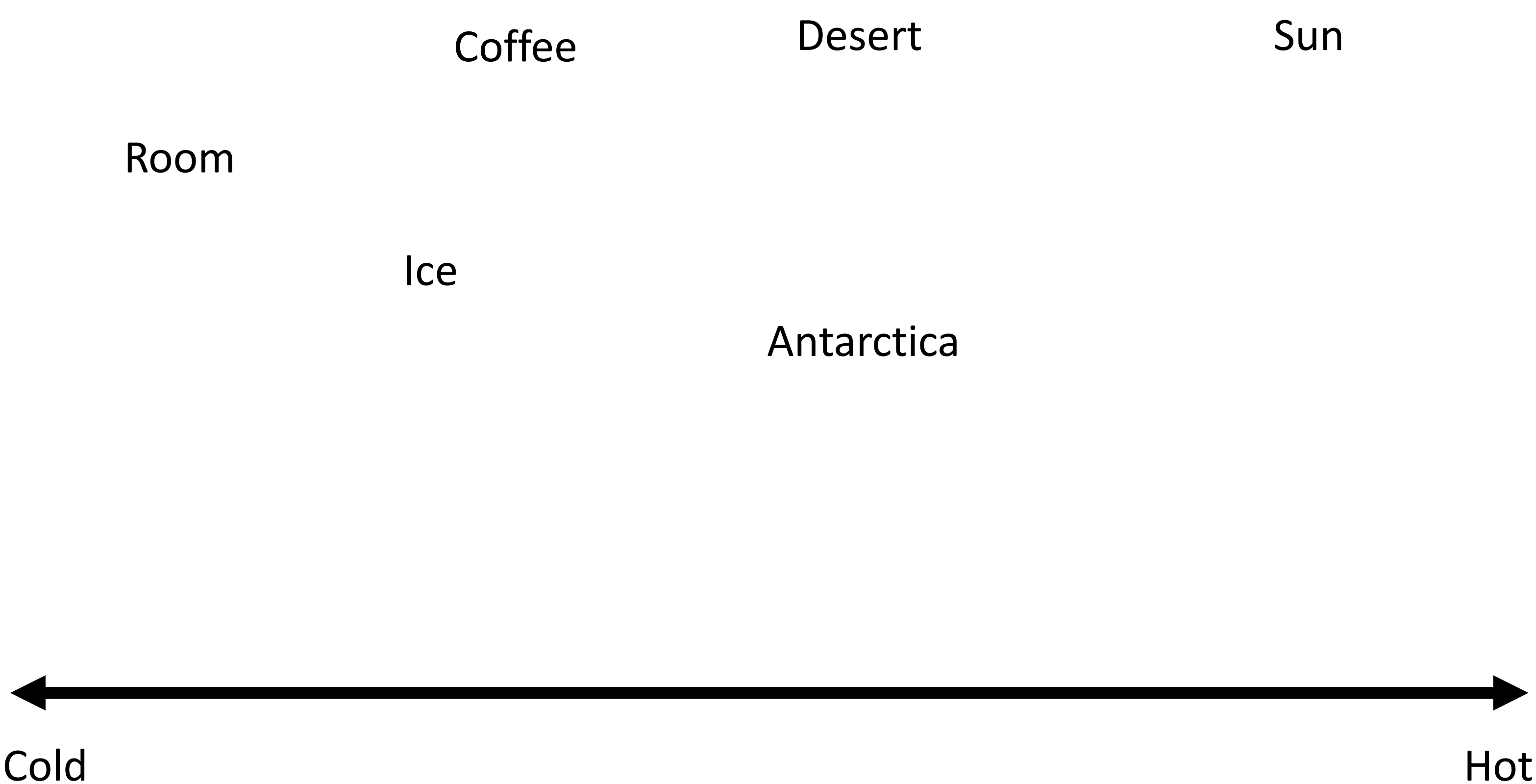
VECTOR DATABASES: AN OVERVIEW

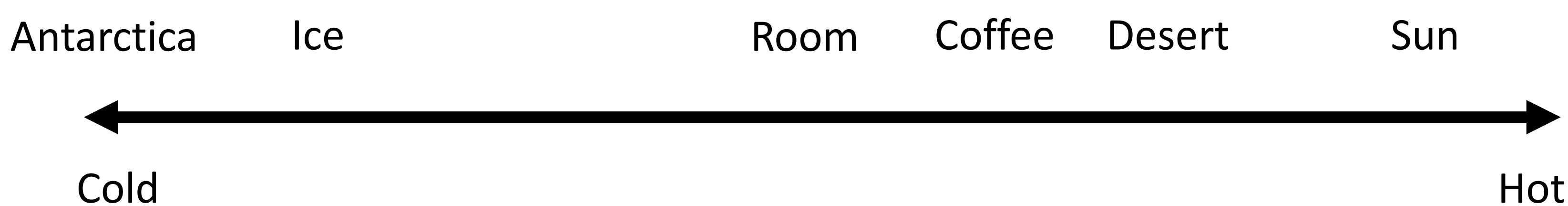
Agenda

1. Introduction to Vector Databases
2. Core Concepts and Architecture
3. Retrieval Techniques Overview
4. Categorization of Retrieval Methods
5. Benchmarking Vector Databases
6. Key Considerations for Selection
7. Current Challenges and Future Directions
8. Q&A

You Could Have Invented VectorDBs



You Could Have Invented VectorDBs



What are Vector Databases?

- **Definition**: Specialized databases designed to store, index, and query high-dimensional vector embeddings
- **Purpose**: Enable efficient similarity search and retrieval of unstructured data
- **Key Insight**: Transform semantic similarity into geometric proximity in high-dimensional space

Why Vector Databases Matter

- Traditional databases struggle with unstructured data (text, images, audio)
- ML models generate embeddings that capture semantic meaning
- Need for fast, scalable similarity search at production scale

The Semantic Gap Problem

Traditional Information Retrieval Limitations

- Keyword matching cannot capture semantic meaning
- Car =/= Automobile in lexical search
- Query-document mismatch even when semantic similarity is high

Example Medical Literature Search

- Query: “heart attack symptoms”
- Relevant document: “myocardial infarction presentation”
- Traditional IR techniques: No lexical overlap -> Missed
- Vector Search: Semantic similarity -> found

Core Architecture Components

Vector Embeddings

- Dense numerical representations (typically 100-2000 dimensions)
- Generated by ML models (transformers, CNNs, etc.)
- Capture semantic relationships in continuous space
