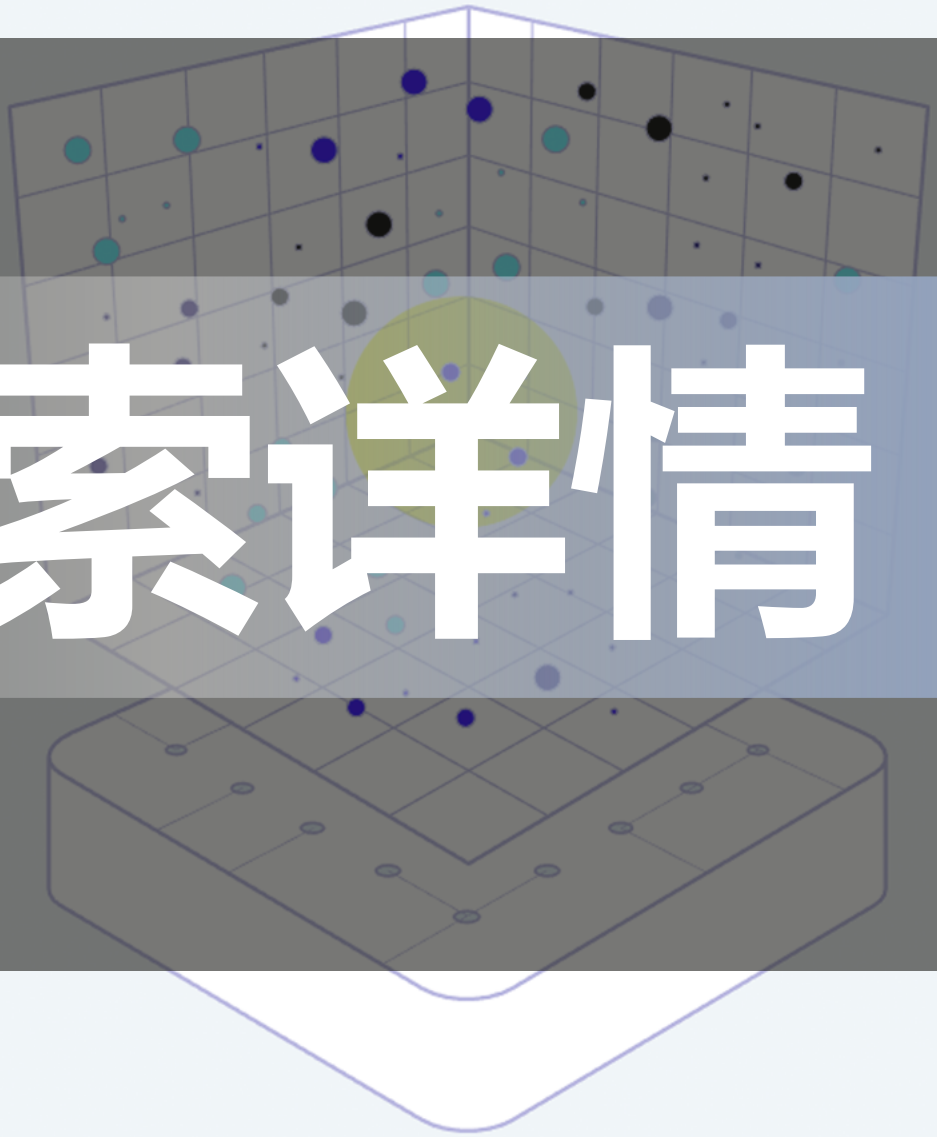
大模型系列 – 数据处理

相似性搜索详情



大模型业务全流程

数据 & 模型算法

2. 数据处理

- 开源数据
- 数据预处理
- 向量数据库

3. 模型算法

- LLM模型架构
- 多模态一切皆Tokens

模型训练 & 微调

4. 模型训练

- 混合精度
- 梯度检查
- 梯度累积
- …
- 5. 分布式并行
- 训练集群稳定性

6. 模型微调

- 全参微调
- 低参微调
- 指令微调

模型验证 & 推理部署

7. 模型验证

- 下游任务
- 测评标准
- bench mark

8. 推理与智能体

- 量化压缩
- 推理加速
- 9. Agent 智能体

集群算力准备

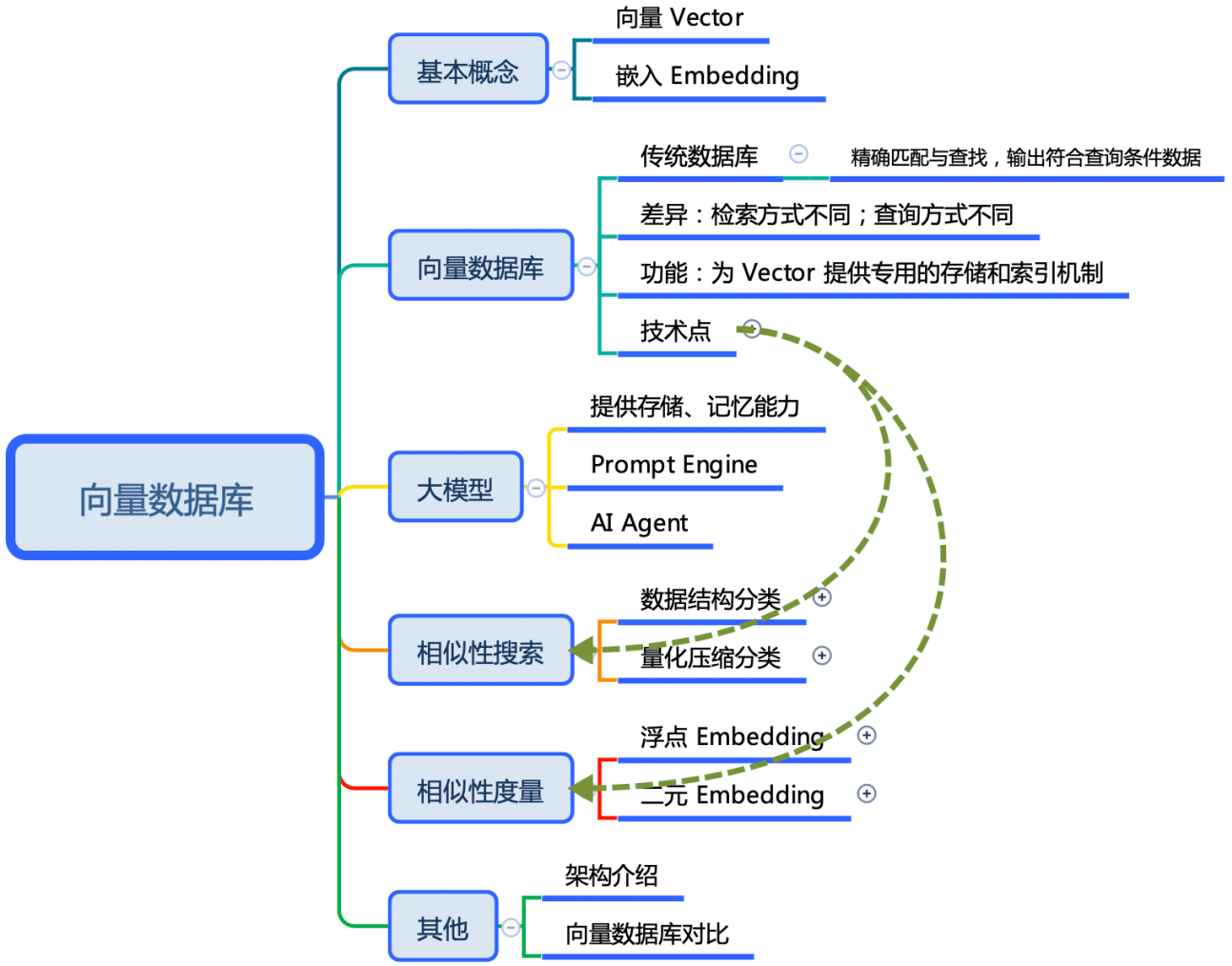
1. AI 集群建设

- 计算、存储、网络
- AI 集群机房建设
- AI 集群上线与运维

大模型系列 – 数据处理之向量数据库

- 具体内容
  - **向量与检索：**向量 Vector 的表示 -- Embedding 原理
  - **向量数据库：**向量数据库原理、功能、特点 -- Vector-DB 应用场景
  - **大模型关系：**向量数据库遇到大模型 – 大模型与 Vector-DB 应用场景
  - **相似性搜索：**K-Means 聚类 -- Faiss 算法 -- PQ 算法 -- IVF 算法 -- HNSW 算法
  - **相似性度量：**欧氏距离（L2） -- 内积（IP） -- 其他度量方式
  - **通用性架构：**通用 Vector-DB 架构 -- KDB 架构示例
  - **对比与小结：**业界向量数据库横向对比 -- Vector-DB 小结

