Vector Search test at scale

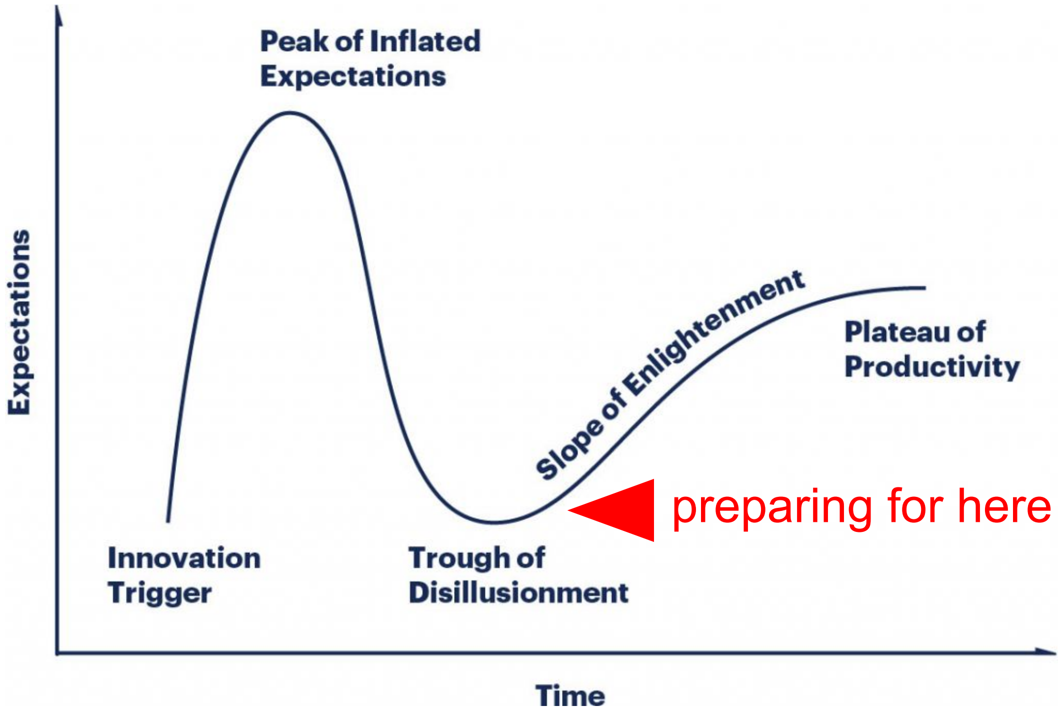
Mohit Sidana, Search Architect
Tom Burgmans, Technology Product Owner Search

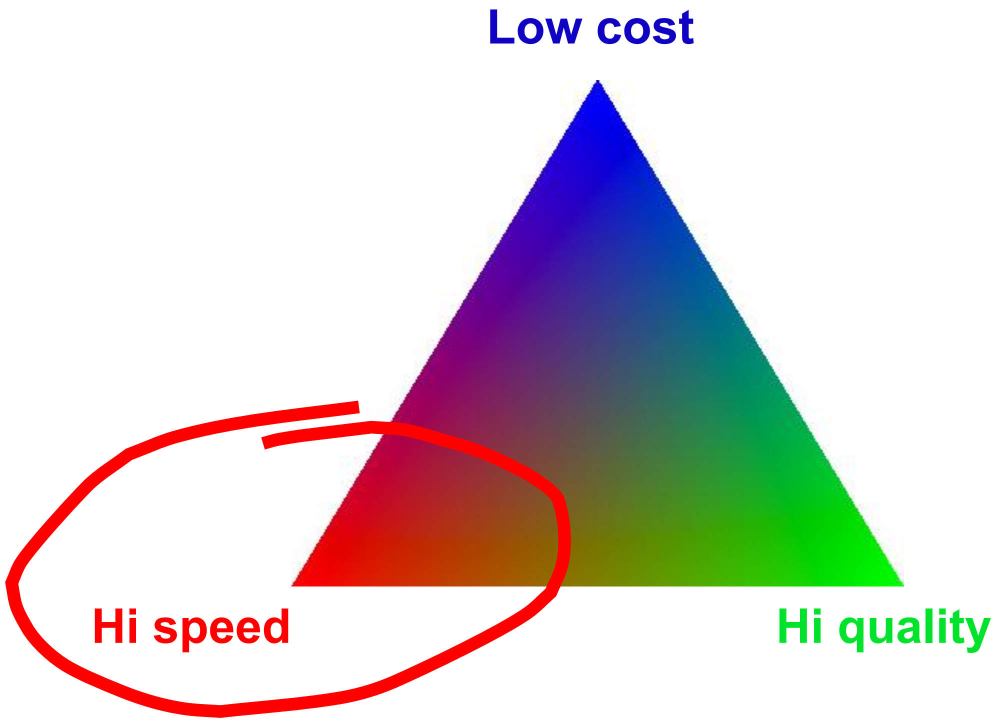
April 23rd 2024

Wolters Kluwer

Introduction

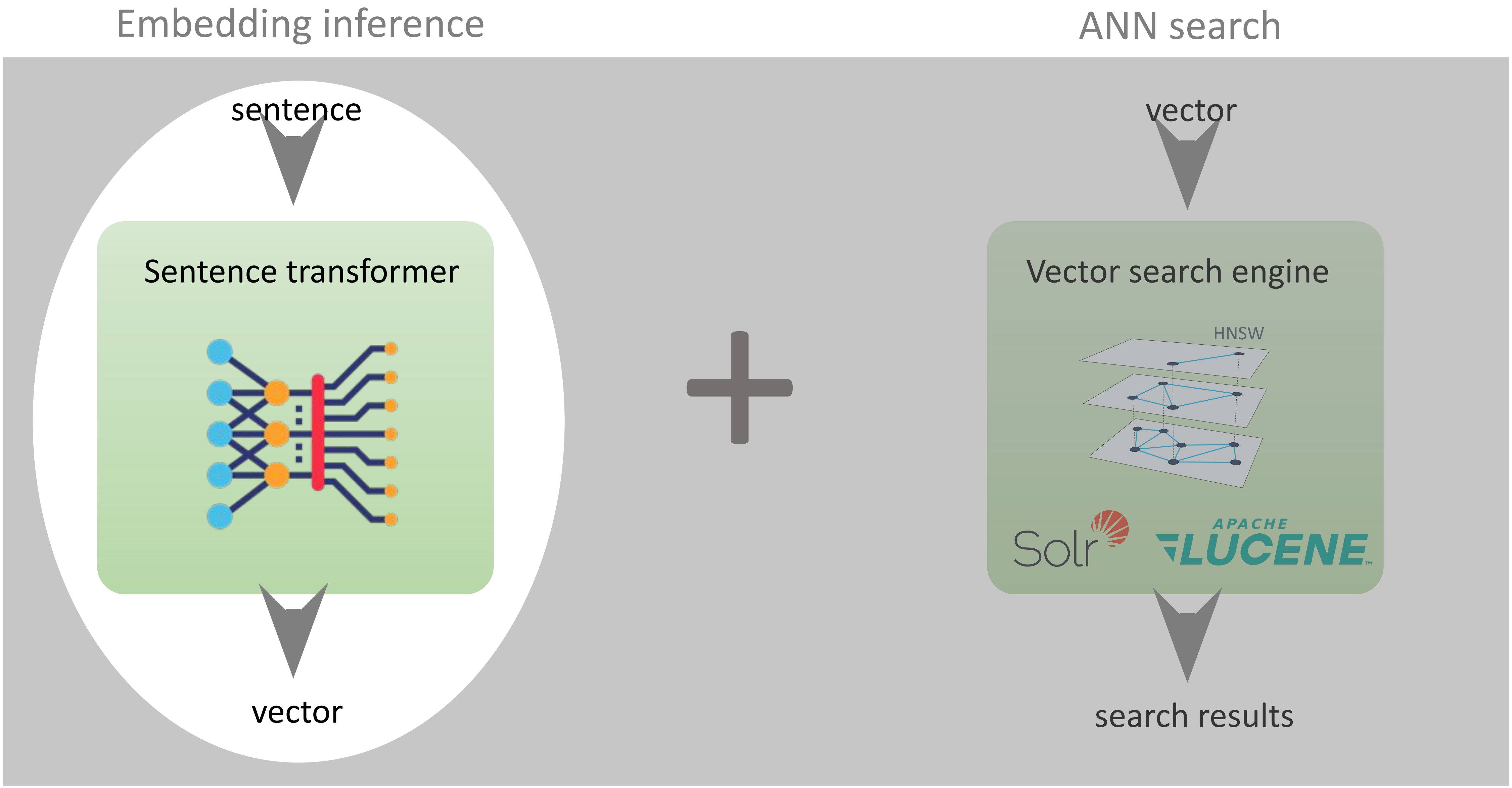
Vector search enables searching for meaning. It has great potential for information retrieval.
Let's pierce through the hype and get prepared for production-like use cases.





Wolters Kluwer

Query latency =



Wolters Kluwer

The defaults for performance testing embedding inference

Query test set	
# of queries	**1000**
Queries sizes (in tokens)	**varying between 4 and 32**
Queries text	**English phrases**

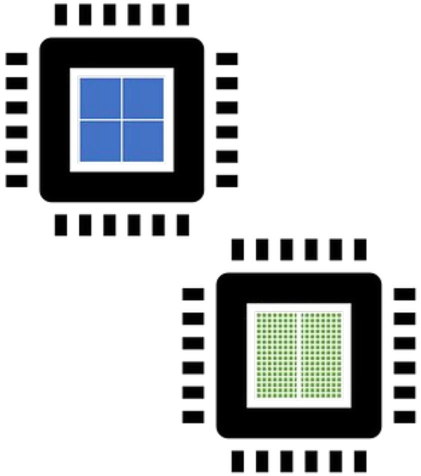
Load test settings	
# of threads	**100**
Pause between transactions	**1000 ms**
Duration per test	**5 min**

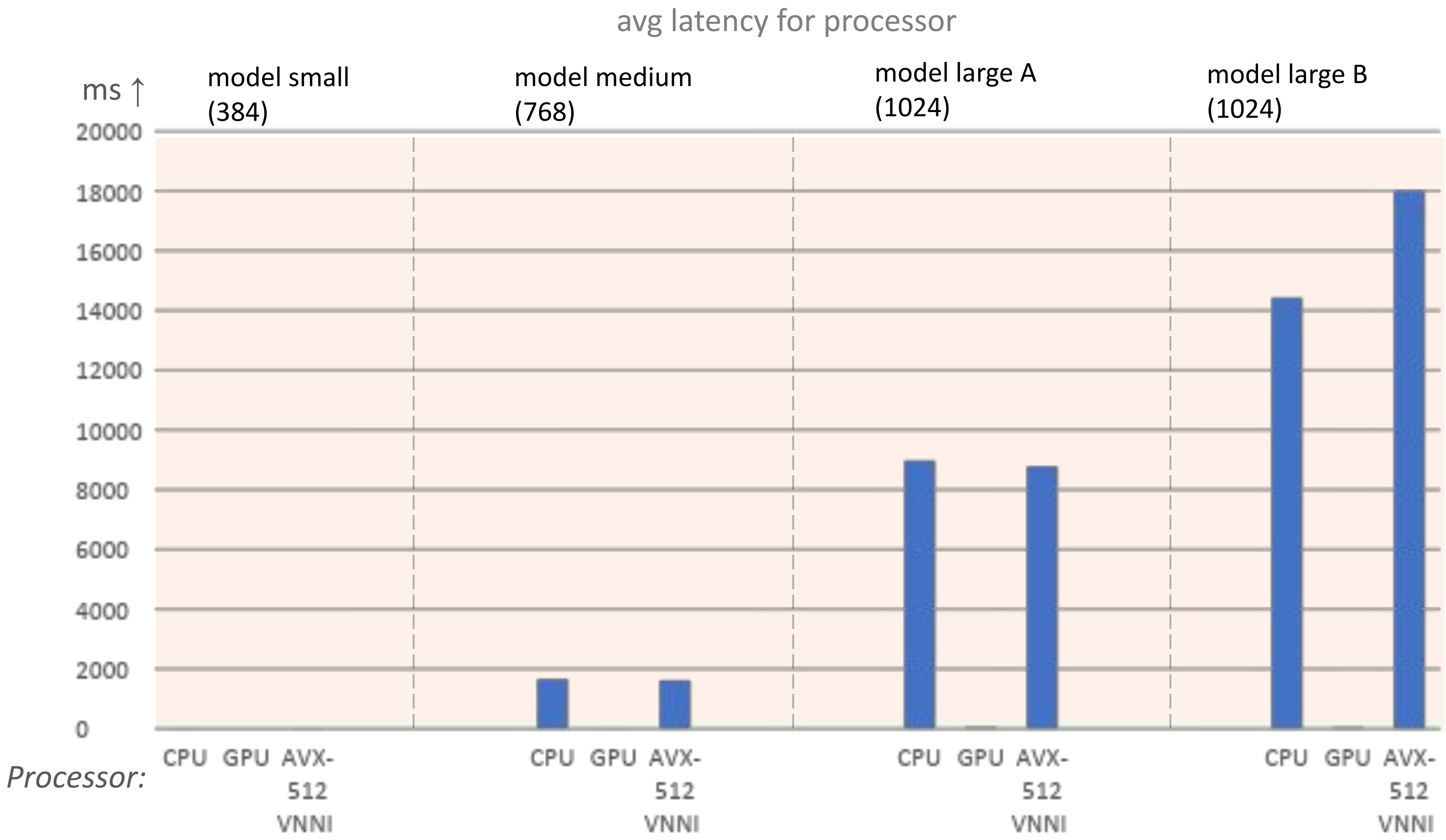
Sentence transformer models	
Small (384 dimensions)	**tavakolih/all-MiniLM-L6-v2-pubmed-full**
Medium (768 dimensions)	**pritamdeka/S-PubMedBert-MS-MARCO**
Large A (1024 dimensions)	**E5-large-v2**
Large B (1024 dimensions)	**thenlper/gte-large**

