

DENSE RETRIEVAL WITH APACHE SOLR NEURAL SEARCH



Alessandro Benedetti – R&D software engineer and director at Sease

Elia Porciani – R&D software engineer at Sease

INTRODUCTION

Lucene has introduced vector indices in late 2020. Our work consists of the integration of dense vector search in Apache Solr. Thanks to this new feature it is now possible to index vectors and perform vector searches.

This opens the doors for several neural search algorithms leveraging language transformers.

New Solr capabilities

- fast indexing of Dense Vectors
- K-Nearest Neighbor search
- Reranking (to be improved)

2020-12 Lucene introduces HNSW

2022-01 Solr implements dense vector search using the Lucene HNSW implementation

2022-01 Lucene makes HNSW hierarchical

2022-02 Lucene adds support for filter queries in HNSW

SOLR IMPLEMENTATION

Overview

The main class introduced in the solr implementation is `DenseVectorFields`. This class inherits from `FloatPointField`.

Index field

Under the hood, the `DenseVectorField` instantiate a `KnnVectorField`. This is responsible of index the vector in the graph and perform the vector search

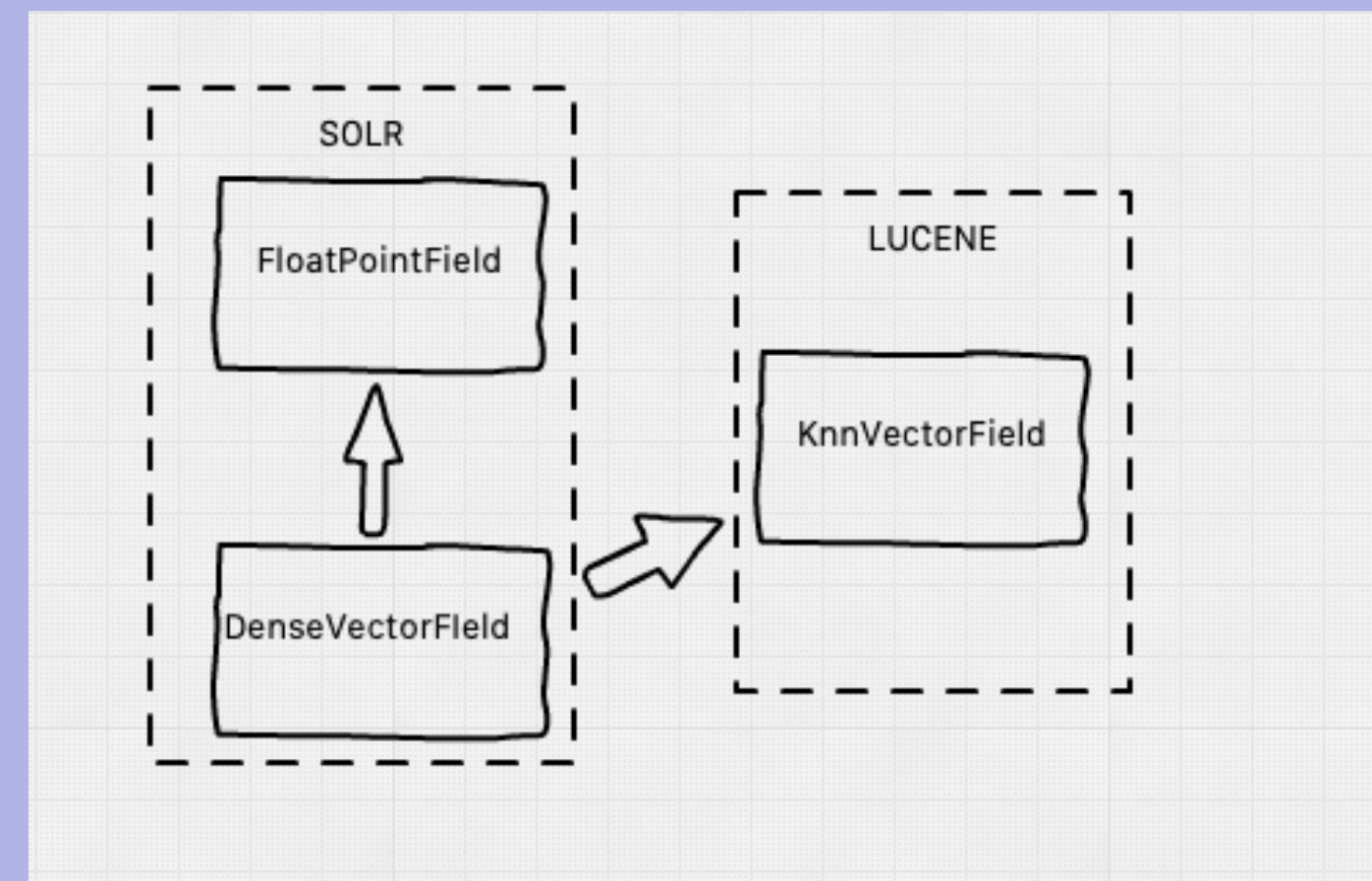
Stored field

The vector is stored using the parent class. The float values of the vector are stored as multivalued `FloatPointField`

Query parser

To search over the vector index, it is necessary to use the `KnnQueryParser`. It takes 3 parameters:

- `f`: field name
- `v`: vector value
- `topK`: number of vectors to return



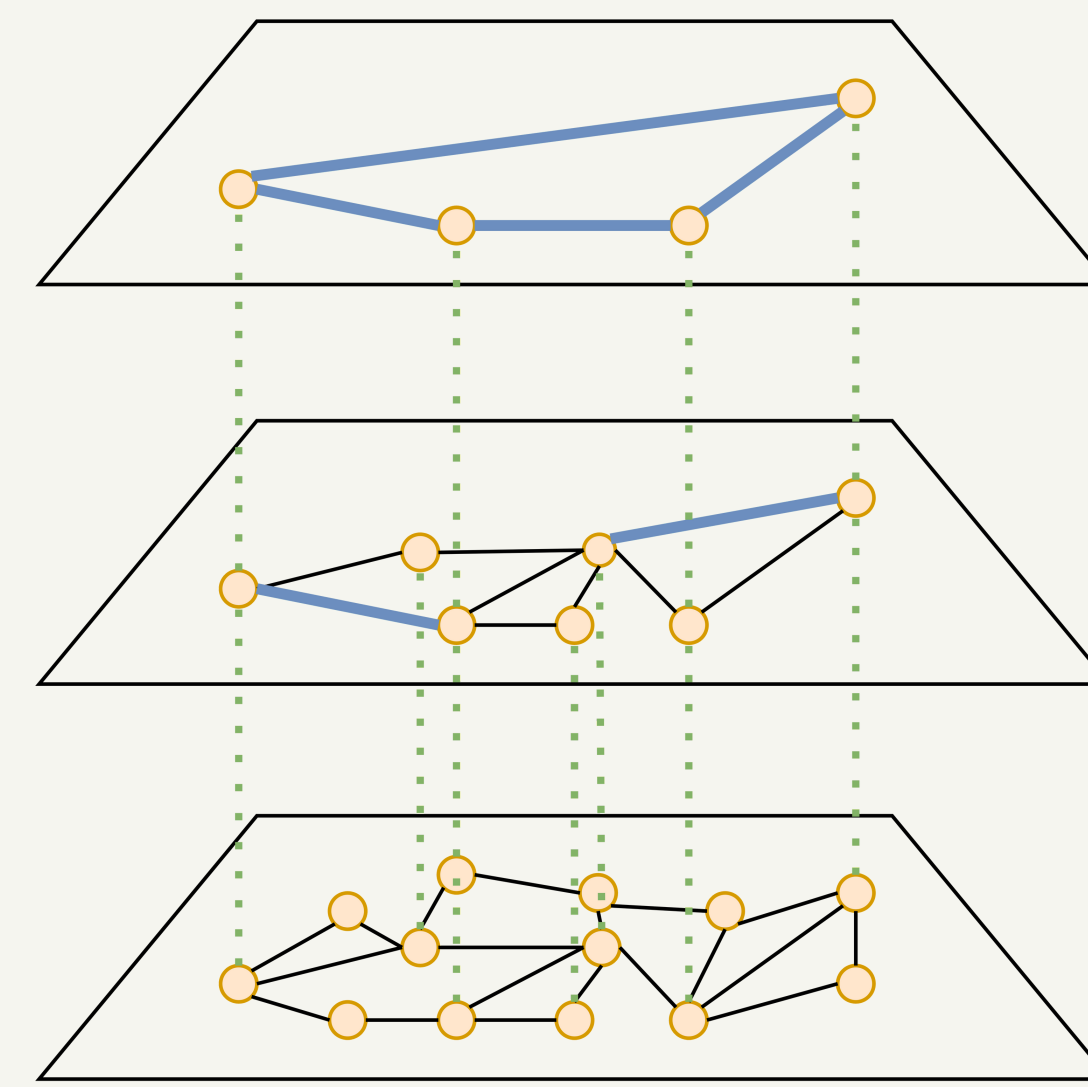
Field type parameters

- **similarityFunction**:
 - EUCLIDEAN – default
 - DOT_PRODUCTS
 - COSINE
- **vectorDimension**: length of the dense vector
- **hnswMaxConnections**: Controls how many of the nearest neighbor candidates are connected to the new node.
 - Default is 16
- **hnswBeamWidth**: the number of candidate neighbors to track while searching the graph for each newly inserted node.
 - Default is 100

HIERARCHICAL NAVIGABLE SMALL WORLDS (HNSW)

Hierarchical Navigable Small World is the top-performing index for vector similarity. This index allows finding the most similar vectors to a target point for a given similarity function.