大模型系列 – 数据处理之向量数据库



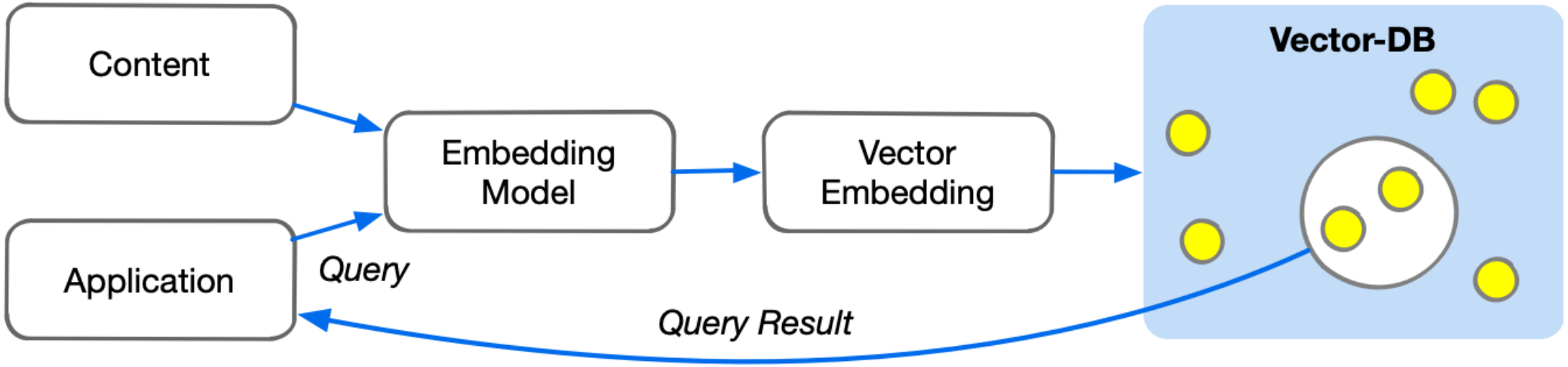
3. 相似性搜索算法

原则

1. 除暴力搜索能精确索引到近邻，所有搜索算法只能在**性能、召回率、内存**三者进行权衡。

向量检索算法

- 向量数据库使用近似最相邻（Approximate Nearest Neighbor，ANN），评估相似向量间相似度。
- 为了解决逐个索引向量相比，效率低问题。出现了 IVF、HNSW、LSH 等算法。

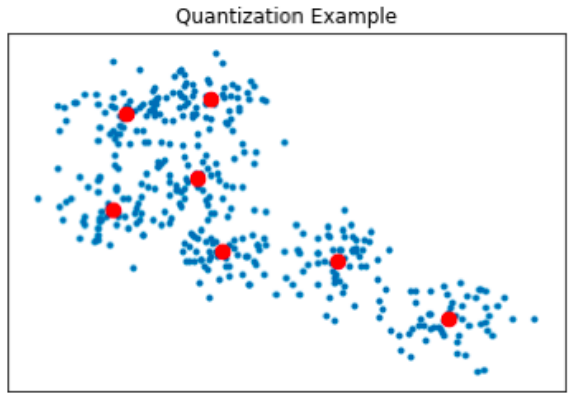


A. Similarly Search

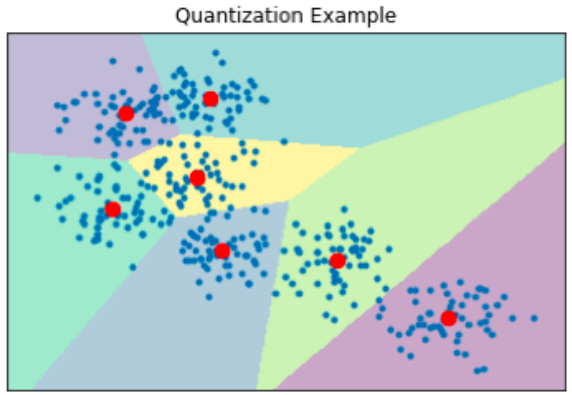
ANN 近似最相邻

ANN

- **先对 DB 向量进行聚类：**
  - e.g. 下图坐标系划定 4 个聚类中心
  - 将每个向量分配到最近聚类中心
  - 聚类算法调整聚类中心位置
  - 最终将向量分成 4 个
- **再进行向量索引：**
  - 先判断搜索向量属于哪个簇
  - 在该聚类簇中进行搜索
- 从 4 个簇减少到了 1 个簇，减少搜索范围



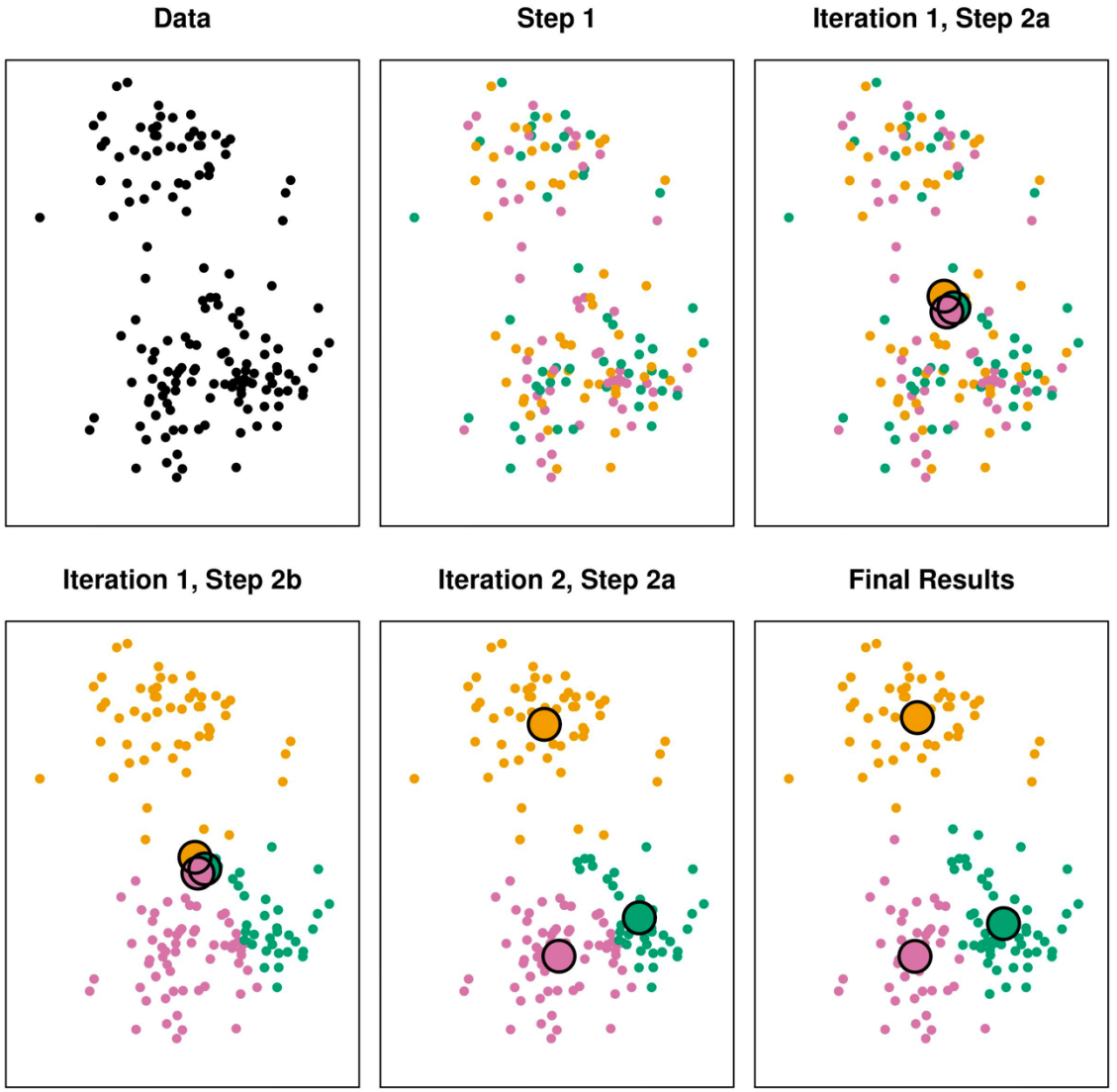
quantizer = KMeans(n_clusters=7).fit(X)



Partition vector space according to closest centroid

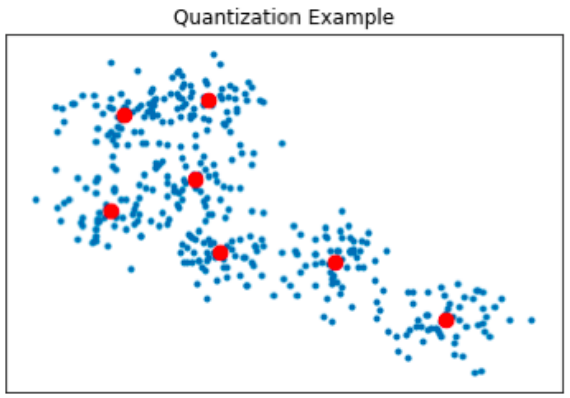
ANN: K-Means Algorithm

- **K-Means 通过迭代搜索算法，将数据分成 K 个类别。下面为其算法步骤：**
  1. 选择 k 个初始聚类中心；
  2. 将每个数据点分配到最近聚类中心；
  3. 计算每个聚类的新中心；
  4. 重复 2&3 至聚类中心不变或达最大迭代。

