chen, and W. tau Yih, *Dense passage retrieval for open-domain question answering*, *EMNLP'2020* **ArXiv abs/2010.08191** (2020).

[11] L. Wang, N. Yang, X. Huang, B. Jiao, L. Yang, D. Jiang, R. Majumder, and F. Wei, *SimLM: Pre-training with representation bottleneck for dense passage retrieval*, *ACL* (2023).

[12] S. Xiao, Z. Liu, Y. Shao, and Z. Cao, *RetroMAE: pre-training retrieval-oriented transformers via masked auto-encoder*, *EMNLP* (2022).

[13] Z. Liu, S. Xiao, Y. Shao, and Z. Cao, *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)* (A. Rogers, J. Boyd-Graber, and N. Okazaki, eds.), (Toronto, Canada), pp. 2635–2648, Association for Computational Linguistics, July, 2023.

[14] J. Johnson, M. Douze, and H. Jégou, *Billion-scale similarity search with gpus*, *arXiv preprint arXiv:1702.08734* (2017).

[15] T. Formal, B. Piwowarski, and S. Clinchant, *SPLADE: Sparse lexical and expansion model for first stage ranking*, *SIGIR* (2021).

[16] T. Formal, C. Lassance, B. Piwowarski, and S. Clinchant, *SPLADE v2: Sparse lexical and expansion model for information retrieval*, *ArXiv* **abs/2109.10086** (2021).

[17] T. Formal, C. Lassance, B. Piwowarski, and S. Clinchant, *From distillation to hard negative sampling: Making sparse neural IR models more effective*, *SIGIR* (2022).

[18] T. Shen, X. Geng, C. Tao, C. Xu, X. Huang, B. Jiao, L. Yang, and D. Jiang, *LexMAE: Lexicon-bottlenecked pretraining for large-scale retrieval*, in *The Eleventh International Conference on Learning Representations*, 2023.

[19] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. S. Maarek, and A. Soffer, *Static index pruning for information retrieval systems*, in *Proc. of SIGIR*, SIGIR '01, (New York, NY, USA), p. 43–50, Association for Computing Machinery, 2001.

[20] R. Blanco and A. Barreiro, *Boosting static pruning of inverted files*, in *Proc. of SIGIR*, SIGIR '07, (New York, NY, USA), p. 777–778, Association for Computing Machinery, 2007.

[21] S. Büttcher and C. L. A. Clarke, *A document-centric approach to static index pruning in text retrieval systems*, in *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, (New York, NY, USA), p. 182–189, Association for Computing Machinery, 2006.

[22] J. Johnson, M. Douze, and H. Jégou, *Billion-scale similarity search with GPUs*, *IEEE Trans. on Big Data* **7** (2019), no. 3 535–547.

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

[24] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini, *Faster blockmax wand with variable-sized blocks*, in *Proc. of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 625–634, 2017.

[25] H. Zamani, M. Dehghani, W. B. Croft, E. G. Learned-Miller, and J. Kamps, *From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing*, *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (2018).

[26] Z. Dai and J. Callan, *Context-aware term weighting for first stage passage retrieval*, *SIGIR* (2020).

[27] A. Mallia, O. Khattab, N. Tonellotto, and T. Suel, *Learning passage impacts for inverted indexes*, *SIGIR* (2021).

[28] J. J. Lin and X. Ma, *A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques*, *ArXiv* **abs/2106.14807** (2021).

[29] L. Gao, Z. Dai, and J. Callan, *COIL: revisit exact lexical match in information retrieval with contextualized inverted list*, *NAACL* (2021).

[30] T. Formal, C. Lassance, B. Piwowarski, and S. Clinchant, *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* (2022).

[31] A. Mallia, J. Mackenzie, T. Suel, and N. Tonellotto, *Faster learned sparse retrieval with guided traversal*, in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1901–1905, 2022.

[32] L. Xiong, C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. N. Bennett, J. Ahmed, and A. Overwijk, *Approximate nearest neighbor negative contrastive learning for dense text retrieval*, in *ICLR*, 2021.

[33] K. Santhanam, O. Khattab, J. Saad-Falcon, C. Potts, and M. A. Zaharia, *ColBERTv2: Effective and efficient retrieval via lightweight late interaction*, *NAACL'22* **ArXiv abs/2112.01488** (2022).

