RADIUS-BASED APPROXIMATE NEAR-NEIGHBOR SEARCH USING HNSW GRAPHS

ABOUT

Kaival Parikh
Software Engineer
Amazon

linkedin.com/in/kaivalnp

Aditya Prakash
Principal Data Scientist
Amazon

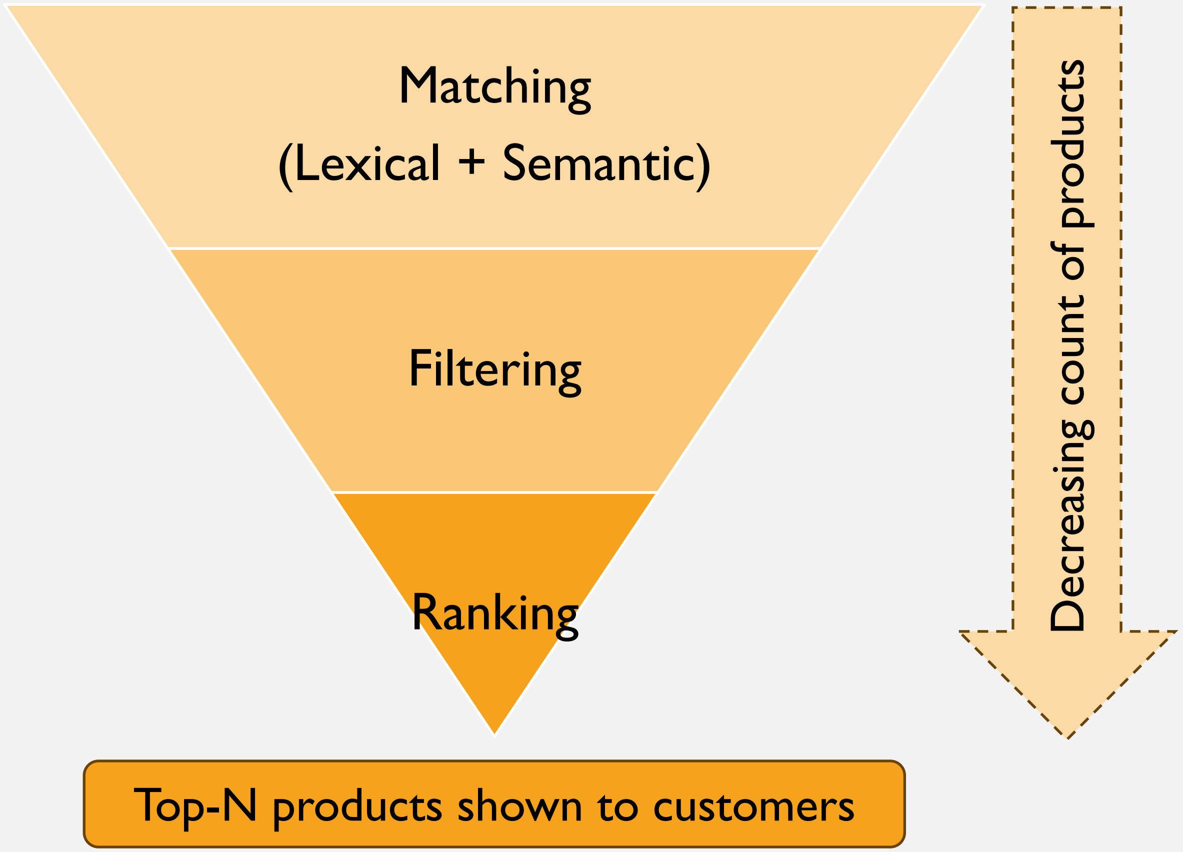
linkedin.com/in/aditya-prakash-6b234410

AGENDA

- Product Search at Amazon
- Vector Search
- Considerations in a production system
- Lucene implementation using HNSW graphs
- Performance
- Q&A

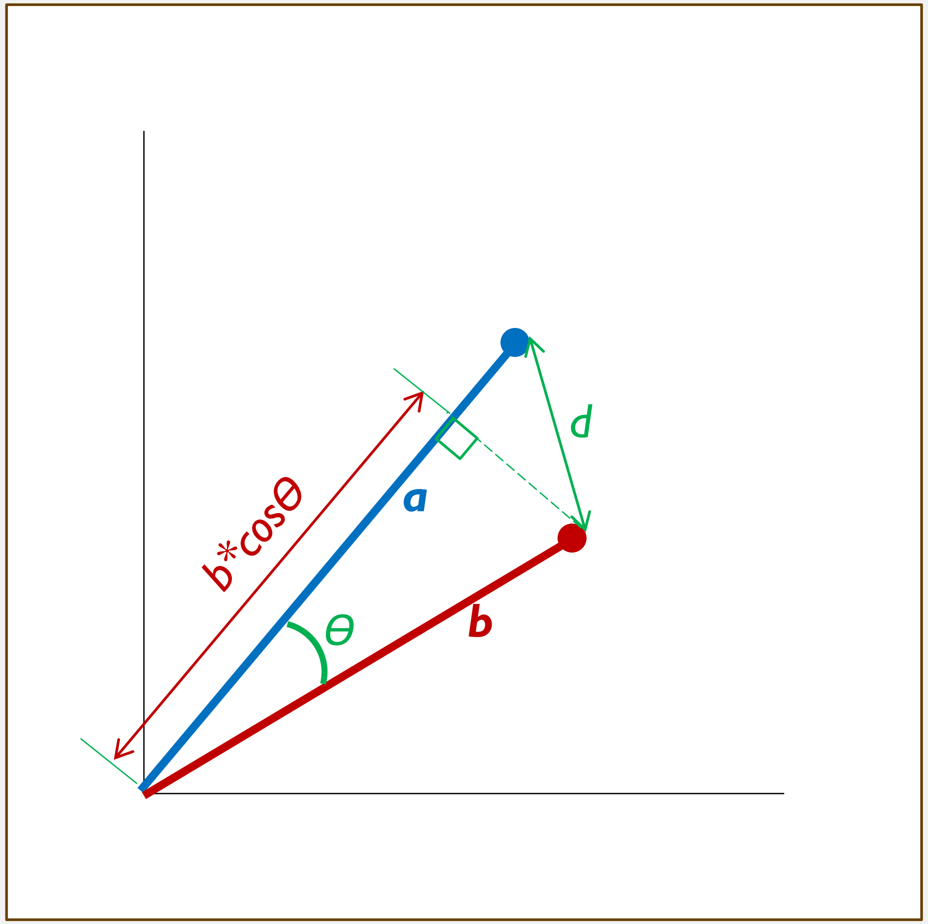
PRODUCT SEARCH AT AMAZON

- Lexical + Semantic matching, filtering and ranking stages
- Query-time filters such as availability, deliverability, brand-preference, etc. are applied
- Multi-stage ranking models are used, which have access to arbitrary runtime and indexed features
- Cosine-similarity scores from semantic matching have a limited role in the final ranking of products