- $(\text{uncle} - \text{man}) + \text{woman} \approx \text{aunt}$

Indexing Structures

- **Flat Index**: Brute force, exact search
- **Tree-based**: KD-trees, Ball trees
- **Hash-based**: LSH (Locality Sensitive Hashing)
- **Graph-based**: HNSW, NSG
- **Quantization-based**: PQ, OPQ, ScaNN

Retrieval Techniques: Overview

Exact vs. Approximate Search

Exact Search (KNN)

- Guarantees finding true nearest neighbors
- Computationally expensive: O(nd) complexity
- Suitable for small datasets or offline processing

Approximate Nearest Neighbor (ANN)

- Trade-off accuracy for speed
- Sublinear time complexity
- Production-ready for large-scale applications

Theoretical Foundations of Approximate Search

Accuracy-Speed Trade-Off

Exact Search Problem: Given query q and dataset $X \in \{x_1, \ldots, x_n\}$ find k points with smallest $d(\mathrm{q}, x_i)$

Computational Complexity

- Brute force: $\mathcal{O}(nd)$
- Best known lower bound: $\Omega(n)$ for general metric spaces
- High dimensions: All points appear equidistant (concentration of measure)

Approximate Nearest Neighbor (ANN): Trade accuracy for speed - enables sublinear algorithms: $\mathcal{O}(n^{\rho})$where $\rho < 1$

Common Approximation Types:

- Distance-based: Return points within $(1+\epsilon)$factor of optimal
- Probabilistic: Find true NN with probability p
- Recall-based: Return α fraction of true k nearest neighbors

The Curse of Dimensionality

Concentration of Measure Phenomenon

Key Result: in high dimensions, distances concentrate around their mean

For random vectors in $\frac{Var\left[||X||_2\right]}{E\left[||X||_2\right]^2} \to 0$ as $d \to \infty$