Its efficiency is due to the hierarchical layout of the graph. This makes the search algorithm able to rapidly converge towards the vectors close to the target.



```
<fieldType name="dense_vector" class="solr.DenseVectorField"
  vectorDimension="3" similarityFunction="cosine" omitNorms="true"/>
<field name="my_vector" type="dense_vector" indexed="true" stored="true" multiValued="false"/>
```

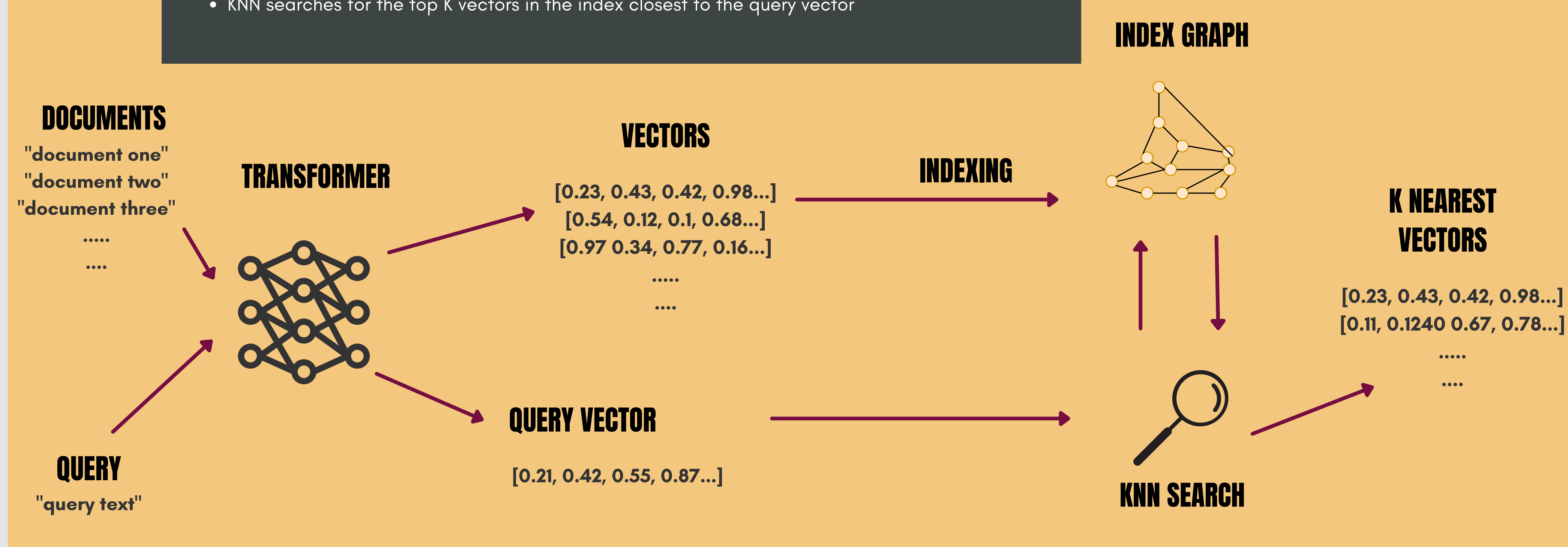
```
curl --location --request POST \
  'http://solrhost/solr/dense_vector_index/update/json/docs?commit=true' \
  --header 'Content-Type: text/xml' \
  --data-raw '{\''id\''': 1, '\''my_vector\''': [0.13, 0.31, 0.52]}
{\''id\''': 2, '\''my_vector\''': [0.55, 0.32, 0.41]}
{\''id\''': 3, '\''my_vector\''': [0.65, 0.97, 0.42]}
{\''id\''': 4, '\''my_vector\''': [0.48, 0.12, 0.75]}
```