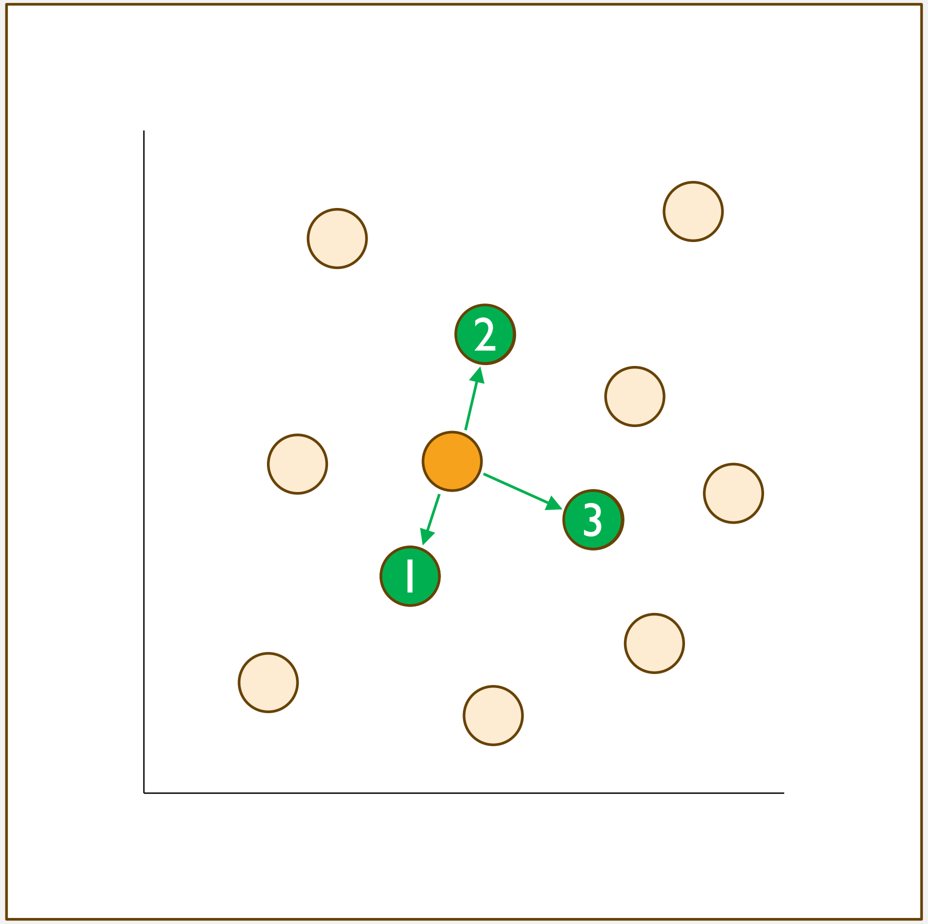


VECTOR SEARCH

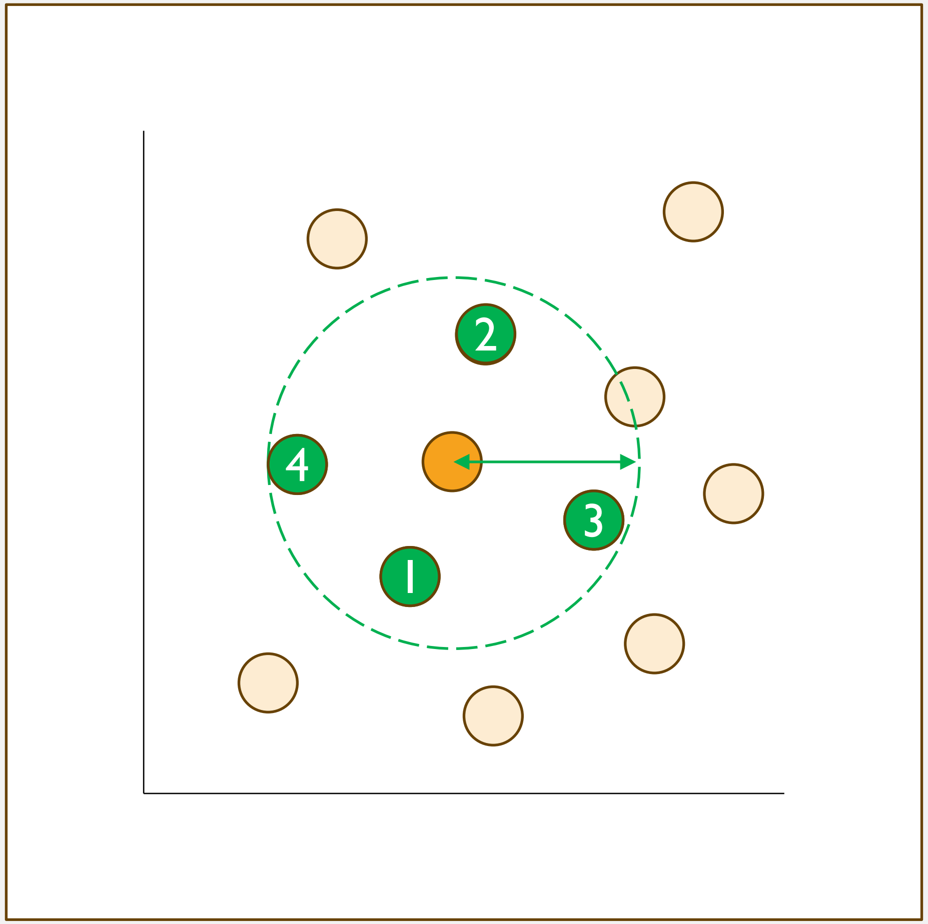
- Vectors are points in an N-dimensional space, and are usually represented as a list of integer or float values.
- Machine learning models typically represent text, image, and videos as vectors, allowing us to easily find similar text, images etc. using vector-similarity search.
- The following similarity measures are commonly used:
  - Euclidean Distance (*d*)
  - Cosine-Similarity (*Θ*)
  - Inner-Product (*IP = a * b * cosΘ*)



TYPES OF VECTOR SEARCH



K-Nearest Neighbor (KNN) Vector Search



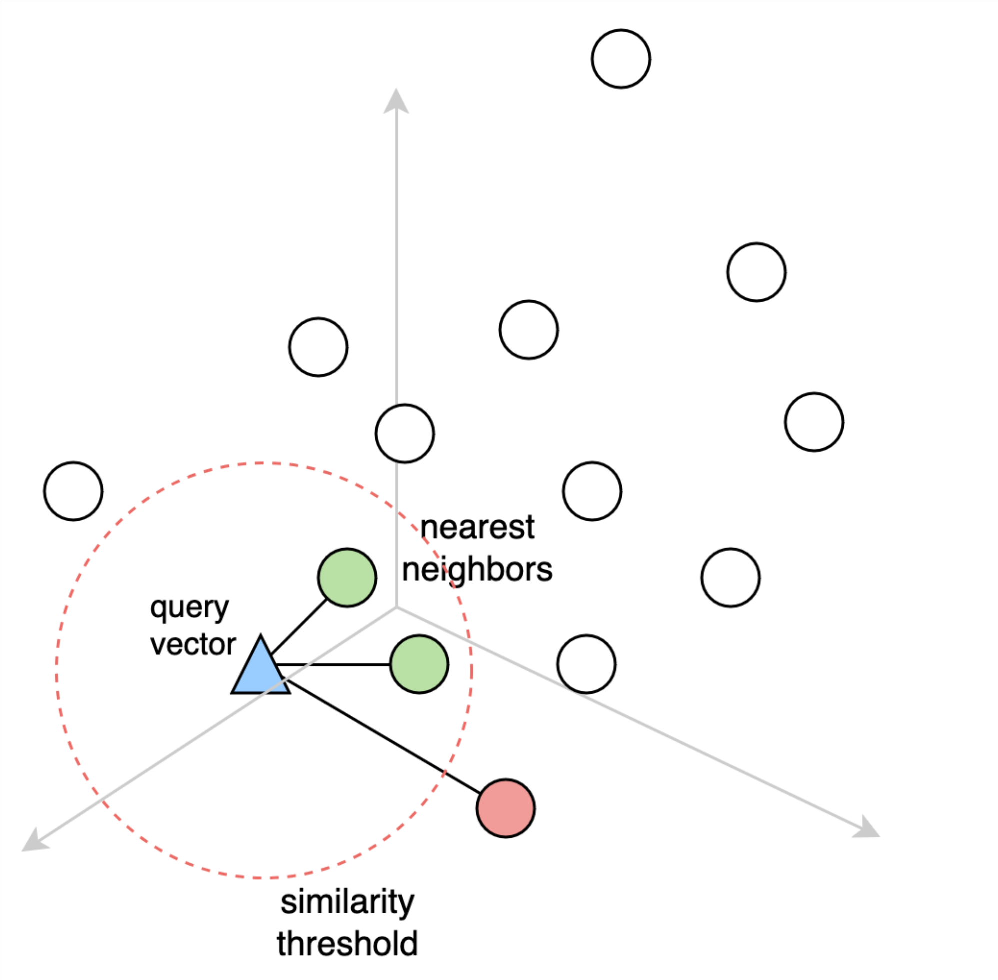
Radius Based Vector Search

CONSIDERATIONS

Reduce irrelevant results

- Query vector present in a sparse part of the space
  - Obscure query
  - Relevant documents not present in the catalog (or not indexed for vector search)
- Show fewer results instead of unrelated ones
- Minimum similarity threshold with the query