[34] J. Zhan, J. Mao, Y. Liu, J. Guo, M. Zhang, and S. Ma, *Learning discrete representations via constrained clustering for effective and efficient dense retrieval*, in *Proc. of Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, pp. 1328–1336, 2022.

[35] S. Xiao, Z. Liu, W. Han, J. Zhang, D. Lian, Y. Gong, Q. Chen, F. Yang, H. Sun, Y. Shao, D. Deng, Q. Zhang, and X. Xie, *Distill-vq: Learning retrieval oriented vector quantization by distilling knowledge from dense embeddings*, *Proc. of SIGIR* (2022).

[36] N. Craswell, B. Mitra, E. Yilmaz, D. F. Campos, and E. M. Voorhees, *Overview of the trec 2020 deep learning track*, *ArXiv* **abs/2102.07662** (2020).

[37] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych, *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)*, 2021.

[38] J. Yang, X. Ma, and J. Lin, *Sparsifying sparse representations for passage retrieval by top-k masking*, *CoRR* **abs/2112.09628** (2021) [arXiv:2112.0962].

[39] S. Hofstätter, S. Althammer, M. Schröder, M. Sertkan, and A. Hanbury, *Improving efficient neural ranking models with cross-architecture knowledge distillation*, *ArXiv* **abs/2010.02666** (2020).

[40] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi, *Soft threshold weight reparameterization for learnable sparsity*, in *International Conference on Machine Learning*, pp. 5544–5555, PMLR, 2020.

[41] J.-W. Park and J.-S. Lee, *Dynamic thresholding for learning sparse neural networks*, in *ECAI 2020*, pp. 1403–1410. IOS Press, 2020.

[42] J. LIU, Z. XU, R. SHI, R. C. C. Cheung, and H. K. So, *Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers*, in *International Conference on Learning Representations*, 2020.

[43] E. Frantar and D. Alistarh, *SPDY: Accurate pruning with speedup guarantees*, in *Proceedings of the 39th International Conference on Machine Learning* (K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, eds.), vol. 162 of *Proceedings of Machine Learning Research*, pp. 6726–6743, PMLR, 17–23 Jul, 2022.

[44] G. Georgiadis, *Accelerating convolutional neural networks via activation map compression*, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7085–7095, 2019.

[45] M. Kurtz, J. Kopinsky, R. Gelashvili, A. Matveev, J. Carr, M. Goin, W. Leiserson, S. Moore, N. Shavit, and D. Alistarh, *Inducing and exploiting activation sparsity for fast inference on deep neural networks*, in *Proceedings of the 37th International Conference on Machine Learning* (H. D. III and A. Singh, eds.), vol. 119 of *Proceedings of Machine Learning Research*, pp. 5533–5543, PMLR, 13–18 Jul, 2020.

[46] K. Järvelin and J. Kekäläinen, *Cumulated gain-based evaluation of ir techniques*, *ACM Transactions on Information Systems (TOIS)* **20** (2002), no. 4 422–446.

[47] C. Lassance and S. Clinchant, *An efficiency study for splade models*, in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2220–2226, 2022.

[48] A. Mallia, M. Siedlaczek, J. Mackenzie, and T. Suel, *PISA: Performant indexes and search for academia*, *Proceedings of the Open-Source IR Replicability Challenge* (2019).

[49] D. Lemire and L. Boytsov, *Decoding billions of integers per second through vectorization*, *Softw. Pract. Exp.* **45** (2015), no. 1 1–29.

[50] A. Mallia, M. Siedlaczek, and T. Suel, *An experimental study of index compression and DAAT query processing methods*, in *Proc. of 41st European Conference on IR Research, ECIR' 2019*, pp. 353–368, 2019.

[51] Y. Qiao, Y. Yang, H. Lin, and T. Yang, *Optimizing guided traversal for fast learned sparse retrieval*, in *Proceedings of the ACM Web Conference 2023*, WWW '23, (Austin, TX, USA), ACM, 2023.

[52] Y. Qiao, Y. Yang, H. Lin, T. Xiong, X. Wang, and T. Yang, *Dual skipping guidance for document retrieval with learned sparse representations*, *ArXiv* **abs/2204.11154** (April, 2022).

[53] J. Lin and A. Trotman, *Anytime ranking for impact-ordered indexes*, in *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, ICTIR '15, (New York, NY, USA), p. 301–304, Association for Computing Machinery, 2015.