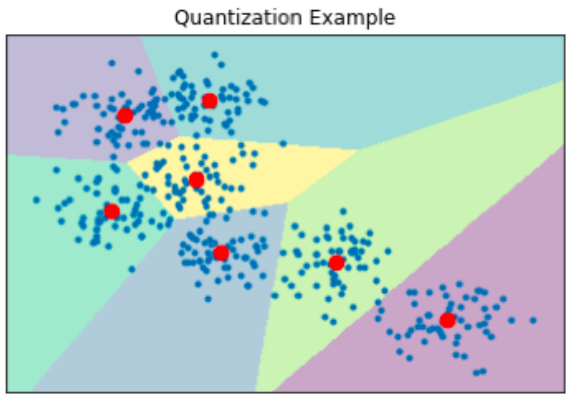


ANN: K-Means Algorithm Problem

- 现实情况，向量分布区域边界相邻。K-Means 搜索缺点，搜索向量处于两个聚类区域的中间，容易遗漏掉部分相似向量。



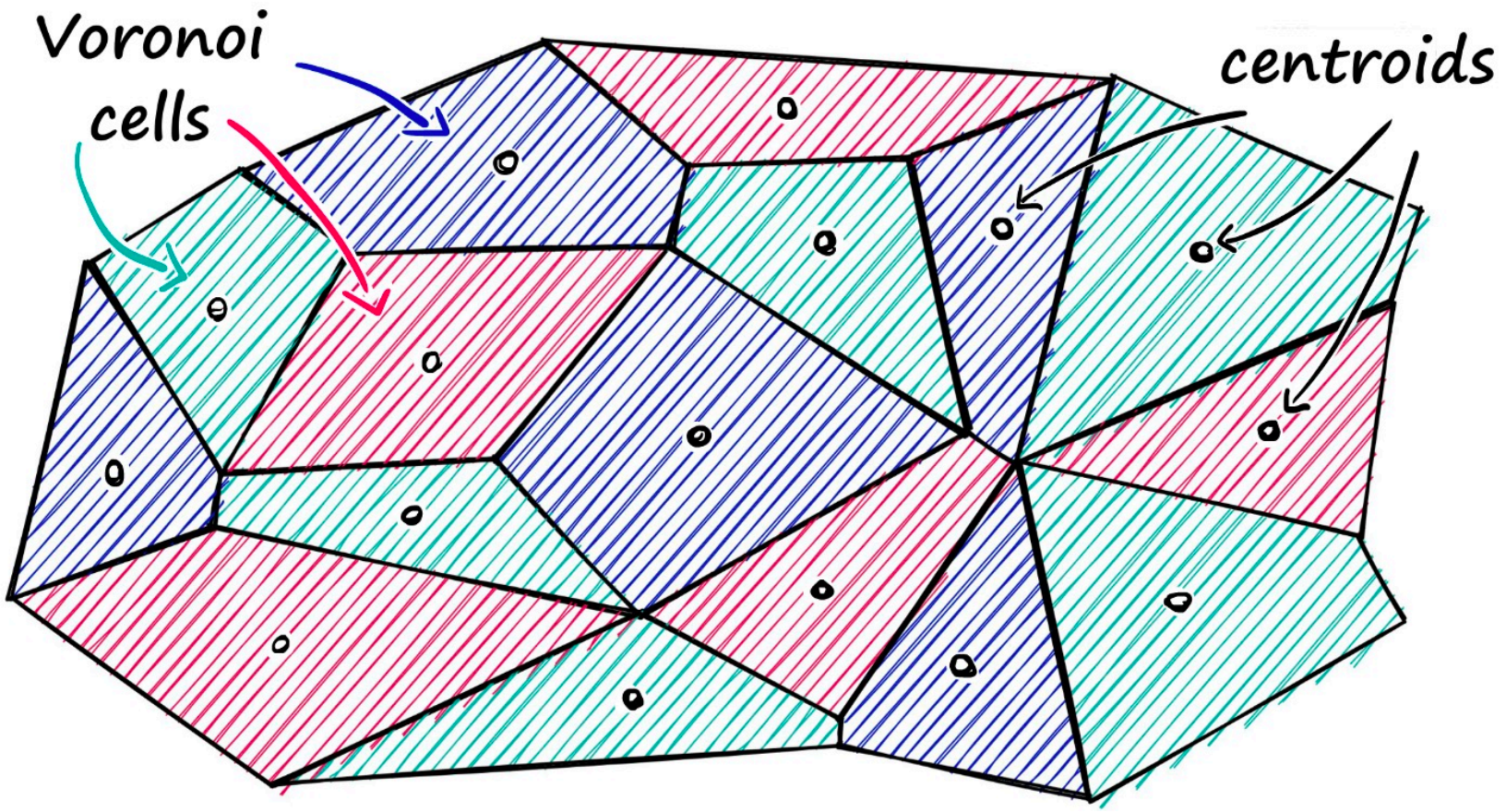
quantizer = KMeans(n_clusters=7).fit(X)