System under test	
EC2 type	**g4dn.xlarge**
CPU cores	**4**
GPU	**1**
Total Memory	**16 Gb**

Model/Vector conversions	
ONNX	**yes**
Vector to numpy	**yes**
Vector normalized to unit length	**No**
Quantized to int8	**No**
Graph Optimization	**No**

Processing Unit on embedding inference latency

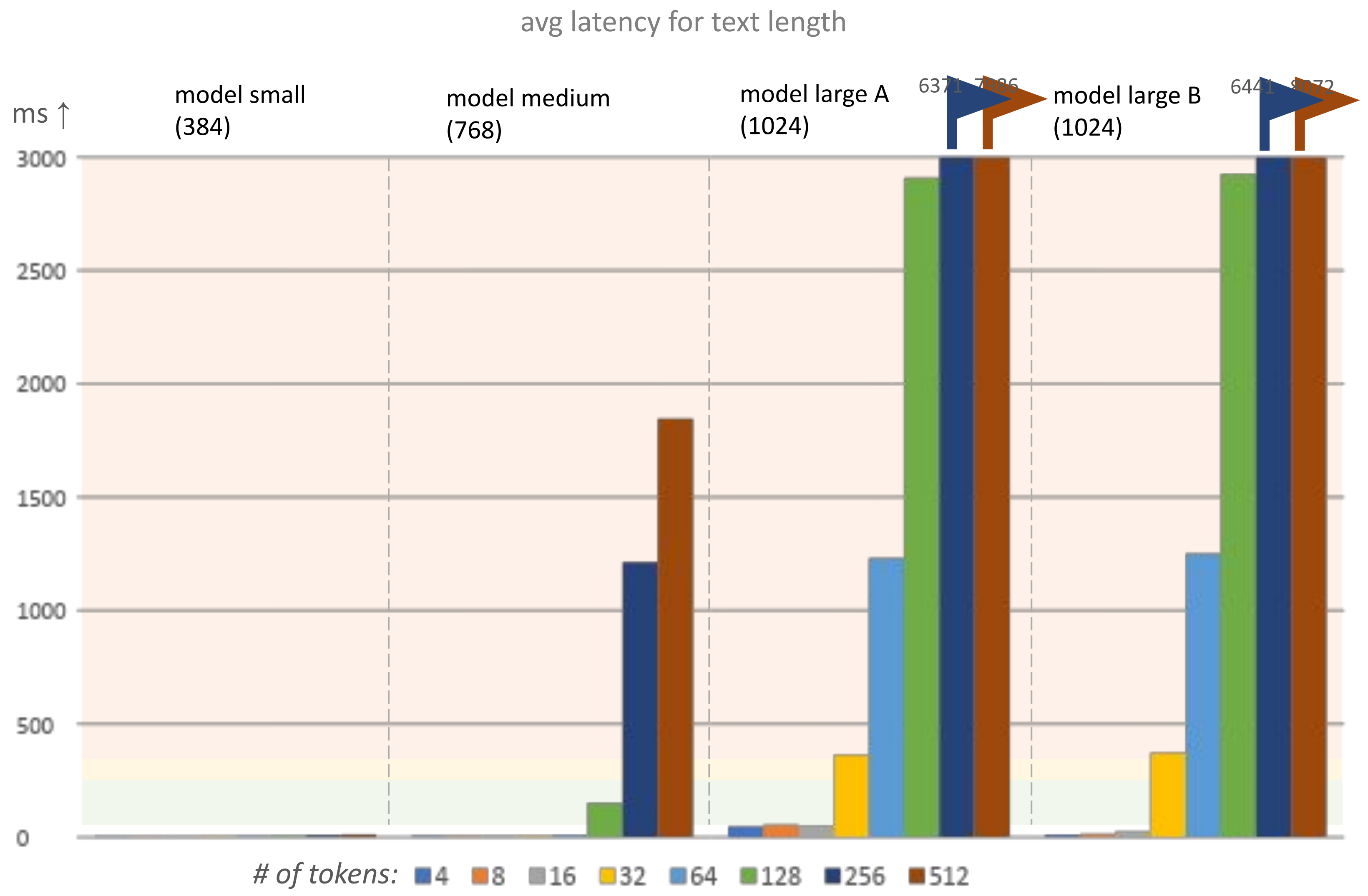




Lessons learned: GPU is optimal for embedding inference tasks

Text length on embedding inference latency

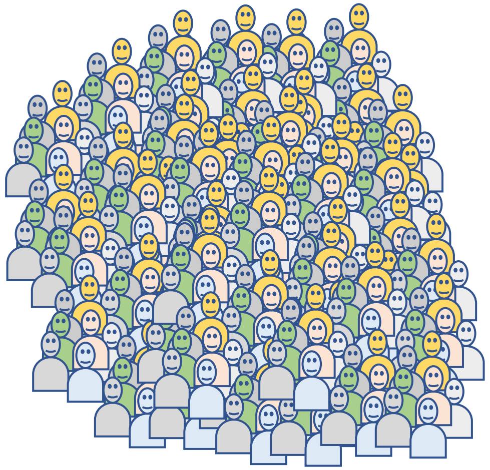

```
Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut
enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut
enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum
dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit
```

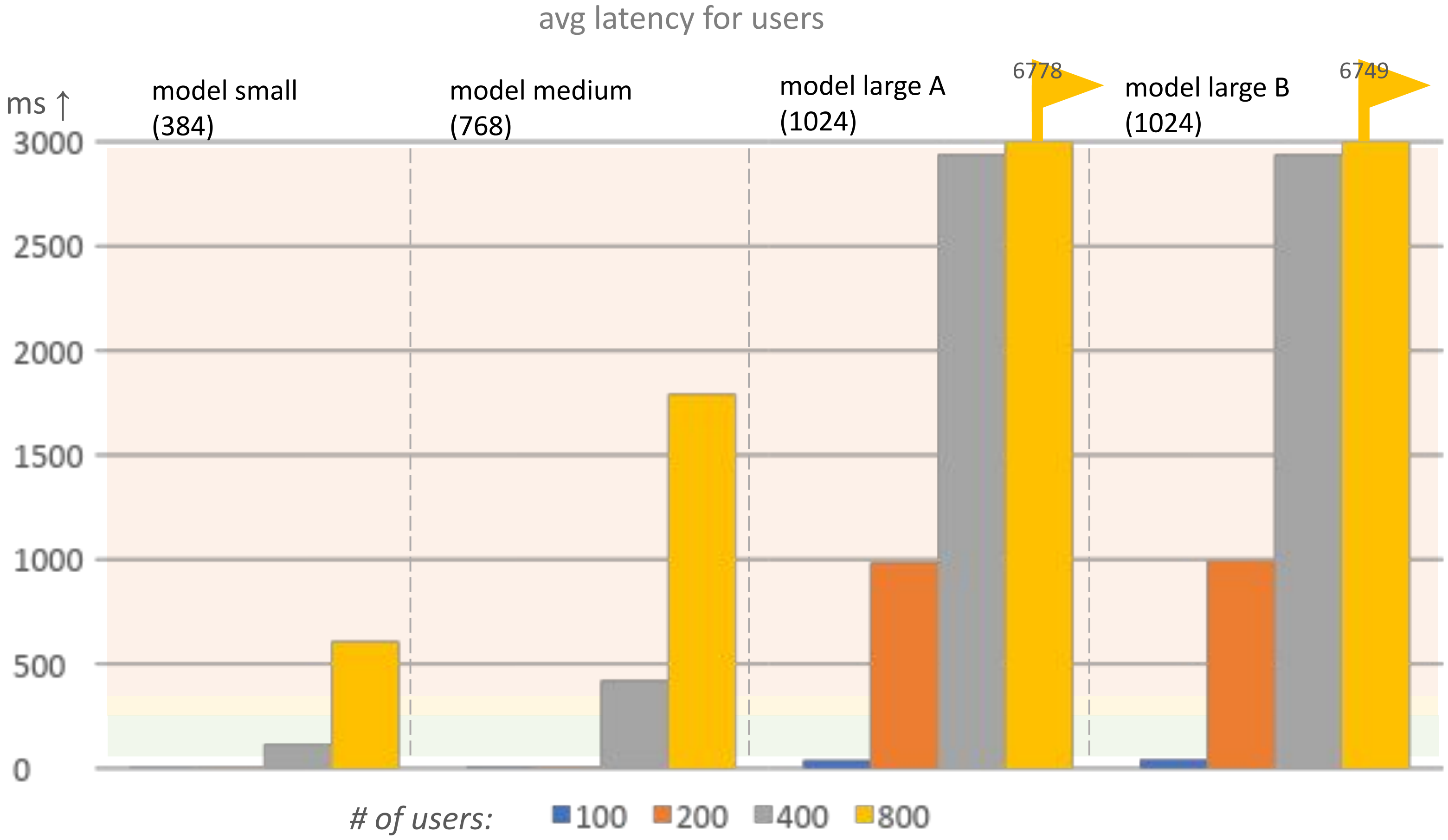


Lessons learned: *text length has a linear relationship with embedding inference latencies*

Simultaneous users on embedding inference latency

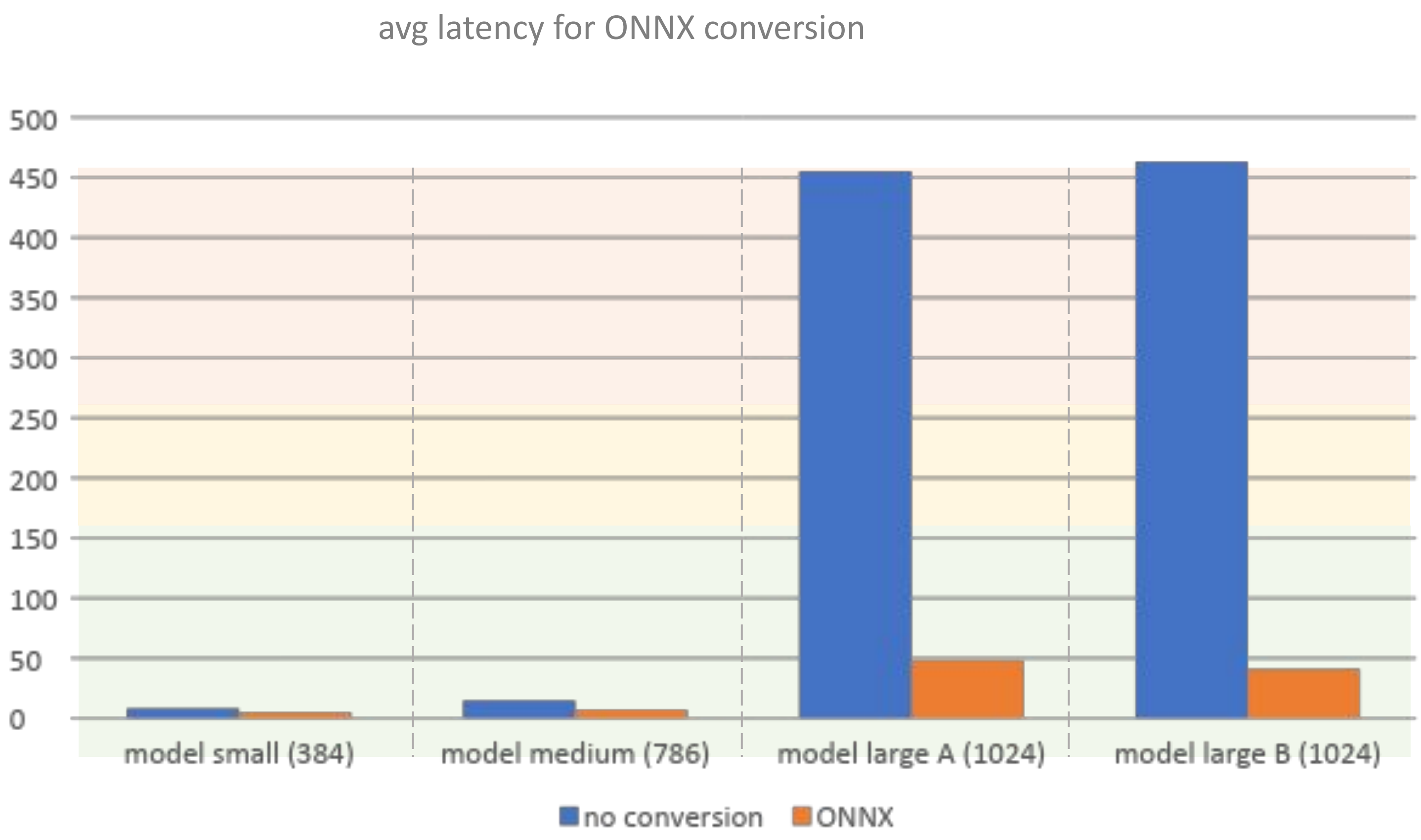
Wolters Kluwer





Lessons learned: # of threads has an exponential relationship with embedding inference latencies