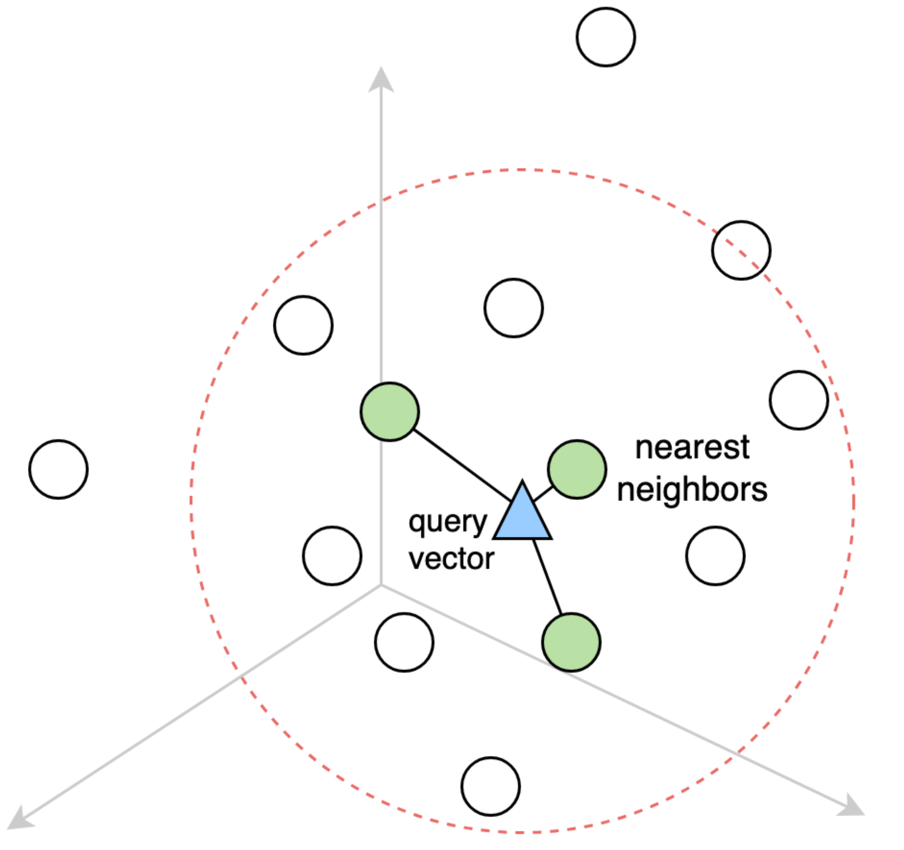
- Unnecessary computations in KNN queries where some of top K results lie outside similarity threshold



CONSIDERATIONS

Multiple matching sources + result rescoring

- Matching sources like lexical terms, behavioral data, more than one vector field, etc.
- Final rescoring has a larger feature set than semantic model
- Potential of missed results because hits outside top K (but still similar enough to the query) are valid candidates for matching

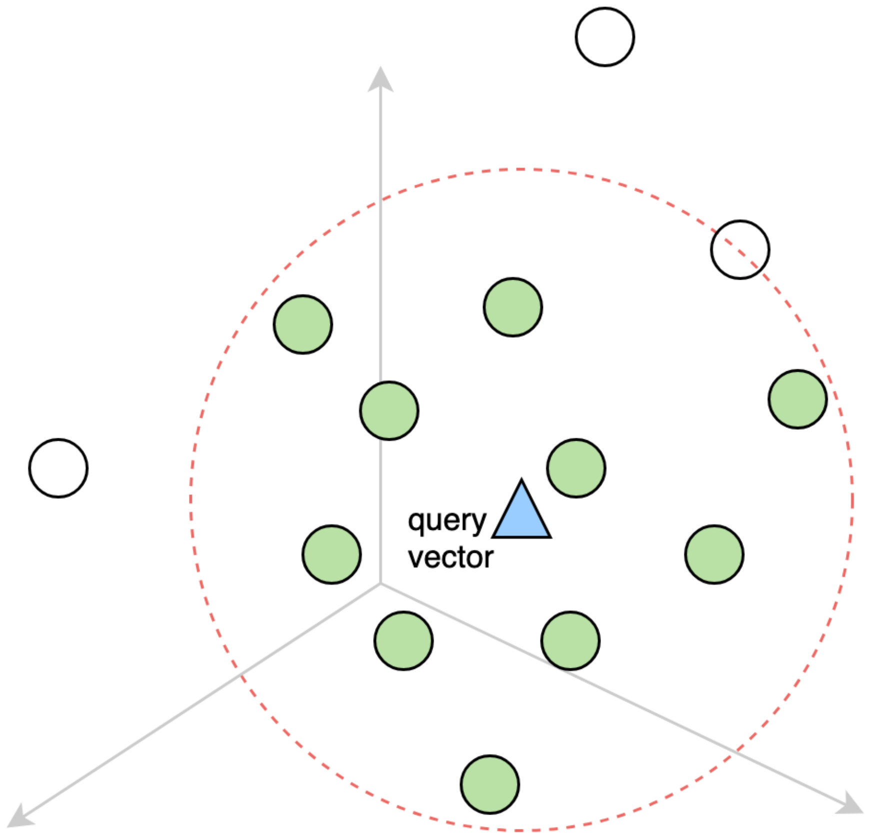


K-Nearest Neighbor (KNN) Vector Search

CONSIDERATIONS

Multiple matching sources + result rescoring

- Matching sources like lexical terms, behavioral data, more than one vector field, etc.
- Final rescoring has a larger feature set than semantic model
- Reduced potential of missed results because all hits similar enough to the query are considered as candidates for matching