Partition vector space according to closest centroid

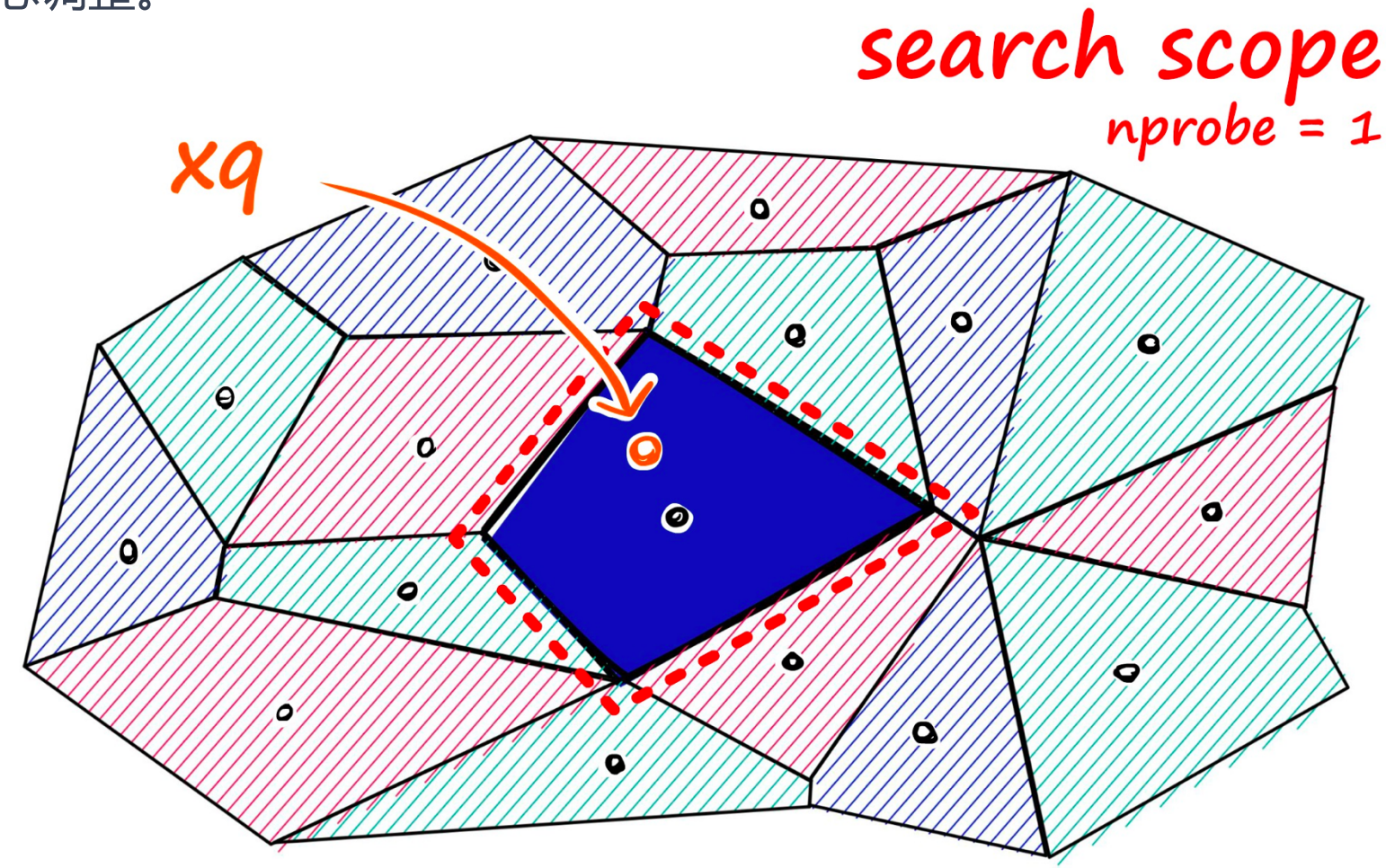
ANN: Faiss Algorithm

- 找到近似多个质心



ANN: Faiss Algorithm

- 搜索范围动态调整。

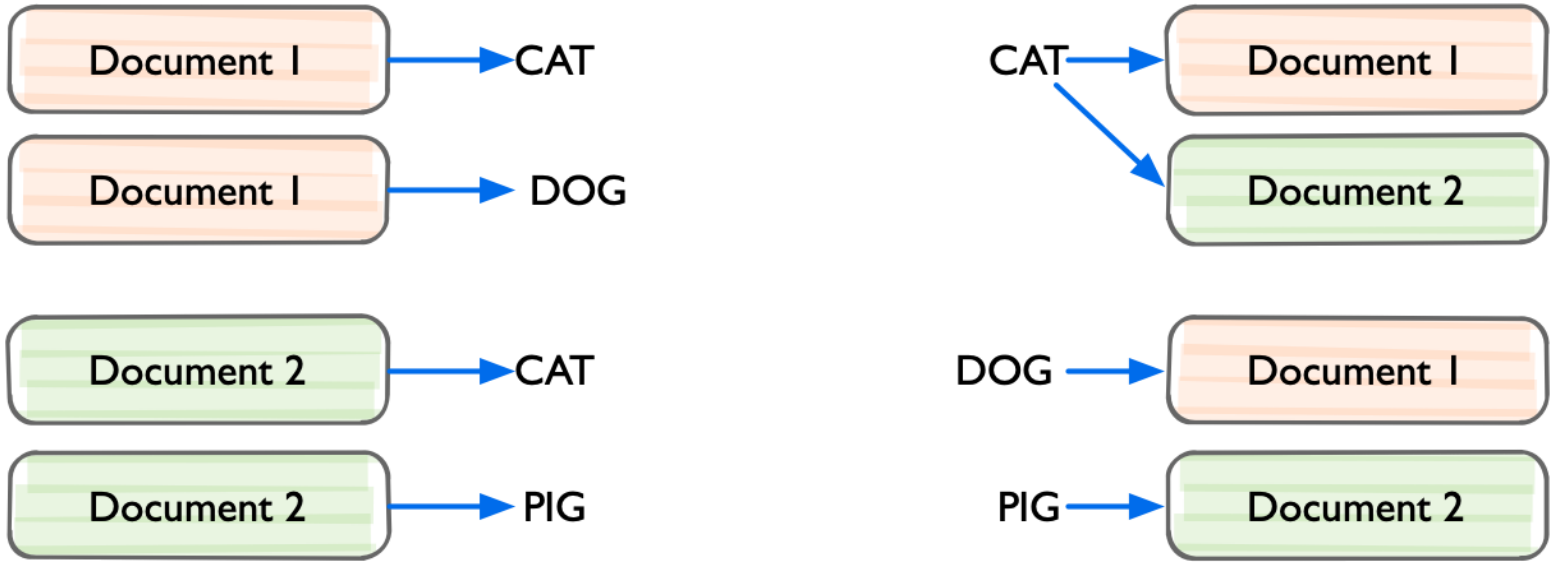


B. Similarly Search

IVF 倒排文件

倒排索引 IVF：全文检索常用技术

- 如下文档1中包含单词 Cat 和 Dog，文档 2 中包含了单词 Cat 和 Pig。
  1. 需要查询文档中包含 Cat 时, 只能线性扫描每一个文档进行判断。
  2. **倒排索引构建单词到文档的映射。**当查询 Cat 时，可快速定位到文档 1 和文档 2 。



倒排文件索引 IVF

- **倒排文件索引 IVF**：
  - 将向量数据集划分为 M 个子集（簇），每个簇都有一个聚类中心向量 a。通过构建倒排表，将聚类中心向量 a 与属于该簇其他向量进行关联。
  - 搜索时，先根据查询向量 q 找到与之最相似聚类中心向量 a，在该聚类中心 a 对应倒排表中查找更接近 q 的具体向量 p。

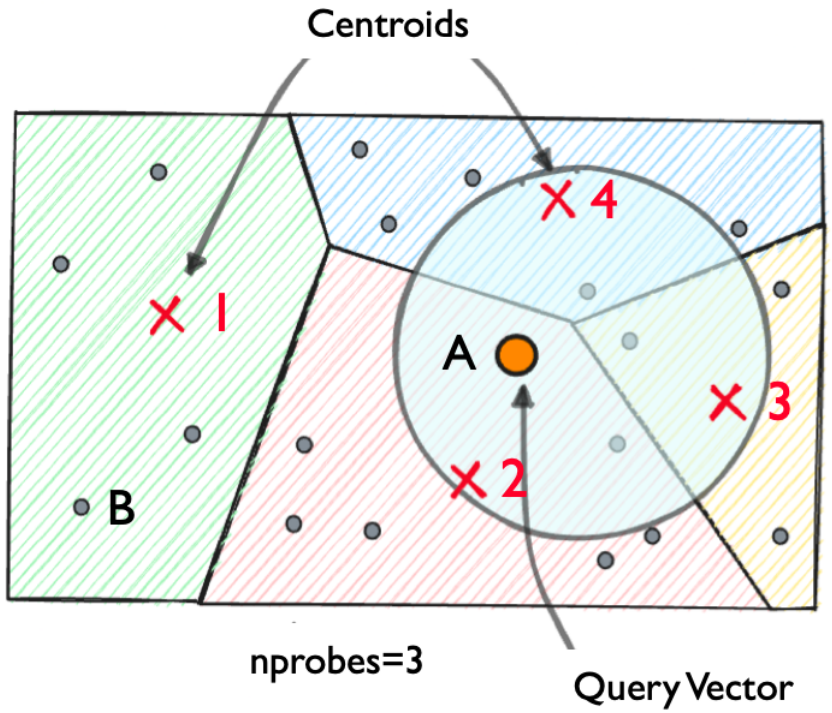
- 两级索引结构极大地减少搜索计算量，提高搜索效率。
- 需要具备一定数据后才开始训练（热启），DB 拥有一定量数据后需重建 IVF 索引表。

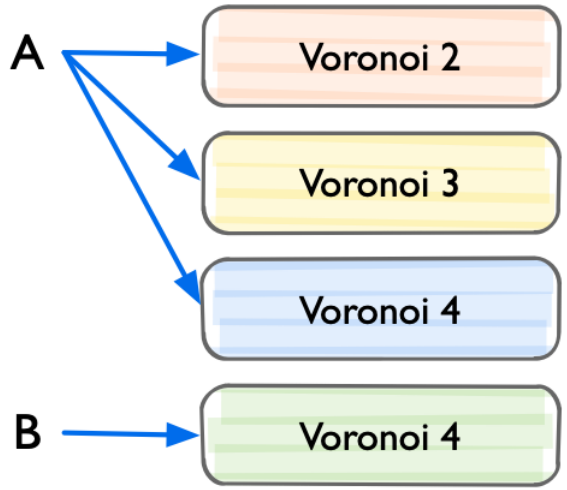
思考

1. 对单词很容易构建倒排，如何对向量构建倒排表？

倒排索引 IVF

- 空间中每个点，都有一个相应区域（Voronoi 单元），空间中距离源点 A 比其他点更近的所有点组成。用种子点 A 创建倒排索引，将每个质心 X 与空间中的向量列表相关联。