[54] J. Mackenzie, M. Petri, and A. Moffat, *Anytime ranking on document-ordered indexes*, *ACM Trans. Inf. Syst.* **40** (sep, 2021).

[55] Y. Bai, X. Li, G. Wang, C. liang Zhang, L. Shang, J. Xu, Z. Wang, F. Wang, and Q. Liu, *Sparterm: Learning term-based sparse representation for fast text retrieval*, *ArXiv* **abs/2010.00768** (2020).

[56] A. Mallia, O. Khattab, T. Suel, and N. Tonellotto, *Learning passage impacts for inverted indexes*, in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1723–1727, 2021.

[57] D. R. Cheriton, *From doc2query to doctttttquery*, 2019.

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

[59] K. Chakrabarti, S. Chaudhuri, and V. Ganti, *Interval-based pruning for top-k processing over compressed lists*, *2011 IEEE 27th International Conference on Data Engineering* (2011) 709–720.

[60] C. Dimopoulos, S. Nepomnyachiy, and T. Suel, *Optimizing top-k document retrieval strategies for block-max indexes*, in *WSDM '13*, 2013.

[61] T. Strohman and W. B. Croft, *Efficient document retrieval in main memory*, in *Proc. of the 30th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 175–182, 2007.

[62] C. Macdonald, N. Tonellotto, and I. Ounis, *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, (New York, NY, USA), pp. 1051–1052, Association for Computing Machinery, 2012.

[63] N. Tonellotto, C. Macdonald, and I. Ounis, *Efficient and effective retrieval using selective pruning*, in *Proc. of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pp. 63–72, ACM, 2013.

[64] M. Crane, J. S. Culpepper, J. Lin, J. Mackenzie, and A. Trotman, *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, (New York, NY, USA), pp. 201–210, ACM, 2017.

[65] Y. Yang, Y. Qiao, J. Shao, X. Yan, and T. Yang, *Lightweight composite re-ranking for efficient keyword search with BERT*, *Proc. of ACM WSDM'22* (2022).

[66] S.-C. Lin, J.-H. Yang, and J. J. Lin, *In-batch negatives for knowledge distillation with tightly-coupled teachers for dense retrieval*, in *REPL4NLP*, 2021.

[67] L. Gao, Z. Dai, T. Chen, Z. Fan, B. V. Durme, and J. Callan, *Complementing lexical retrieval with semantic residual embedding*, 2021.

[68] X. Ma, K. Sun, R. Pradeep, and J. Lin, *A replication study of dense passage retriever*, 2021.

[69] H. Li, S. Wang, S. Zhuang, A. Mourad, X. Ma, J. Lin, and G. Zuccon, *To interpolate or not to interpolate: Prf, dense and sparse retrievers*, *SIGIR* (2022).

[70] N. Tonellotto, C. Macdonald, I. Ounis, *et. al.*, *Efficient query processing for scalable web search*, *Foundations and Trends® in Information Retrieval* **12** (2018), no. 4-5 319–500.

[71] D. F. Campos, T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, L. Deng, and B. Mitra, *Ms marco: A human generated machine reading comprehension dataset*, *ArXiv* **abs/1611.09268** (2016).

[72] J. Lin, X. Ma, S.-C. Lin, J.-H. Yang, R. Pradeep, and R. Nogueira, *Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations*, in *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pp. 2356–2362, 2021.

[73] coCondenser, *https://huggingface.co/luyu/co-condenser-marco*, .

[74] MiniLM-L-6-v2, *https://huggingface.co/cross-encoder/ms-marco-minilm-l-6-v2*, .

[75] O. Khattab, M. Hammoud, and T. Elsayed, *Finding the best of both worlds: Faster and more robust top-k document retrieval*, in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1031–1040, 2020.

[76] J. Mackenzie, J. S. Culpepper, R. Blanco, M. Crane, C. L. A. Clarke, and J. Lin, *Query driven algorithm selection in early stage retrieval*, in *Proc. of the 11th ACM International Conference on Web Search and Data Mining*, pp. 396–404, 2018.

[77] R. Ren, Y. Qu, J. Liu, W. X. Zhao, Q. She, H. Wu, H. Wang, and J.-R. Wen, *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*, (Online and Punta Cana, Dominican Republic), pp. 2825–2835, ACM, Nov., 2021.

[78] H. Kulkarni, S. MacAvaney, N. Goharian, and O. Frieder, *Lexically-accelerated dense retrieval*, in *Proc. of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, (New York, NY, USA), p. 152–162, Association for Computing Machinery, 2023.