INDEXING

- The documents are transformed into vectors using a sentence transformer (like BERT)
- Vectors are indexed in the graph index in solr

QUERYING

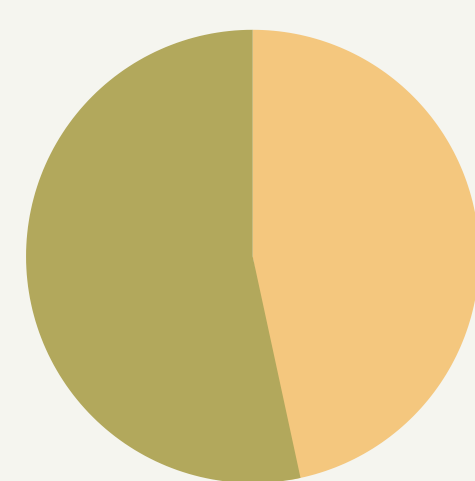
- The query text is transformed into a query vector
- KNN searches for the top K vectors in the index closest to the query vector



```
{
  "responseHeader": {
    "status": 0,
    "QTime": 60,
    "params": {
      "q": "{!knn f=my_vector v=[0.1,0.2,0.3] topK=3}",
      "indent": "true",
      "q.op": "OR",
      "_:": "1648855092945"}},
  "response": { "numFound": 3, "start": 0, "numFoundExact": true, "docs": [
    {
      "id": "1",
      "my_vector": [0.13, 0.31, 0.52]},
    {
      "id": "4",
      "my_vector": [0.48, 0.12, 0.75]},
    {
      "id": "2",
      "my_vector": [0.55, 0.32, 0.41]}
  ]}
}
```