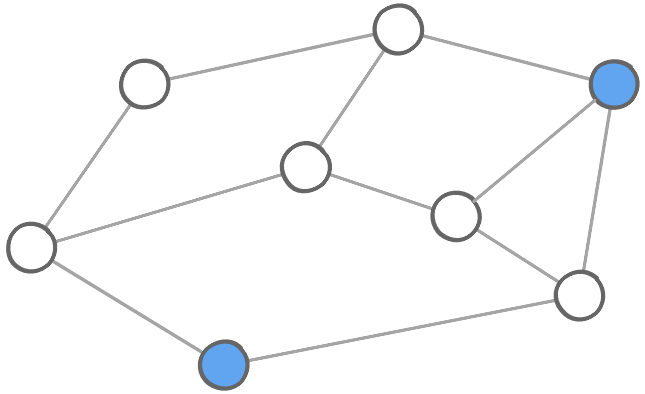


C. Similarly Search

HNSW

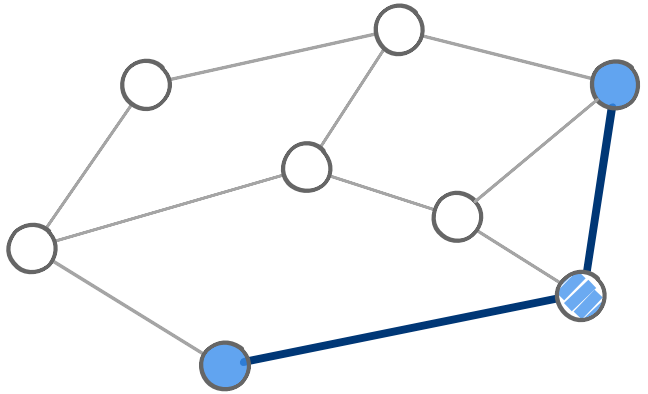
Graph 图算法原理

- **小世界理论**：基于小世界理论（六度空间），在一个充分连接的图，通过六跳可以连通两个点。



- **构图方式**：高维空间看作图，每个节点是一个向量，每条边是距离者相似度。

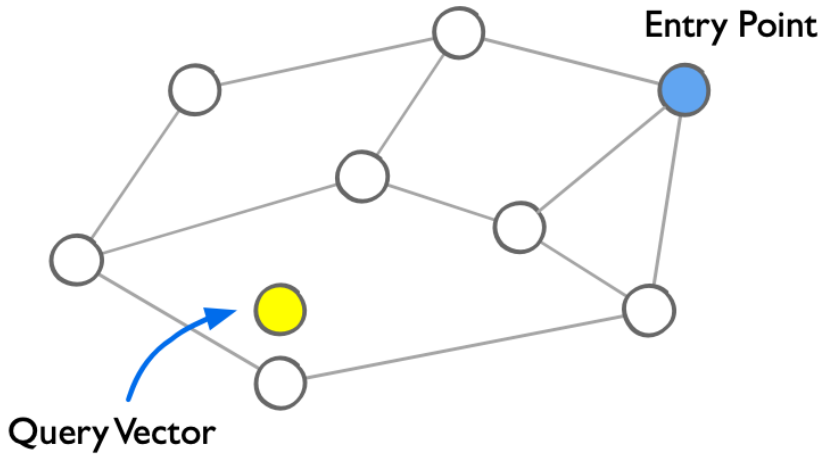
- **检索方式**：使用启发式算法构建和遍历图，从而找到最相似的向量。

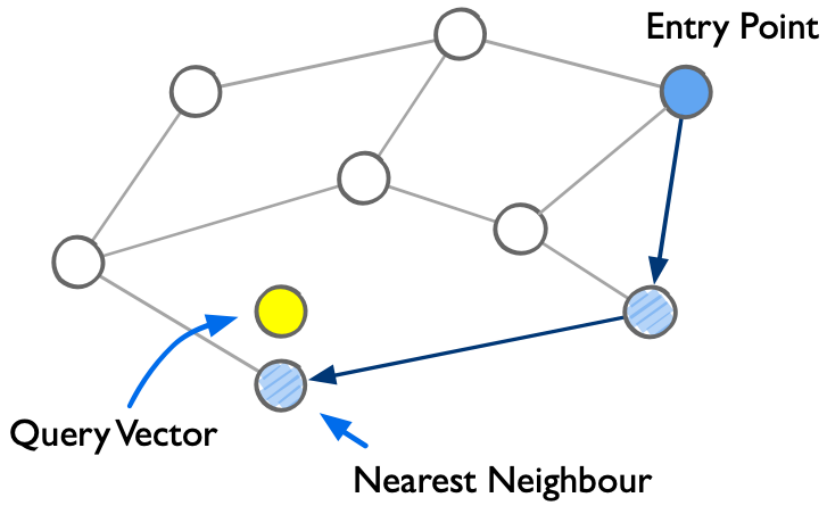


图算法原理

- **构图：**
  1. 逐个插入顶点
  2. 每次挑选 M 个近邻向量，构建链接

- **检索：**
  1. 随机挑选起始点
  2. 找到距离目标近的邻居，并移动
  3. 当所有邻居没有更近的距离，结束算法





图算法的问题

I. 数据增加 >>> 空间点越密集

- 1）构图速度变慢
- 2）跳转数增加

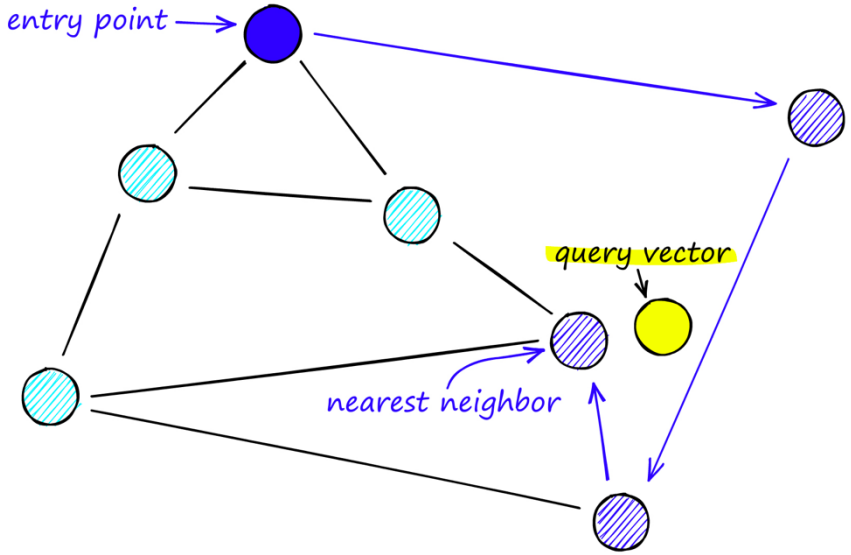
I. 图中所有点都以最短路径相连

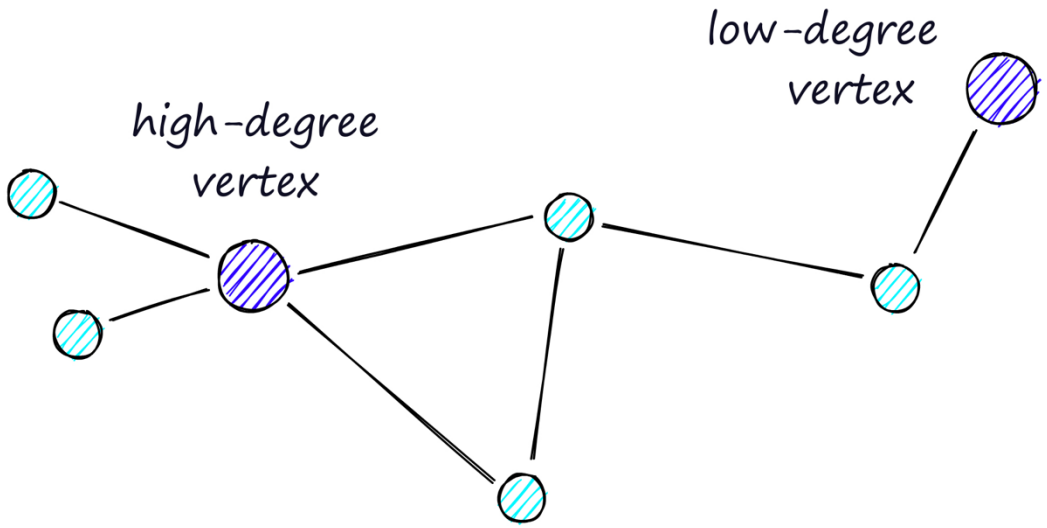
- 搜索时，需要遍历所有节点

NSW 原理

https://arxiv.org/abs/1603.09320

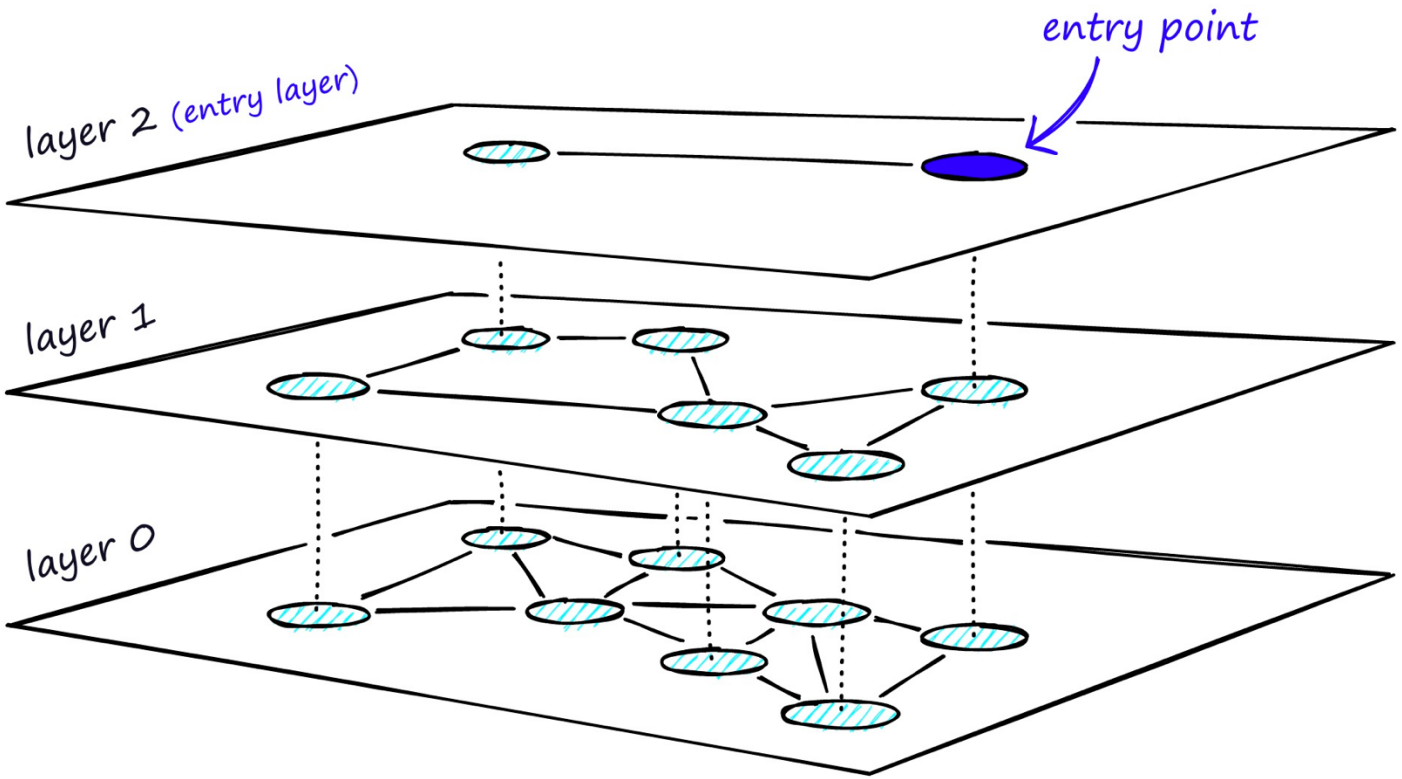
- **构建无向图，即给定任意入口点，可以到达图中的任何节点：**
  1. 首先形成长边（连接相距较远且需要多次遍历的节点），提高搜索*效率*
  2. 然后形成短边（连接附近的节点），提高搜索*精度*
  3. 通过遍历该图找到距给定查询向量最近的节点。





