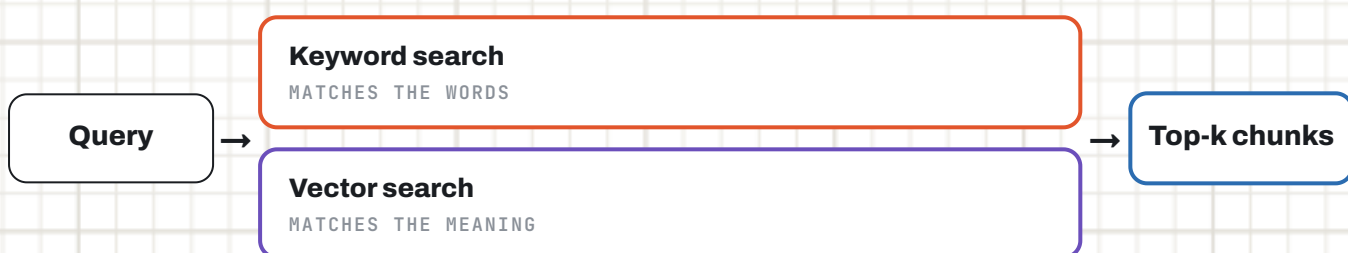


10 Search Algorithms

BEHIND EVERY RAG RETRIEVER

How each one finds the right chunk, what it trades away, and which one to pick for your RAG system.

**KEYWORD**

Which documents share the query's words?

- 01 **BM25**
- 03 **TF-IDF**
- 05 **Query Likelihood**

GRAPH

Walk neighbour links toward the query.

- 02 **HNSW**
- 08 **DiskANN**

CLUSTER + COMPRESS

Search fewer vectors, and smaller ones.

- 04 **IVF**
- 06 **ScaNN**
- 07 **IVF-PQ**

HASHING

Put similar vectors in the same bucket.

- 09 **LSH**

EXACT

Check every vector. No shortcuts.

- 10 **Brute Force**

The cheat sheet.

ONE LINE EACH

#	ALGORITHM	FAMILY	RESULT	IN ONE LINE
01	BM25	KEYWORD	RANKED	Scores documents by matching words: rare words count more, repeats count less.
02	HNSW	GRAPH	APPROX	Layered neighbour graph, searched coarse to fine. The usual default.
03	TF-IDF	KEYWORD	RANKED	Weighs a word by how often it's here times how rare it is everywhere.
04	IVF	CLUSTER	APPROX	Groups vectors into clusters, then searches only the nearest few.
05	Query Likelihood	KEYWORD	RANKED	Ranks documents by how likely each one is to have produced the query.
06	ScaNN	CLUSTER	APPROX	Google's library: partition, score compressed vectors, rescore the best.
07	IVF-PQ	CLUSTER	APPROX	IVF plus product quantization: each vector shrinks to a few bytes.
08	DiskANN	GRAPH	APPROX	A graph index that lives on SSD: a billion vectors on one machine.
09	LSH	HASHING	APPROX	Hashes similar vectors into the same bucket, then searches only that bucket.
10	Brute Force	EXACT	EXACT	Compares the query with every vector. Perfect recall, linear cost.

The one idea to keep: keyword search finds the exact words (product codes, error IDs, names), vector search finds the meaning. Good RAG retrievers use both. Every vector index except brute force is **approximate**: it trades a little recall for a lot of speed or memory.

How to use this guide: read the cheat sheet, then the card for anything you couldn't explain out loud. Each card ends with a one-line interview answer and a link to the original paper.

01 BM25 Best Matching 25

KEYWORD

WHAT IT IS

Scores a document by the query words it contains. Rare words count more, each repeat of a word adds less than the last, and long documents don't win just by being long.

WHY IT MATTERS

Still the strongest simple baseline, and it catches what embeddings miss: product codes, error IDs, names. It's the default ranking in Lucene, Elasticsearch and OpenSearch.

Knobs: $k_1 \approx 1.2-2.0$ (repeats), $b = 0.75$ (length)

SAY IT "TF-IDF with two fixes: repeats saturate, and long documents don't win by size."