Implication

- Signal-to-noise ratio degrades: $\frac{d_{max}-d_{min}}{d_{min}} \to 0$
- All points appear nearly equidistant
- Traditional indexing structures (k-d trees) become ineffective

Distance Metrics

Common Similarity Measures

1. **Euclidean Distance (L2)**
   - $d = \sqrt{\sum(x_i - y_i)^2}$
   - Sensitive to magnitude
2. **Cosine Similarity**
   - $\cos(\theta) = (A \cdot B)/(|A||B|)$
   - Direction-focused, magnitude-invariant
3. **Dot Product**
   - $A \cdot B = \sum(a_i \times b_i)$
   - Fast computation, used in attention mechanisms
4. **Manhattan Distance (L1)**
   - $d = \sum|x_i - y_i|$
   - Robust to outliers

Retrieval Method Categories

1. Tree-Based Methods

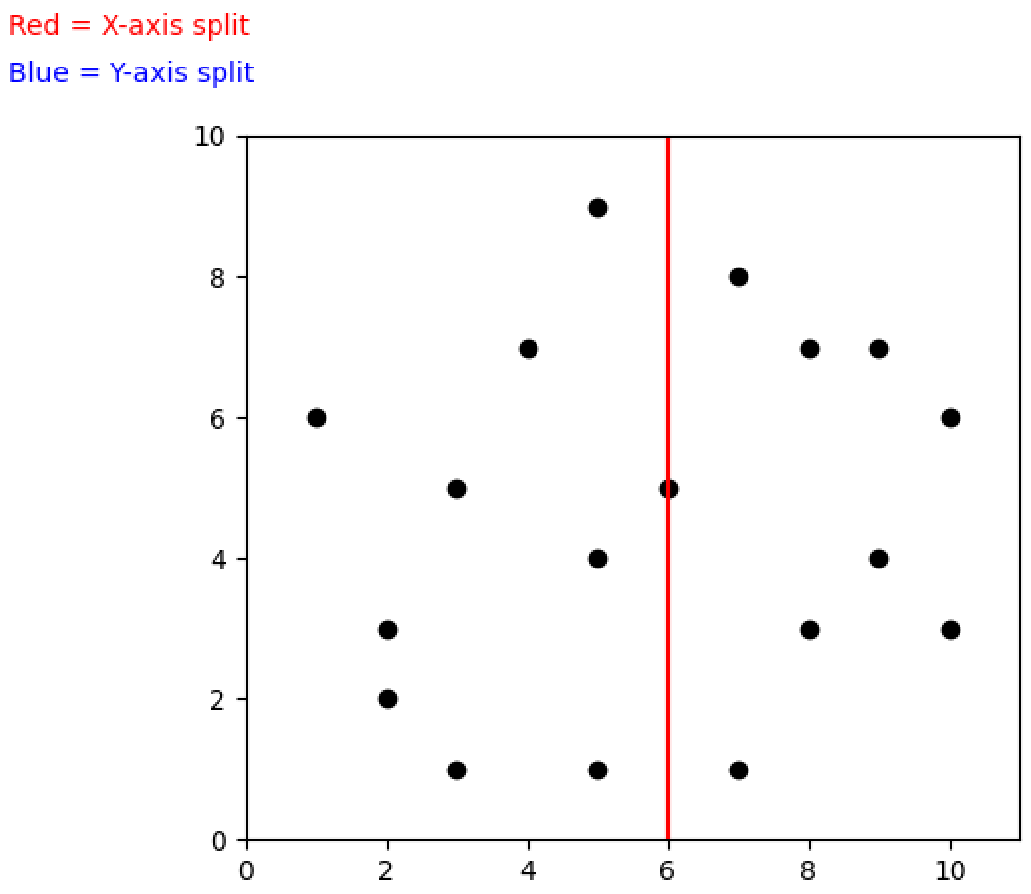
KD-Trees

- Binary space partitioning
- Effective in low dimensions (< 20)
- Degrades with dimensionality (curse of dimensionality)
- Rarely used in modern embeddings, exact recall

Ball Trees

- Hierarchical clustering approach
- Better for high-dimensional data than KD-trees
- $\mathrm{O}(\log n)$ average case complexity

k-d Trees



Source: https://medium.com/@notesbymuneeb/k-d-trees-for-nearest-neighbor-search-df4fc459da51