ONNX conversion on embedding inference latency

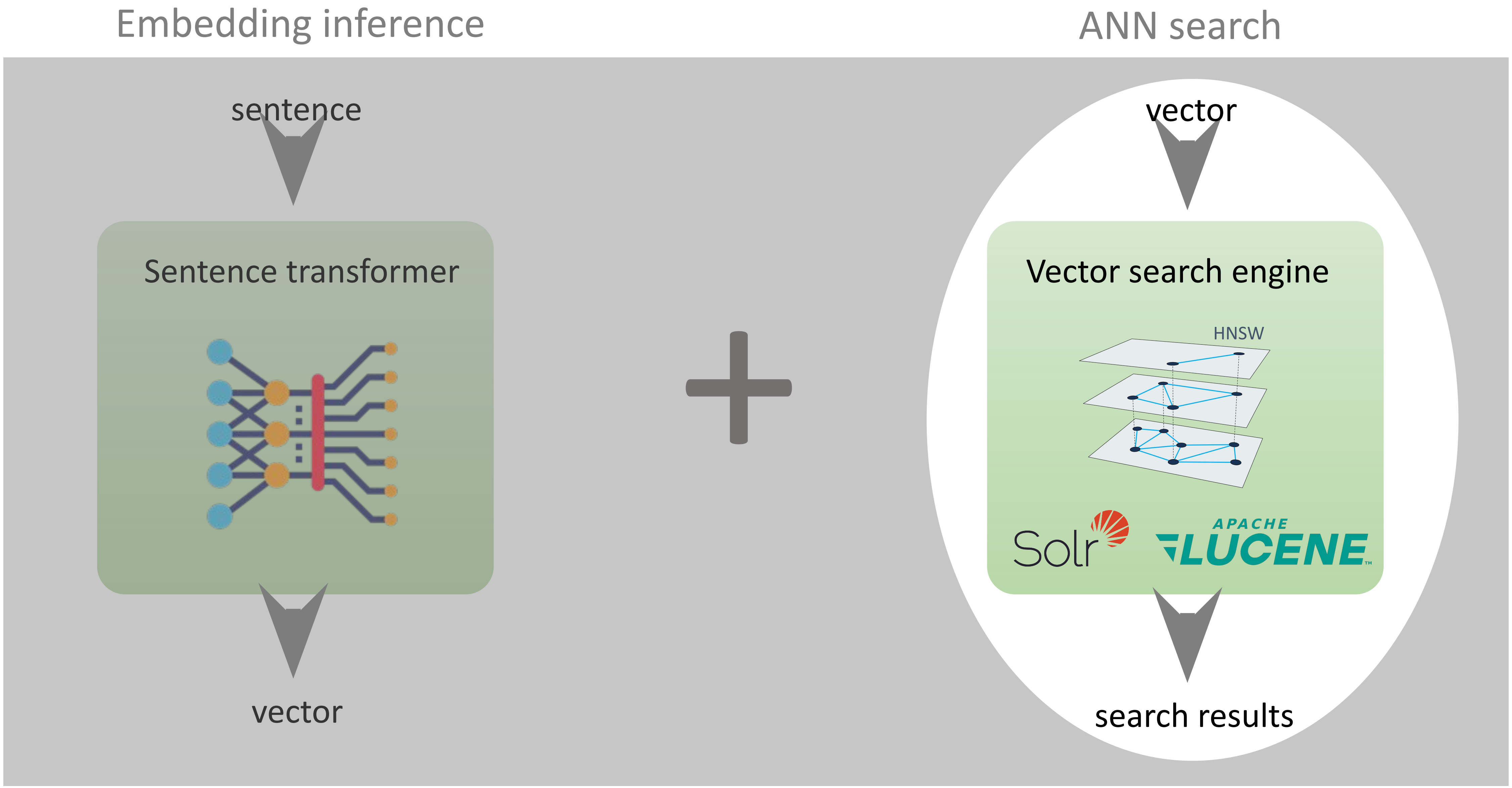


💡 *Lessons learned: ONNX conversion is a great performance booster*

Still in progress....

- Graph optimizations
  - Constant Folding
  - Redundant node eliminations
- Model quantization
  - Float32 to Int8
  - Binary quantization
- Model conversion to TensorRT
- Scaling out the embedding service

Query latency =



The defaults for performance testing ANN search

Document set	
# of documents	**2.600.000**
Avg doc size (w/o vectors)	**6,8 kb**
Vectorized text	**English phrases**

Load test settings	
# of threads	**100**
Pause between transactions	**1000 ms**
Duration per test	**15 min**
Vector of every query is unique	**Yes**
k	**50**
Embedding inference included	**No**

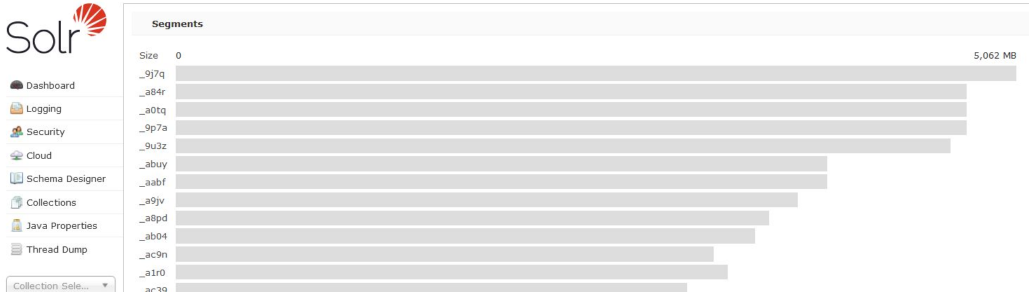
Sentence transformer models	
Small (384 dimensions)	**tavakolih/all-MiniLM-L6-v2-pubmed-full**
Medium (768 dimensions)	**pritamdeka/S-PubMedBert-MS-MARCO**
Large A (1024 dimensions)	**E5-large-v2**
Large B (1024 dimensions)	**thenlper/gte-large**

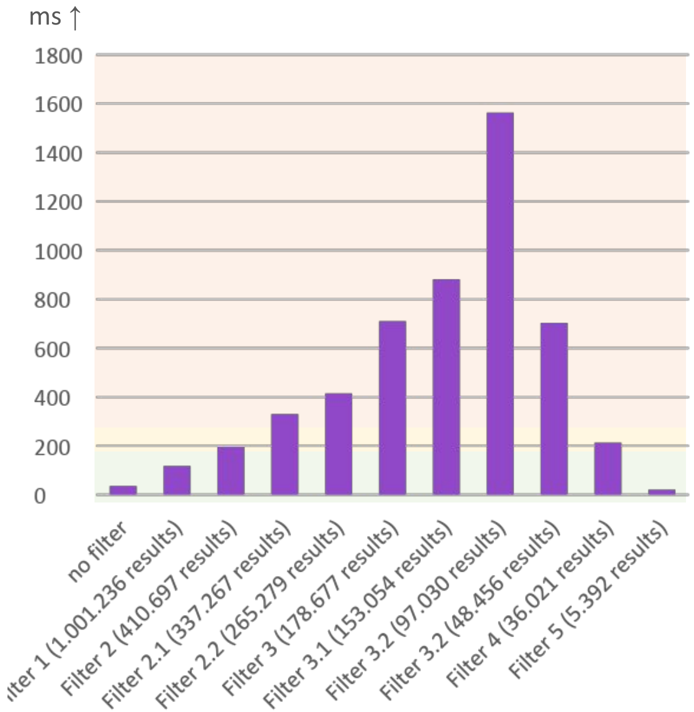
System under test	
EC2 type	**r5.4xlarge**
CPU cores	**16**
Total Memory	**128 Gb**
Memory reserved for JVM	**32 Gb**
Solr / Lucene version	**9.3.0 / 9.7.0**
Shards	**1**
Replicas per shard	**1**
Segments per collection	**1** (fully optimized)
Warmed up memory	**yes**

Field(type)	
indexed	**true**
stored	**false**
Class	**solr.DenseVectorField**
similarityFunction	**euclidean**
vectorEncoding	**FLOAT32**
hnswMaxConnections	**16**
hnswBeamWidth	**100**

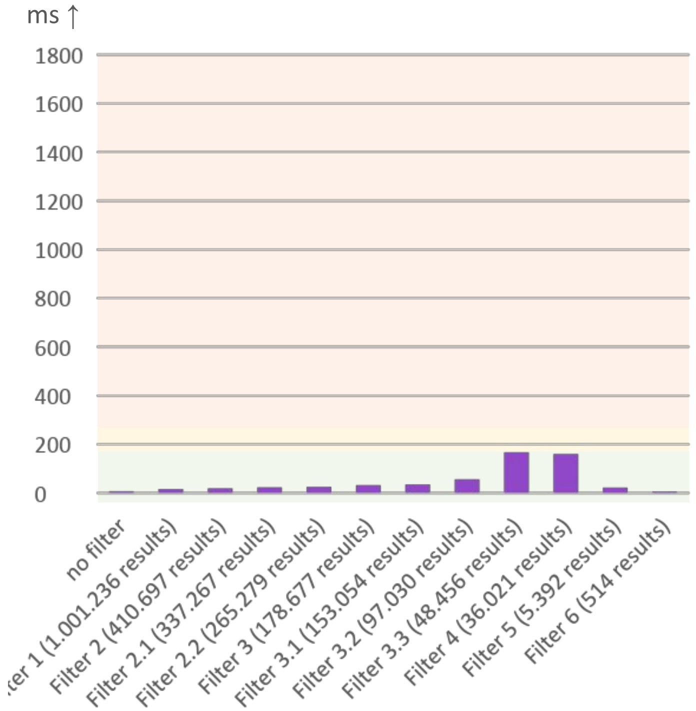
Get you s… together!

