03/16/26

Modern Index Design for Efficient Learned Sparse Retrieval

Parker Carlson

About Me

Parker Carlson
PhD Student at UC Santa Barbara

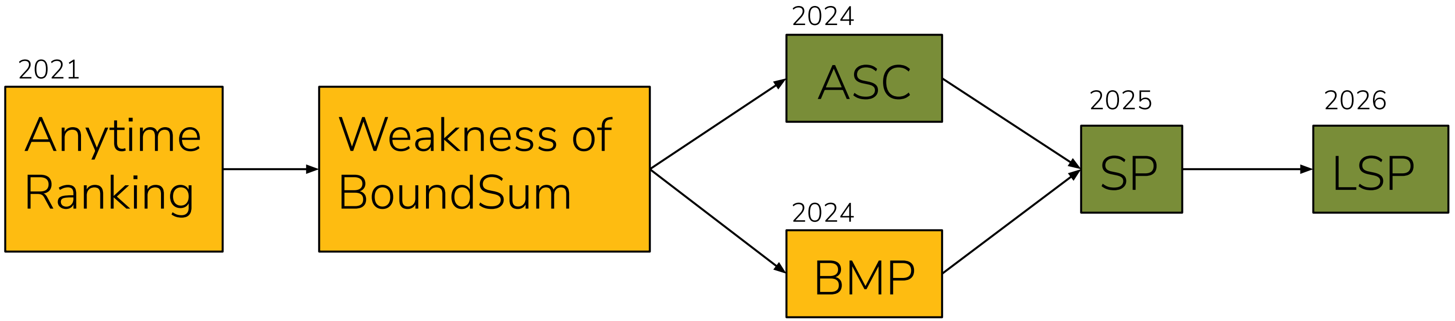
Email: parker_carlson@ucsb.edu
Website: thefxperson.github.io

I’m a PhD student at UCSB advised by Prof. Tao Yang. My research is focused on improving the efficiency of large-scale neural text retrieval systems.

Previously, I earned my BS at Oregon State University, where I worked with Prof. Patrick Donnelly on applications of machine learning to sound and music.

Overview

- Brief background on fundamentals [5 min]
  - Sparse vs Dense, Inverted Index, DAAT vs TAAT, LSR
- Modern Index Design for LSR [40 min]



Background

Problem

Search is easy!

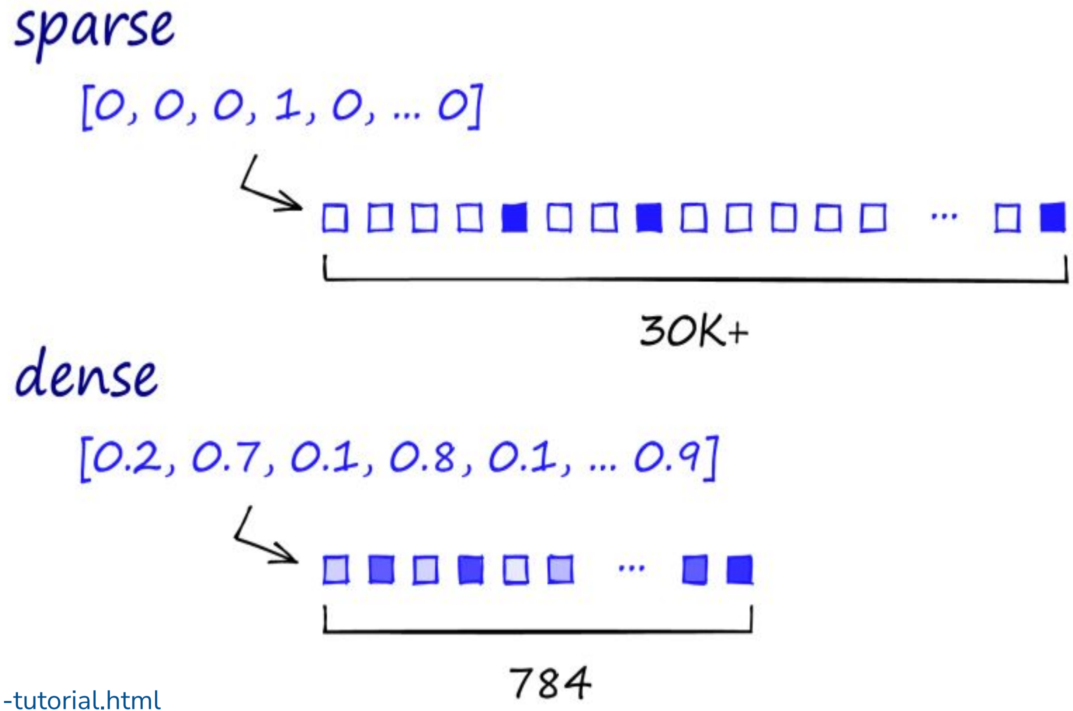
```
for document in collection:
        s = score(query, doc)
        topk_heap.insert(s, doc_id)
```

- Score() is generally dot product or cosine similarity
- When the collection is large (10M, 100M, 1B, 10B …?), it’s harder!

Sparse & Dense Retrieval

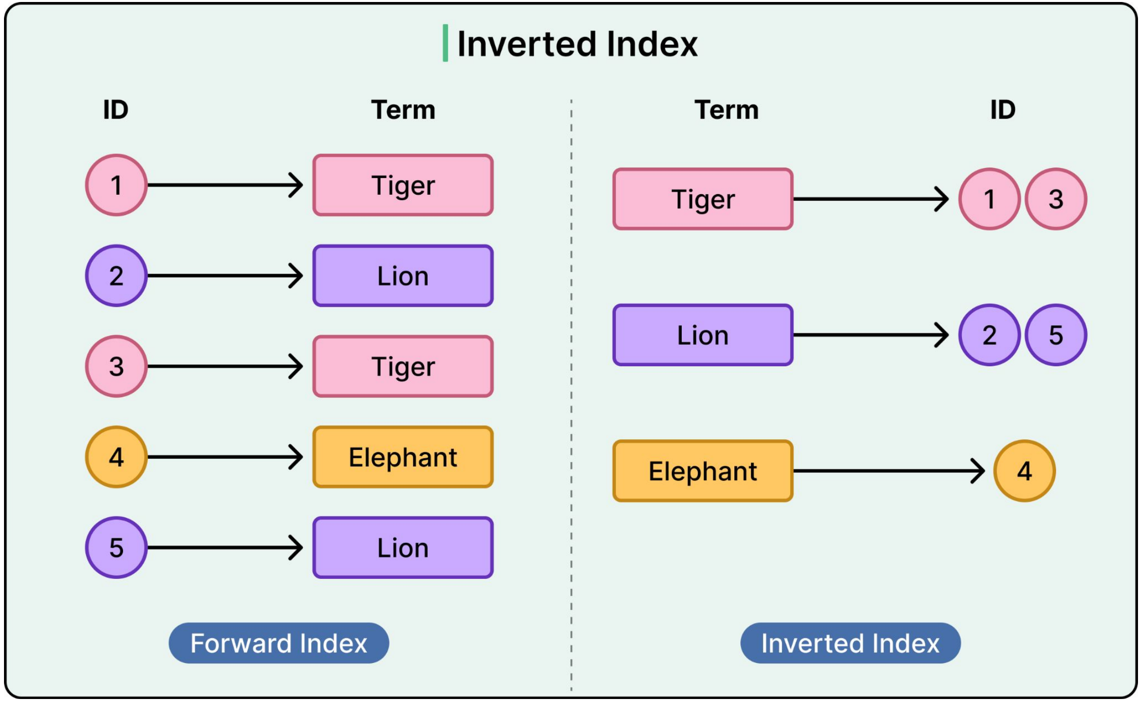
Academia: Sparse Retrieval
(aka Lexical Retrieval)
Industry: Sparse Vector Search

Academia: Dense Retrieval
Industry: Vector Search



img: https://sease.io/2024/07/opensearch-neural-sparse-search-tutorial.html

Inverted Index



img: https://blog.bytebytego.com/p/database-index-internals-understanding

DAAT vs TAAT

Document-at-a-Time (DAAT)

a	(d1, 1.0)	(d4, 2.0)	(d7, 0.2)
b	(d4, 1.0)	(d7, 2.0)	(d8, 0.2)
c	(d4, 3.0)	(d5, 1.5)	(d7, 1.0)

Term-at-a-Time (TAAT)*

*on an impact-ordered index

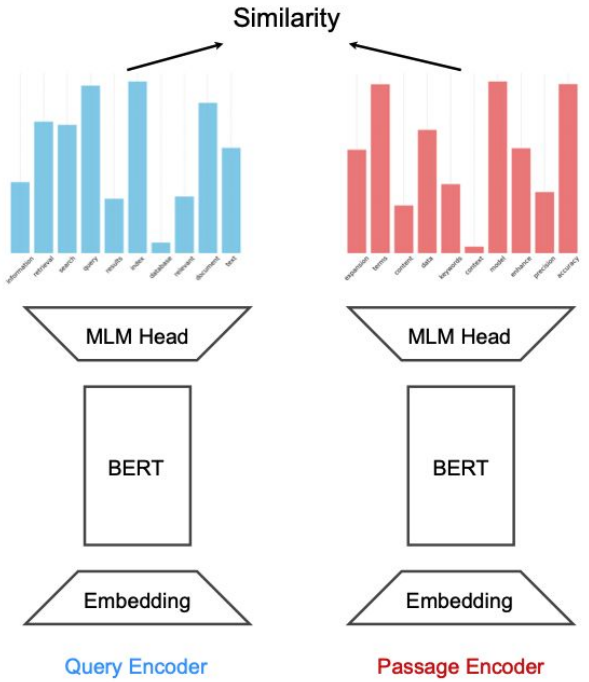
a	(d4, 2.0)	(d1, 1.0)	(d7, 0.2)
b	(d7, 2.0)	(d4, 1.0)	(d8, 0.2)
c	(d4, 3.0)	(d5, 1.5)	(d7, 1.0)

Accumulators:

d1	d2	d3	d4	d5	d6	d7	d8
0	0	0	0	0	0	0	0

Learned Sparse Retrieval (LSR)

- Sparse (Lexical) retrieval uses term frequency statistics for query and doc
  - E.g. tf-idf, BM25
- LSR generates term weights using a PLM
- LSR also does query and document expansion

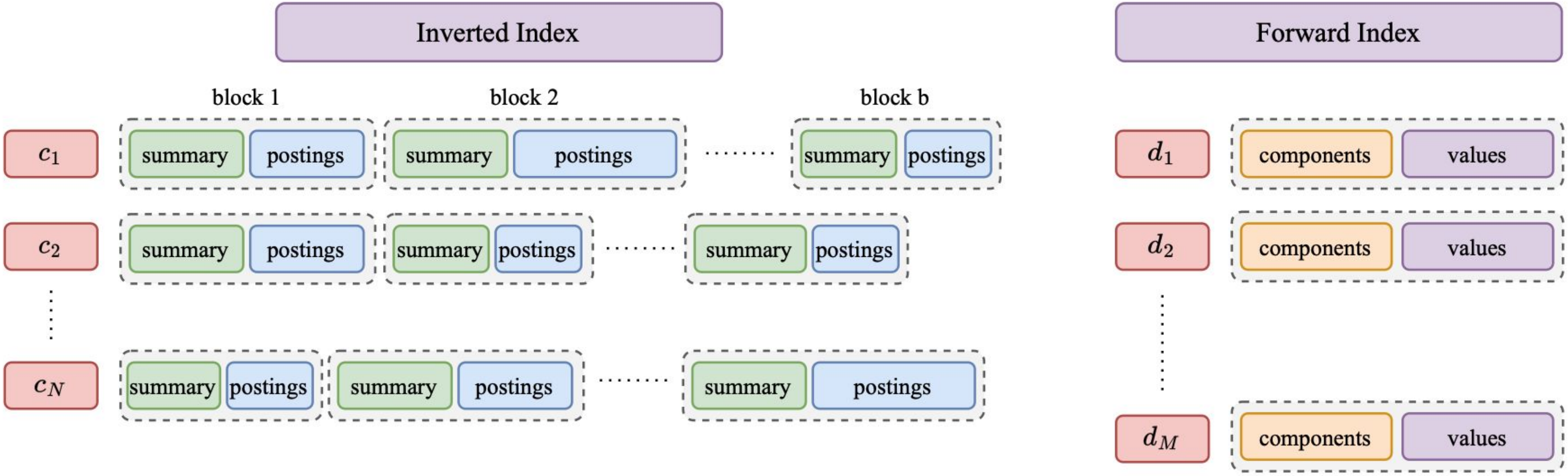


img: https://arxiv.org/pdf/2408.11119

Recent Work on Indexing for LSR

Seismic (SIGIR 24)