Retrieval Method Categories

Ball Trees
Structure:
- Recursive binary tree over nested hyperspheres ("balls")
- Each node covers a subset of points with a center and radius

Querying:
- Uses triangle inequality to prune subtrees
- Efficient for exact k-NN and radius queries

Complexity:
- Build time: O(n log n)
- Avg. query time**:** O(log n)
- Worst-case: O(n) in high dimensions

Trade-offs:
- Handles higher dimensions better than KD-Trees
- Still degrades with dimensionality
- Exact recall, but slower than hashing for large-scale search

Retrieval Method Categories

2. Hash-Based Methods

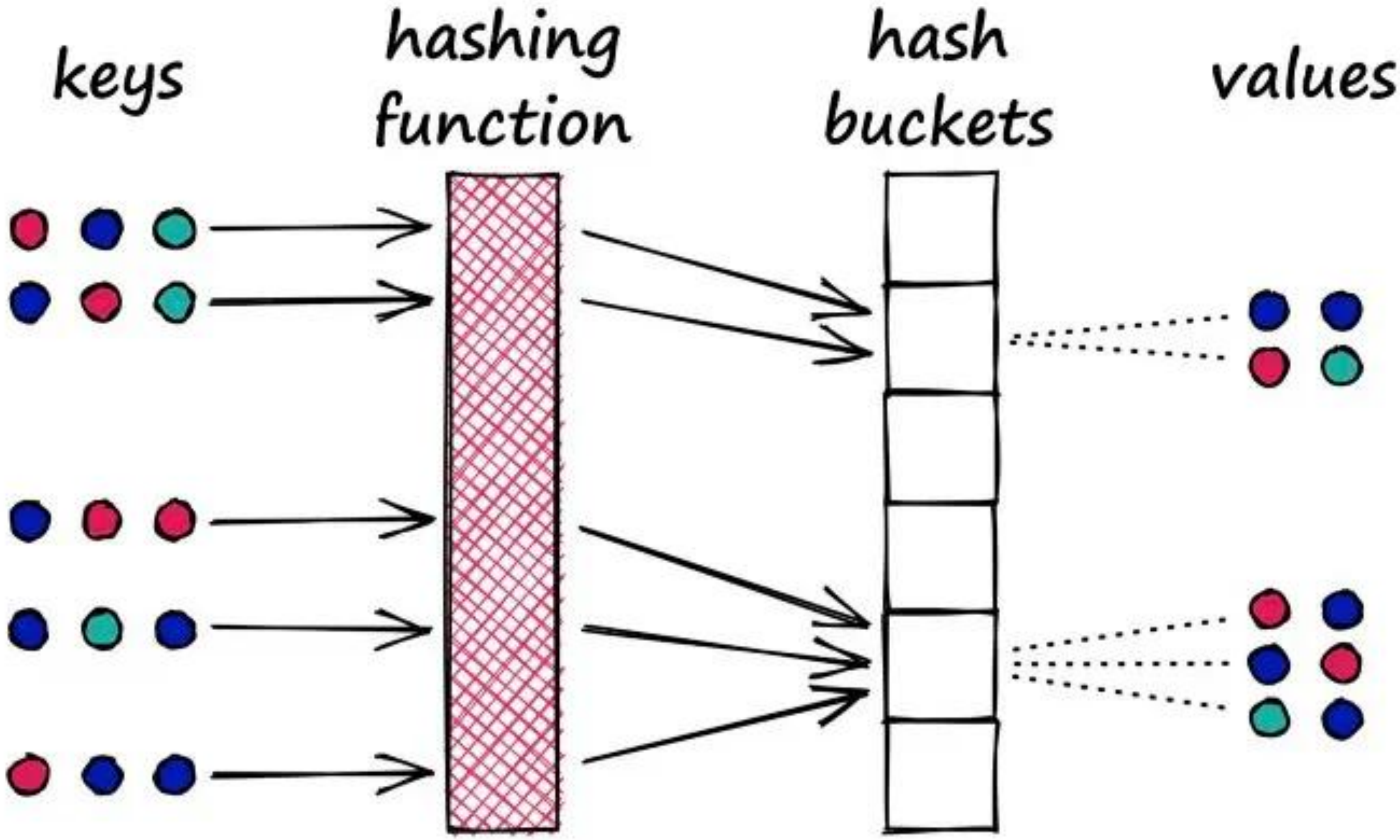
Locality Sensitive Hashing (LSH)