Not optimized





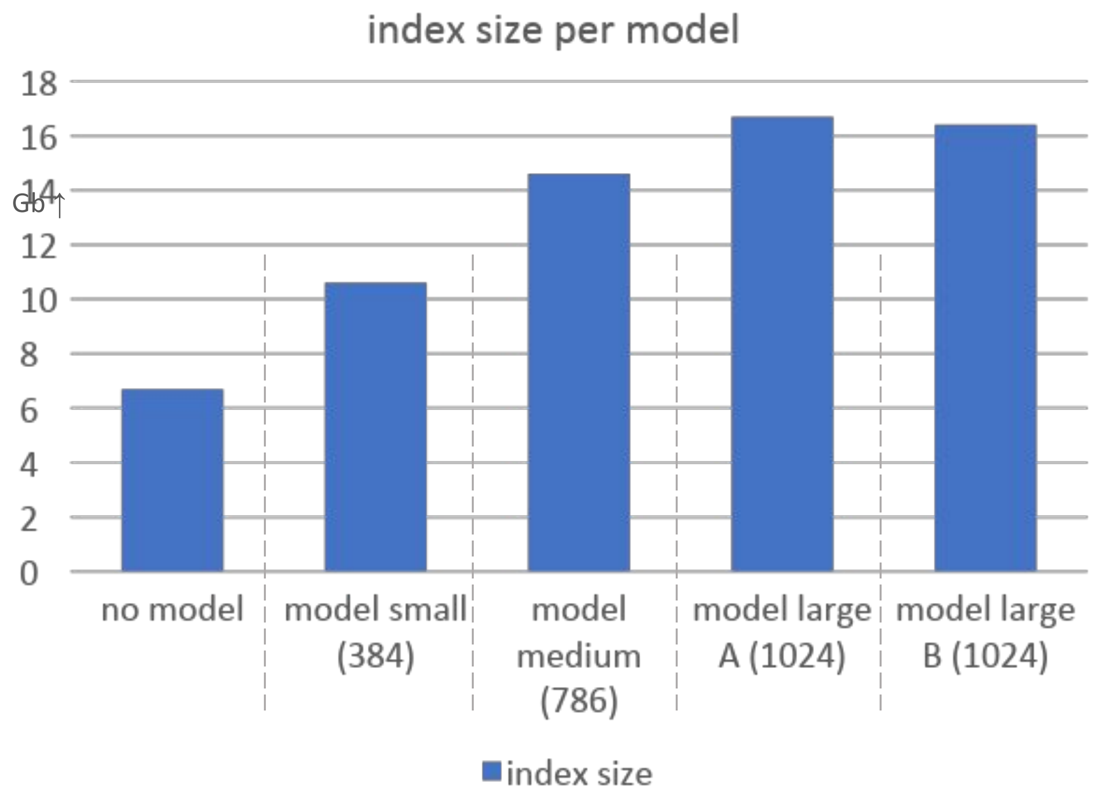
Fully optimized

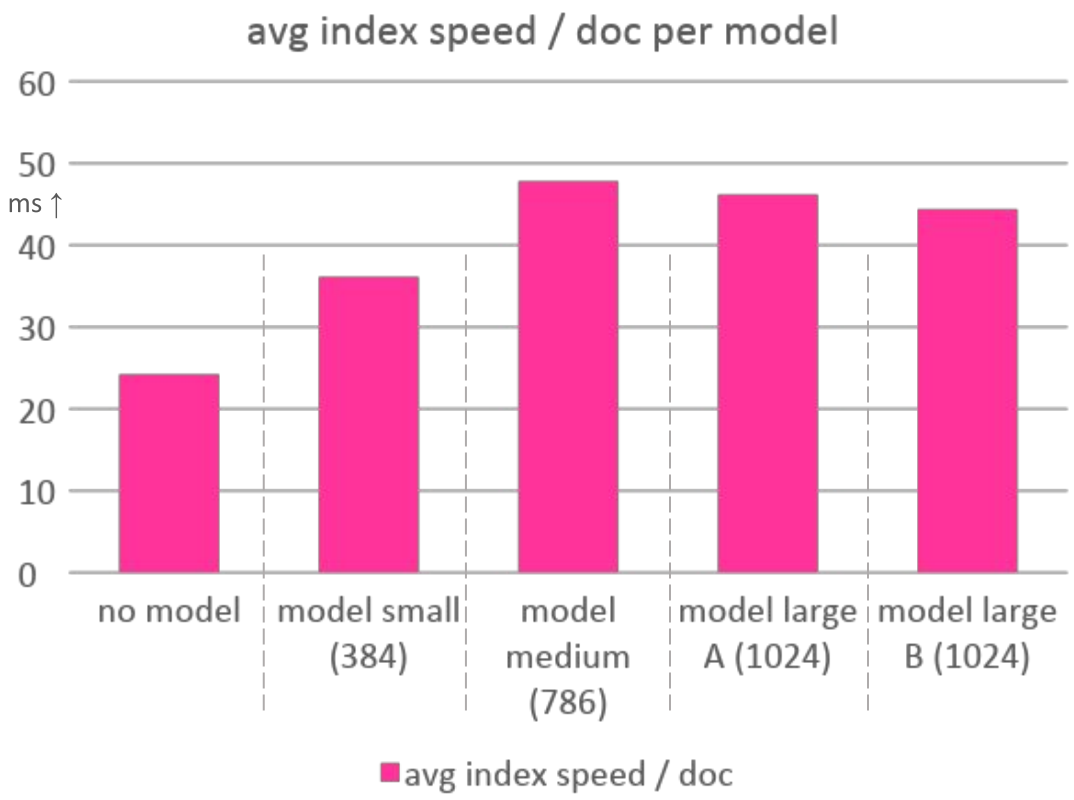


Lessons learned: *avoid a segmented index for vector search*

Vector length on index metrics

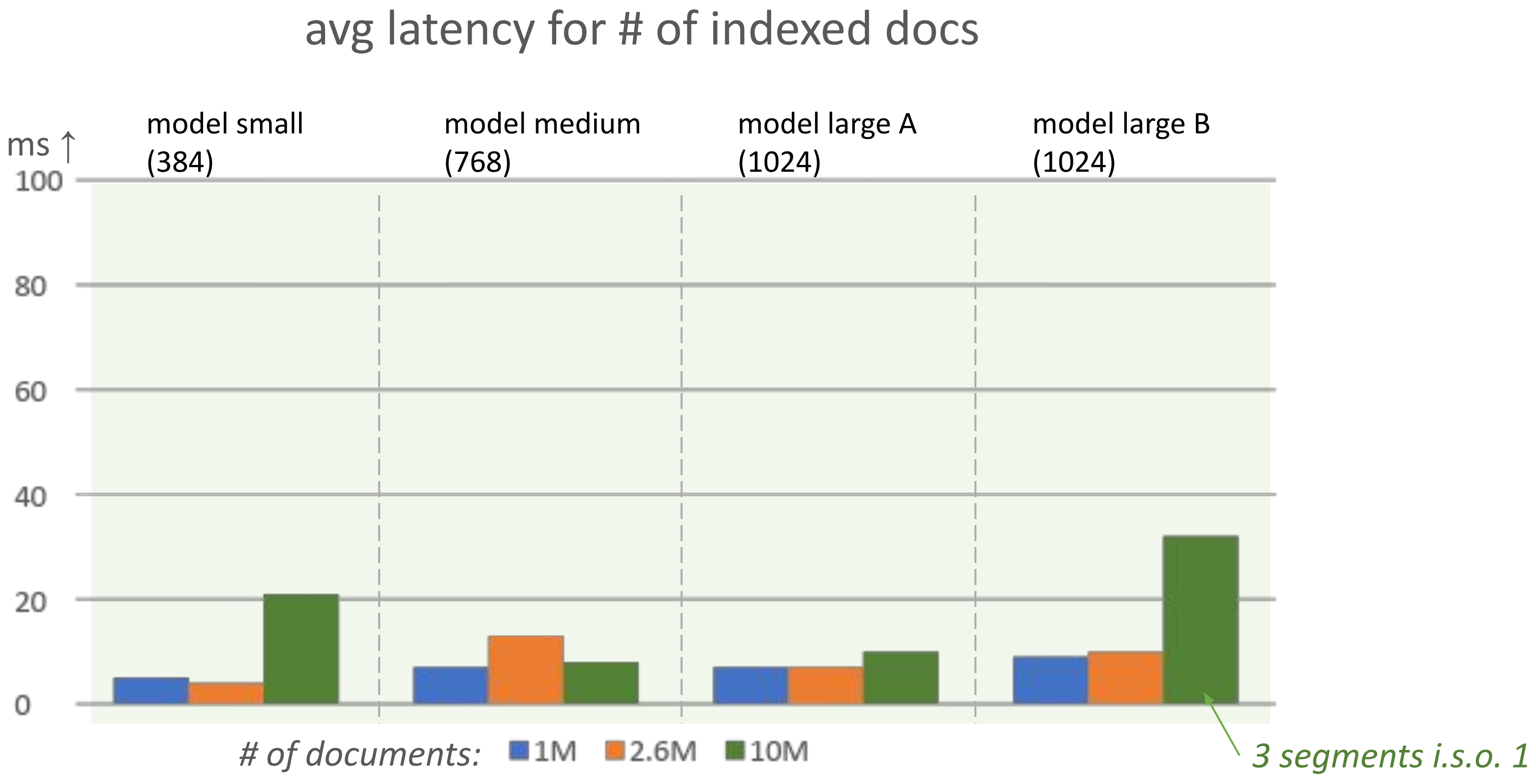
Indexing 2.6M documents with vectors and other metadata





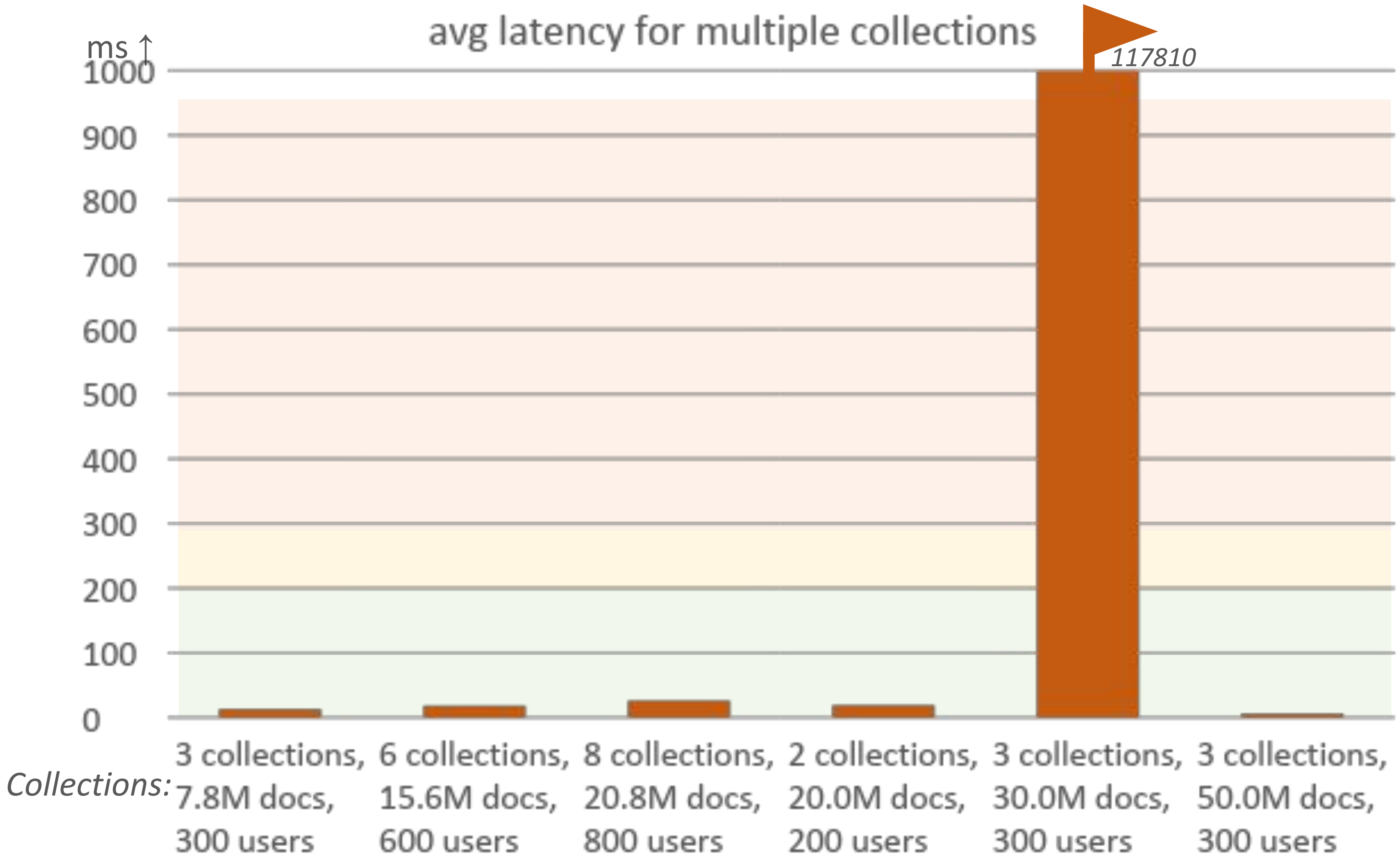
Lessons learned: vectors as metadata have a big impact on index size and index speed

Vector length, # of documents on query latency



Lessons learned: *vector search could perform well for large content sets*

Simultaneous collections on query latency



Lessons learned: not a problem to query multiple collections under high load… as long as they all fit into memory.

k

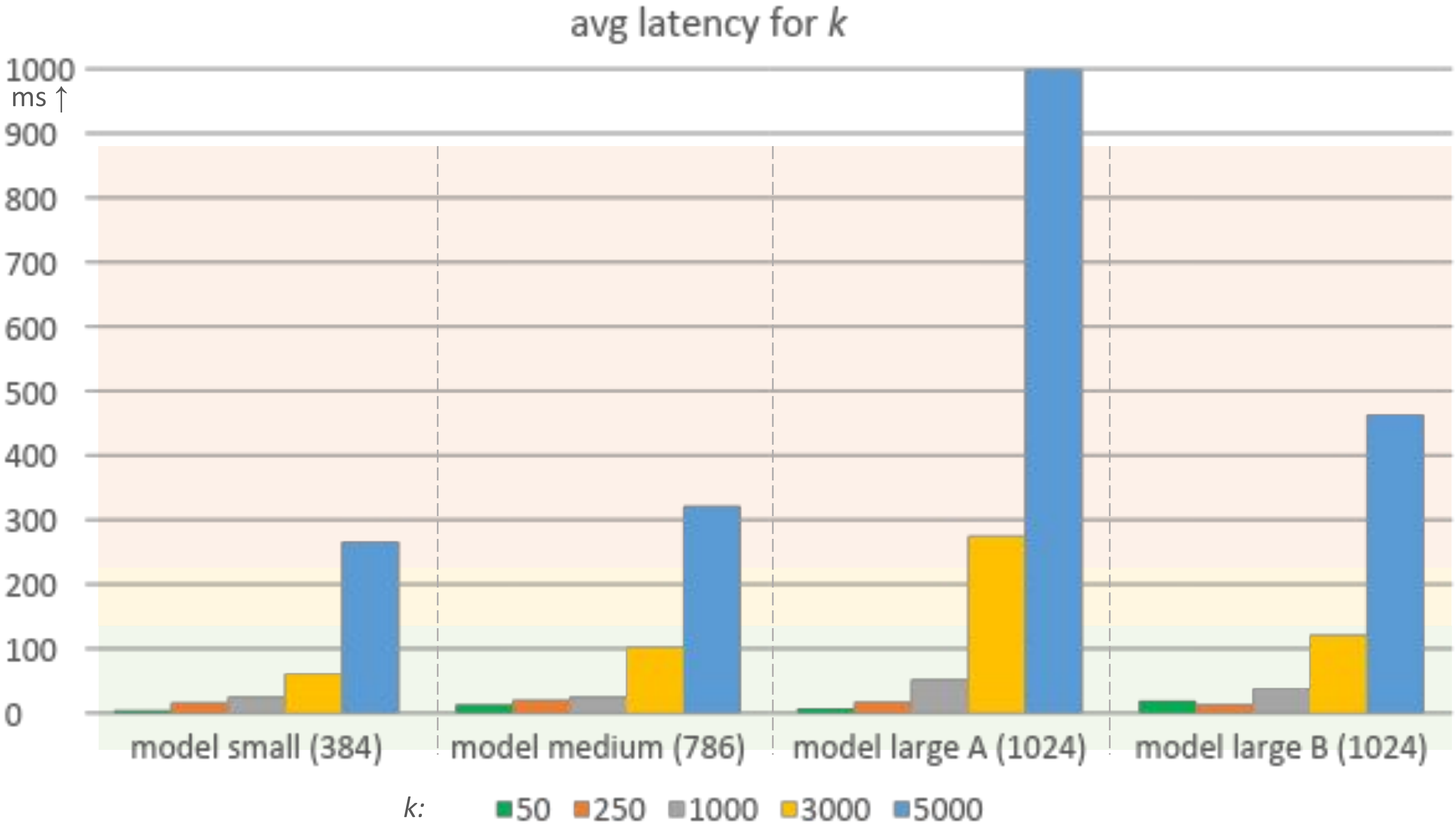
k = the number of approximate nearest neighbours to return

q={!knn f=vectorfield **topK=50**}[-0.32371908, -0.49656674, …]&rows=10

k not only defines the (maximum) number of results, but also impacts relevancy!
The higher *k*, the more likely that the nearest neighbours are in the top results.