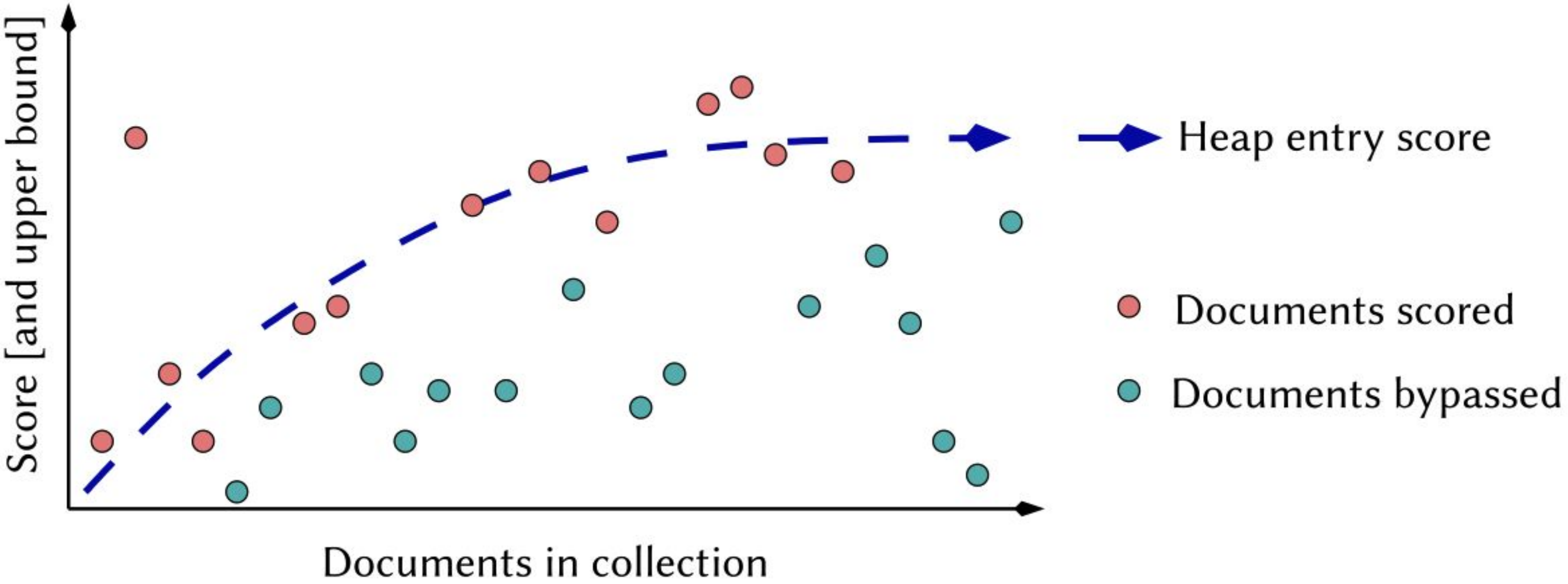
A modern approach to TAAT retrieval



Anytime Ranking (for DAAT) (TOIS, 2021)

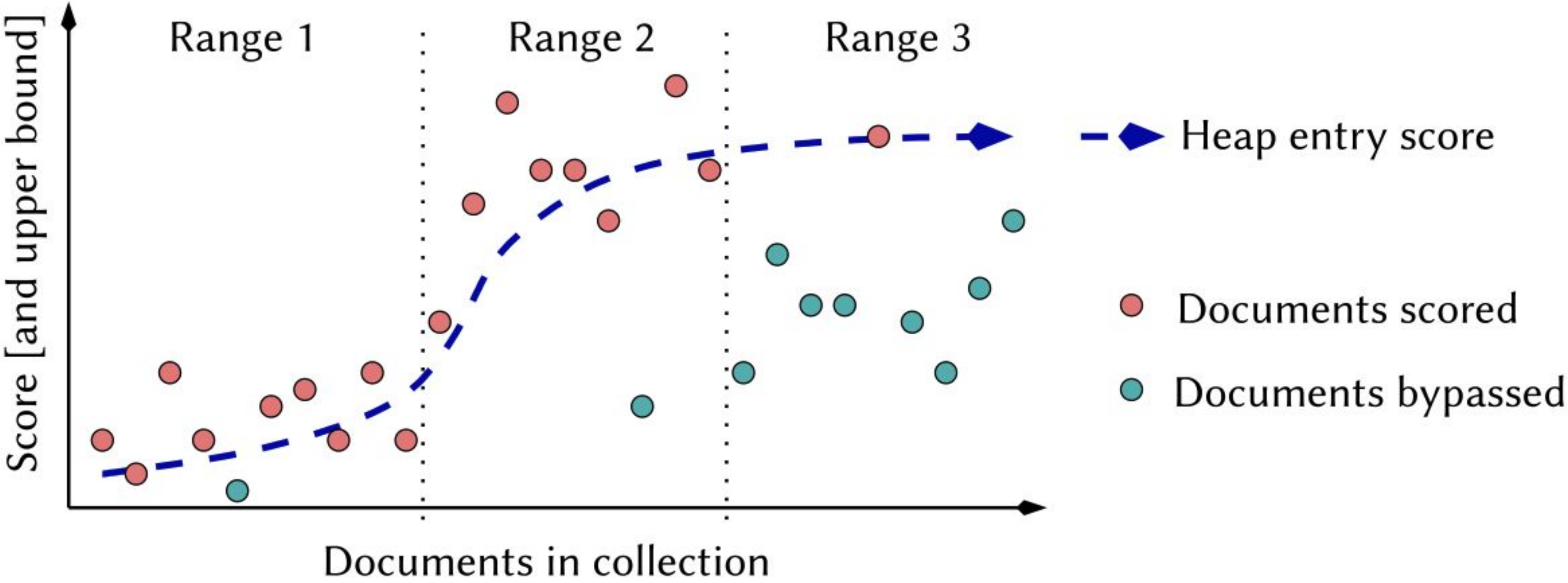
1. (Offline:) Cluster documents by similarity
2. Compute a score heuristic for every cluster
3. Sort clusters by estimated score
4. Score documents within top clusters
5. When top-k score ϴ is larger than the next cluster score, stop search

Anytime Ranking (for DAAT) (TOIS, 2021)



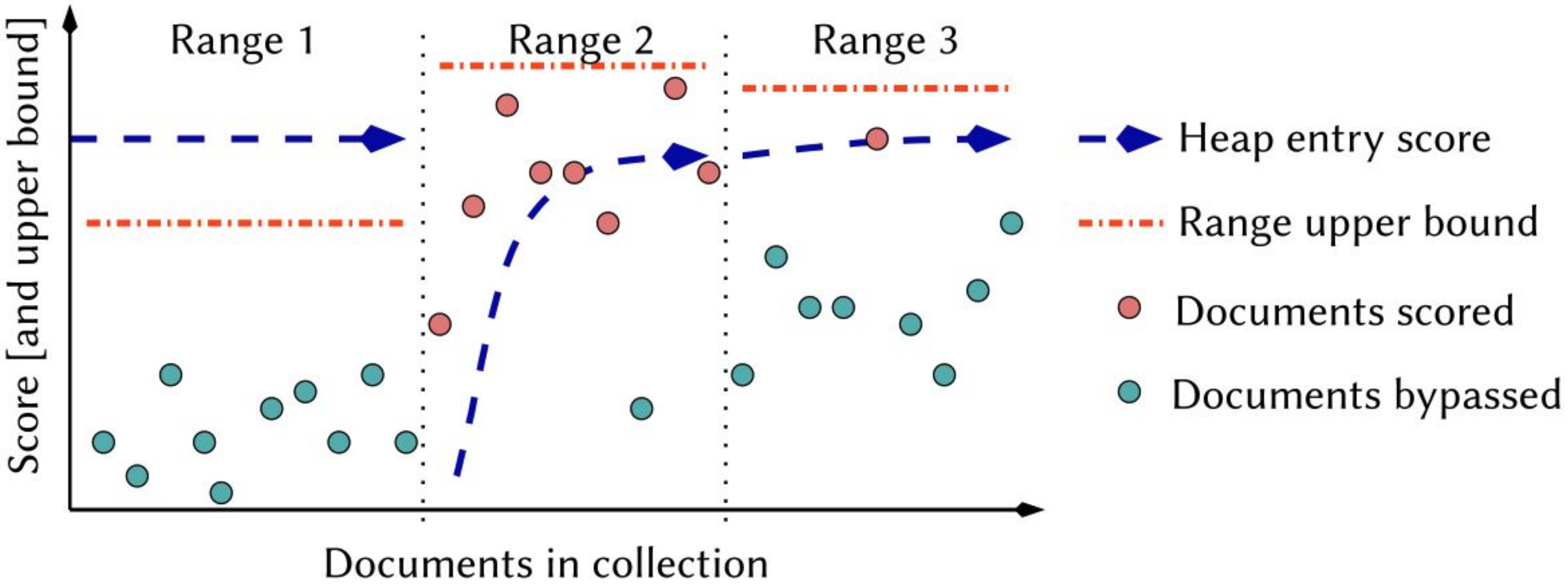
(a) Normal pruning, with the heap threshold in blue.

Anytime Ranking (for DAAT) (TOIS, 2021)



(b) Pruning in a topically-coherent collection.

Anytime Ranking (for DAAT) (TOIS, 2021)



(c) Range-prioritized processing, based on upper bounds.

Anytime Ranking (for DAAT) (TOIS, 2021)

Cluster Heuristic: BoundSum

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

Advantages:

- **Safe** – if Θ >= BoundSum(B), then B has no top-k documents
- **Simple** – fast to compute, low storage overhead

Note: Similar methods have been used for non-clustered index pruning for many years (e.g. BMW (Ding & Suel, SIGIR 11), VBMW (Mallia et al., SIGIR ‘17))

Problem: Loose BoundSums

	Cluster 1			Cluster 2		
doc_id	1	2	3	4	5	6
term_1	50	100	150	5	10	180
term_2	150	100	50	180	10	1
Score(q,d)	**200**	**200**	**200**	**185**	**20**	**181**
BoundSum	300			360		
Bound Tightness	200/300 = 66%			185/360 = 51%		

Problem: Loose BoundSums

Search

k=3

	Cluster 1			Cluster 2		
doc_id	1	2	3	4	5	6
term_1	50	100	150	5	10	180
term_2	150	100	50	180	10	1
Score(q,d)	**200**	**200**	**200**	**185**	**20**	**181**
BoundSum	300			360		
Bound Tightness	200/300 = 66%			185/360 = 51%		

$\theta = 0$

Problem: Loose BoundSums

ϴ = 200 < BoundSum(C2) = 360,
so we need to score all docs

Search
k=3

	Cluster 1			Cluster 2		
doc_id	1	2	3	4	5	6
term_1	50	100	150	5	10	180
term_2	150	100	50	180	10	1
Score(q,d)	**200**	**200**	**200**	**185**	**20**	**181**
BoundSum	300			360		
Bound Tightness	200/300 = 66%			185/360 = 51%		

ϴ = 200

Problem: Loose BoundSums

No new documents entered the top-k!