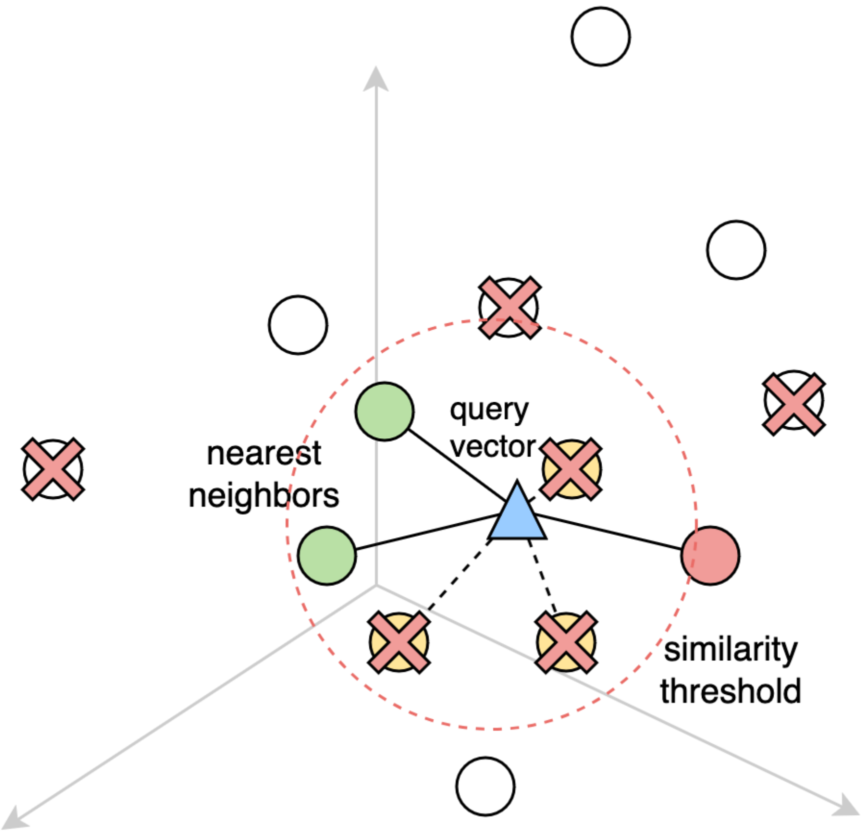


Radius Based Vector Search

CONSIDERATIONS

Query-time filters

- Like availability, delivery speed, brand, etc.
- The K-Nearest Neighbors of the query may not satisfy these constraints
- Lucene solves for this using pre-filtering, but comes at a cost
  - Collect all docs matching the constraints in a set
  - Only match on these documents during retrieval
  - Even if this set is cached and re-used across queries, becomes cost inhibitive due to heap usage with a large number of unique filters

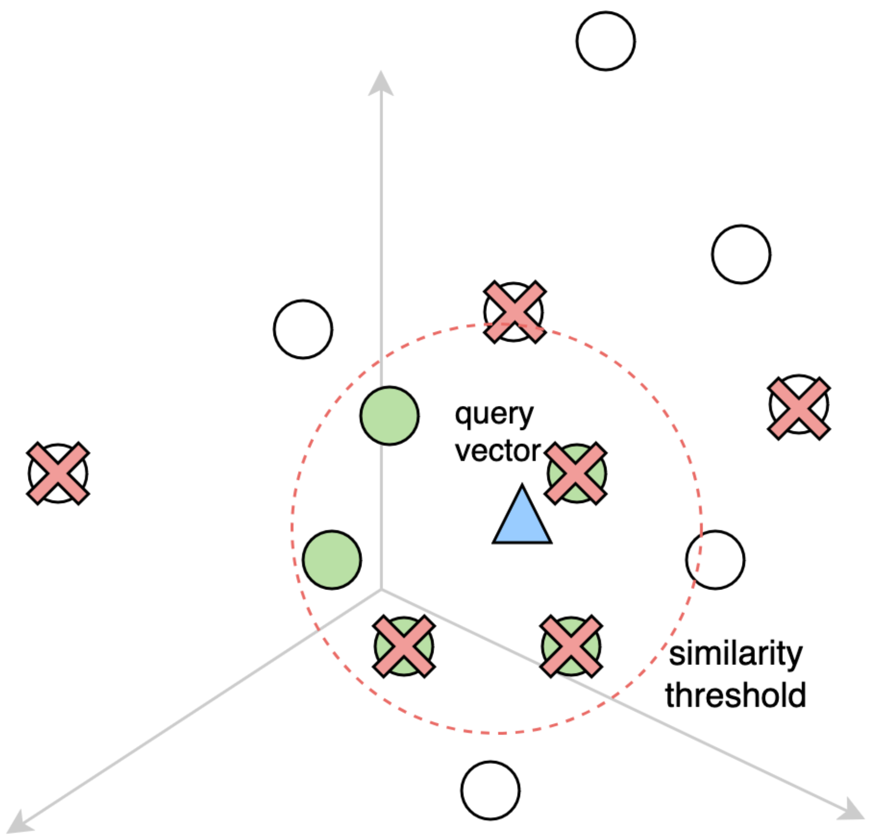


K-Nearest Neighbor (KNN) Vector Search

CONSIDERATIONS

Query-time filters

- Like availability, delivery speed, brand, etc.
- All vectors similar enough to the query are already matched
- No explicit need for pre-filtering, rely on post-filtering



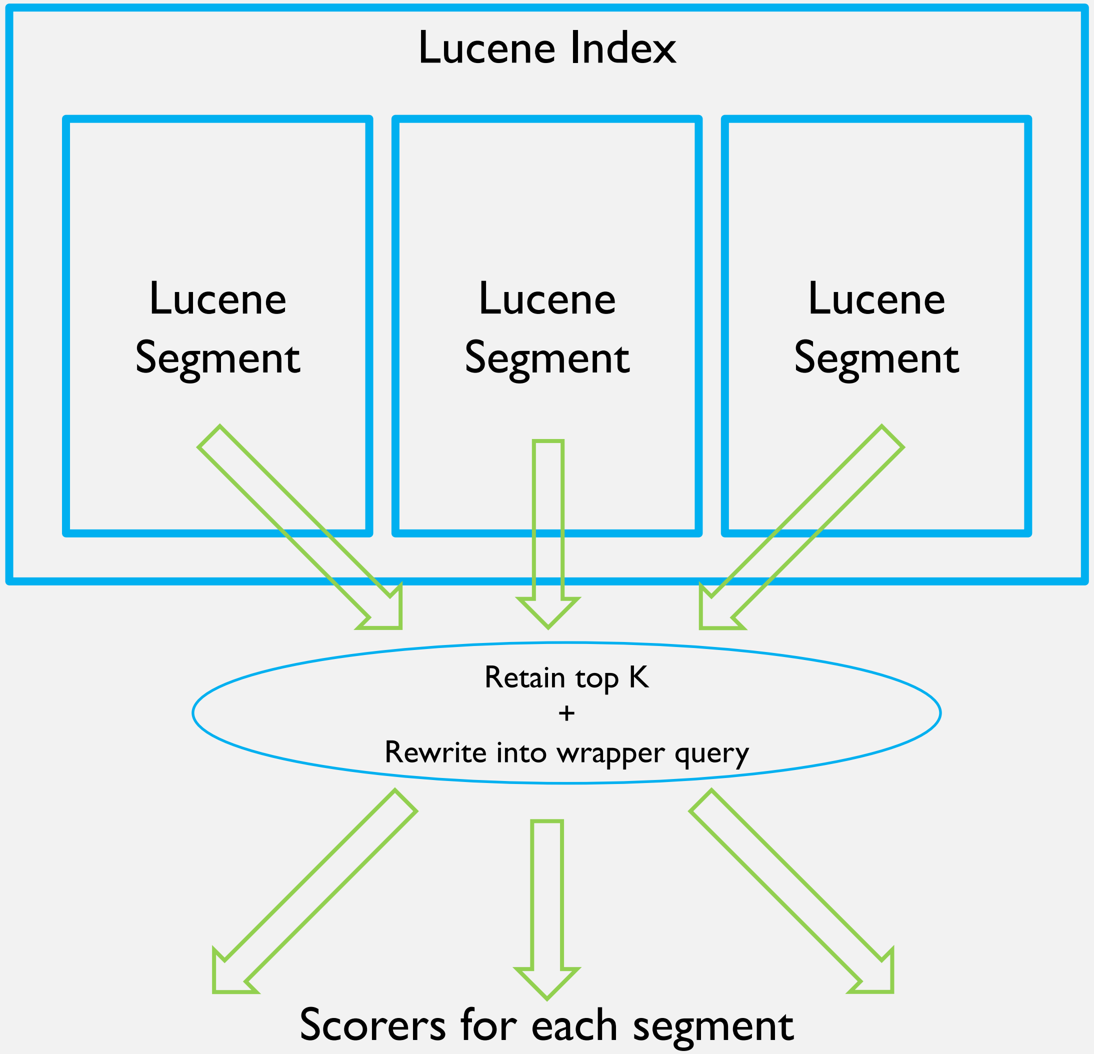
Radius Based Vector Search

CONSIDERATIONS

Distributed nature

- Lucene has independently searchable sub-indexes called segments, and vectors are spread across them
- Segment-level top K vectors need to be collected at a single place to determine results
- Caveats like custom parallelism and non-cacheable