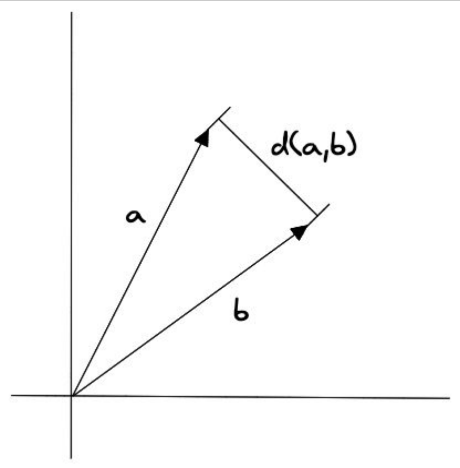
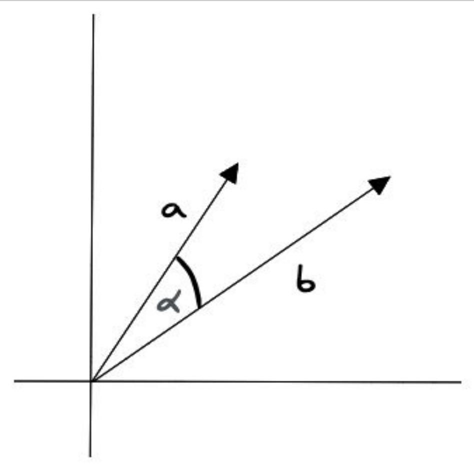
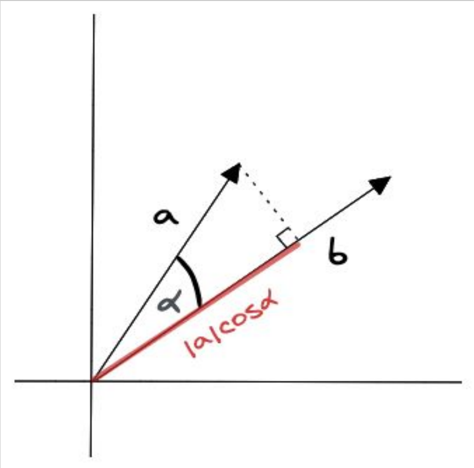
Expectation: higher k means higher avg latency

K on query latency



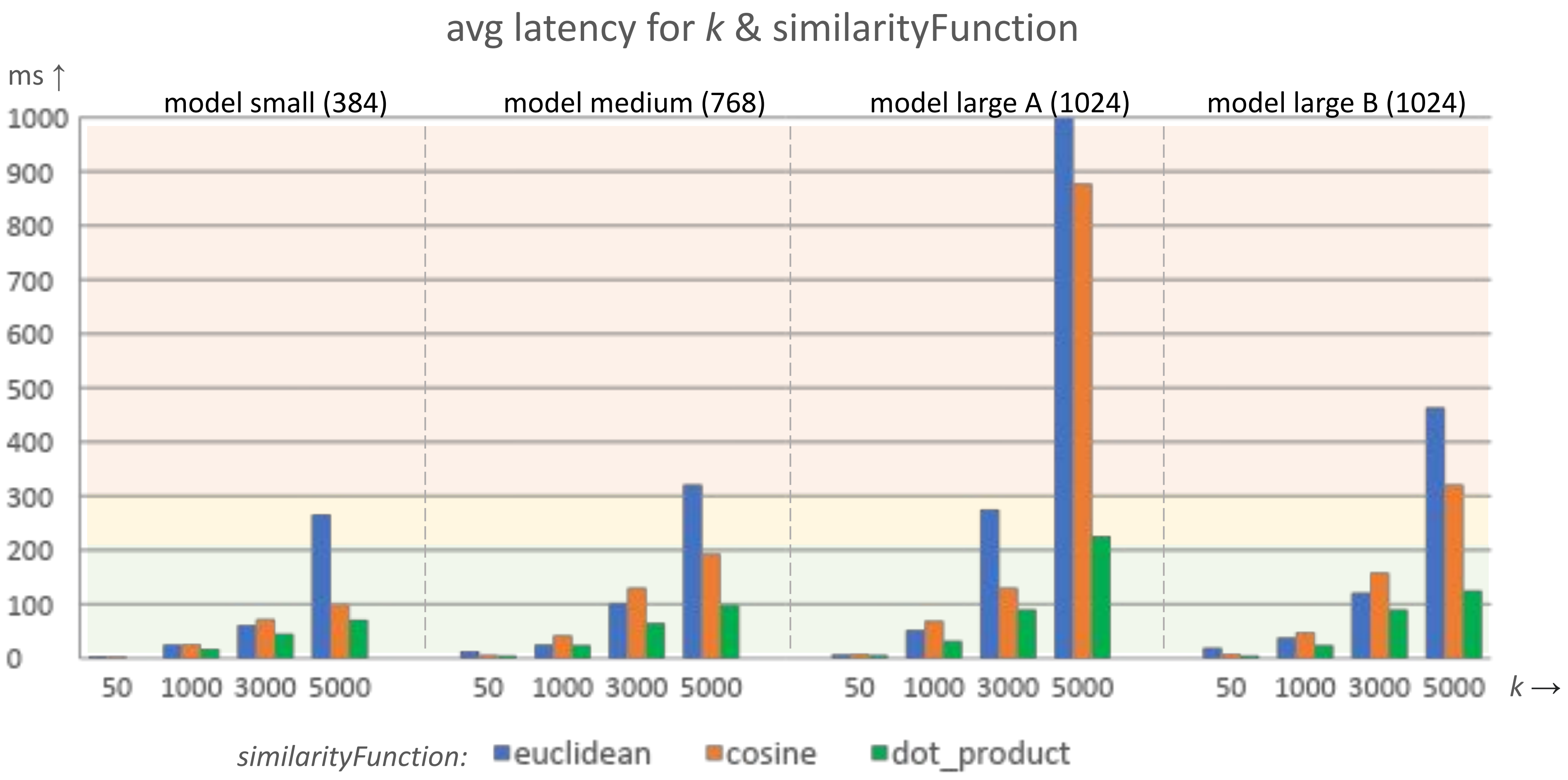
Lessons learned: *don't make k higher than needed for acceptable relevance.*

SimilarityFunction

	euclidean	**cosine**	**dot_product**
Measures	distance	angle	projection
Notation	||a - b||	(a * b) / (|a| * |b|)	(a * b)
			

Expectation: dot_product is fastest, then euclidean, and cosine the slowest

SimilarityFunction on query latency



Lessons learned: dot_product is the best performing.
Cosine is slowest for low k values, euclidean is slowest for high k values

VectorEncoding

FLOAT32

-0.21449316, -0.7045389, -0.67822456, -0.29824427, -0.23921804, -0.0809364, -0.5233864, 0.7305913, 0.09852978, 0.50574046, 0.3282113, 0.2059273, -0.031108191, 0.035400968, -0.22698092, -0.32095635, 0.21415716, 0.09343966, 0.08683256, 0.19313174, 0.63785744, 0.298874, -0.28171337, 0.18531613, -0.6641149, 0.19386779, -0.31794095, 0.4402138, 0.3466606, -0.2858599, -0.22758806, 0.5094929, 0.046053726, 0.75082016, -0.07399338, -0.2844224, 0.40751144, -0.20799315, 0.14701228, -0.08118942, 0.50932866, -0.28915992, 0.19562256, 0.21961893, 0.20695217, 0.10814471, 0.2393254, -0.8819913, 0.16113488, -0.5311082, -0.1953351, -0.13989331, 0.10564095, 0.40680933, 0.042414997, 0.07088098, -0.020308852, -0.0022723621, -0.043205384, 0.12104646, 0.08444527, 0.64572316, 0.08393095, -0.19806932, -0.04344313, 0.4255652, -0.42429543, -0.41475034, -0.36487082, -0.09986199, -0.13209495, 0.06342443, 0.027432332, -0.27986363, 0.3010312, -0.103268646, 0.37407556, 0.11932395, -0.58556277, 0.059918627, 0.4299334, 0.4327116, 0.101633854, -0.05603434, -0.36993638, 0.13854954, 0.34047017, 0.20950834, 0.34301245, 0.048450783, 0.50535196, 0.044725284, -0.17060715, -0.37688974, 0.20206492, 0.04468606, 0.14183544, -0.2736002, -0.0658742

BYTE

-3, -4, -7, 2, -7, 2, -10, 5, 8, 4, 2, -8, -12, 7, -9, 8, -20, -1, 1, -7, -4, 7, 2, 1, -9, 0, 2, -3, -1, -3, -5, 11, 4, 8, 0, 4, -2, 4, 17, 1, 0, -1, -1, 6, 2, 4, -4, -6, 1, -2, -2, -2, -15, 5, 1, 1, -7, -6, 6, -2, 8, -3, -14, 9, 20, -5, 10, 0, 3, 6, 2, 1, 0, 10, 8, 5, -6, 5, 5, -4, -1, 8, 4, 8, 7, -6, 4, 3, -11, 3, -7, -11, 3, -2, -8, 6, -2, -2, -2, -15, 5

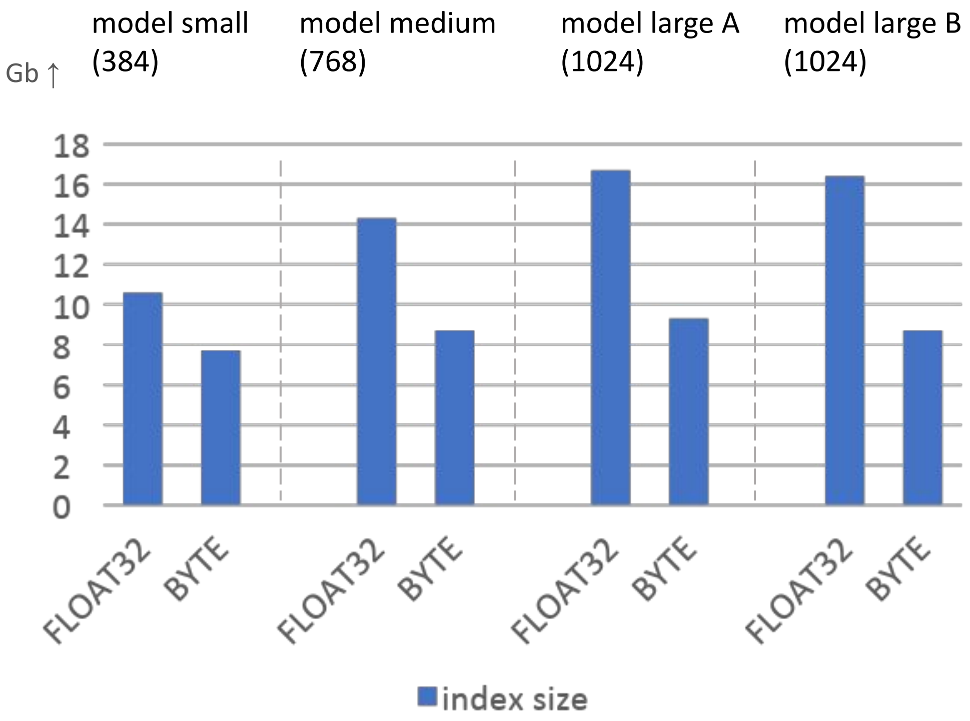
Scalar Quantization:

- ◉ Maintaining the angle
- ○ Maintainting the relative distance

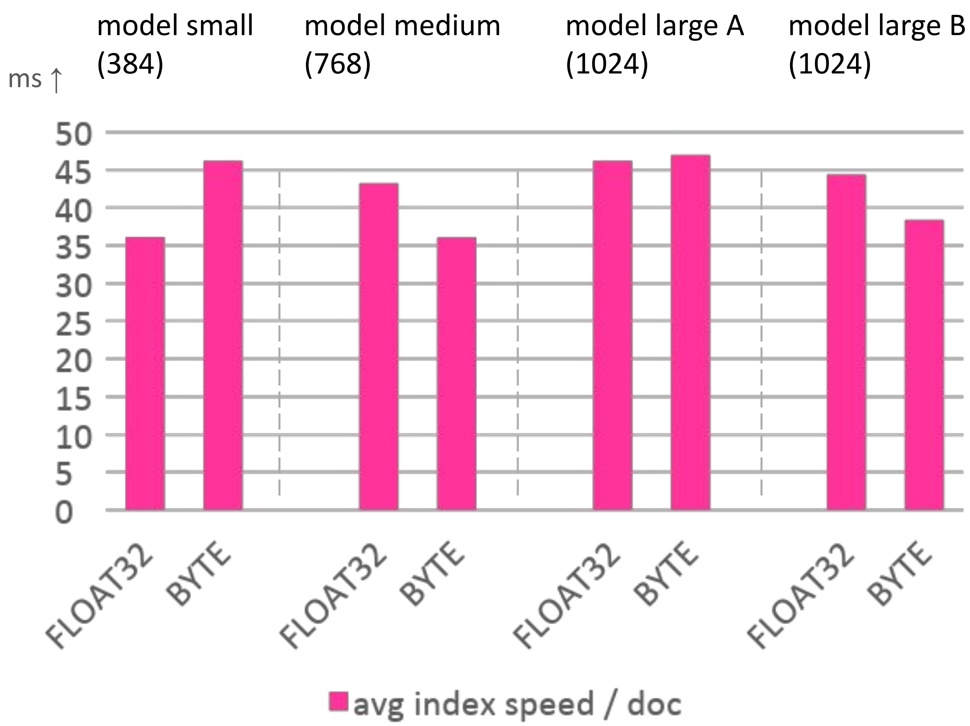
Expectation: both index size and avg query latency are lower with BYTE compared to FLOAT32