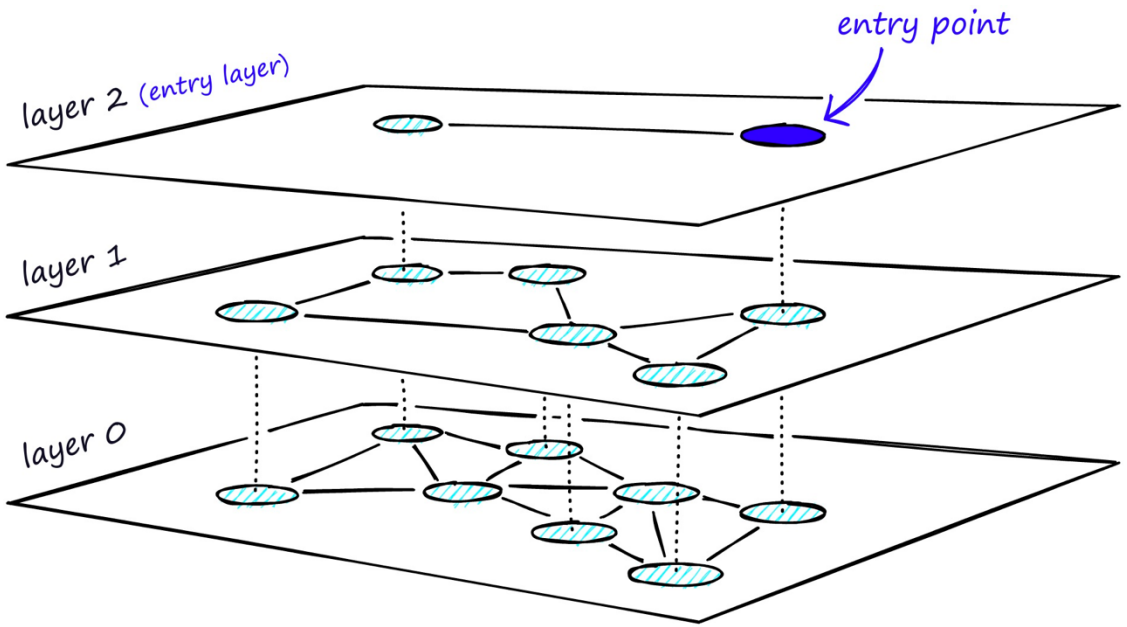
HNSW 构图原理

- HNSW 分层图结构解决遍历效率低，修复每个节点邻居数量上限，搜索复杂度降低 $Log(m)$。



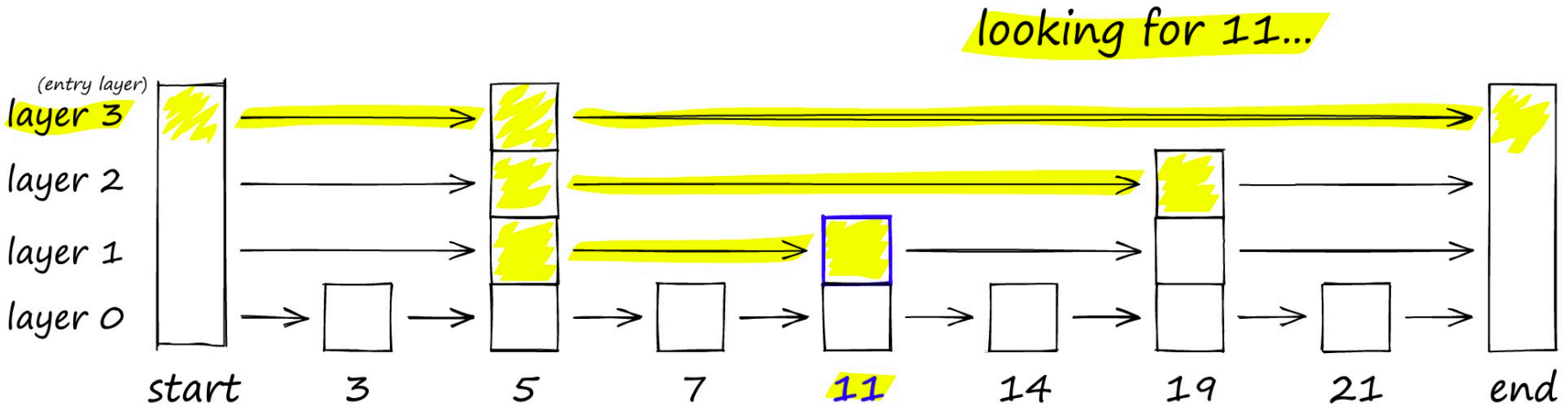
HNSW 构图原理

- **图结构**：将图分为多层，每一层作为一个SW（小世界），图中节点相互连接。每层节点都与上层节点相连。
- **构图**：按距离相似度将近邻分成不同层。长边在顶层，层数越低边越短，最底层构成完整图。所有层都相互折叠，则 HNSW 图本质为 NSW 图。



HNSW 检索原理

- **检索：**
  - 从最高层 l 开始沿最长边检索
  - 当前节点 n 值超出向量相似度，转移下一层图 $l-1$
  - 在 $l-1$ 层图回退到 l 层图 n 点沿最长边检索



HNSW 检索原理

- **检索：**
  - 从最高层 l 开始沿最长边检索
  - 当前节点 n 值超出向量相似度，转移下一层图 $l-1$
  - 在 $l-1$ 层图回退到 l 层图 n 点沿最长边检索

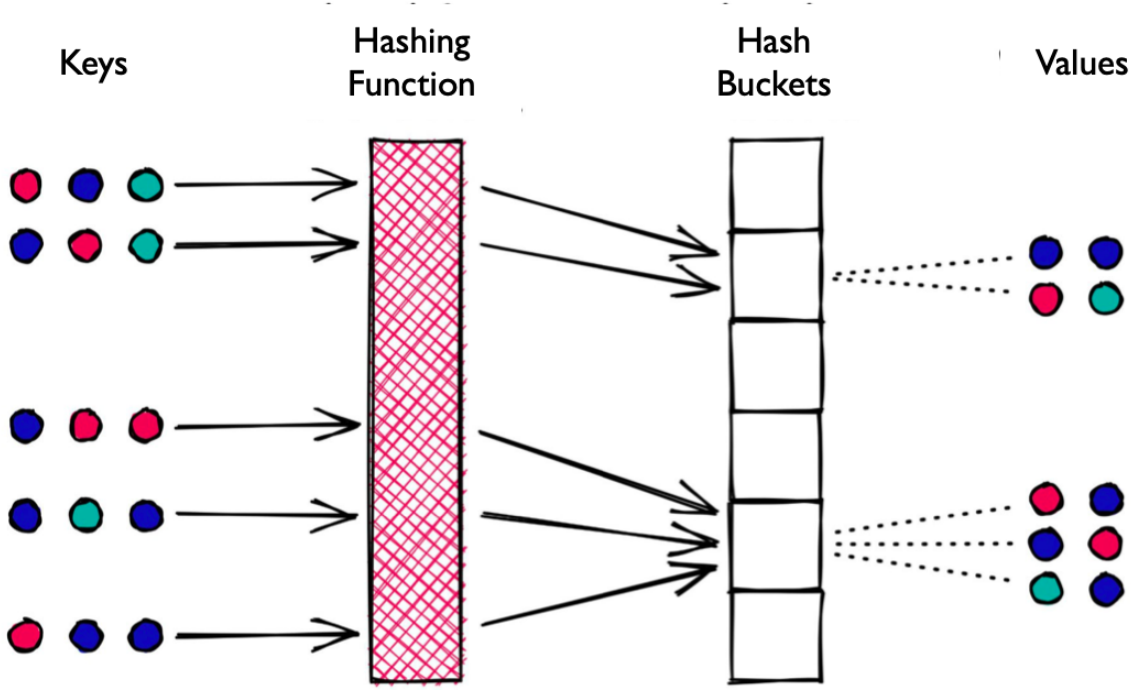
- 特点：空间换时间算法，搜索召回率和搜索速度较高，但内存开销较大。
- 内存：存储所有向量，还需维护多层图结构。

D. Similarly Search

LSH 局部敏感哈希

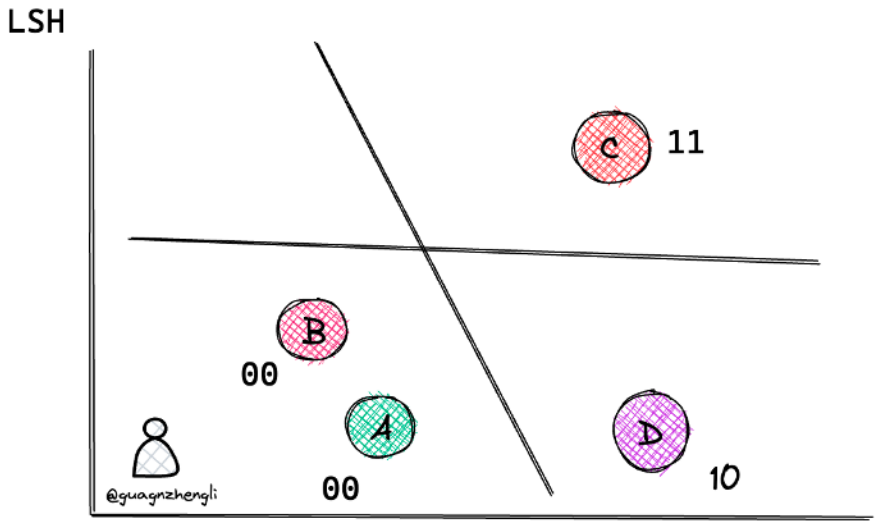
Locality Sensitive Hashing (LSH)