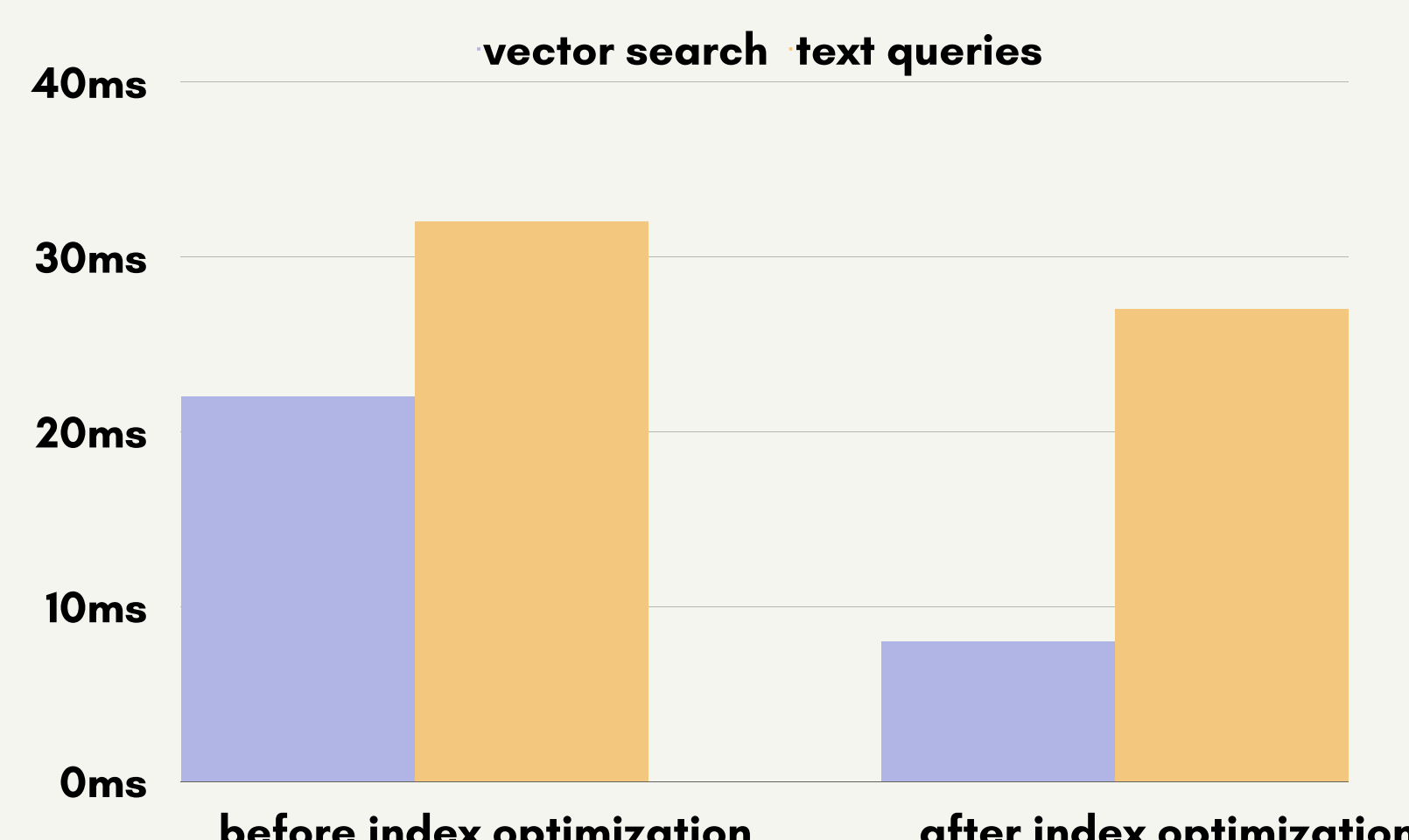
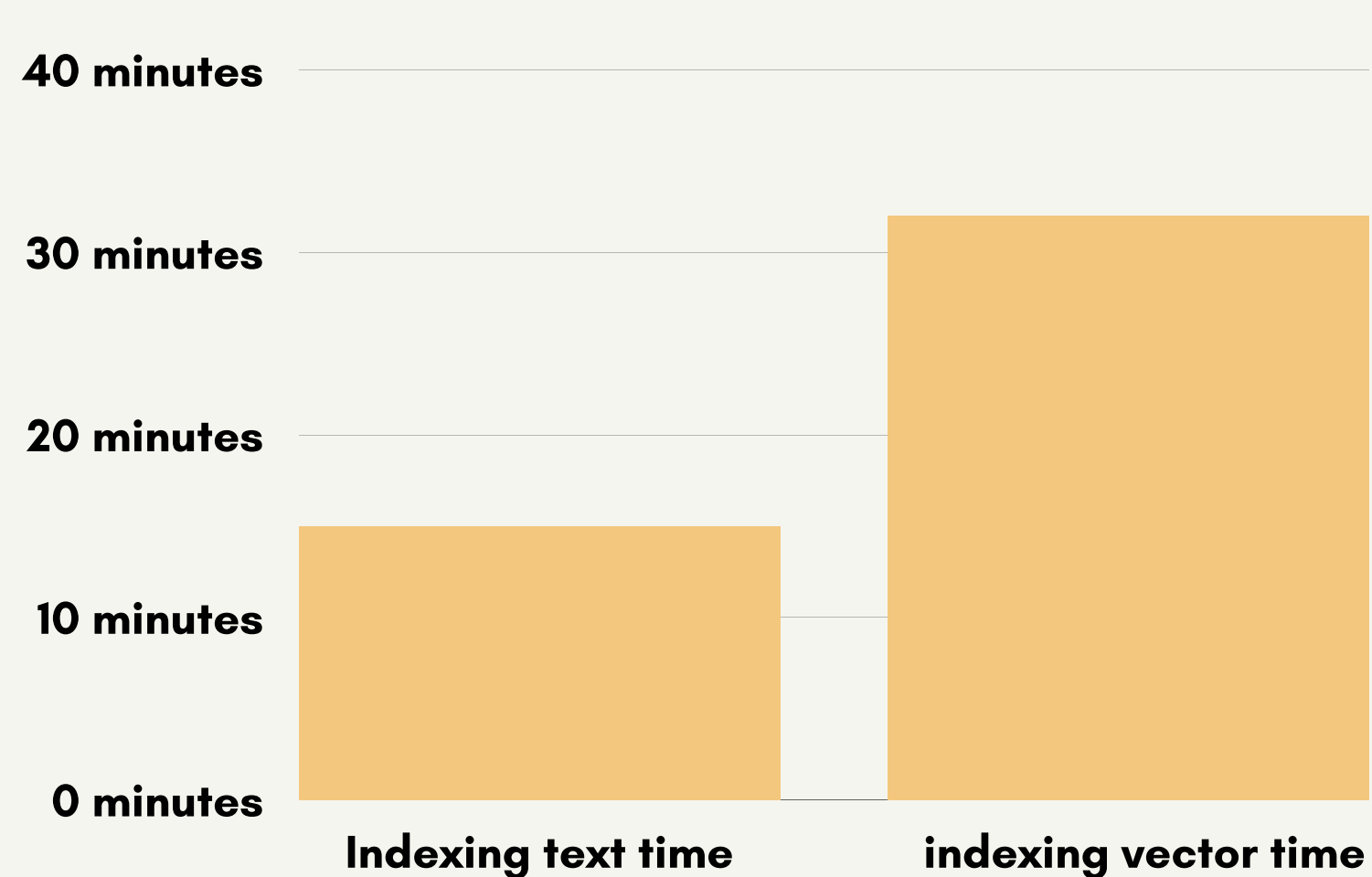
BENCHMARKS