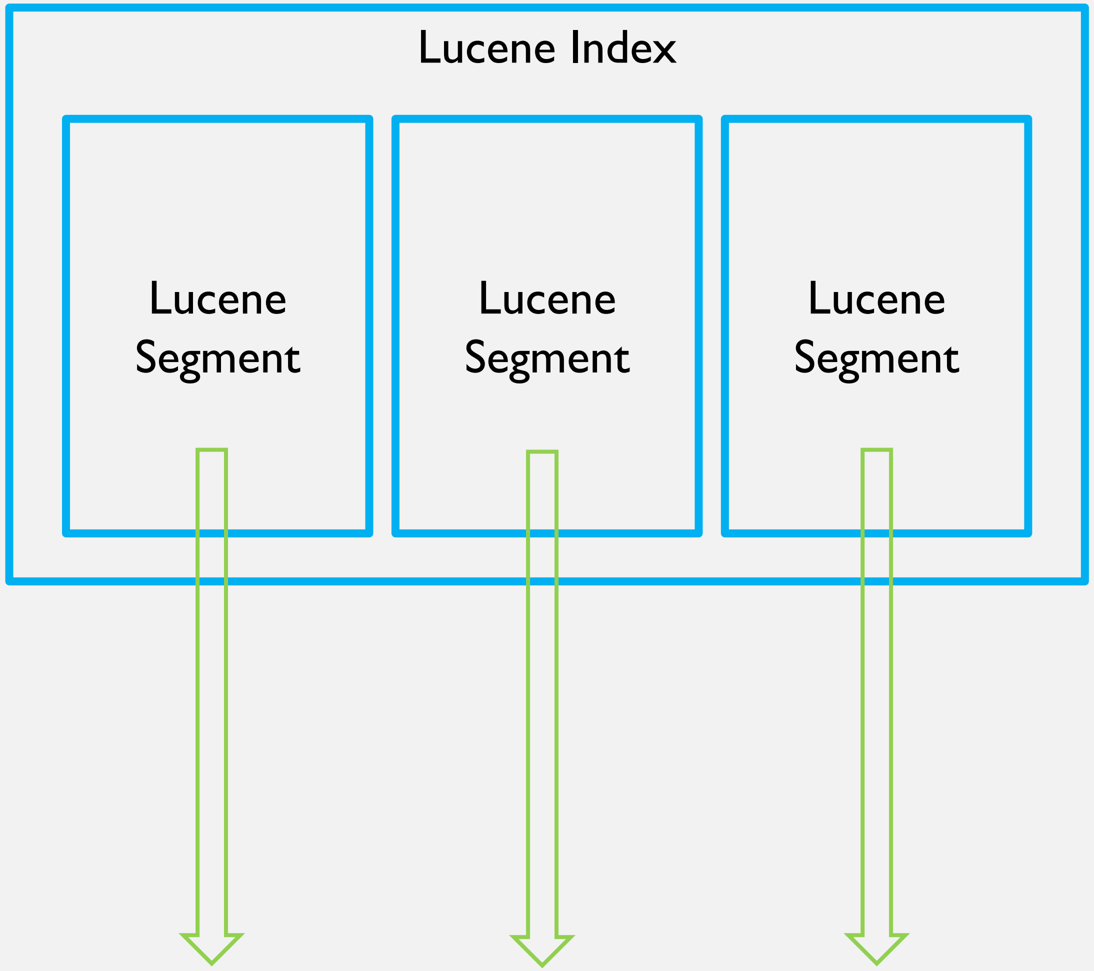
K-Nearest Neighbor (KNN) Vector Search



CONSIDERATIONS

Distributed nature

- Lucene has independently searchable sub-indexes called segments, and vectors are spread across them
- Each document is independently evaluated as a hit (without needing scores of other documents)
- Segment-level results are additive and need not be collected at a single place
- Simpler query implementation and cacheable!



Radius Based Vector Search

Scorers for each segment

LUCENE IMPLEMENTATION

Hierarchical Navigable Small World (HNSW) Graphs (https://arxiv.org/pdf/1603.09320)

- Already implemented since Lucene 9.1 (LUCENE-9004, LUCENE-10054) to perform KNN search
- Relies on document-document similarity to connect each vector to its closest (and diverse) neighbors
- Documents are spread across multiple layers, with each layer having an exponentially increasing superset of documents of the layer above
- Upper layers provide a suitable entry point for actual search in the last layer
- A priority queue of K results is maintained in the last layer, and search stops when the best available candidate cannot replace any collected result