- Maps similar vectors to same hash buckets
- Very fast, low memory, acceptable recall
- Works better in high dimensional spaces
- Probabilistic guarantees
- Examples: Random projections, MinHash
- Trade-Off: More hash functions = better recall but slower queries

Learning to Hash

- Deep learning approaches
- Learn optimal hash functions
- Examples: Deep Supervised Hashing (DSH)

LSH



Retrieval Method Categories

LSH Mathematical Foundations

LSH Family Definition: Family H is (r, cr, p_1, p_2)-sensitive if:

- $d(x, y) \leq r \rightarrow \Pr[h(x) = h(y)] \geq p_1$
- $d(x, y) \geq cr \rightarrow \Pr[h(x) = h(y)] \leq p_2$

Query Complexity:

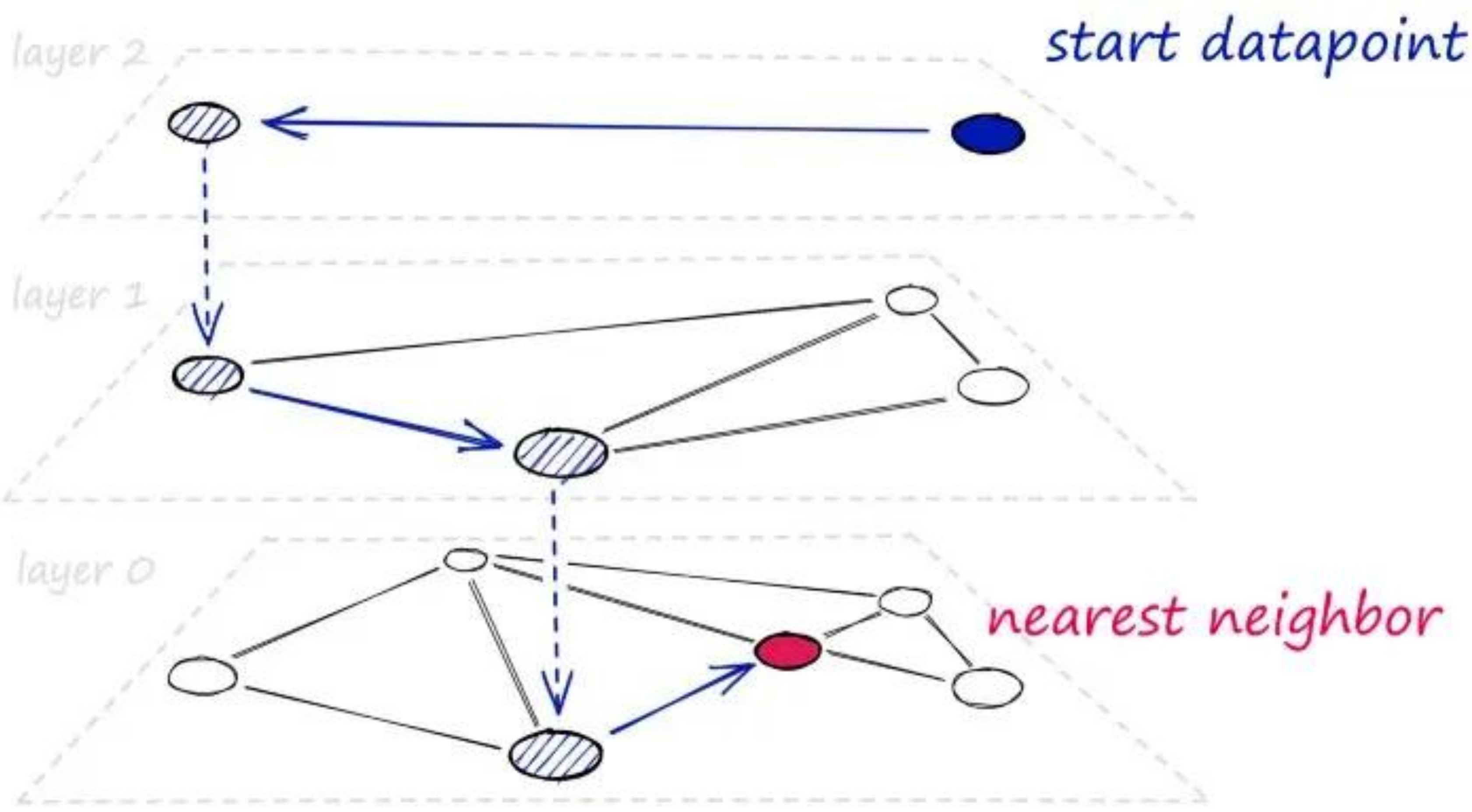
- L hash tables, K functions each
- Expected query time: $O(L \cdot n^{\rho})$ where $\rho = \ln(1/p_1) \,/\, \ln(1/p_2) < 1$
- Space: $O(LKn)$