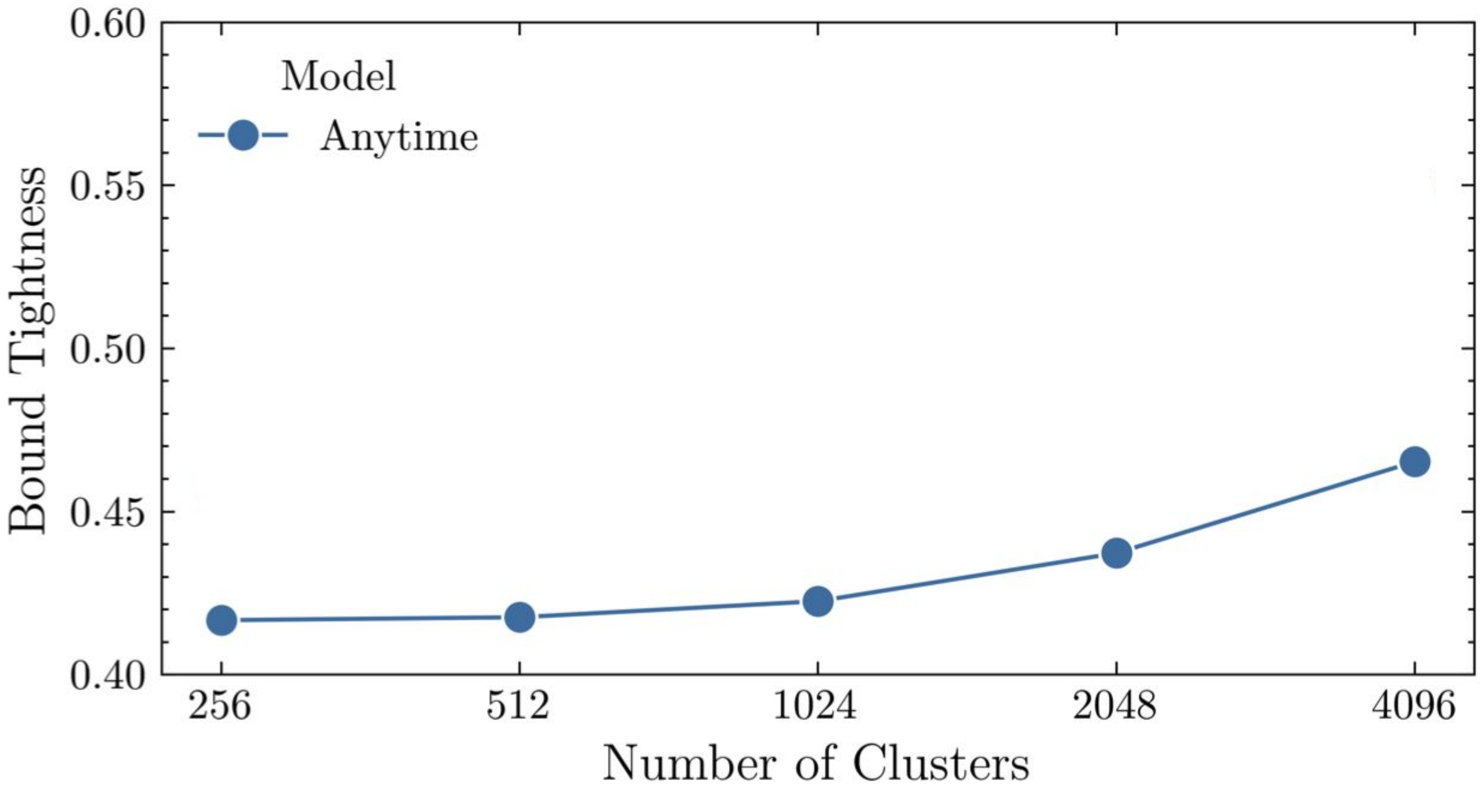
Search
k=3

	Cluster 1			Cluster 2		
doc_id	1	2	3	4	5	6
term_1	50	100	150	5	10	180
term_2	150	100	50	180	10	1
Score(q,d)	**200**	**200**	**200**	**185**	**20**	**181**
BoundSum	300			360		
Bound Tightness	200/300 = 66%			185/360 = 51%		

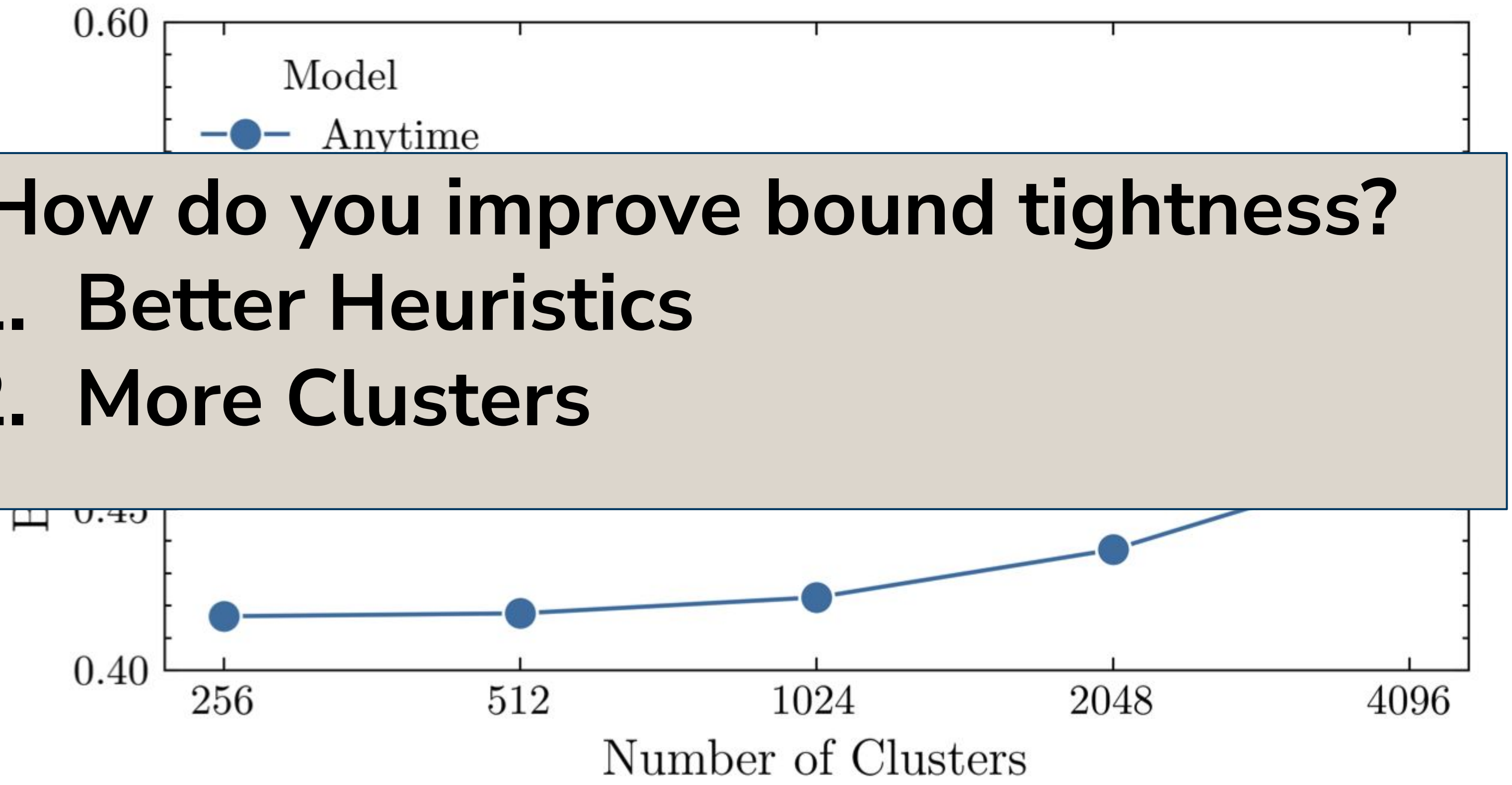
If Bound Tightness was >=93%,
we could have skipped cluster 2!

Θ = 200

Problem: Loose BoundSums



Problem: Loose BoundSums



How do you improve bound tightness?

1. Better Heuristics
2. More Clusters

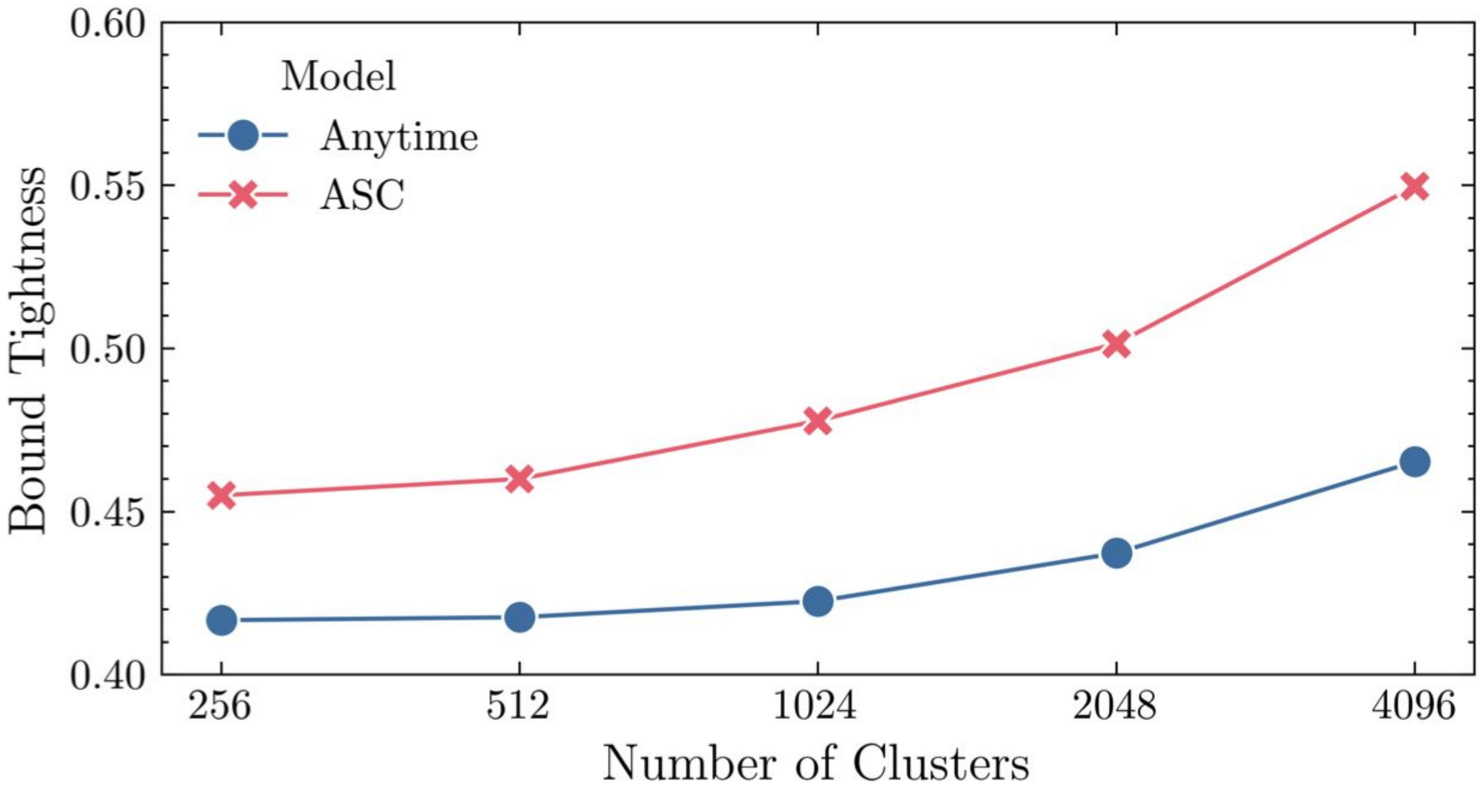
Better Heuristic – ASC (EMNLP 24)

1. Divide clusters into n sub-clusters (typically 8 or 16)
2. Compute BoundSum for each sub-cluster ($B_{i,j}$)
3. Compute MaxSBound and AvgSBound
4. Visit cluster if MaxSBound > Θ or
5. Visit cluster if (MaxS - AvgS) <= Θ

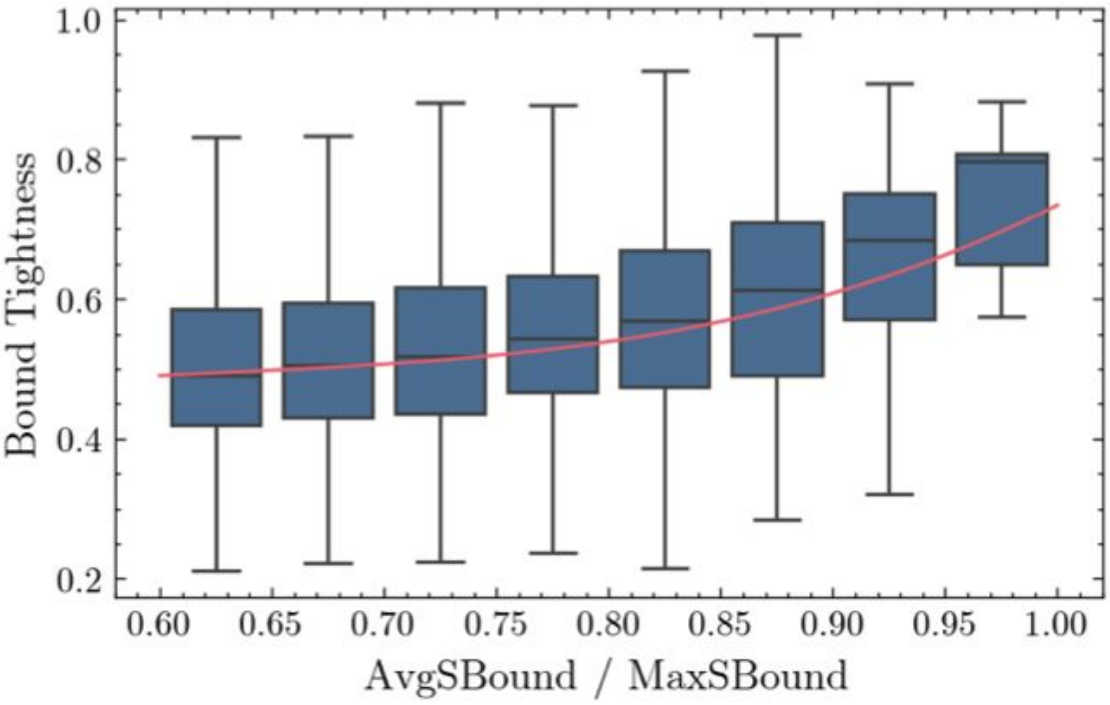
$$MaxSBound(C_i) = \max_{j=1}^{n} B_{i,j},$$