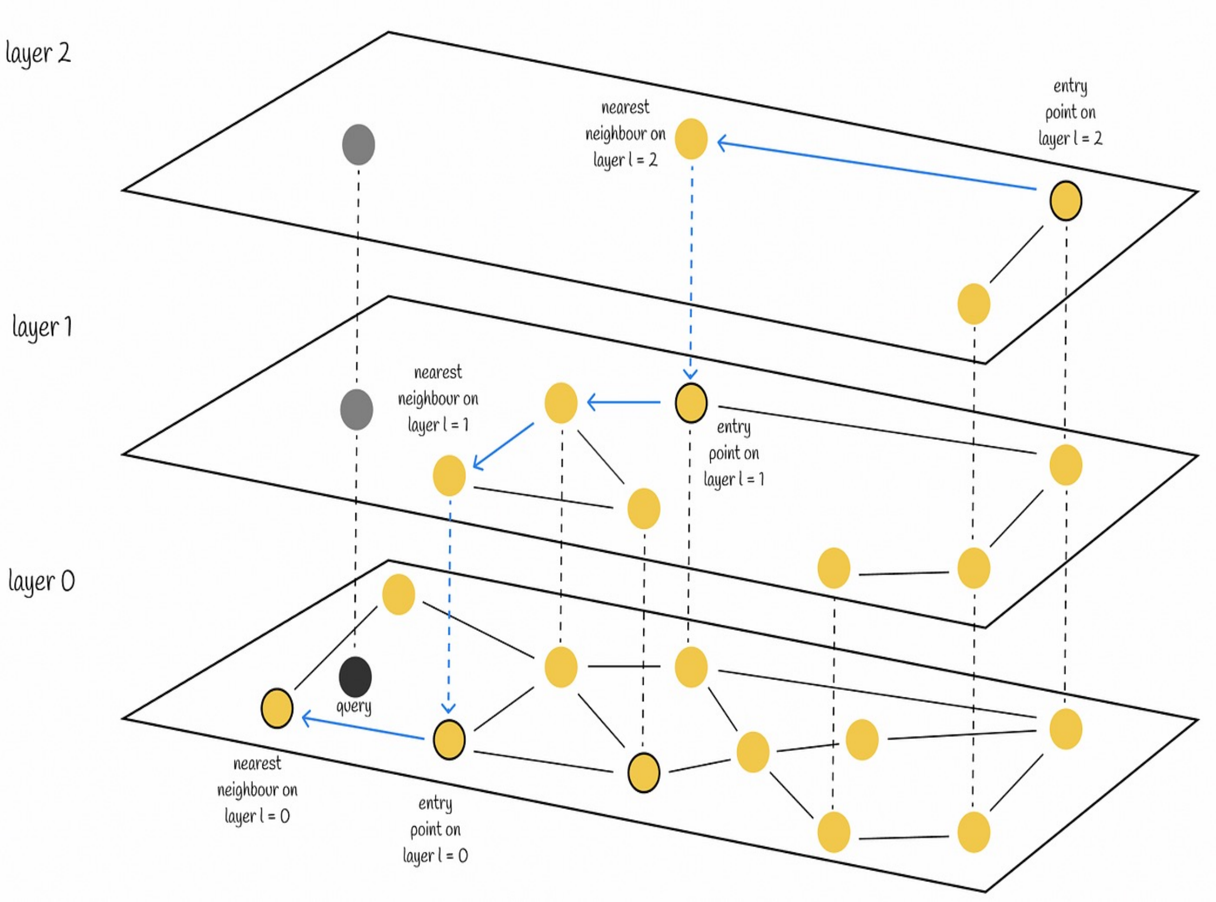
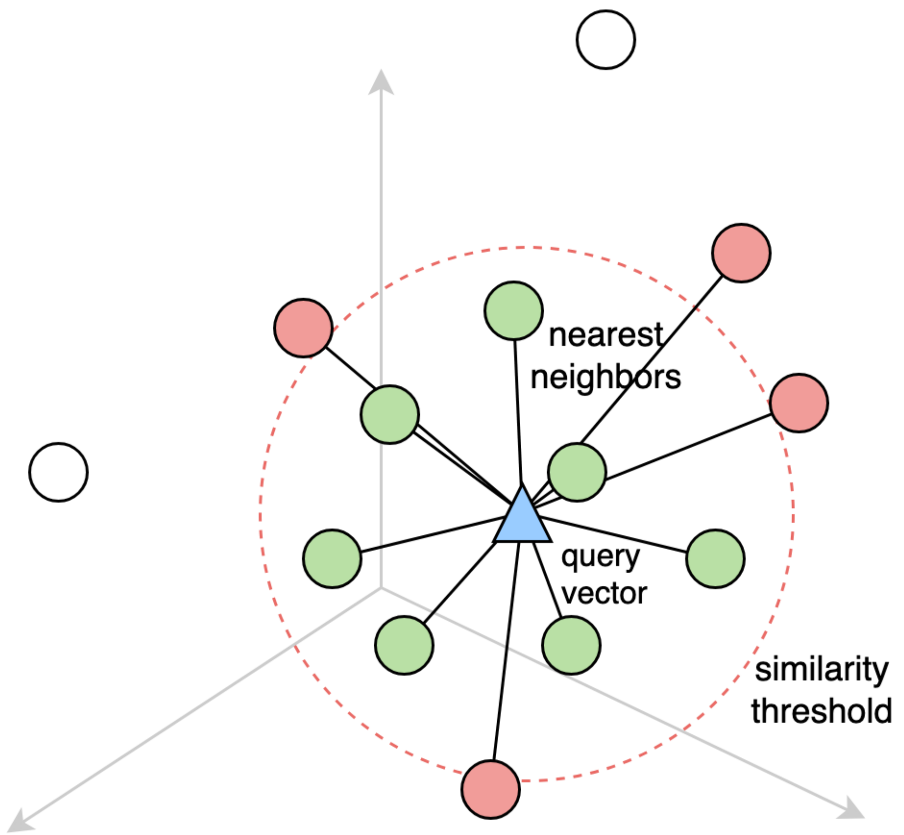


Image Source

LUCENE IMPLEMENTATION

Simulating a Radius-based vector search

- Large K with post-filtering?
  - Incurs additional latency
  - Missed results if not large enough
- Predictive query-level K?
  - Another layer of approximation + complexity