VectorEncoding on index metrics

index size for vectorEncoding

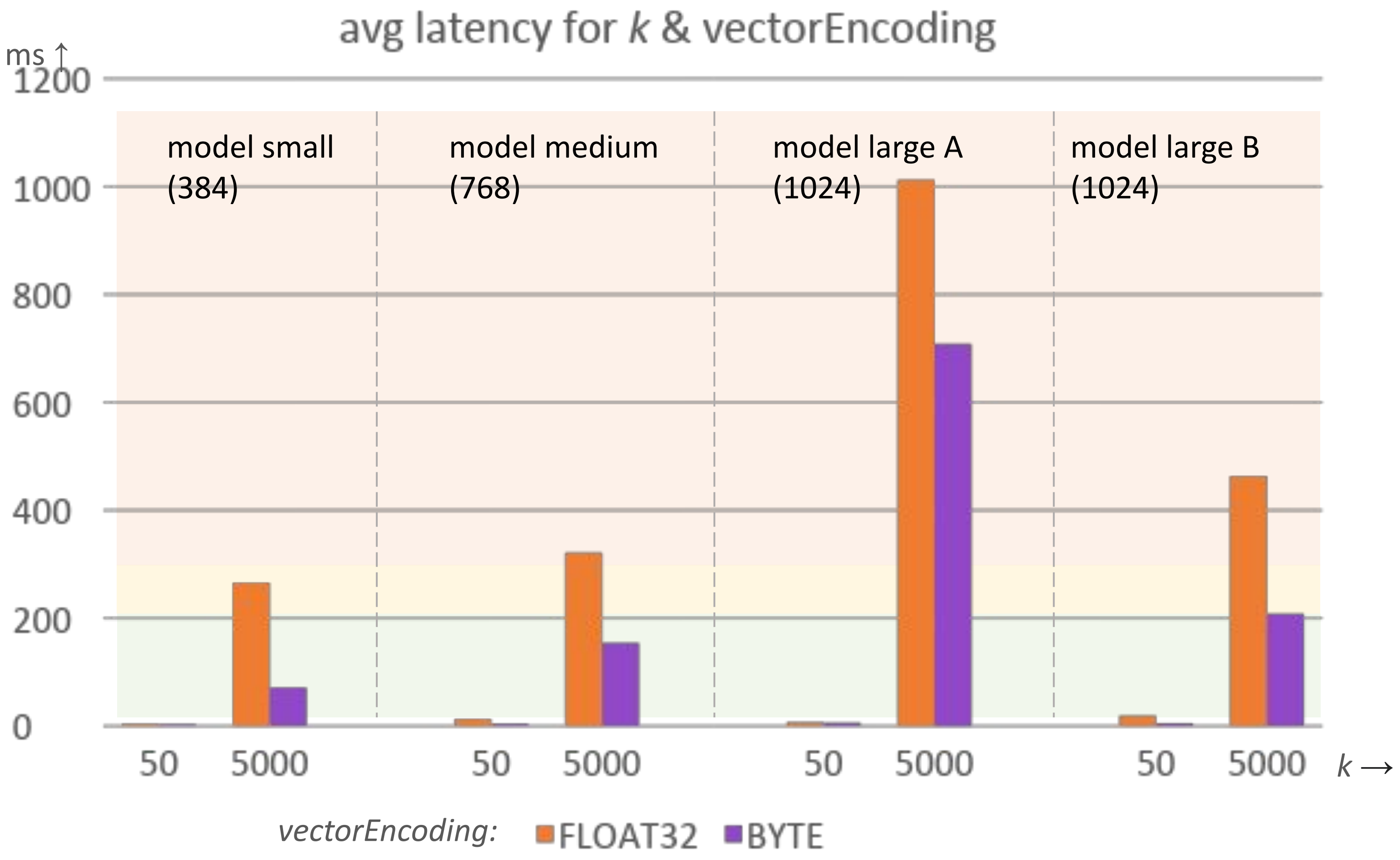


avg index speed / doc for vectorEncoding



Lessons learned: encoding vector values as BYTE i.s.o. FLOAT32 could greatly reduce the index size, especially for large vectors

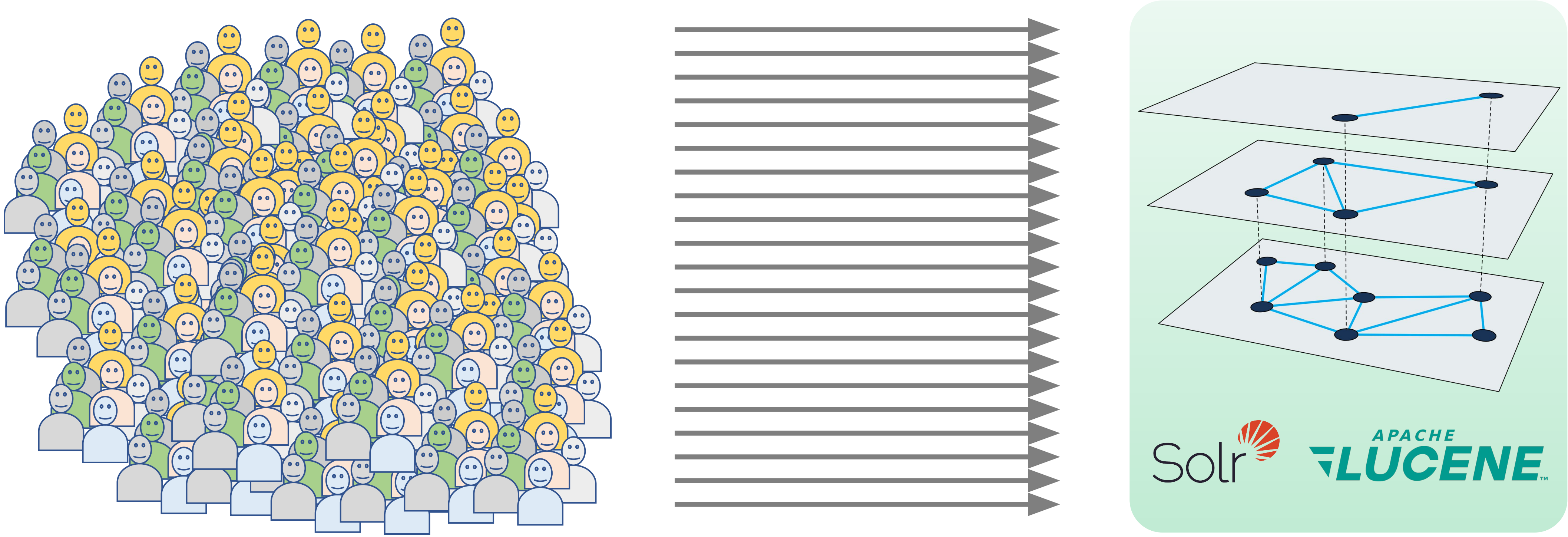
vectorEncoding on query latency



Lessons learned: encoding vector values as BYTE i.s.o. FLOAT32 is an excellent way to lower latencies especially for high k values

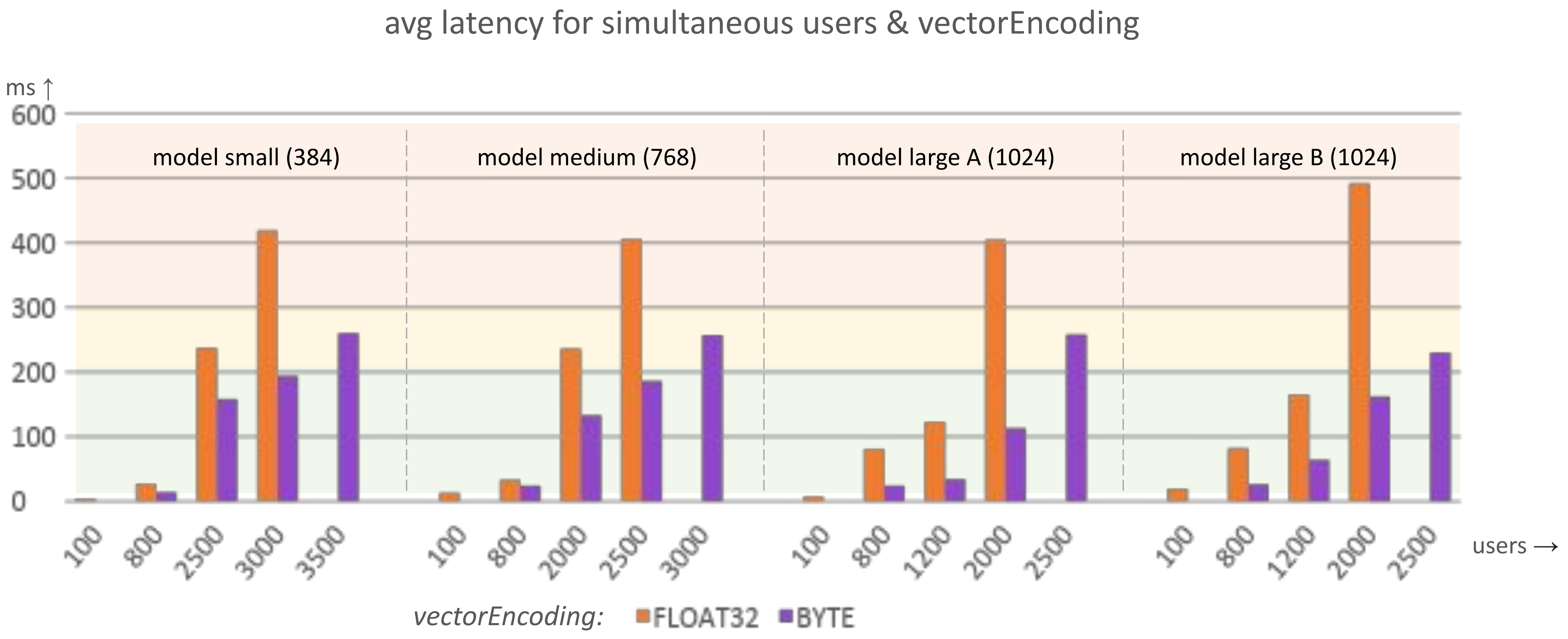
Wolters Kluwer

Stress test



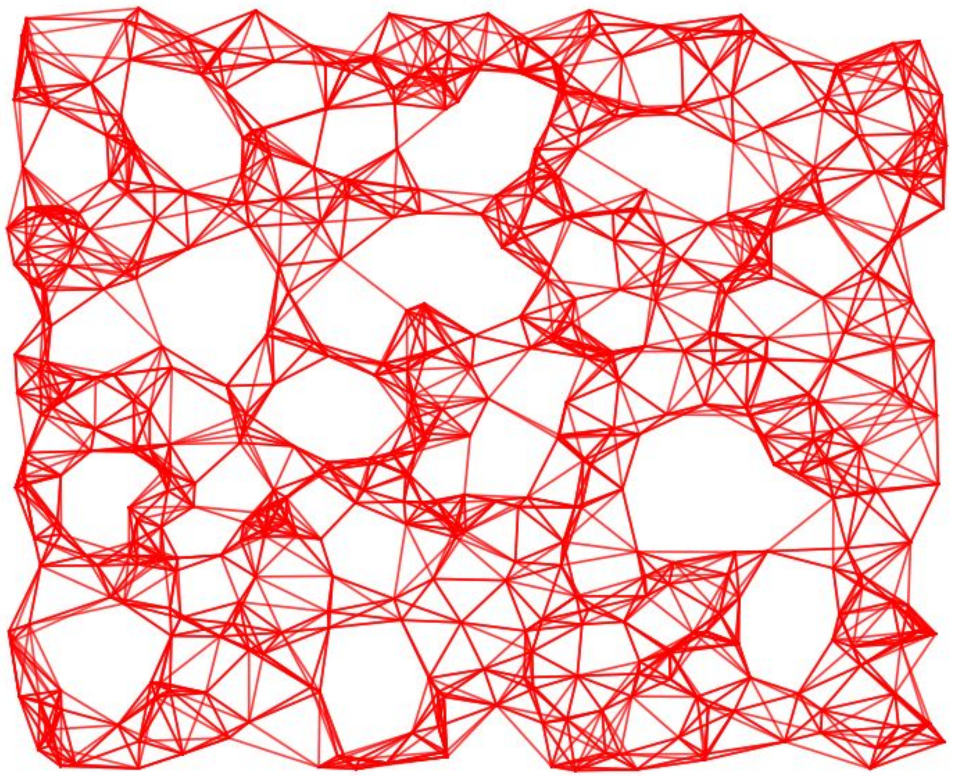
Expectations: *more simultaneous users means higher avg latency*

Simultaneous user on query latency



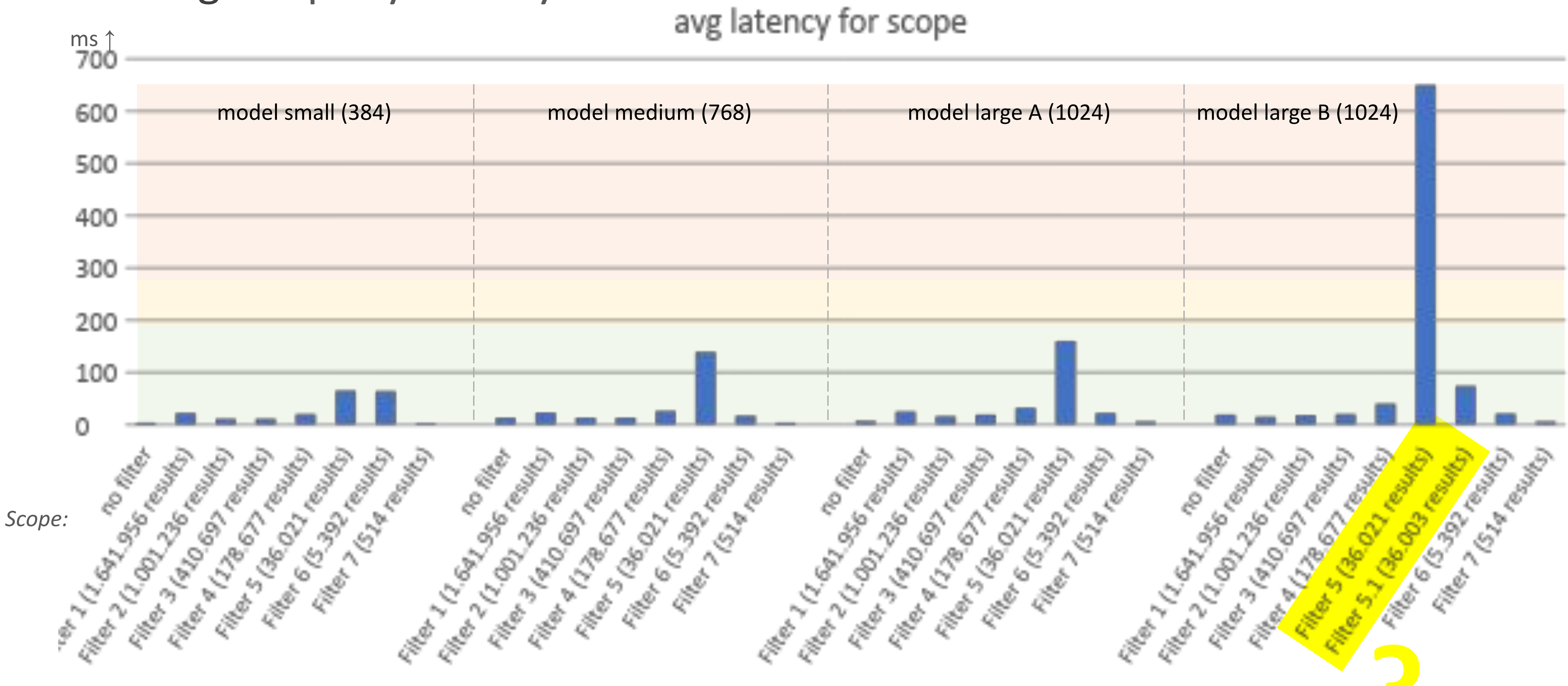
Lessons learned: encoding vector values as BYTE i.s.o. FLOAT32 is an excellent way to handle bigger stress (up to thousands of QPS)

Filtering



Expectation: filtered searches are slower than unfiltered.