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

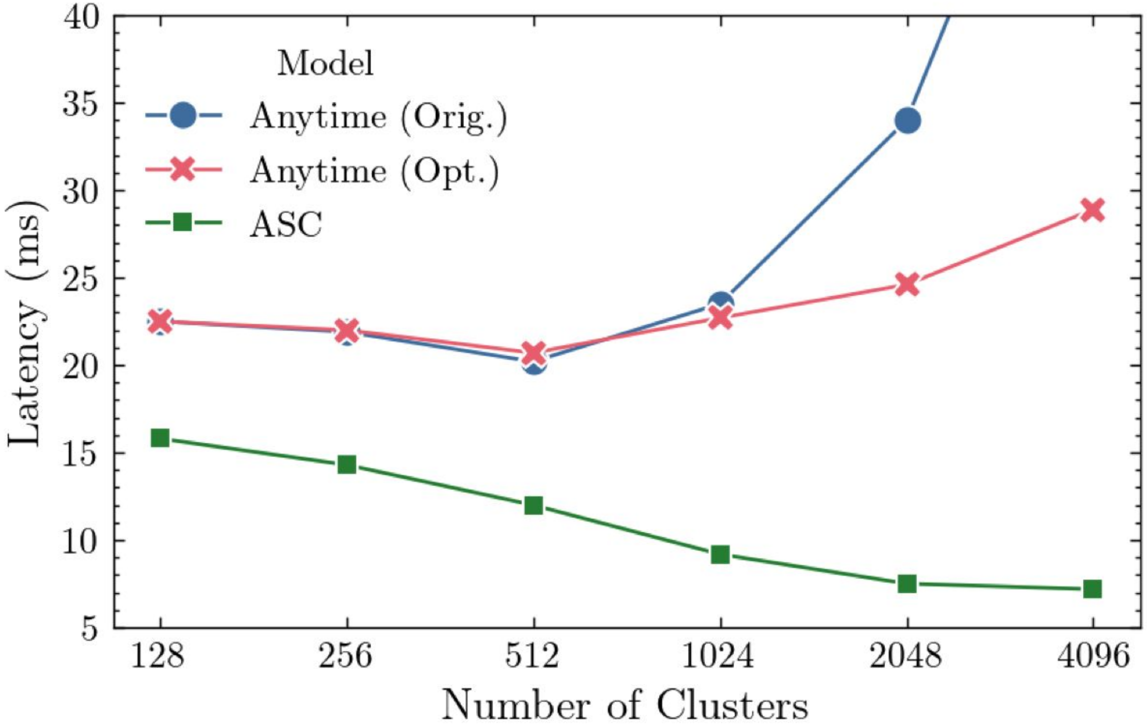
ASC Improve BoundSum tightness



Ratio of MaxSBound and AvgSBound Loosely Predict Bound Tightness



Overhead of Additional Clusters Outweighs the Benefits of ASC's Pruning Heuristic

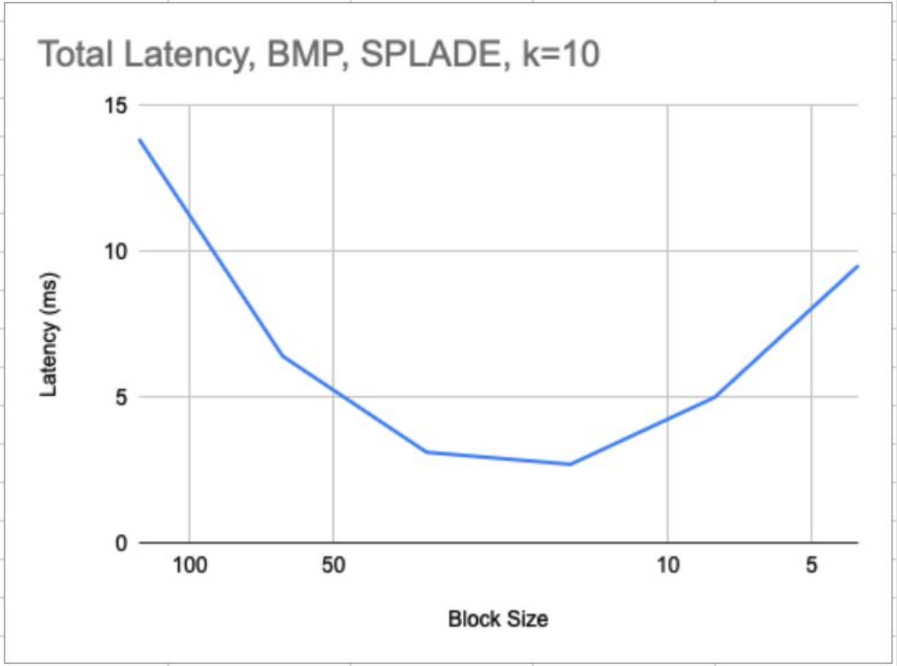


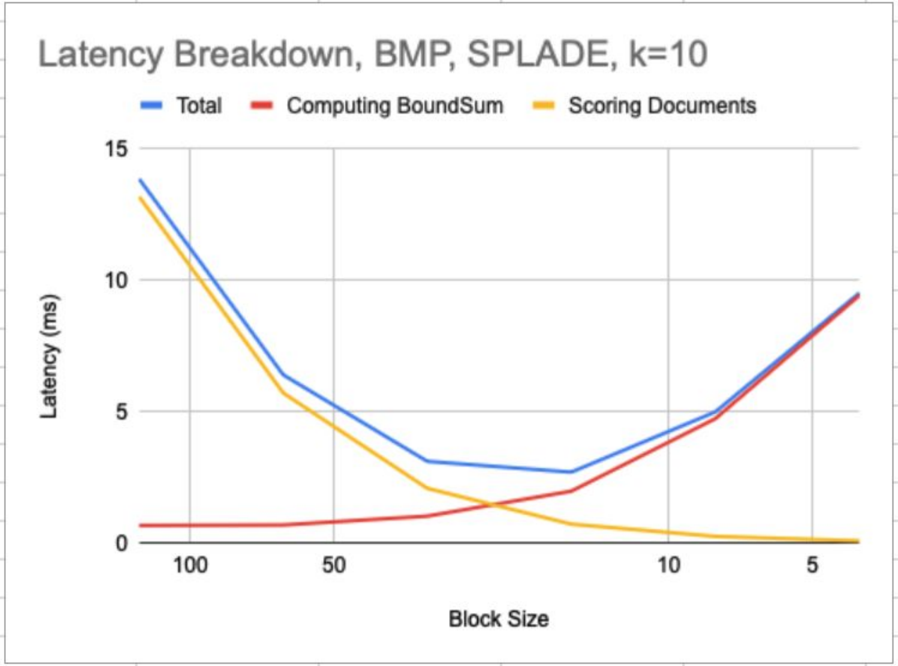
4096 clusters ~= 32K BoundSum computations

More Clusters – BMP (SIGIR 24)

1. Candidate Generation
    a. Compute cluster BoundSums using SIMD
    b. Partially-sort clusters using O(n) counting sort
2. Document Scoring
    a. Score documents within top clusters
    b. When top-k score ϴ is larger than the next cluster score, stop search

Balance of Block Size

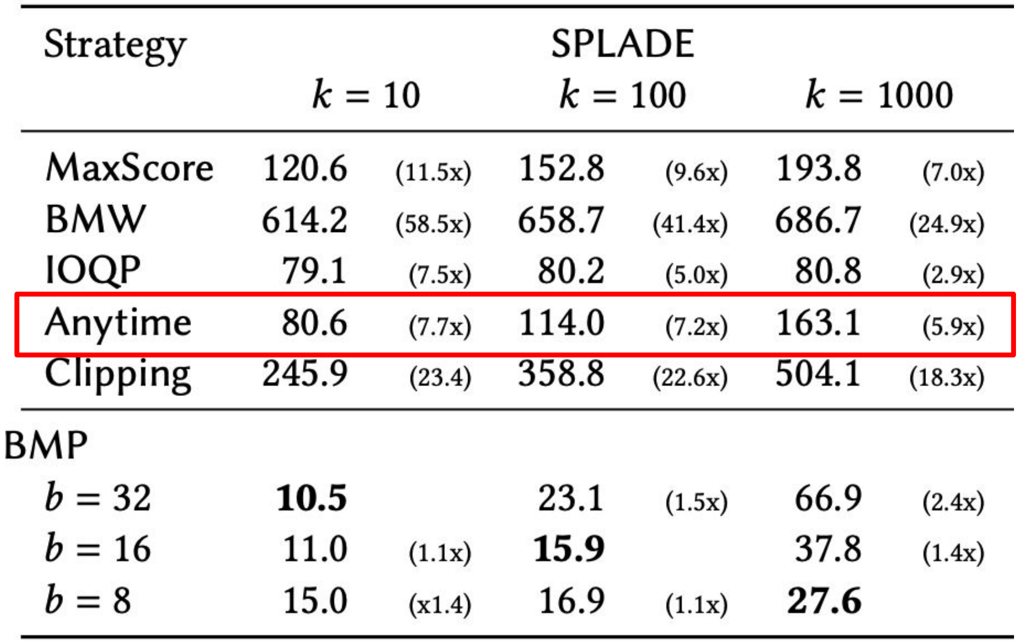




Note: Block Size = 1 / Number of Clusters (with fixed corpus)

Block Size = 8 => **~1.1M clusters**

BMP is better than Anytime and Traditional Inverted Index Methods



Strategy	SPLADE					
	$k = 10$		$k = 100$		$k = 1000$	
MaxScore	120.6	(11.5x)	152.8	(9.6x)	193.8	(7.0x)
BMW	614.2	(58.5x)	658.7	(41.4x)	686.7	(24.9x)
IOQP	79.1	(7.5x)	80.2	(5.0x)	80.8	(2.9x)
Anytime	80.6	(7.7x)	114.0	(7.2x)	163.1	(5.9x)
Clipping	245.9	(23.4)	358.8	(22.6x)	504.1	(18.3x)
BMP						
$b = 32$	**10.5**		23.1	(1.5x)	66.9	(2.4x)
$b = 16$	11.0	(1.1x)	**15.9**		37.8	(1.4x)
$b = 8$	15.0	(x1.4)	16.9	(1.1x)	**27.6**	

256K clusters

512K clusters

1.1M clusters

Query Pruning is very Effective for LSR

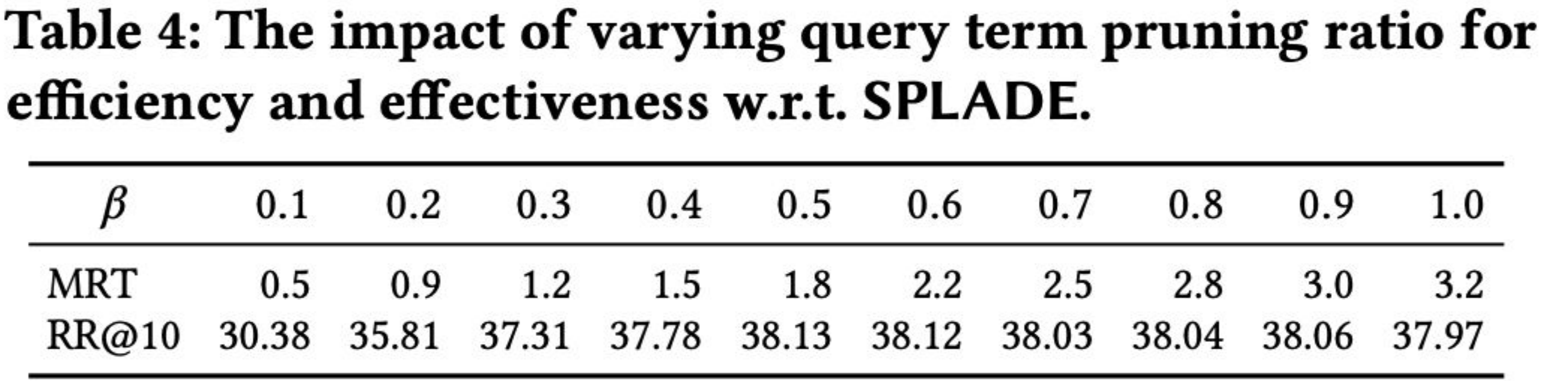


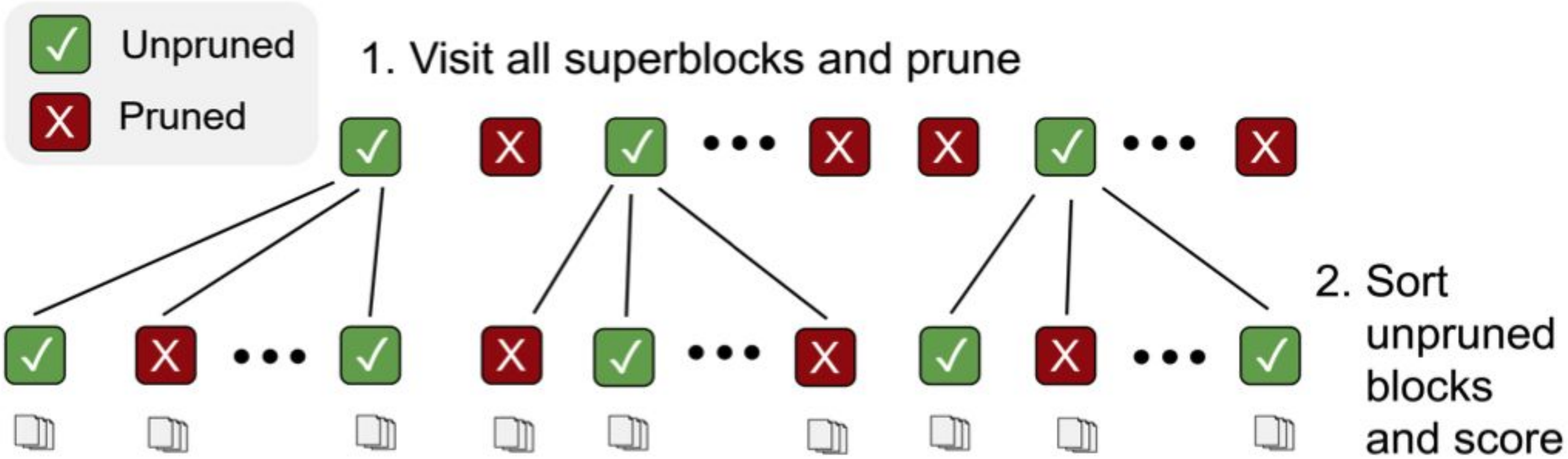
Table 4: The impact of varying query term pruning ratio for efficiency and effectiveness w.r.t. SPLADE.