Retrieval Method Categories

3. Graph-Based Methods
Navigable Small World (NSW)
- Single-layer graph
- Inspired by small-world networks
- Foundation for HNSW

Hierarchical Navigable Small World (HNSW)
- Multi-layer graph structure
- Greedy search with skip connections
- State-of-the-art performance/speed trade-off
- O(n) space

HNSW



Navigable Small World Networks

Kleinberg's Model (2000):

- Grid graph + long-range connections
- Connection probability $\propto d(u, v)^{(-\alpha)}$
- Optimal α = d (dimension) for greedy routing

HNSW Insight: Multi-layer graph mimics hierarchical structure

- Layer 0: All points (NSW graph)
- Layer i: Exponentially decreasing density
- Search: Start at top, navigate down

Theoretical Guarantee: Expected search complexity: O(log n) with high probability

Retrieval Method Categories

4. Learned Indices

Deep Learning Approaches

- Learn optimal indexing structures
- Examples: Learned LSH, Neural Information Retrieval
- Promising but computationally intensive

Hybrid Methods

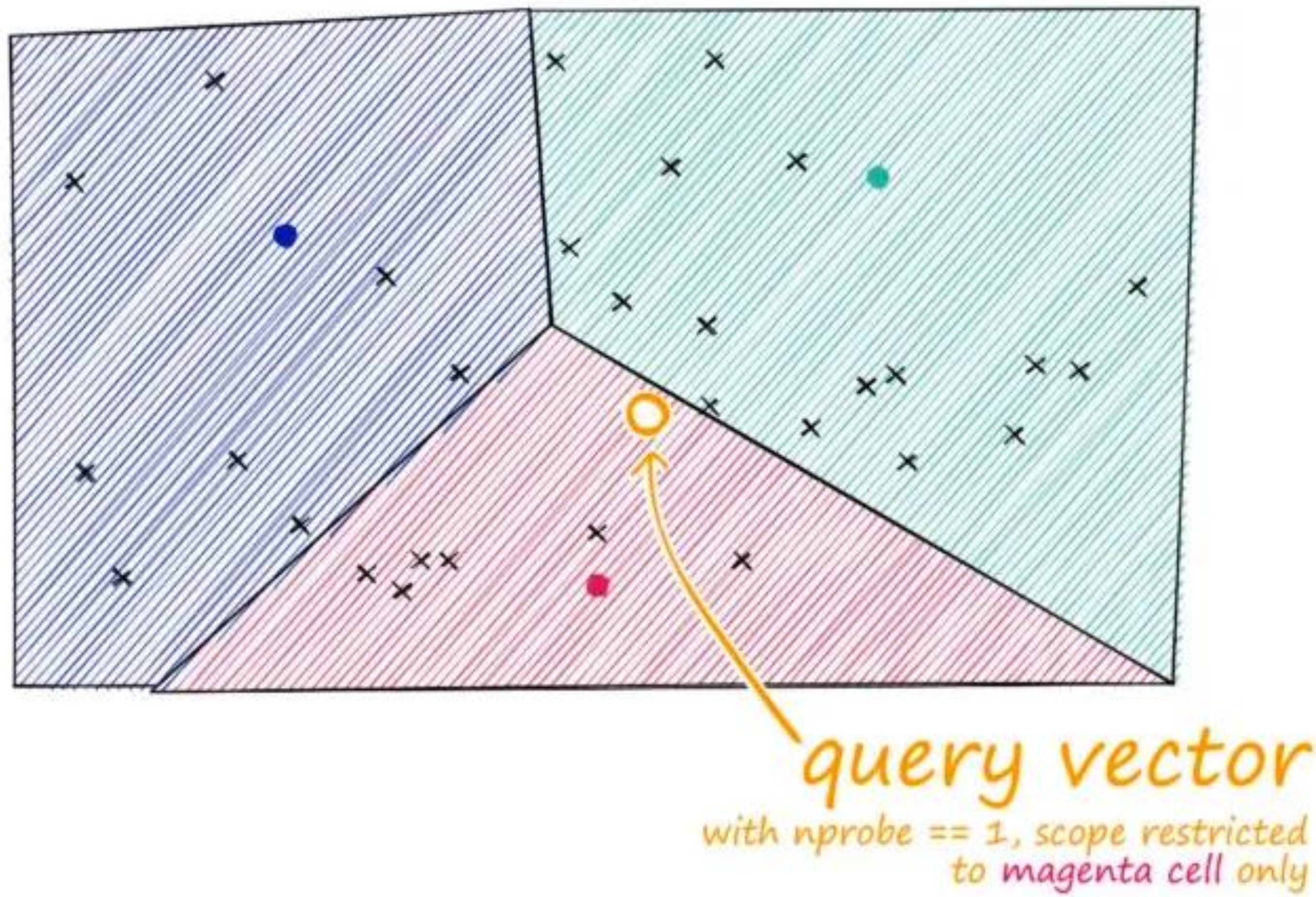
- Combine traditional indexing with ML
- Learn query-specific optimizations
- Balance between performance and complexity