Robertson & Zaragoza, 2009 · The Probabilistic Relevance Framework: BM25 and Beyond · doi.org/10.1561/15000000019

02 HNSW Hierarchical Navigable Small World

GRAPH

WHAT IT IS

Links every vector to its nearest neighbours in a stack of graphs, sparse at the top and complete at the bottom. A search enters at the top and hops greedily toward the query, layer by layer.

WHY IT MATTERS

High recall at low latency, which is why pgvector, Qdrant, Weaviate and Milvus all offer it. The cost is memory: full vectors plus the graph sit in RAM.

Knobs: M (links per node), $efSearch$ (recall vs speed)

SAY IT "Layers of neighbour graphs, searched coarse to fine: approximate, but very fast."

Mal'kov & Yashunin, 2016 · Efficient and robust approximate nearest neighbor search using HNSW graphs · arxiv.org/abs/1603.09320

03 TF-IDF Term Frequency × Inverse Document Frequency

KEYWORD

WHAT IT IS

Weighs each word by how often it appears in this document (tf) times how rare it is across all documents (idf): $tf \times \log(N / df)$.

WHY IT MATTERS

The idea under almost all keyword search: "the" is everywhere so it scores near zero, "chargeback" is rare so it carries the match.

idf: in 10 of 1M docs = 5 · in every doc = 0

SAY IT "Frequent here, rare elsewhere: that's the word this document is about."

Sparck Jones, 1972 · A Statistical Interpretation of Term Specificity and Its Application in Retrieval · doi.org/10.1108/eb026526

04 IVF Inverted File Index

CLUSTER

WHAT IT IS

k-means splits the vectors into n clusters. A query is compared with the cluster centres, then only the vectors in the n_{probe} nearest clusters are searched.

WHY IT MATTERS

Less memory than a graph and quick to build. The risk: a true neighbour sitting just across a cluster border gets missed.

n_{probe} 10 of 1,000 clusters $\approx$ 1% of vectors

SAY IT "Cluster first, then search only the closest few. n_{probe} trades recall for speed."

Douze et al., 2024 · The Faiss library · arxiv.org/abs/2401.08281

05 Query Likelihood

KEYWORD

WHAT IT IS

Builds a small language model for each document and ranks documents by how likely that model is to generate the query: $P(\text{query} \mid \text{document})$.

WHY IT MATTERS

The probabilistic view of keyword search. Smoothing with collection-wide word counts stops one missing word from zeroing the score. LLM rerankers reuse the idea: score a passage by how likely it makes the question.

Smoothing: a missing word lowers the score, not zeroes it

SAY IT **"Rank each document by how likely it is to have produced the query."**

Ponte & Croft, 1998 · A Language Modeling Approach to Information Retrieval · doi.org/10.1145/290941.291008

06 ScaNN Scalable Nearest Neighbors

CLUSTER

WHAT IT IS

Google's vector search library. It partitions the data, scores candidates with compressed vectors, then rescores the best with full ones. Its anisotropic quantization keeps small the errors that would change the ranking.

WHY IT MATTERS

Led the public ann-benchmarks when it was released, and powers Google Cloud's Vertex AI Vector Search and AlloyDB's ScaNN index. Built for inner-product search at scale.

Partition → score compressed → rescore top with full vectors

SAY IT **"Compress for what ranking cares about, not for overall reconstruction error."**

Guo et al., 2019 · Accelerating Large-Scale Inference with Anisotropic Vector Quantization · arxiv.org/abs/1908.10396

07 IVF-PQ IVF + Product Quantization

CLUSTER

WHAT IT IS

IVF clustering, plus every vector compressed: split it into m sub-vectors and replace each with the ID of its nearest of 256 learned centroids, one byte each.

WHY IT MATTERS

Fits hundreds of millions of vectors in RAM. Distances become approximate, so production systems usually re-rank the top results with the full vectors.