β	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
MRT	0.5	0.9	1.2	1.5	1.8	2.2	2.5	2.8	3.0	3.2
RR@10	30.38	35.81	37.31	37.78	38.13	38.12	38.03	38.04	38.06	37.97

My Contributions

Superblock Pruning (SIGIR 25)

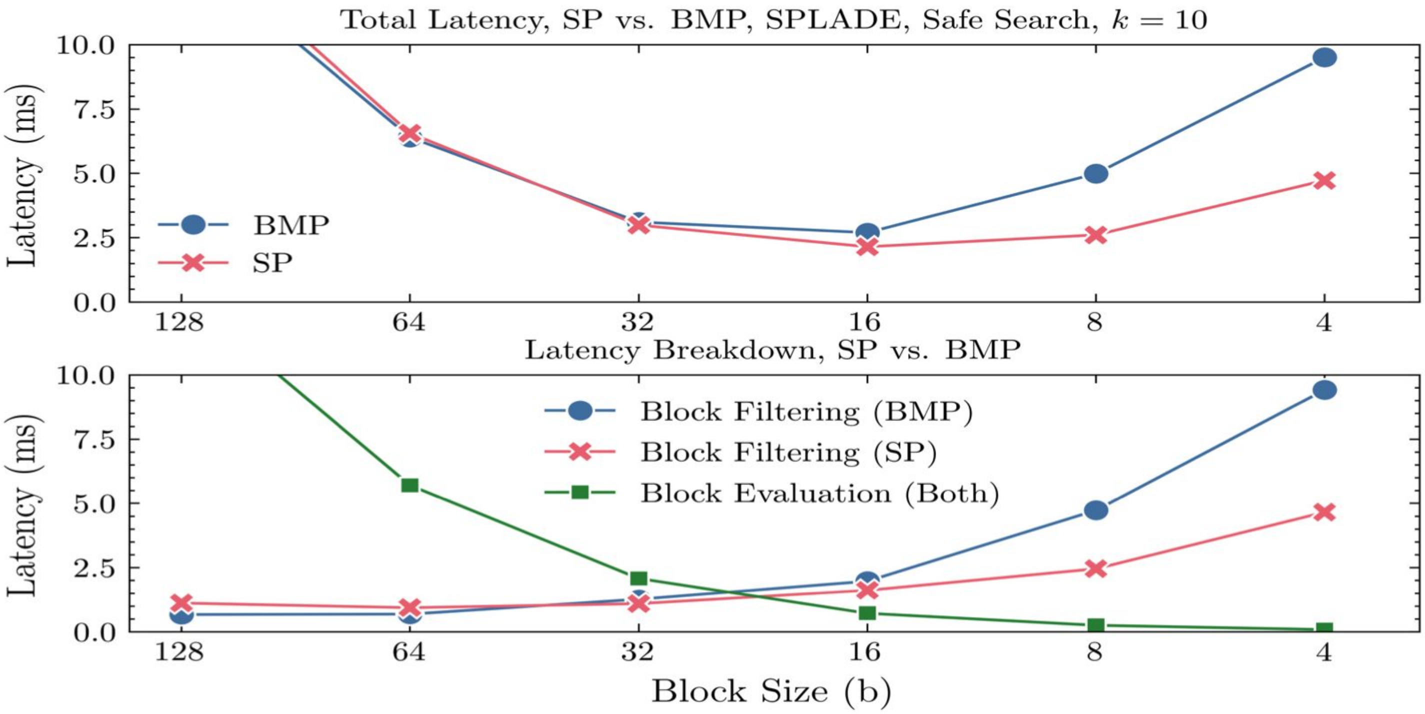
Use a cluster hierarchy to avoid scoring known-bad clusters


✓ Unpruned
X Pruned
1. Visit all superblocks and prune
2. Sort unpruned blocks and score

Superblock Pruning (SIGIR 25)

1. Candidate Generation
    a. Compute Superblock Max & Avgs using SIMD
    b. For unpruned Superblocks, compute their child blocks’ BoundSum with SIMD
    c. Partially-sort clusters using O(n) counting sort
2. Document Scoring
    a. Score documents within top clusters
    b. When top-k score ϴ is larger than the next cluster score, stop search

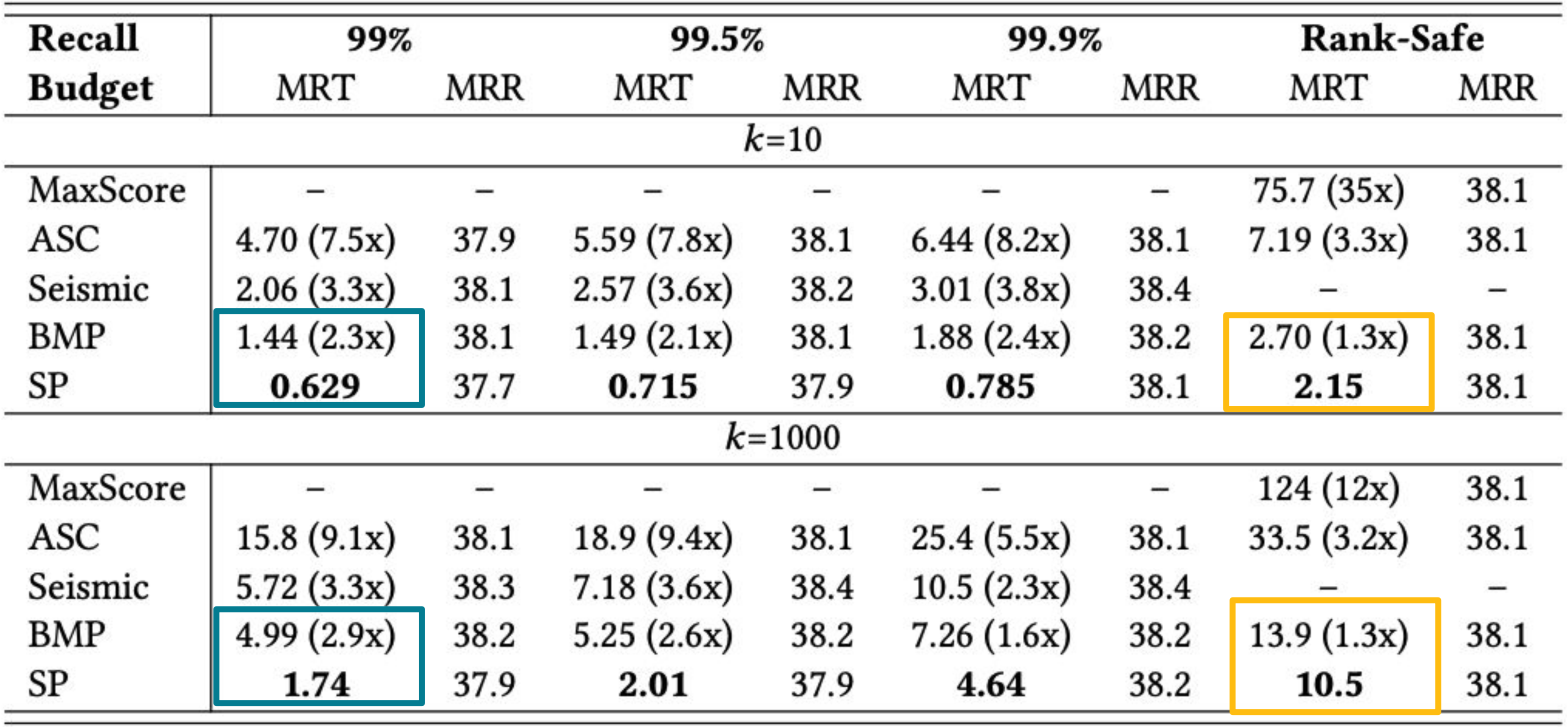
Reduce Candidate Generation Overhead, Maintain Document Scoring Advantages



A Note on Approximate Retrieval

- We can compensate for loose BoundSums using *threshold overestimation.*
- This can significantly reduce latency, but we may miss important documents.
- μ = 1.0 is safe search
- μ < 1.0 is approximate
- We prune a cluster if BoundSum(C) <= ϴ / μ

SP vs BMP and ASC

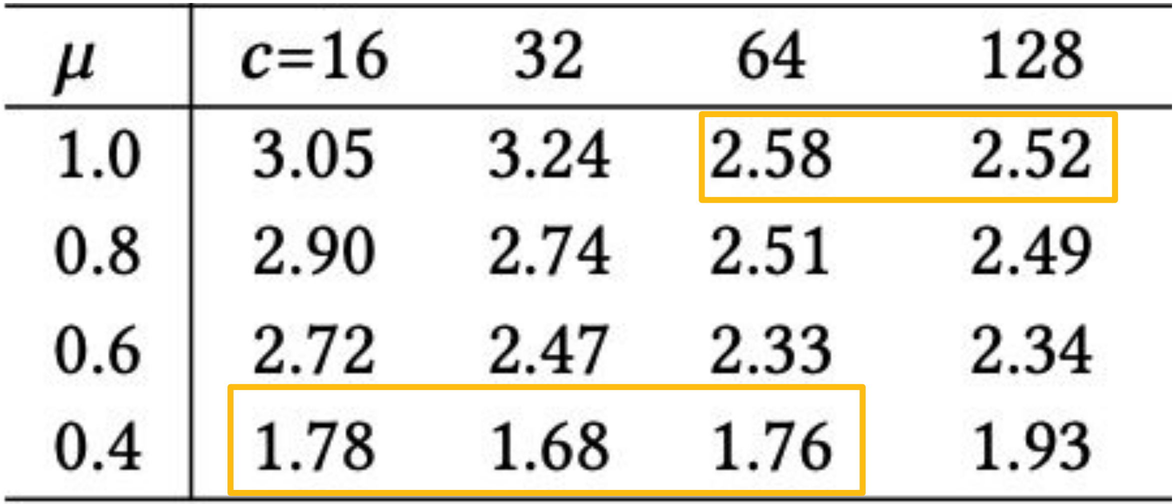


Recall Budget	**99%**		**99.5%**		**99.9%**		**Rank-Safe**	
	MRT	MRR	MRT	MRR	MRT	MRR	MRT	MRR
				k=10				
MaxScore	–	–	–	–	–	–	75.7 (35x)	38.1
ASC	4.70 (7.5x)	37.9	5.59 (7.8x)	38.1	6.44 (8.2x)	38.1	7.19 (3.3x)	38.1
Seismic	2.06 (3.3x)	38.1	2.57 (3.6x)	38.2	3.01 (3.8x)	38.4	–	–
BMP	1.44 (2.3x)	38.1	1.49 (2.1x)	38.1	1.88 (2.4x)	38.2	2.70 (1.3x)	38.1
SP	**0.629**	37.7	**0.715**	37.9	**0.785**	38.1	**2.15**	38.1
				k=1000				
MaxScore	–	–	–	–	–	–	124 (12x)	38.1
ASC	15.8 (9.1x)	38.1	18.9 (9.4x)	38.1	25.4 (5.5x)	38.1	33.5 (3.2x)	38.1
Seismic	5.72 (3.3x)	38.3	7.18 (3.6x)	38.4	10.5 (2.3x)	38.4	–	–
BMP	4.99 (2.9x)	38.2	5.25 (2.6x)	38.2	7.26 (1.6x)	38.2	13.9 (1.3x)	38.1
SP	**1.74**	37.9	**2.01**	37.9	**4.64**	38.2	**10.5**	38.1

How Many Blocks Per Superblock?