Retrieval Method Categories

5. Inverted File (IVF) Methods

- Partitions data into clusters or “cells”
- Uses a coarse quantizer (e.g., k-means) to assign vectors to clusters
- At query time, searches only a subset of clusters near the query
- Reduces search space dramatically, speeding up retrieval
- Often combined with Product Quantization (PQ) for efficient compression

Inverted File (IVF) Methods



Performance on Sift1M (d=128, k=10)

Index	Memory (MB)	Query Time (ms)	Recall	Notes
Flat (L2 or IP)	~500	~18	1.0	Good for small datasets or where query time is irrelevant
LSH	20 - 600	1.7 - 30	0.4 - 0.85	Best for low dimensional data, or small datasets
HNSW	600 - 1600	0.6 - 2.1	0.5 - 0.95	Very good for quality, high speed, but large memory usage
IVF	~520	1 - 9	0.7 - 0.95	Good scalable option. High-quality, at reasonable speed and memory usage

Hardware used
- M1 chip with 8-core CPU
- 8GB unified memory

https://www.pinecone.io/learn/series/faiss/vector-indexes/

Retrieval Methods Decision Tree

- **Small + Low-dim + Exact**: Flat, Trees
- **Large + Memory constraints**: LSH or IVF
- **Production + High performance**: HNSW
- **Massive scale + Memory efficient**: IVF + Quantization
- **Research + Optimization**: Learned Indices

Vector Database Systems

Open-Source Options

- **Faiss** (Facebook): High-performance similarity search library
- **Annoy** (Spotify): Approximate nearest neighbors, memory-mapped
- **Hnswlib**: Fast HNSW implementation
- **Milvus**: Distributed vector database
- **Weaviate**: GraphQL-based vector search engine

Commercial Solutions

- **Pinecone**: Managed vector database service
- **Vespa**: Distributed search and storage
- **Qdrant**: Neural search engine
- **Chroma**: AI-native embedding database

Benchmarking Vector Databases

Key Performance Metrics
Recall@K
- Percentage of true neighbors found in top-K results
- Most important accuracy metric
- Usually measured at K=1, 10, 100

Queries Per Second (QPS)
- Throughput under concurrent load
- Measured at different recall levels

Latency Percentiles
- P50, P95, P99 response times
- Important for user-facing applications

Memory Usage
- Index size vs. dataset size ratio
- Critical for large-scale deployments

Standard Benchmarking Datasets

Research Datasets

- **SIFT1M/SIFT1B**: 128-dimensional SIFT descriptors
- **GIST1M**: 960-dimensional GIST descriptors
- **GloVe**: Word embeddings (25-300 dimensions)
- **Deep1B**: 1 billion deep neural network features