Filtering on query latency

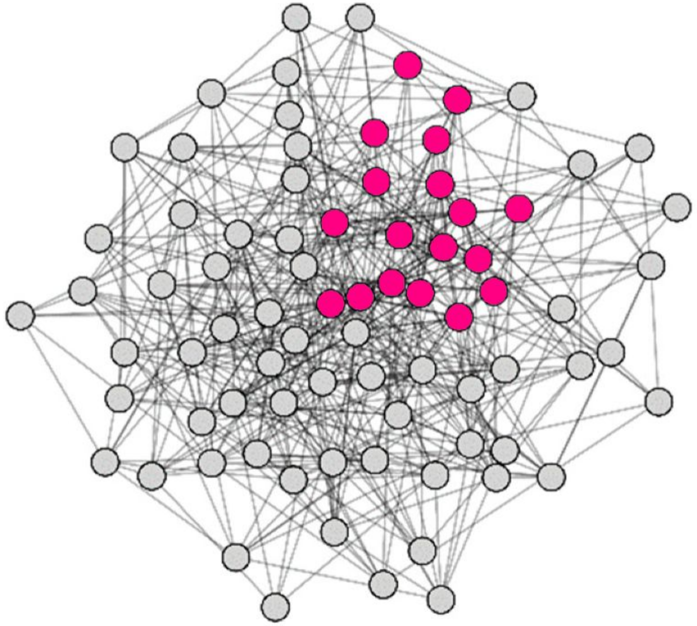


💡 *Lessons learned: filters impact performance, sometimes dramatically*

Wolters Kluwer

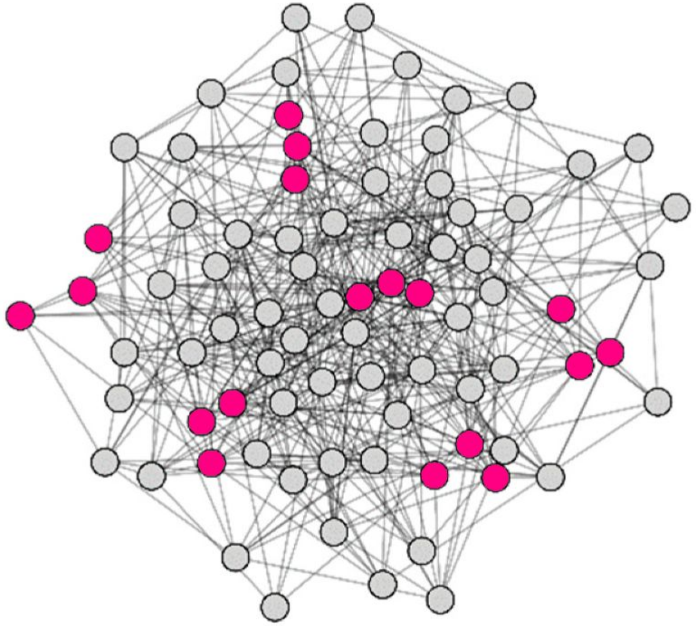
Filter 5 (36.021 results)
Avg latency: 650 ms

```
fq=folder_s:("JTN01_deconstruced")
```



Filter 5.1 (36.003 results)
Avg latency: 74 ms

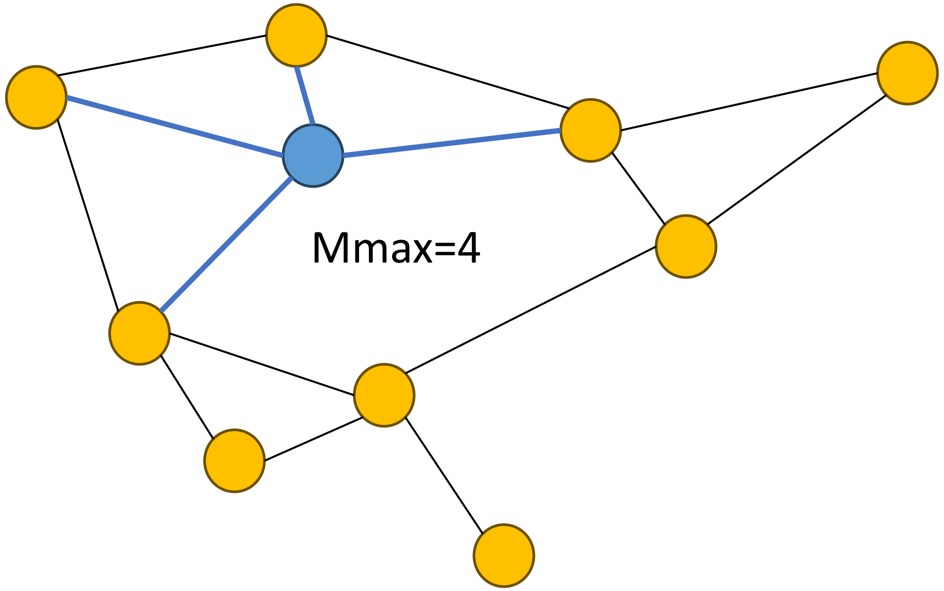
```
fq=docfolder:("x463" OR "x494" OR "x548"
OR "x708" OR "x772" OR "x773" OR "x370"
OR "x424" OR "x541" OR "x926" OR "x272"
OR "x562" OR "x817" OR "x925" OR "x213"
OR "x23" OR "x317" OR "x321" OR "x511" OR
"x55")
```



Lessons learned: the narrowness of a filter drives latency but also the distribution of the filtered candidates

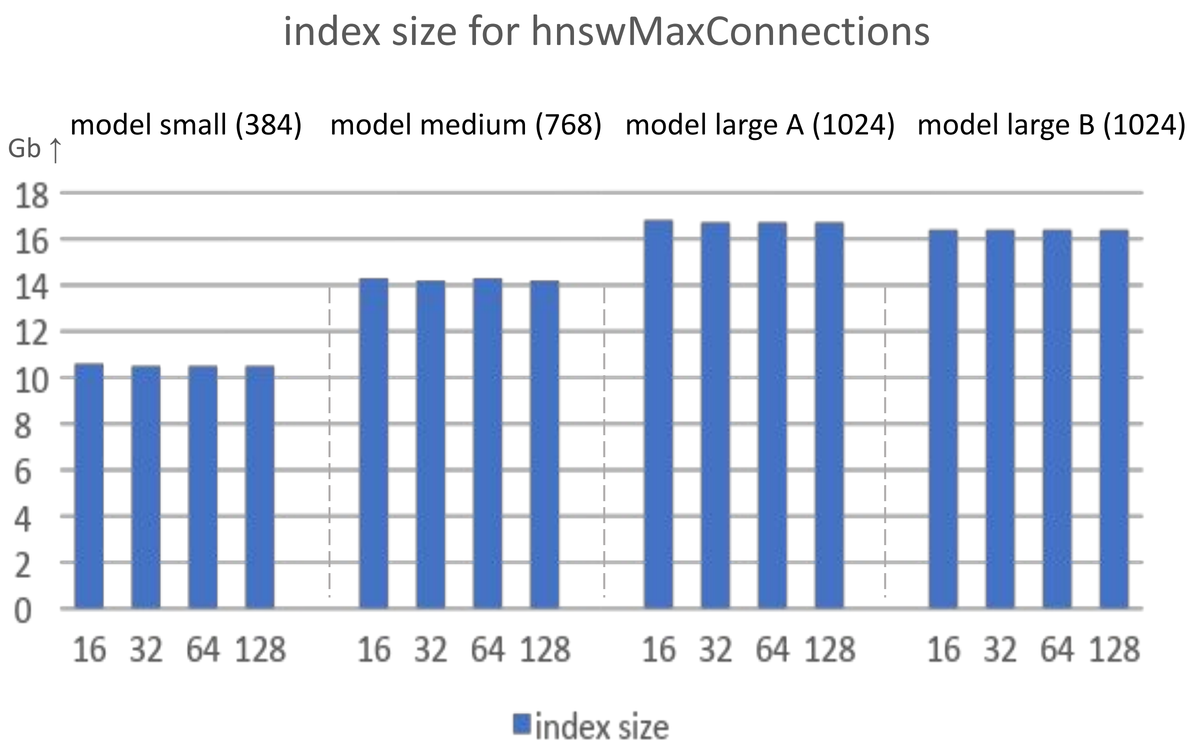
hnswMaxConnections

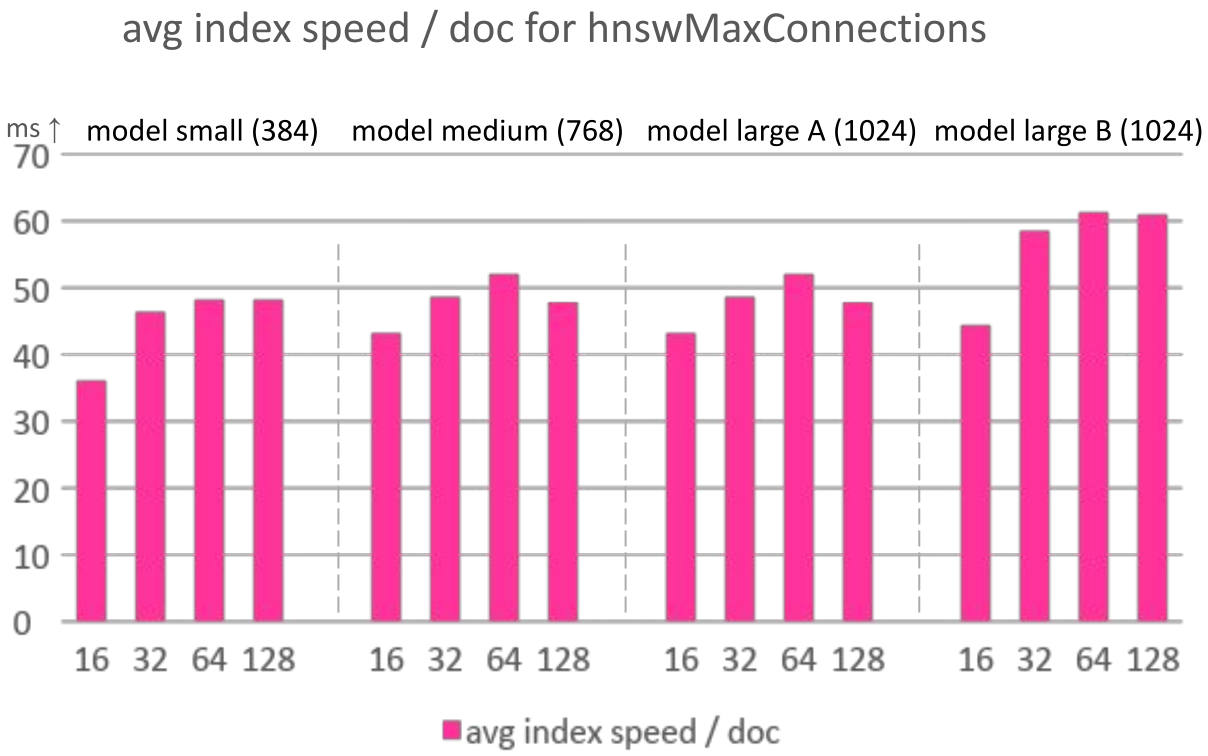
hnswMaxConnections (a.k.a. Mmax or just M) defines the maximum amount of connections each node in the graph could get