[79] P. Zhang, Z. Liu, S. Xiao, Z. Dou, and J. Yao, *Hybrid inverted index is a robust accelerator for dense retrieval*, in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (H. Bouamor, J. Pino, and K. Bali, eds.), (Singapore), pp. 1877–1888, Association for Computational Linguistics, Dec., 2023.

[80] C. Dimopoulos, S. Nepomnyachiy, and T. Suel, *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*, pp. 723–732, 2013.

[81] A. Mallia, M. Siedlaczek, and T. Suel, *Fast disjunctive candidate generation using live block filtering*, in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, (New York, NY, USA), p. 671–679, Association for Computing Machinery, 2021.

[82] J. Mackenzie, M. Petri, and A. Moffat, *Anytime ranking on document-ordered indexes*, *ACM Trans. Inf. Syst.* **40** (sep, 2021).

[83] F. Can, I. Altingövde, and E. Demir, *Efficiency and effectiveness of query processing in cluster-based retrieval*, *Information Systems* **29** (12, 2004) 697–717.

[84] F. Hafizoglu, E. C. Kucukoglu, and . S. Altıngövde, *On the efficiency of selective search*, in *ECIR 2017*, vol. 10193, pp. 705–712, 2017.

[85] A. Mallia, T. Suel, and N. Tonellotto, *Faster learned sparse retrieval with block-max pruning*, 2024.

[86] Y. Qiao, Y. Yang, S. He, and T. Yang, *Representation sparsification with hybrid thresholding for fast SPLADE-based document retrieval*, *ACM SIGIR'23* (2023).

[87] C. Lassance, S. Lupart, H. Déjean, S. Clinchant, and N. Tonellotto, *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, (New York, NY, USA), p. 1771–1775, Association for Computing Machinery, 2023.

[88] J. Mackenzie, A. Mallia, A. Moffat, and M. Petri, *Accelerating learned sparse indexes via term impact decomposition*, in *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 2830–2842, 2022.

[89] J. Mackenzie, M. Petri, and L. Gallaghera, *Ioqp: A simple impact-ordered query processor written in rust*, in *Proceedings of DESIRES 2022*, 8, 2022.

[90] J. Mackenzie, A. Trotman, and J. Lin, *Efficient document-at-a-time and score-at-a-time query evaluation for learned sparse representations*, *ACM Trans. Inf. Syst.* **41** (mar, 2023).

[91] Big-ANN, *Neurips'23 competition track: https://big-ann-benchmarks.com/neurips23.html*, may, 2024.

[92] S. Bruch, F. M. Nardini, A. Ingber, and E. Liberty, *Bridging dense and sparse maximum inner product search*, *ACM Trans. Inf. Syst.* (may, 2024).

[93] C. Lassance and S. Clinchant, *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, (New York, NY, USA), p. 2431–2435, ACM, 2023.

[94] A. Kulkarni and J. Callan, *Selective search: Efficient and effective search of large textual collections*, *ACM Trans. Inf. Syst.* **33** (2015), no. 4 17:1–17:33.

[95] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat, *Efficient distributed selective search*, *Inf. Retr. J.* **20** (2017), no. 3 221–252.

[96] S. Lloyd, *Least squares quantization in pcm*, *IEEE Trans. on Information Theory* **28** (1982), no. 2 129–137.

[97] Z. Zhang, K. Lange, and J. Xu, *Simple and scalable sparse k-means clustering via feature ranking*, in *NeurIPS 20220: Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 10148–10160, 2020.

[98] S. Dey, S. Das, and R. Mallipeddi, *The sparse minmax k-means algorithm for high-dimensional clustering*, in *Proc. of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 2103–2110, 7, 2020.

[99] J. Callan, *The lemur project and its clueweb12 dataset*, in *Invited talk at the SIGIR 2012 Workshop on Open-Source Information Retrieval*, 2012.

[100] A. Overwijk, C. Xiong, and J. Callan, *Clueweb22: 10 billion web documents with rich information*, in *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*, pp. 3360–3362, 2022.

[101] Y. Yang, P. Carlson, S. He, Y. Qiao, and T. Yang, *Cluster-based partial dense retrieval fused with sparse text retrieval*, in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, (New York, NY, USA), p. 2327–2331, Association for Computing Machinery, 2024.