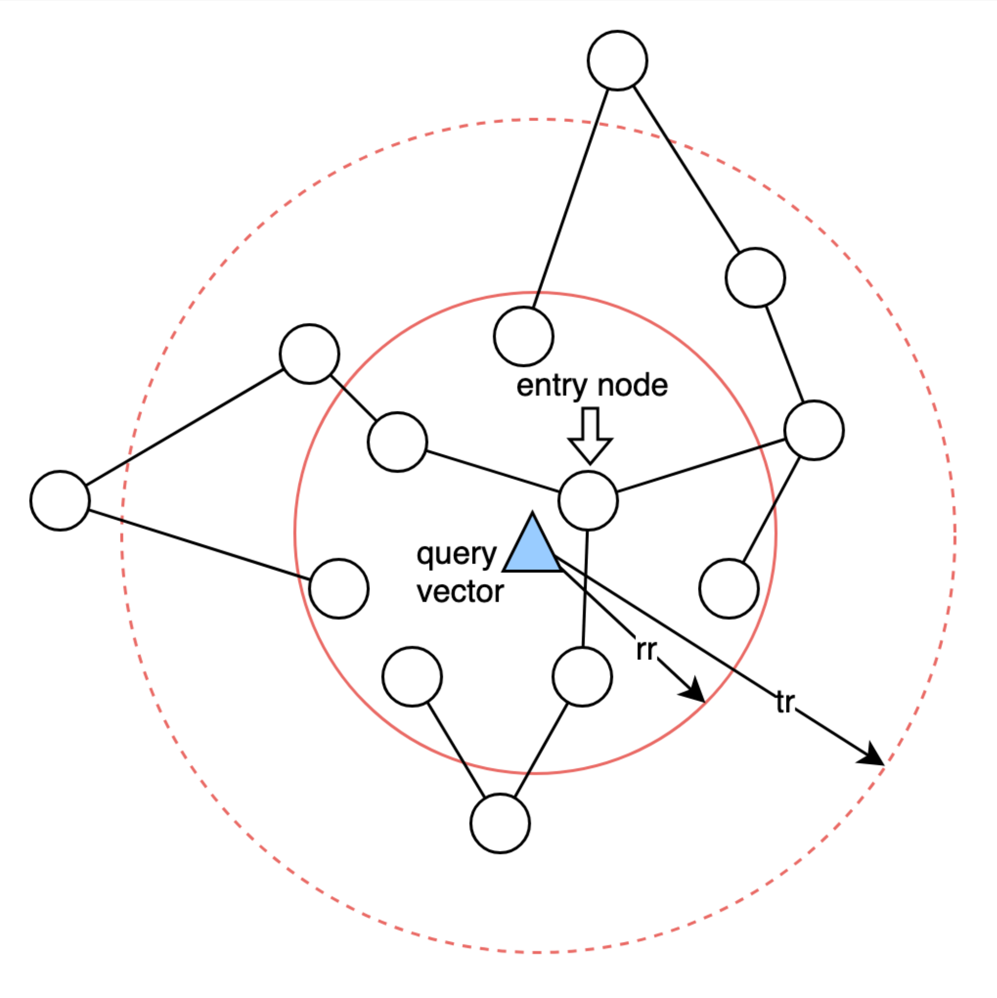


K-Nearest Neighbor (KNN) Vector Search

LUCENE IMPLEMENTATION

Algorithm

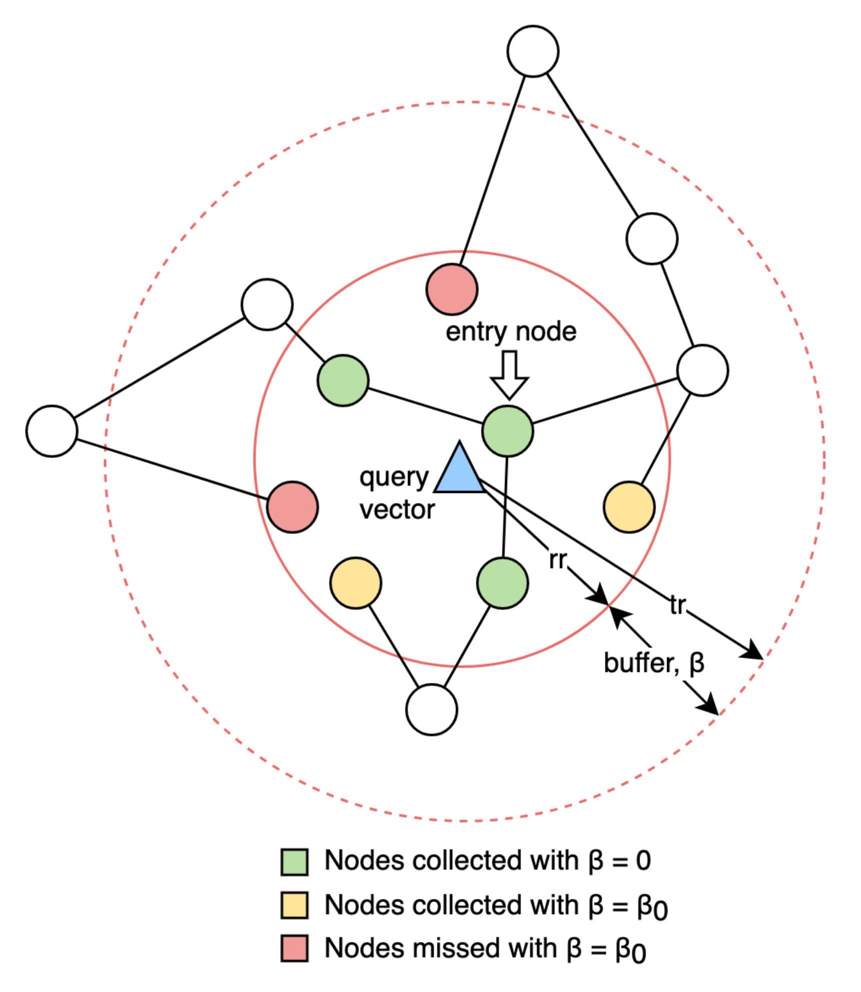
- Released in Lucene 9.10 ([GH#12679](GH#12679))
- Change graph traversal and result collection criteria to be radius-based instead of count-based
    - HNSW graphs are valuable
    - Minimally invasive
- Introduces two parameters, traversalSimilarity and resultSimilarity
- Traverse all nodes with similarity score higher than traversalSimilarity
- Collect all traversed nodes with similarity score higher than resultSimilarity
- Clause to continue traversal as long as better scoring nodes are available (handle edge cases where entry node lies outside traversalSimilarity)



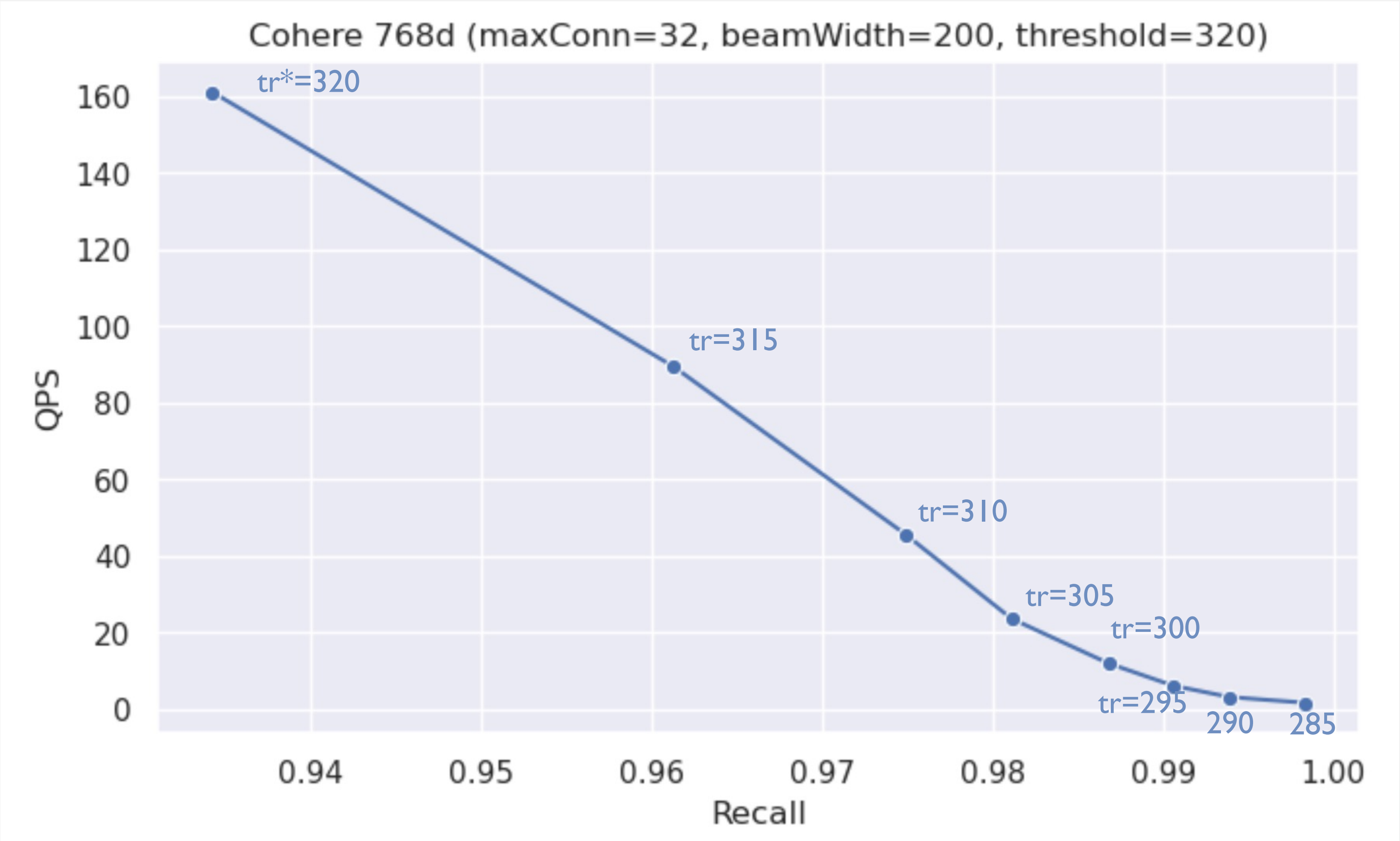
LUCENE IMPLEMENTATION

Benefits

- Exists as a tunable parameter to reach results where some node along the path is lower scoring than resultSimilarity (recall v/s QPS)
- Number of nodes traversed and collected is locality-sensitive (more nodes in dense parts of the graph, and vice versa)
- No need to maintain priority queue of results for highest-scoring top K
- Graph search can be performed in a more appropriate place in the Lucene query flow
- Cacheable!