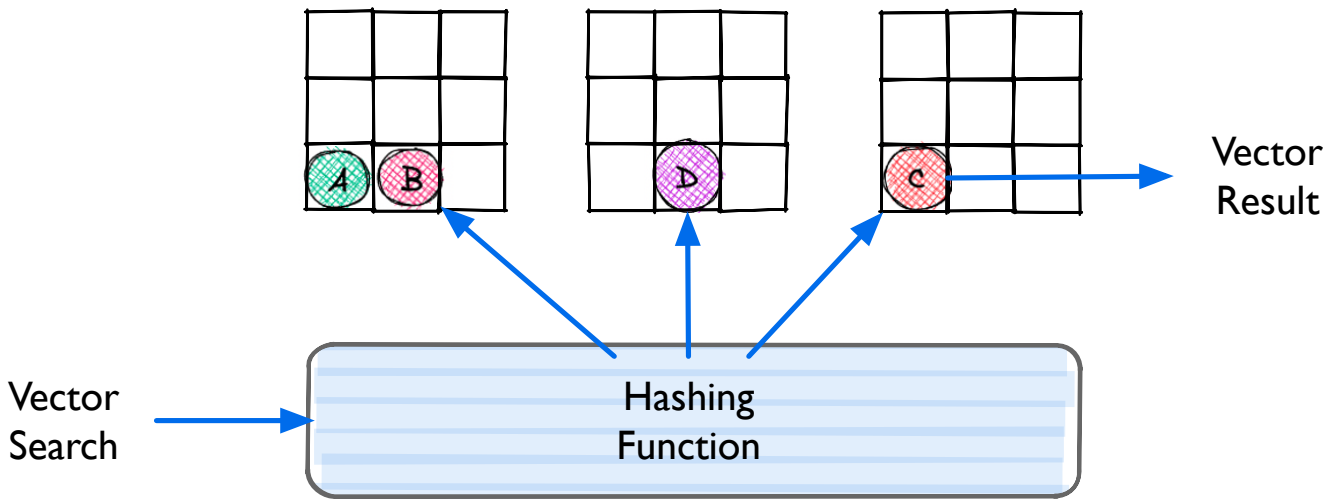
- **原理**：哈希函数将相似向量相同哈希值映射到桶中，通过比较哈希值判断向量间相似度。
- **哈希**：Hash 碰撞概率尽可能高，越相似的向量越容易碰撞，相似向量被映射到同一个桶。
- **特点**：性能高，每个哈希表桶向量远少于整个空间中向量数，并提供一个近似、非穷举结果。



哈希函数设计

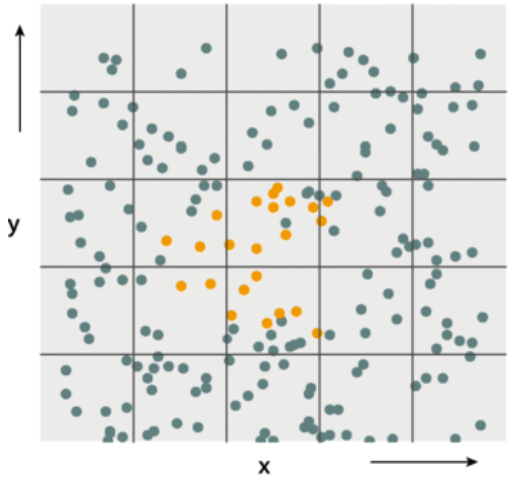
- 若两向量距离相近，则直线同一侧概率变高，如 AB 相似概率大于 AC。搜索向量时，将该向量进行哈希计算，得到相同桶中的向量，然后再通过其他索引方式，找到近邻。

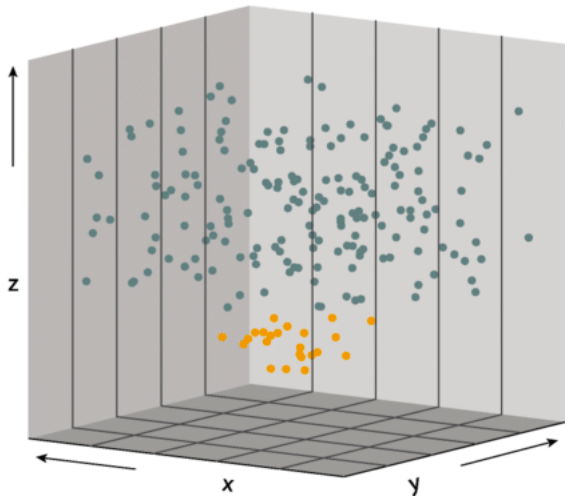




Random Projection for LSH 随机投影

- **高维空间**：二维坐标系通过直线区分相似性，三维坐标系可以通过平面，将三维坐标系划分为多个区域。高维空间可以通过超平面，将多维坐标系划分为多个区域，从而区分相似性。
- **高维问题**：高维空间数据点间距离稀疏，数据点间距离随维度增加指数级增长。实际实现会考虑随机投影，将高维空间数据点投影到低维，减少计算时间和提高查询召回率。



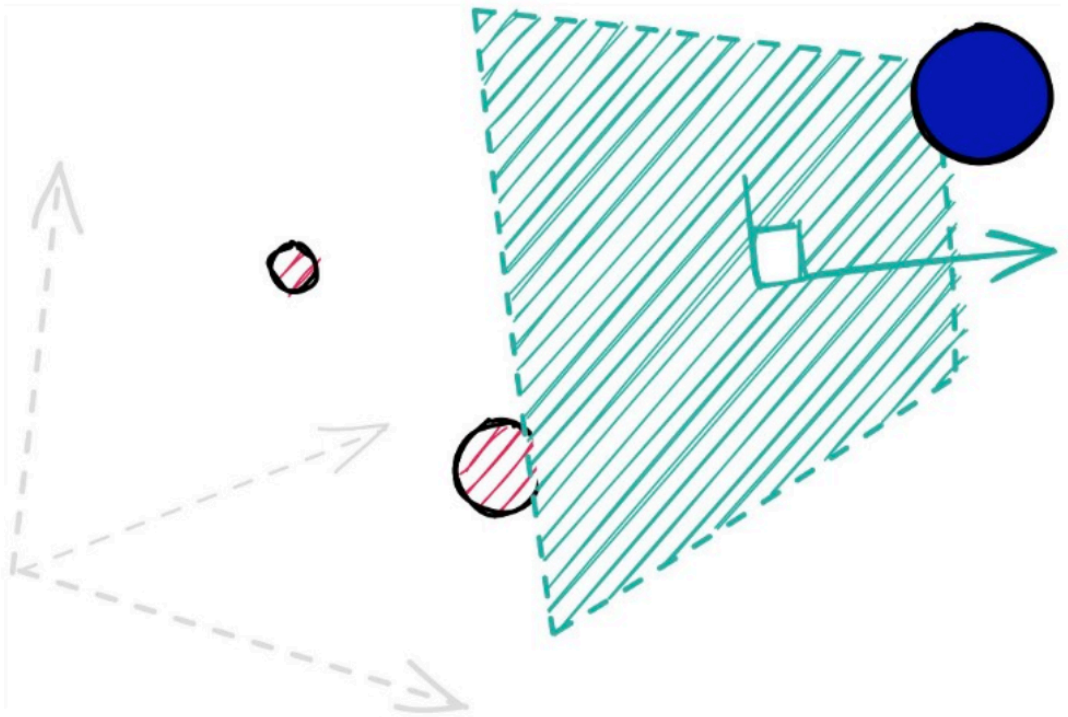


Random Projection for LSH 随机投影

- **随机投影**：使用随机投影矩阵 $\boldsymbol{P}$ 将高维投影到低维。计算 DB 中向量 $\vec{a}$ 和随机投影矩阵 $\boldsymbol{P}$ 间点积 $\boldsymbol{P} \cdot \vec{a}$，得到被投影矩阵 $\boldsymbol{Q}$。

- **具体查询**：使用投影矩阵 $\boldsymbol{P}$ 将查询向量 $\vec{q}$ 投影到低维空间 $\vec{p}$。将投影查询向量 $\vec{p}$ 与 DB 中投影向量进行比较，以找到近邻点。