Latency (ms) for different number of child blocks (c)

# SIMD Registers = 1 2 4 8



μ	c=16	32	64	128
1.0	3.05	3.24	2.58	2.52
0.8	2.90	2.74	2.51	2.49
0.6	2.72	2.47	2.33	2.34
0.4	1.78	1.68	1.76	1.93

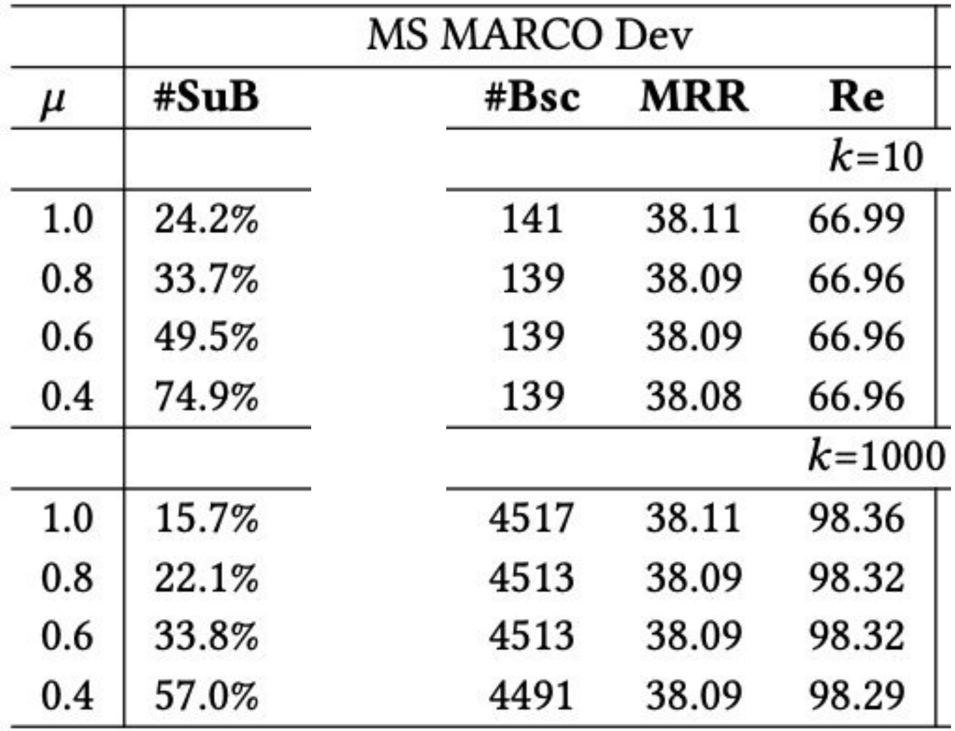
c >= 64 is preferred for safe search

Pruning more decreases latency

c = [16,64] are good choices for approximate retrieval

Pruning Sensitivity

Because we only apply μ to the SB level, we maintain quality

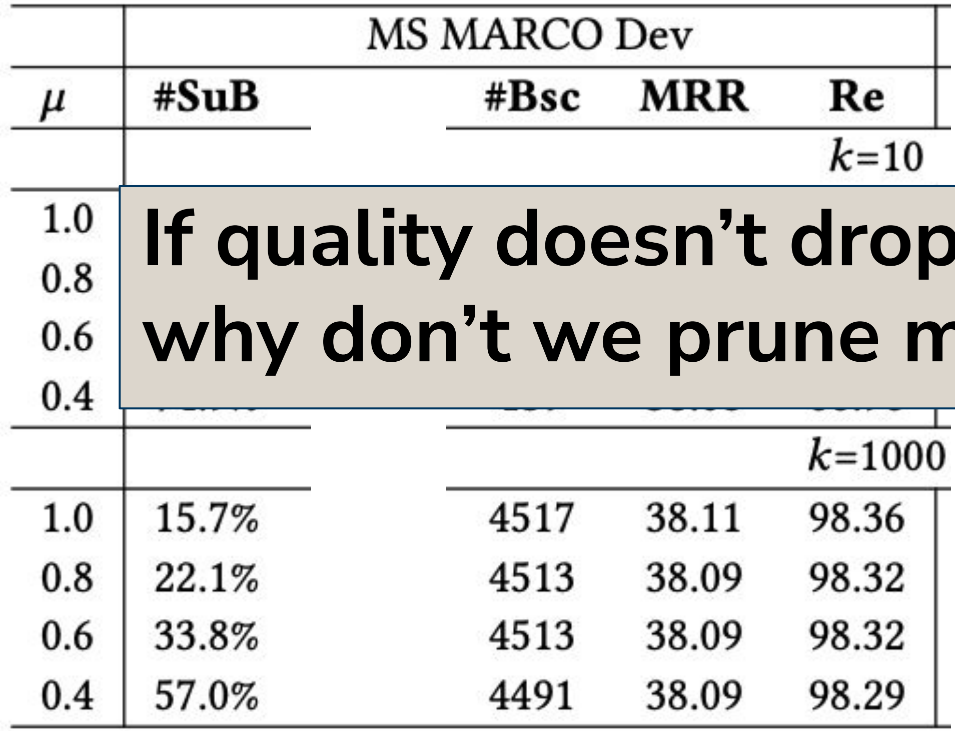


	MS MARCO Dev			
μ	#SuB	#Bsc	MRR	Re
				k=10
1.0	24.2%	141	38.11	66.99
0.8	33.7%	139	38.09	66.96
0.6	49.5%	139	38.09	66.96
0.4	74.9%	139	38.08	66.96
				k=1000
1.0	15.7%	4517	38.11	98.36
0.8	22.1%	4513	38.09	98.32
0.6	33.8%	4513	38.09	98.32
0.4	57.0%	4491	38.09	98.29

BMP with μ = 0.8 only gets 62.6 R@10!

Pruning Sensitivity

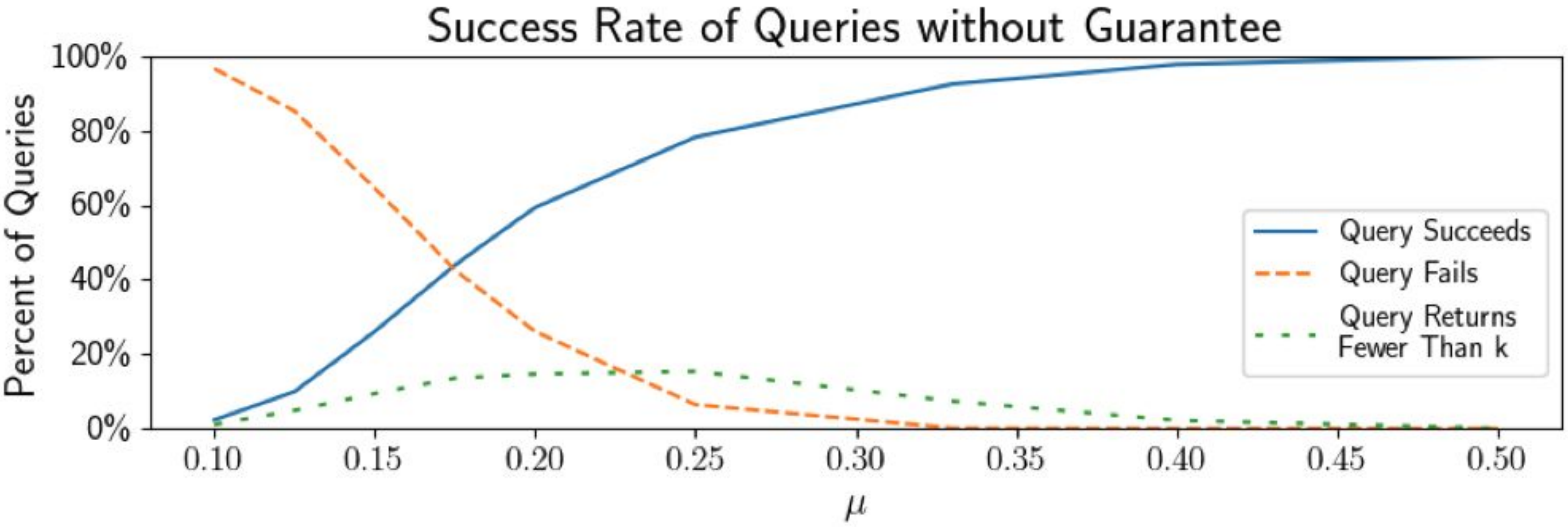
Because we only apply μ to the SB level, we maintain quality



	MS MARCO Dev			
μ	**#SuB**	**#Bsc**	**MRR**	**Re**
				k=10
1.0				
0.8				
0.6				
0.4				
				k=1000
1.0	15.7%	4517	38.11	98.36
0.8	22.1%	4513	38.09	98.32
0.6	33.8%	4513	38.09	98.32
0.4	57.0%	4491	38.09	98.29

If quality doesn't drop with aggressive μ, why don't we prune more?

SP is Susceptible to Overpruning SBs



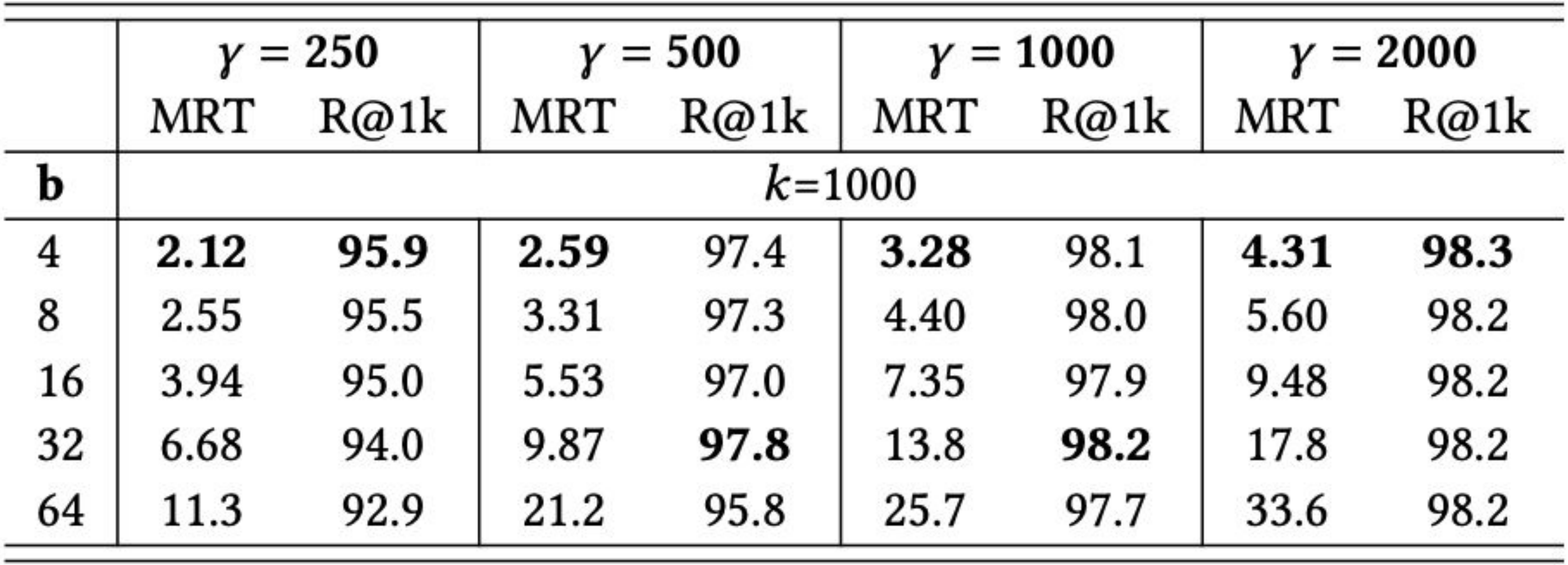
Lightweight Superblock Pruning (Under Review)

Build on SP with Three Goals:

1. Address superblock overpruning
    a. Analyze choice of SB pruning heuristic
2. Compress the index without affecting latency
3. Simplify parameter choices & enable zero-shot approximate retrieval

Address Superblock Overpruning

Guarantee at least γ superblocks are scored for every query



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

Explore SB Pruning Heuristic

LSP compares three SB pruning heuristics

- **LSP/0** (default) – only search the top-gamma superblocks
- **LSP/1** – use BoundSum at the superblock level. Ensure at least gamma superblocks are scored.
- **LSP/2** – use the ASC pruning heuristic at the superblock level. Ensure at least gamma superblocks are scored