Domain-Specific Benchmarks

- **MS MARCO**: Document retrieval
- **BEIR**: Information retrieval benchmark
- **MTEB**: Massive text embedding benchmark

Benchmark Considerations

Evaluation Methodology

- **Cold vs. Warm Queries**: Cache effects
- **Query Distribution**: Random vs. realistic query patterns
- **Dataset Characteristics**: Dimensionality, clustering, outliers

Hardware Dependencies

- **CPU vs. GPU**: Different algorithms perform differently
- **Memory Hierarchy**: L1/L2 cache, RAM, storage
- **SIMD Instructions**: Vectorized operations impact

Selection Considerations

Dataset Characteristics

- **Size**: Millions vs. billions of vectors
- **Dimensionality**: Low (<100) vs. high (>1000) dimensions
- **Update Frequency**: Static vs. dynamic datasets
- **Distribution**: Clustered vs. uniform data

Query Patterns

- **Batch vs. Real-time**: Different optimization strategies
- **Query Volume**: QPS requirements
- **Accuracy Requirements**: Exact vs. approximate tolerance

Selection Considerations (cont.)

Operational Requirements
Scalability - Horizontal vs. vertical scaling capabilities
- Distributed processing support
- Load balancing strategies

Consistency
- ACID properties for updates
- Eventual vs. strong consistency
- Multi-version concurrency control

Integration
- API compatibility (REST, gRPC)
- Language bindings
- Ecosystem integration (Kafka, Spark, etc.)

Performance Optimization Strategies

Index Tuning

- **Parameter Selection**: M, ef_construction in HNSW
- **Memory vs. Accuracy Trade-offs**: Choose appropriate index type
- **Preprocessing**: Dimensionality reduction, normalization

Query Optimization

- **Batching**: Process multiple queries together
- **Caching**: Cache frequent queries and results
- **Filtering**: Pre-filter with metadata before vector search

Hardware Optimization

- **SIMD Utilization**: Vectorized distance calculations
- **GPU Acceleration**: Parallel processing for large batches
- **Memory Layout**: Optimize for cache locality

Current Challenges

Technical Challenges

- **Curse of Dimensionality**: Performance degradation in high dimensions
- **Dynamic Updates**: Maintaining index quality with insertions/deletions
- **Multi-modal Retrieval**: Combining different data types effectively

Operational Challenges

- **Cost Management**: Balance between performance and infrastructure costs
- **Monitoring**: Understanding query patterns and system health
- **Version Management**: Handling model updates and embedding changes

Emerging Trends

Advanced Retrieval Techniques

- **Dense-Sparse Hybrid**: Combining dense embeddings with sparse features
- **Multi-Vector Retrieval**: Representing documents with multiple embeddings

Integration Patterns

- **RAG (Retrieval-Augmented Generation)**: LLM + vector search
- **Multi-modal Search**: Text, image, audio in unified systems
- **Federated Search**: Distributed vector databases

Future Directions

Research Areas

- Learned Indexes: Using ML to optimize index structures and query routing

- Disk-based ANN: Scaling to trillion-vector datasets that don't fit in memory

- Real-time Learning: Dynamic embedding updates and adaptive indexing

Industry Evolution

- Standardization: Common APIs, benchmarks, and interchange formats

- Hybrid Search: Combining vector similarity with traditional filters and ranking

- Edge Deployment: Optimized models and indexes for mobile/IoT devices

Key Takeaways

1. **No One-Size-Fits-All**: Choose based on specific requirements
2. **Trade-offs are Fundamental**: Speed vs. accuracy vs. memory
3. **Benchmarking is Critical**: Use appropriate datasets and metrics
4. **Consider Total Cost of Ownership**: Not just query performance
5. **Stay Updated**: Rapidly evolving field with new techniques

Recommended Approach

1. Start with simple baseline (Faiss flat index)
2. Identify bottlenecks and requirements
3. Experiment with different methods
4. Validate with realistic workloads
5. Monitor and iterate

Thank You

Additional Resources:
https://arxiv.org/html/2310.11703v2

https://www.pinecone.io/learn/series/faiss/vector-indexes/