text index size
1.34 GB



vector index size
1.17 GB

# DOCUMENTS	461K
# QUERIES	5750
AVG DOCUMENT LENGTH	1087 WORDS
AVG QUERY LENGTH	5.9 WORDS



FUTURE WORKS

- Import latest **Lucene 9.1** changes into **Solr**:
 - Support for filtering in KNN search
 - Up to **30% improvement** in index throughput for vectors
 - Up to **10% faster** KNN search
- Dense Vector Fields Multivalued
- index/query time BERT integration in Apache Solr

REFERENCES

Andoni, A.; Indyk, P. (2006-10-01). "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions". 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). pp. 459-468. CiteSeerX 10.1.1.142.5471. doi:10.1109/FOCS.2006.49. ISBN 978-0-7695-2720-8.

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, Vladimir Krylov, Approximate nearest neighbor algorithm based on navigable small world graphs, Information Systems, Volume 45, 2014, Pages 61-68, ISSN 0306-4579

Malkov, Yury; Yashunin, Dmitry (2016). "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs". arXiv:1603.09320 [cs.DS].

Alessandro Benedetti. Apache Solr Neural Search. <https://sease.io/2022/01/apache-solr-neural-search.html>

Elia Porciani. Apache Solr Neural Search Knn benchmark. <https://sease.io/2022/01/apache-solr-neural-search-knn-benchmark.html>

Lucene Jira Issue: <https://issues.apache.org/jira/browse/LUCENE-9004?focusedCommentId=17393216&page=com.atlassian.jira.plugin.system.issuetabpanels%3Acomment-tabpanel#comment-17393216>