Searching a Fixed Number of Superblocks

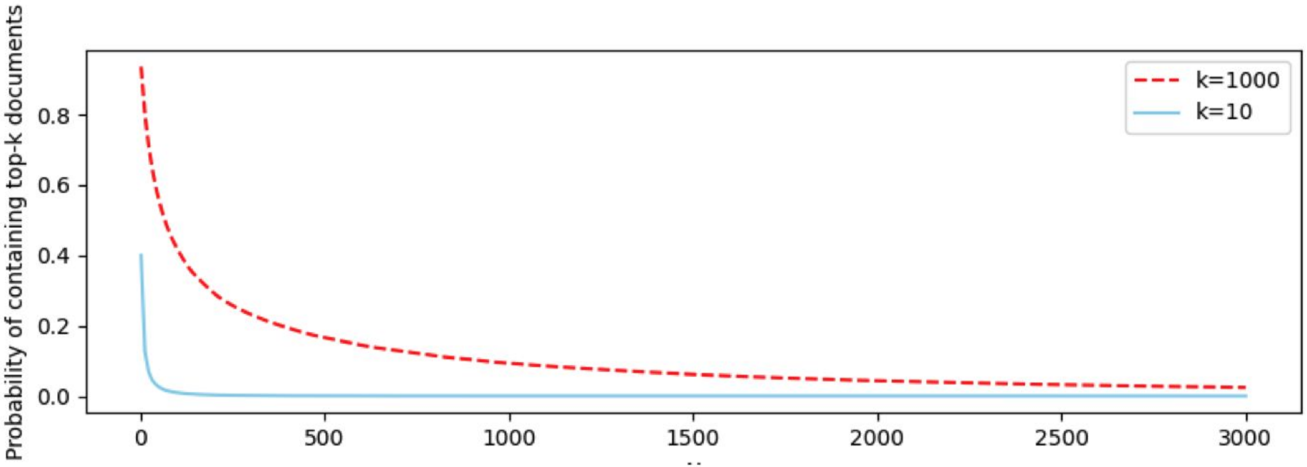
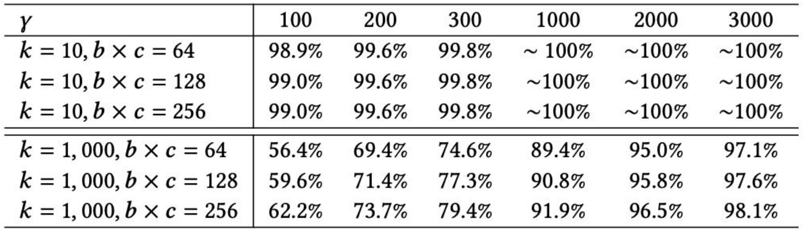


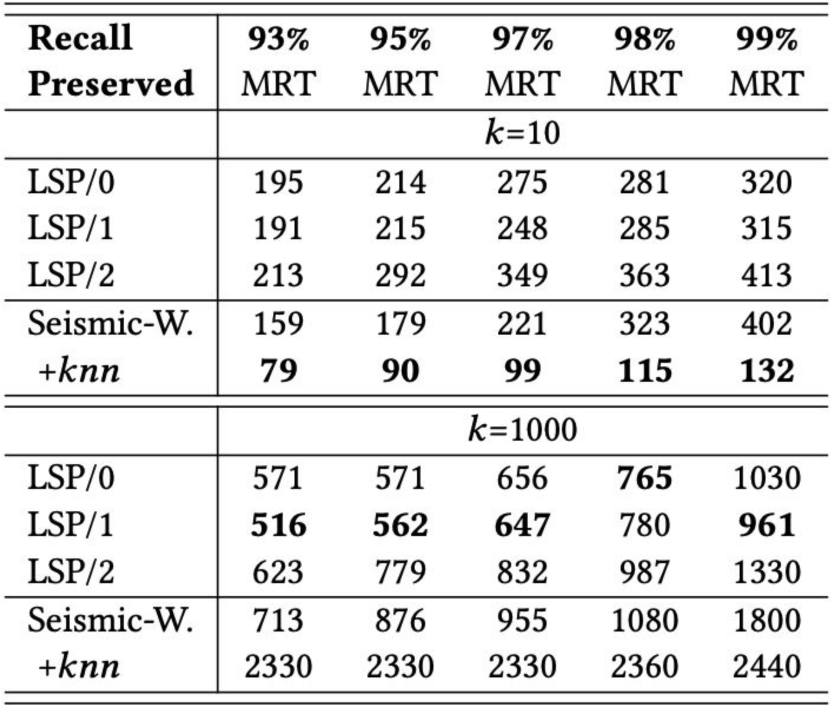
Table 1: Confidence $P_\gamma(I)$ based on MS MARCO training data



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

Which SB Pruning Heuristic is Best?

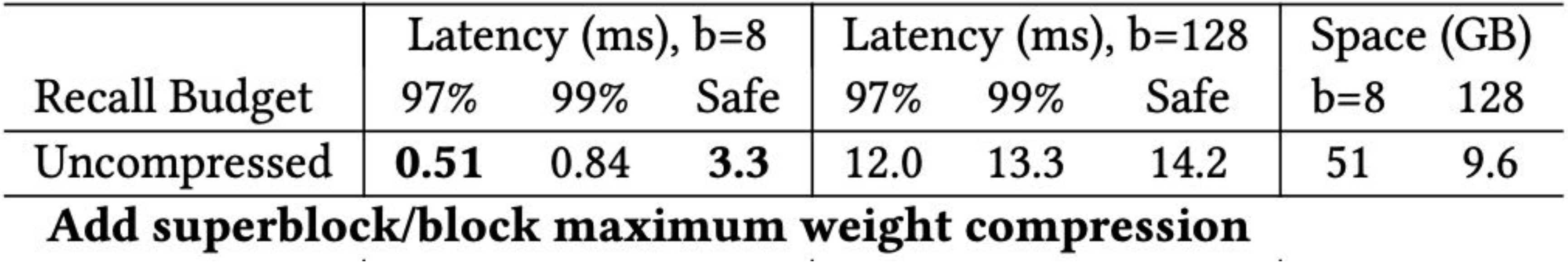
LSP/1 (μ & γ) is best; LSP/0 (γ only) is competitive



Recall Preserved	**93%** MRT	**95%** MRT	**97%** MRT	**98%** MRT	**99%** MRT
			k=10		
LSP/0	195	214	275	281	320
LSP/1	191	215	248	285	315
LSP/2	213	292	349	363	413
Seismic-W.	159	179	221	323	402
+*knn*	**79**	**90**	**99**	**115**	**132**
			k=1000		
LSP/0	571	571	656	**765**	1030
LSP/1	**516**	**562**	**647**	780	**961**
LSP/2	623	779	832	987	1330
Seismic-W.	713	876	955	1080	1800
+*knn*	2330	2330	2330	2360	2440

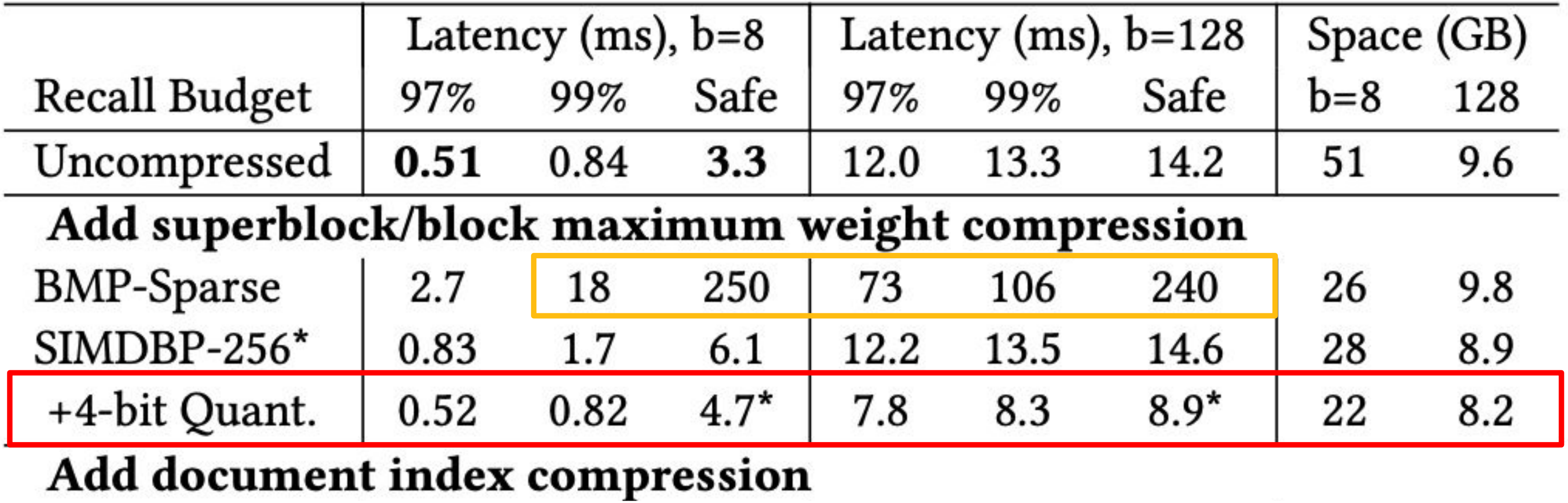
Because LSP/0 is close to LSP/1 after an exhaustive parameter grid search, we make LSP/0 our default configuration because it is easier to select effective parameters

Index Compression



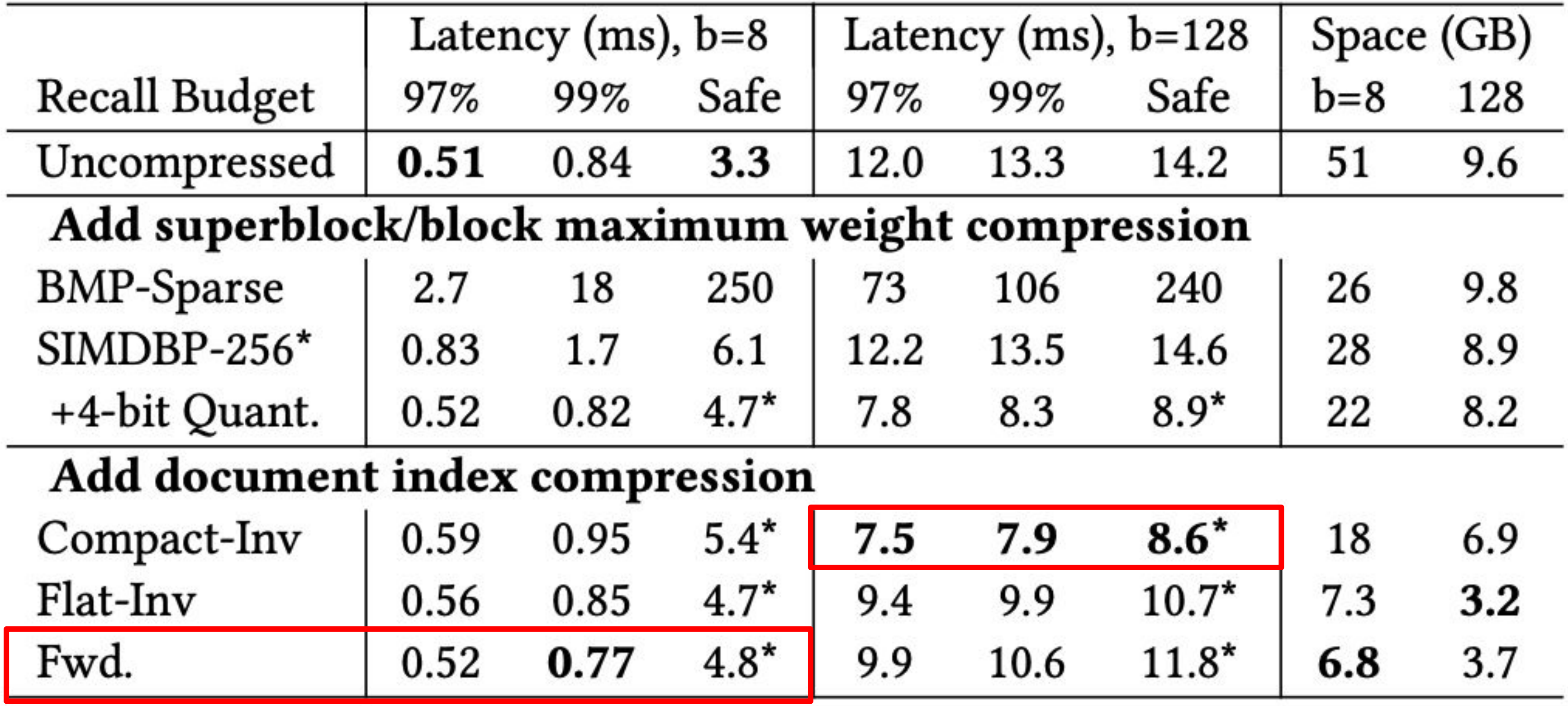
	Latency (ms), b=8			Latency (ms), b=128			Space (GB)	
Recall Budget	97%	99%	Safe	97%	99%	Safe	b=8	128
Uncompressed	**0.51**	0.84	**3.3**	12.0	13.3	14.2	51	9.6
Add superblock/block maximum weight compression								