Expectations: more connections means slower index times, bigger indexes and lower avg query latency

hnswMaxConnections on index metrics

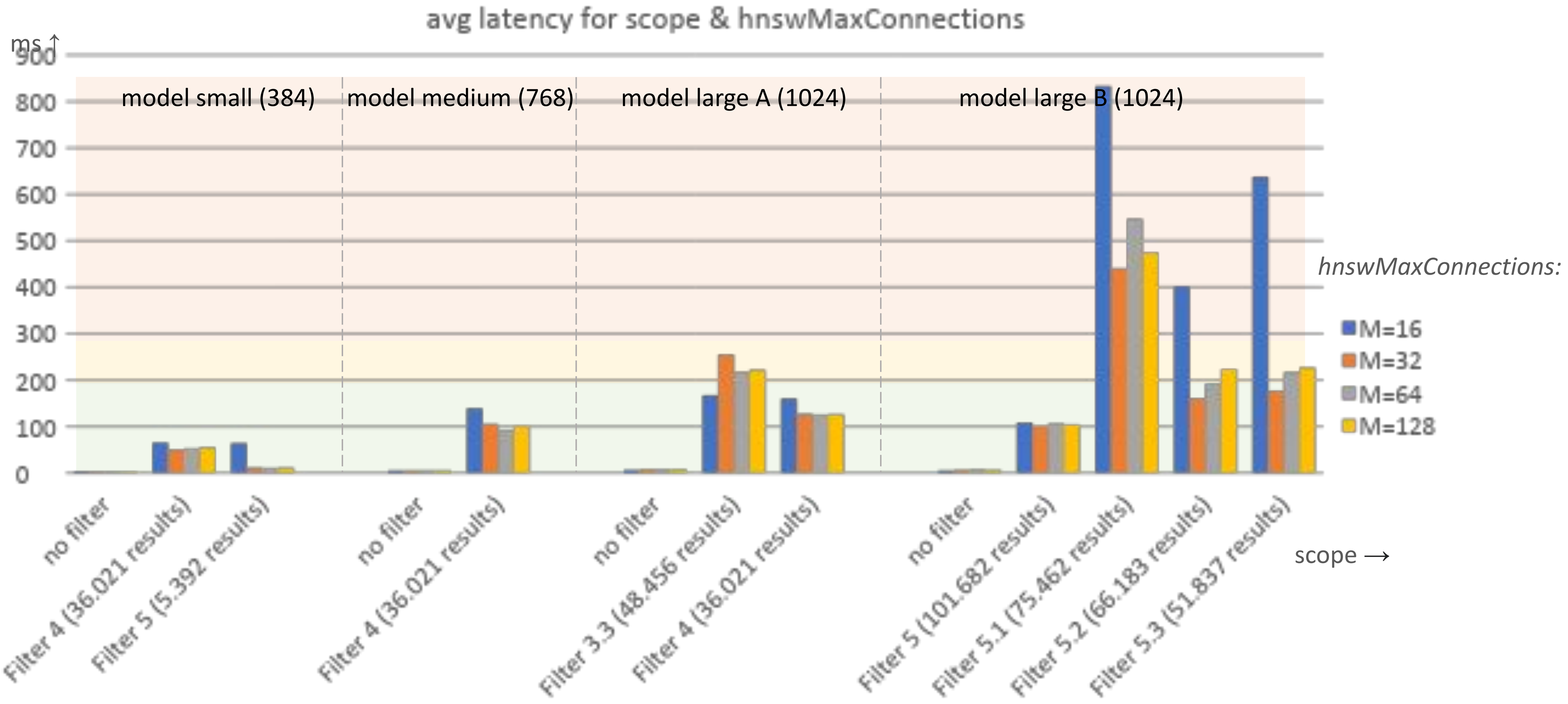




No difference in index size??

?

hnswMaxConnections on query latency



Lessons learned: a higher hnswMaxConnections is not an universal performance booster but could help in some filter scenarios

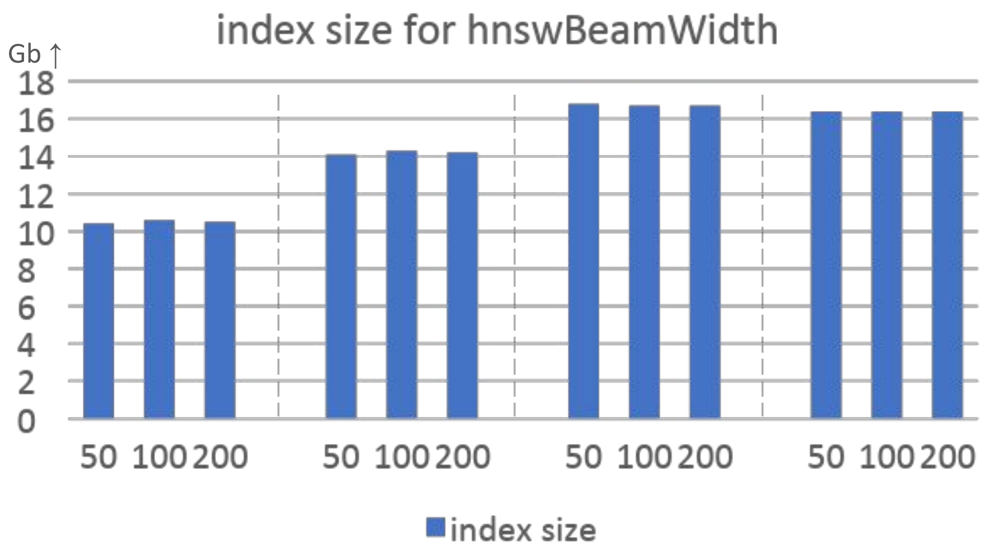
hnswBeamWidth

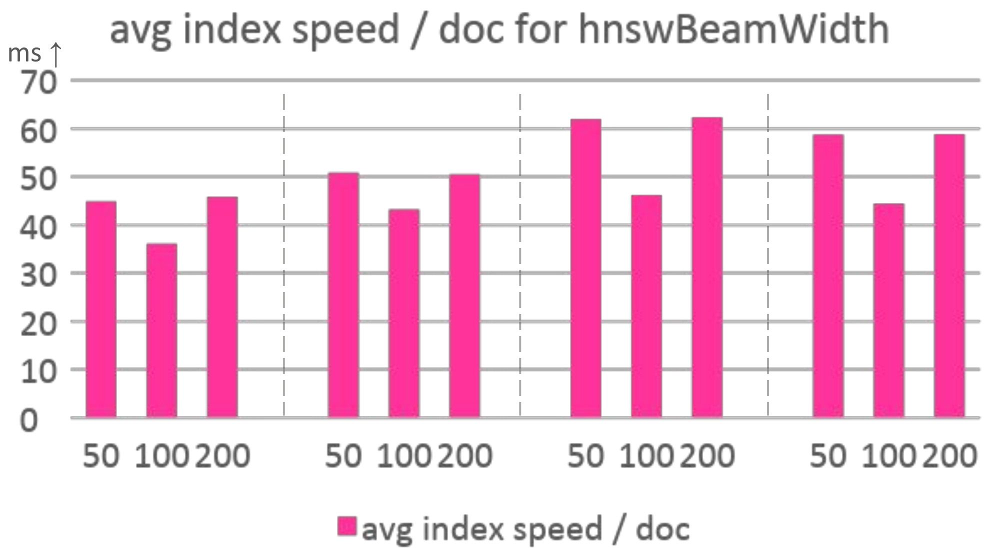
hnswBeamWidth (a.k.a. ef_construction) defines the size of the candidate list used during the index building process

#	Node ID	Distance score
1	985137491	0.2345
2	092475819	0.1586
3	875193457	0.0843
4	183975913	0.0811
5	985159819	0.0770
…		
100	198357914	0.0685

Expectations: a higher hnswBeamWidth slows down the index building process and shouldn't impact avg query latency (but improves relevancy)

hnswBeamWidth on index metrics



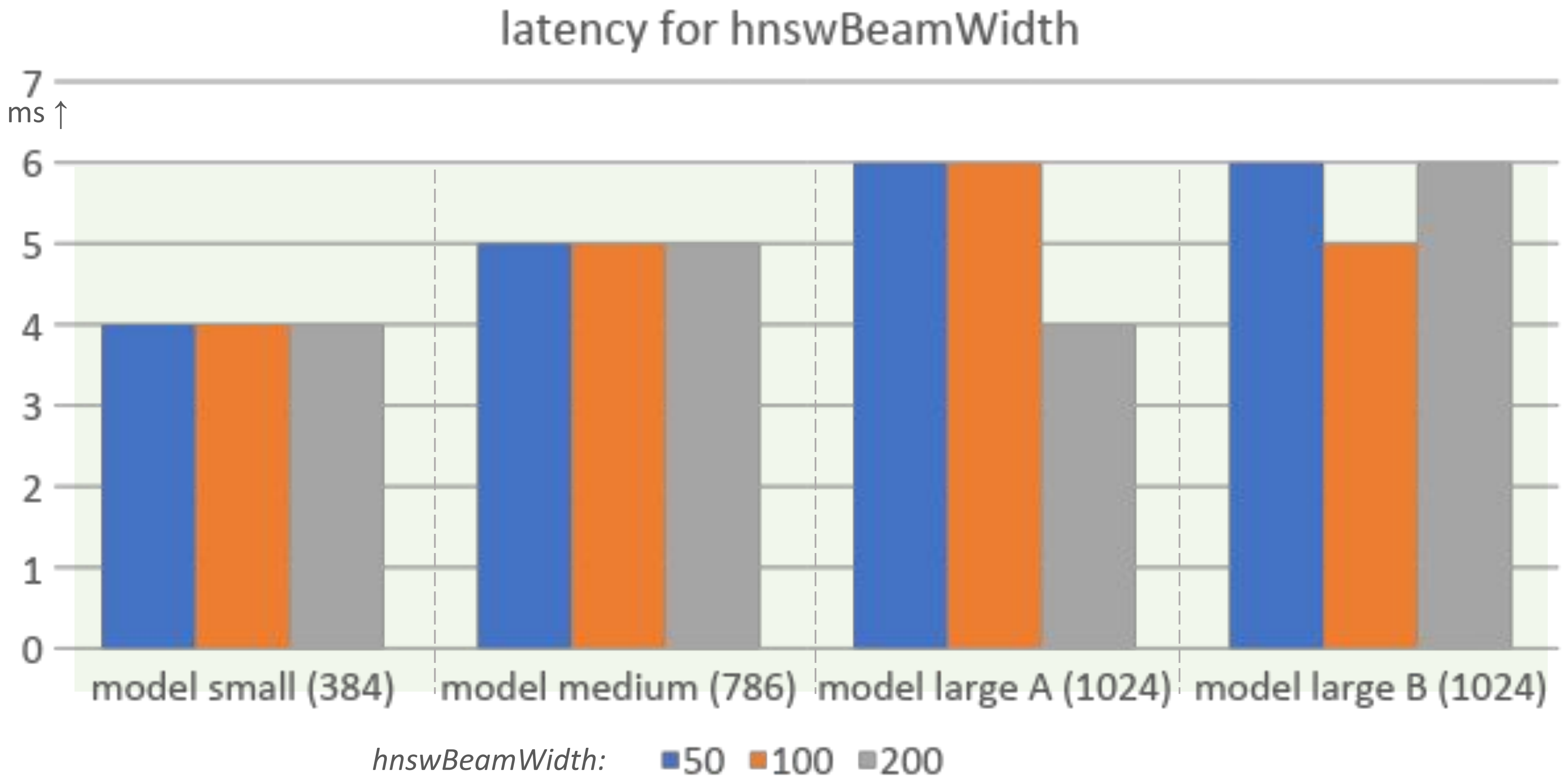


Why does it take longer to index with 50 than with 100?

?

Lessons learned: hnswBeamWidth has no impact on index size but no clear correlation with index speed

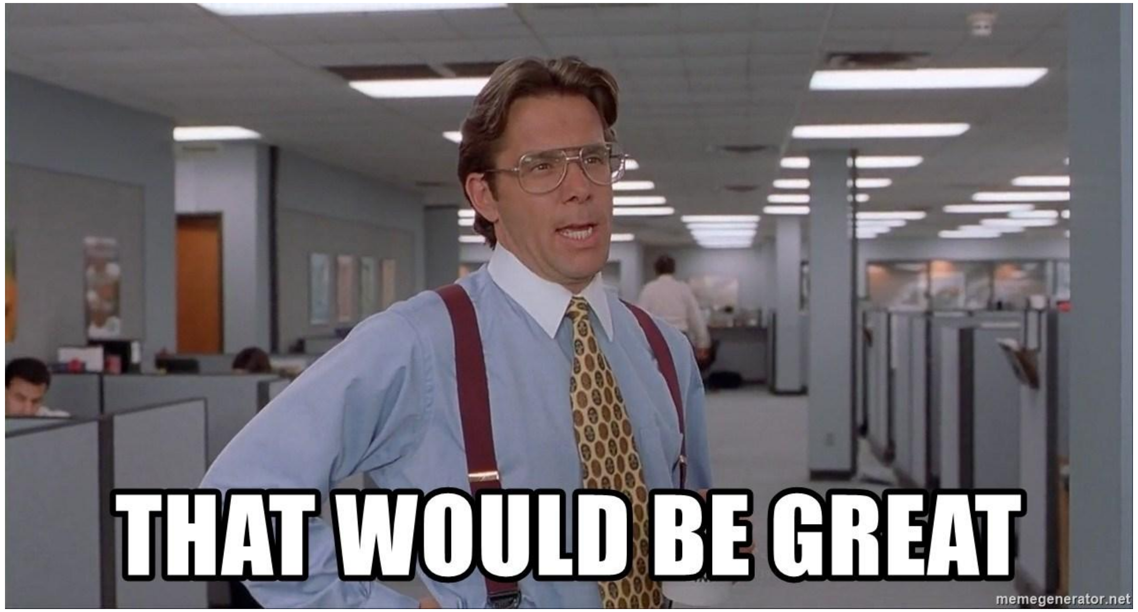
hnswBeamWidth on query latency



Lessons learned: hnswBeamWidth has no impact on latency

Wish list

- No segmented HNSW graph
- Internal embedding inference
- Internal vector quantization
- Friendly hybrid search support
- Multi-valued vector fields



Thank you!

Questions?

Or contact us later:

Mohit Sidana
Search Architect
Wolters Kluwer

Tom Burgmans
Technology Product Owner Search
Wolters Kluwer