768-d float32: 3,072 bytes → 96 bytes with $m = 96$ (32× smaller)

SAY IT **"IVF searches fewer vectors, PQ makes each one tiny. Re-rank to win back accuracy."**

Jégou, Douze & Schmid, 2011 · Product Quantization for Nearest Neighbor Search · doi.org/10.1109/TPAMI.2010.57

08 DiskANN

GRAPH

WHAT IT IS

A graph index (Vamana) built to live on SSD. Compressed vectors stay in RAM to steer the search, and the full vectors and graph links are read from disk in a few hops.

WHY IT MATTERS

Billion-vector search on one machine without a huge RAM bill. It's built into Azure Cosmos DB and Timescale's pgvectorscale.

1B vectors · 64 GB RAM · 5,000+ queries/s · <3 ms · 95%+ recall@1

SAY IT **"A compressed copy in RAM, the full graph on SSD: billion-scale on one machine."**

Subramanya et al., 2019 · DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node · [NeurIPS 2019](#)

09 LSH Locality-Sensitive Hashing

HASHING

WHAT IT IS

Hashes vectors so similar ones tend to land in the same bucket. For cosine: one bit per random hyperplane, set by which side the vector falls on. A query checks only its own buckets.

WHY IT MATTERS

Simple, fast to build and provably correct with high probability. Still common for near-duplicate detection; on dense embeddings, graph and IVF indexes usually get more recall per query.

Random-hyperplane bit: $P(\text{same bit}) = 1 - \theta/\pi$

SAY IT **"Hash so that neighbours collide, then search one bucket instead of everything."**

Indyk & Motwani, 1998 · Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality · doi.org/10.1145/276698.276876
Cosine version: Charikar, 2002 · Similarity Estimation Techniques from Rounding Algorithms · doi.org/10.1145/509907.509965

10 Brute Force Exact k-NN · flat index

EXACT

WHAT IT IS

Compares the query with every vector and keeps the top k. No index to build, no training, no approximation.

WHY IT MATTERS

100% recall, so it's the ground truth every other index is measured against. Fast enough for small collections, especially on a GPU, but cost grows linearly with the data.

Per query: $N \times d$ multiply-adds ($1M \times 768 \approx 768M$)

SAY IT **"Exact but linear. Start here, and add an index when latency forces you to."**

Johnson, Douze & Jégou, 2017 · Billion-scale similarity search with GPUs · arxiv.org/abs/1702.08734

Measure it: run brute force on a sample of real queries, then check what share of its top 10 your index also returns (recall@10). Tune efSearch or nprobe until that number is good enough.

Which one should you use?

START FROM YOUR CONSTRAINT

Users type **exact terms**: product codes, error IDs, names?

Keep BM25 in the mix 01

A **small collection**, or you need exact results?

Brute force 10

Vectors **fit in RAM** and latency matters most?

HNSW 02

Memory is tight, or the index must build fast?

IVF or IVF-PQ 04 07

A **billion vectors** on one machine, with an SSD?

DiskANN 08

On **Google Cloud**, or inner-product search at scale?

ScaNN 06

Finding **near-duplicates** rather than answers?

LSH 09

TF-IDF (03) and Query Likelihood (05) are the theory behind BM25. Learn them to explain BM25 in an interview; use BM25 in practice.

Hybrid search: BM25 + HNSW

Run both, then merge the two ranked lists with Reciprocal Rank Fusion. It uses ranks, not scores, so BM25 and cosine scores never need to be on the same scale.

$$\text{score}(\text{doc}) = \sum \frac{1}{(k + \text{rank})} \cdot k = 60$$

```
def rrf(rankings, k=60):
    scores = {}
    for ranking in rankings:          # e.g. [bm25_ids, hnsw_ids]
        for rank, doc_id in enumerate(ranking, start=1):
            scores[doc_id] = scores.get(doc_id, 0) + 1 / (k + rank)
    return sorted(scores, key=scores.get, reverse=True)
```

Cormack, Clarke & Büttcher, 2009 · Reciprocal Rank Fusion outperforms Condorcet and individual rank learning methods · doi.org/10.1145/1571941.1572114

Found this useful?

Follow SKD Neuron for more explainers on RAG, AI agents and AI system design. Save this for your next interview.

@SKDNEURON
SKDNEURON.COM