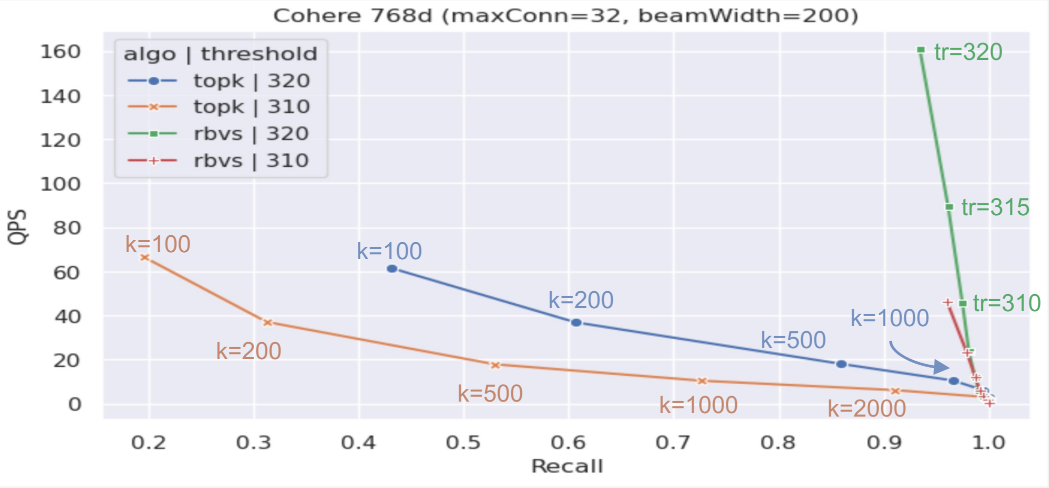
PARAMETER TUNING

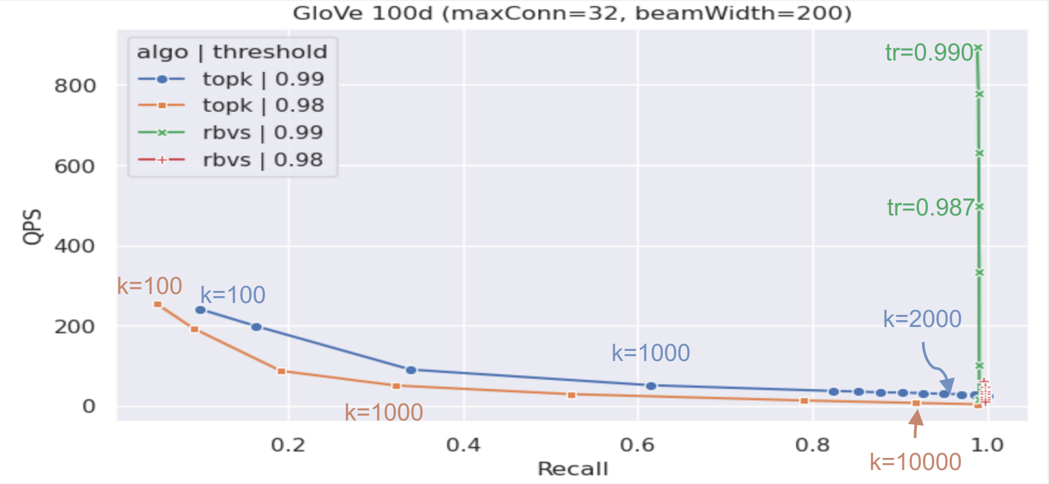


*tr = traversal similarity for Cohere vectors

PERFORMANCE COMPARISON

- Number of documents above a threshold has been used as a baseline for recall calculation
  - As a consequence, low values of top K in the KNN setup result in very low recall values
  - Points plotted in the charts are for varying values of top K for KNN, and varying values of traversalSimilarity for RBVS
- RBVS is capable of providing very high recall without compromising on QPS for applications which require to find all documents above a given threshold



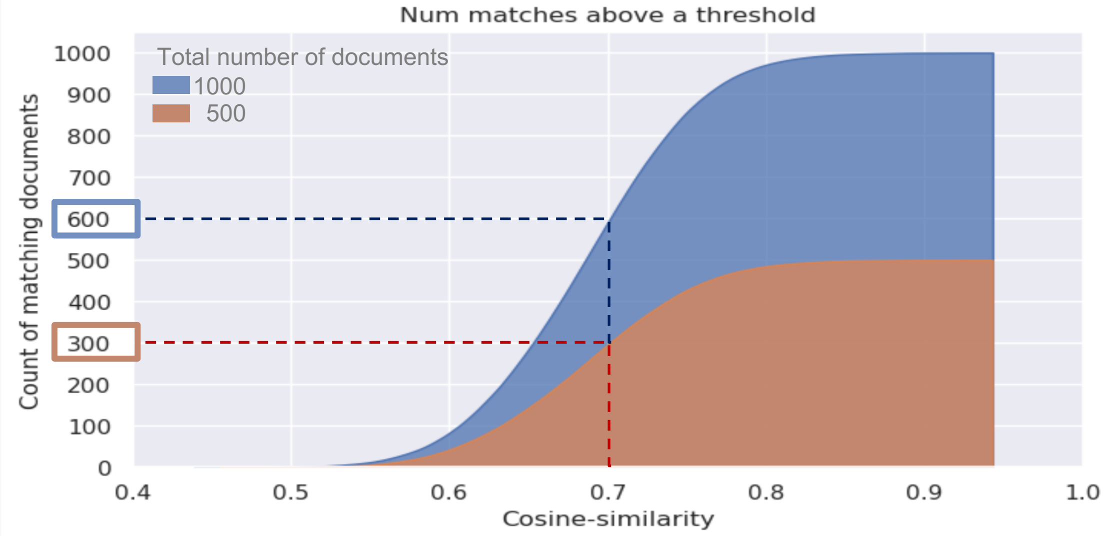


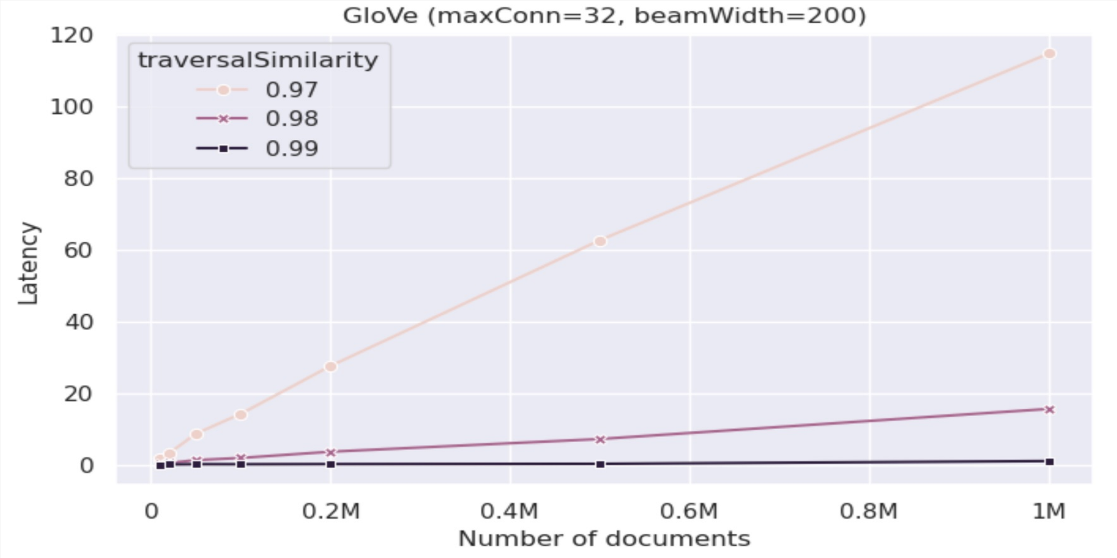
TIME COMPLEXITY

- In brute-force (or exact) search, doubling the number of documents leads to doubling in the number of matches for a fixed threshold

- Time complexity is dictated by the number of nodes visited during graph search, which has an upper bound of actual number of vectors with a score above traversal-threshold

- In actual simulations for RBVS, we found the number of nodes traversed, and thus latency, increases linearly with increase in number of documents





THANK YOU