Index Compression



	Latency (ms), b=8			Latency (ms), b=128			Space (GB)	
Recall Budget	97%	99%	Safe	97%	99%	Safe	b=8	128
Uncompressed	**0.51**	0.84	**3.3**	12.0	13.3	14.2	51	9.6
Add superblock/block maximum weight compression								
BMP-Sparse	2.7	18	250	73	106	240	26	9.8
SIMDBP-256*	0.83	1.7	6.1	12.2	13.5	14.6	28	8.9
+4-bit Quant.	0.52	0.82	4.7*	7.8	8.3	8.9*	22	8.2
Add document index compression								

Index Compression



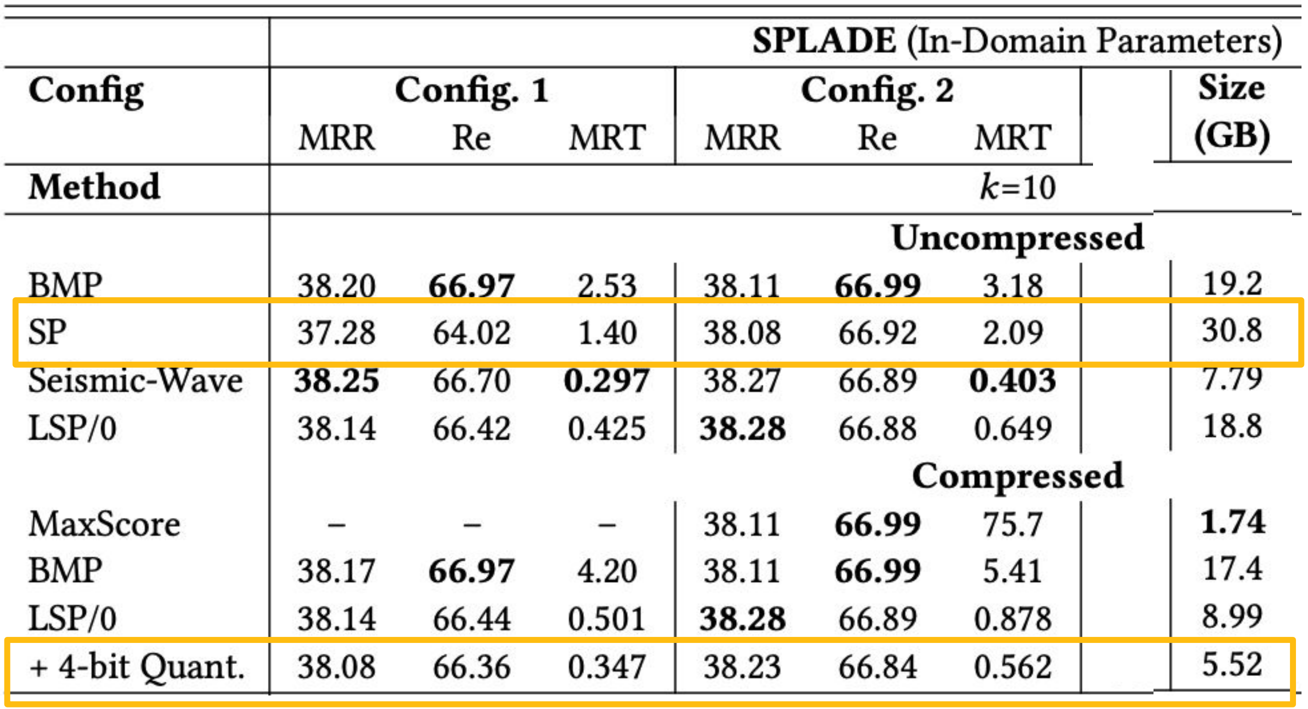
	Latency (ms), b=8			Latency (ms), b=128			Space (GB)	
Recall Budget	97%	99%	Safe	97%	99%	Safe	b=8	128
Uncompressed	**0.51**	0.84	**3.3**	12.0	13.3	14.2	51	9.6
Add superblock/block maximum weight compression								
BMP-Sparse	2.7	18	250	73	106	240	26	9.8
SIMDBP-256*	0.83	1.7	6.1	12.2	13.5	14.6	28	8.9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

Zero-Shot LSP Configuration

For k=10:

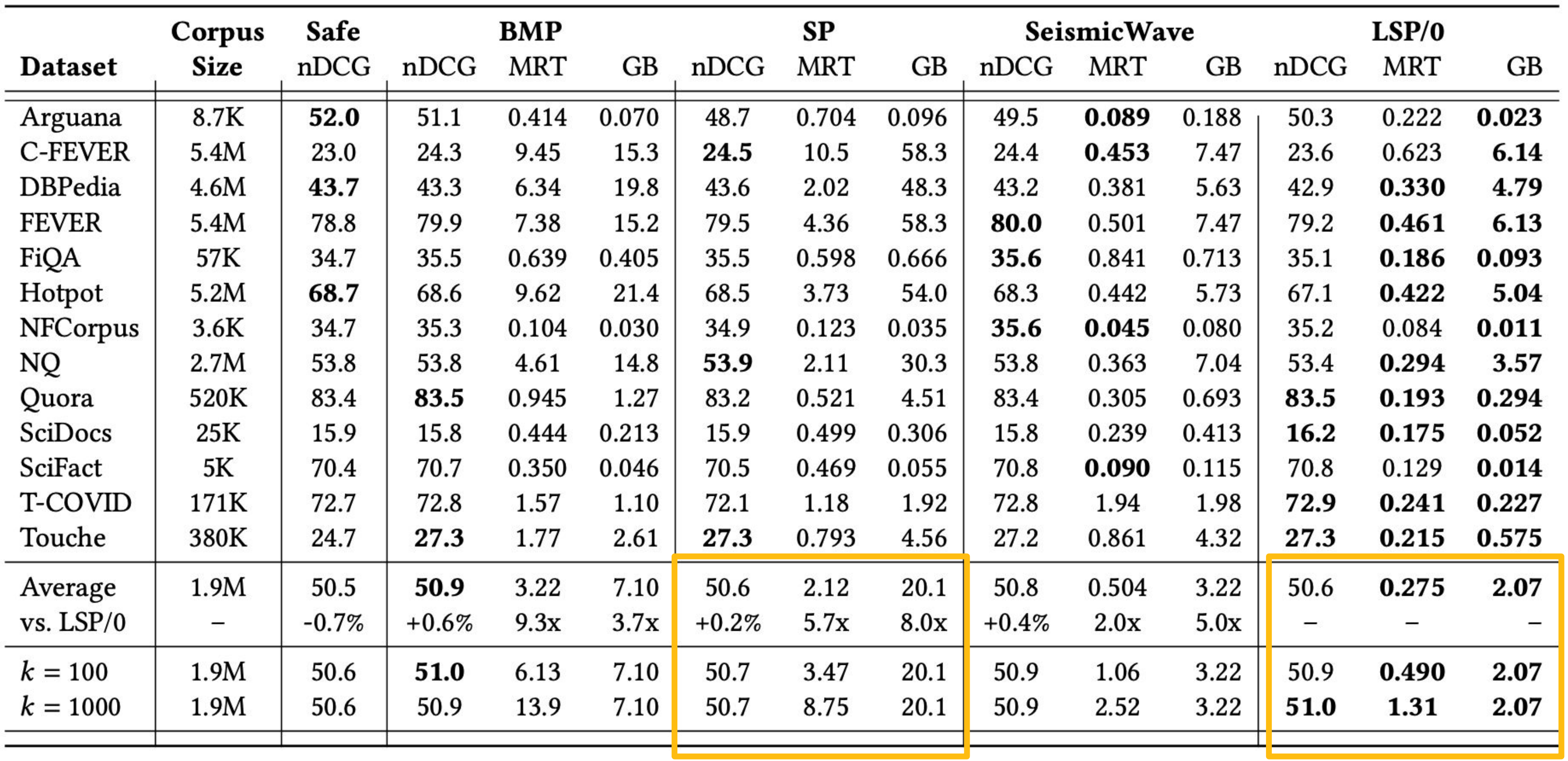
- Index Settings:
    - b = 8, c = 16, 4-bit compressed block max weights, fwd. index
- Query-Processing Settings
    - Config 1 (aggressive):
        - Gamma = 250, beta = 0.33
    - Config 2 (conservative):
        - Gamma = 500, beta = 0.5

Zero-Shot LSP – In Domain



	SPLADE (In-Domain Parameters)						
Config	**Config. 1**			**Config. 2**			**Size**
	MRR	Re	MRT	MRR	Re	MRT	**(GB)**
Method	k=10						
	Uncompressed						
BMP	38.20	**66.97**	2.53	38.11	**66.99**	3.18	19.2
SP	37.28	64.02	1.40	38.08	66.92	2.09	30.8
Seismic-Wave	**38.25**	66.70	**0.297**	38.27	66.89	**0.403**	7.79
LSP/0	38.14	66.42	0.425	**38.28**	66.88	0.649	18.8
	Compressed						
MaxScore	–	–	–	38.11	**66.99**	75.7	**1.74**
BMP	38.17	**66.97**	4.20	38.11	**66.99**	5.41	17.4
LSP/0	38.14	66.44	0.501	**38.28**	66.89	0.878	8.99
+ 4-bit Quant.	38.08	66.36	0.347	38.23	66.84	0.562	5.52

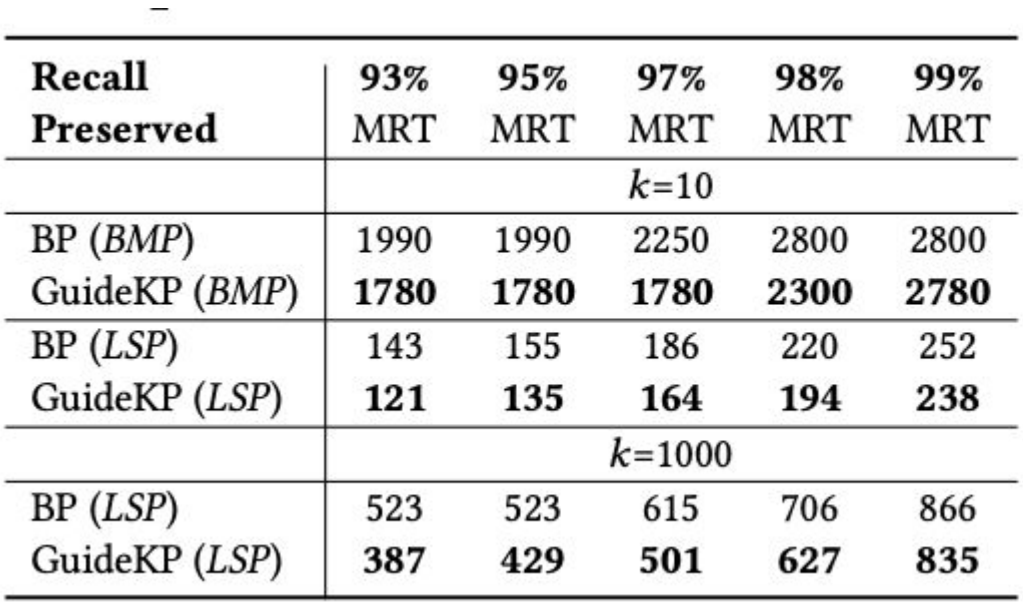
Zero-Shot LSP – Out of Domain



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

Future Work

Better Clustering (Under Review)



Recall Preserved	**93%** MRT	**95%** MRT	**97%** MRT	**98%** MRT	**99%** MRT
			k=10		
BP (*BMP*)	1990	1990	2250	2800	2800
GuideKP (*BMP*)	**1780**	**1780**	**1780**	**2300**	**2780**
BP (*LSP*)	143	155	186	220	252
GuideKP (*LSP*)	**121**	**135**	**164**	**194**	**238**
			k=1000		
BP (*LSP*)	523	523	615	706	866
GuideKP (*LSP*)	**387**	**429**	**501**	**627**	**835**

Larger Datasets, Larger LSR Models

- MS MARCO only has 8.8M passages, MSM v2 has 138M passages.
  - Other, larger web-scale collections have 1B+ passages
- Latest LSR models have longer queries and documents
  - SPLADE has ~43 query terms and ~119 document terms on average
  - Lion-SP has ~293 query terms and ~1250 document terms on average!
- SPLADE's vocabulary is ~27K tokens, Lion's vocabulary is 128K
  - This means storing ~5x more Block Max weights in the index

Beyond the Tokenizer

- LLMs with fixed vocabularies are often not preferred for web-scale retrieval
  - Can’t match obscure terms or numbers
  - Many users still prefer BM25
- DeepImpact applies LSR to an arbitrary vocabulary

Better Scoring Functions

- We can use the sparse candidate generation as a form of retrieval itself, and replace the sparse document index with another document index, such as single-vector dense or multi-vector dense, or a fusion of them.

UC SANTA BARBARA