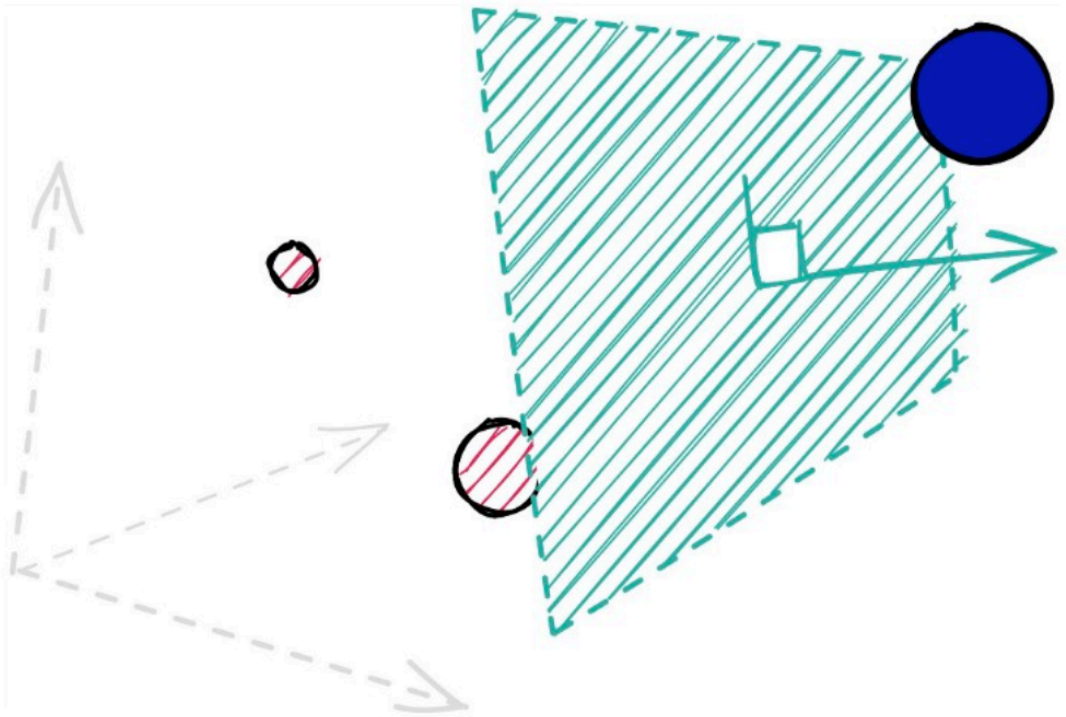
- **由于数据维数降低，搜索过程比在整个高维空间中搜索要快。**



Random Projection for LSH 随机投影

- **基本步骤：**
  1. 高维空间随机选择一超平面，将数据点投影到超平面。
  2. 重复 1 选择多个超平面，将数据点投影到多个超平面上。
  3. 多个超平面投影结果组合成一个向量，作为低维空间的表示。
  4. 使用哈希函数将低维空间中向量映射到哈希桶中。

- 投影矩阵 P 随机性越大，其映射质量就越好，计算成本也会越高。



E. Similarly Search

PQ 乘积量化

Product Quantization (PQ)

- 大规模数据集中，聚类算法最大的问题在于内存占用大：
  1. 需要保存每个向量，向量中元素是浮点数 FP
  2. 需要维护聚类中心和每个向量的聚类中心索引

Product Quantization (PQ)

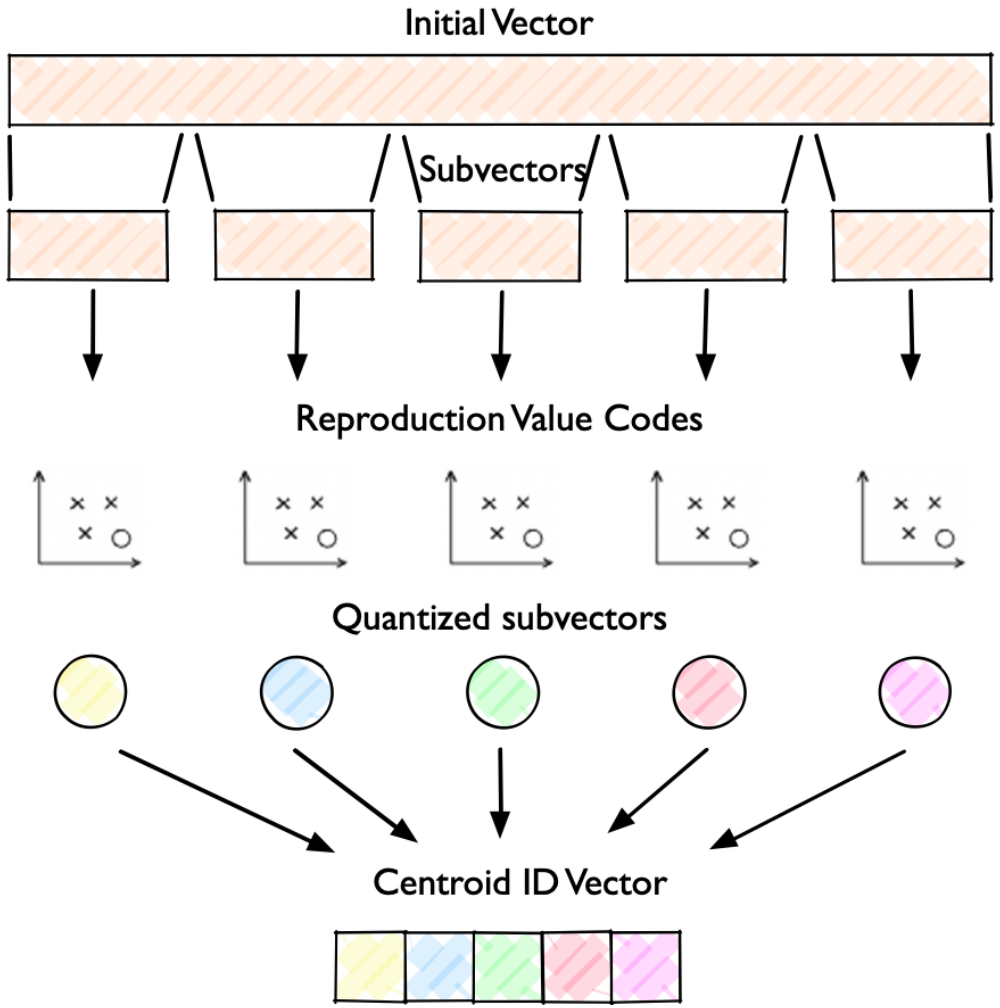
- 随维度增加数据点间距离呈指数增长，高维坐标需要更多聚类中心点将数据点分成更小簇。否则向量和聚类中心距离过远，会降低搜索速度和质量。

  1. 高维坐标系中，容易遇到维度灾难问题怎么整？
  2. 维护数量庞大的聚类中心？

Product Quantization (PQ)

- 将向量分解为 m 个子向量，对每个子向量独立量化，e.g.：
  1. 将 128 维向量分为 8 个 16 维
  2. 8 个 16 维子向量分别聚类
  3. 得到 8 个向量组合成最终编码

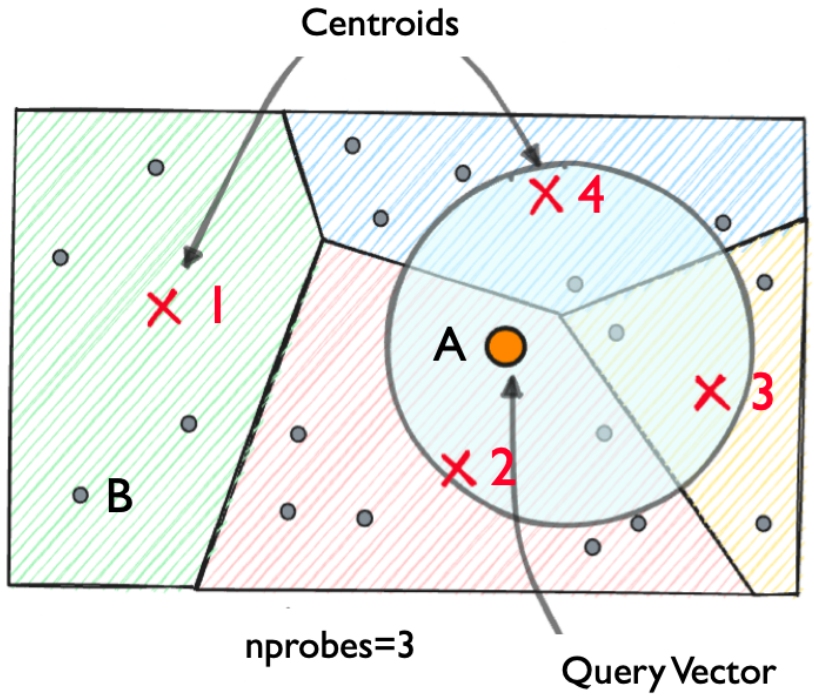
- 16 维子向量大概需 256 个聚类中心，256 维向量大概需 2^64 个聚类中心：
  - 2^64 >>> 8 * 256 = 2048

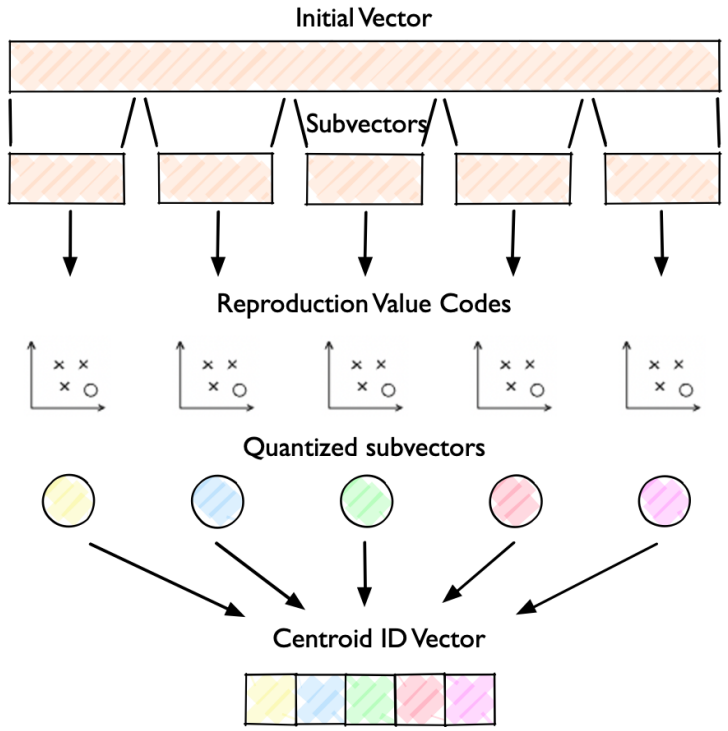


4. 热门 HOT-TOP 相似性搜索算法

IVF-PQ 索引

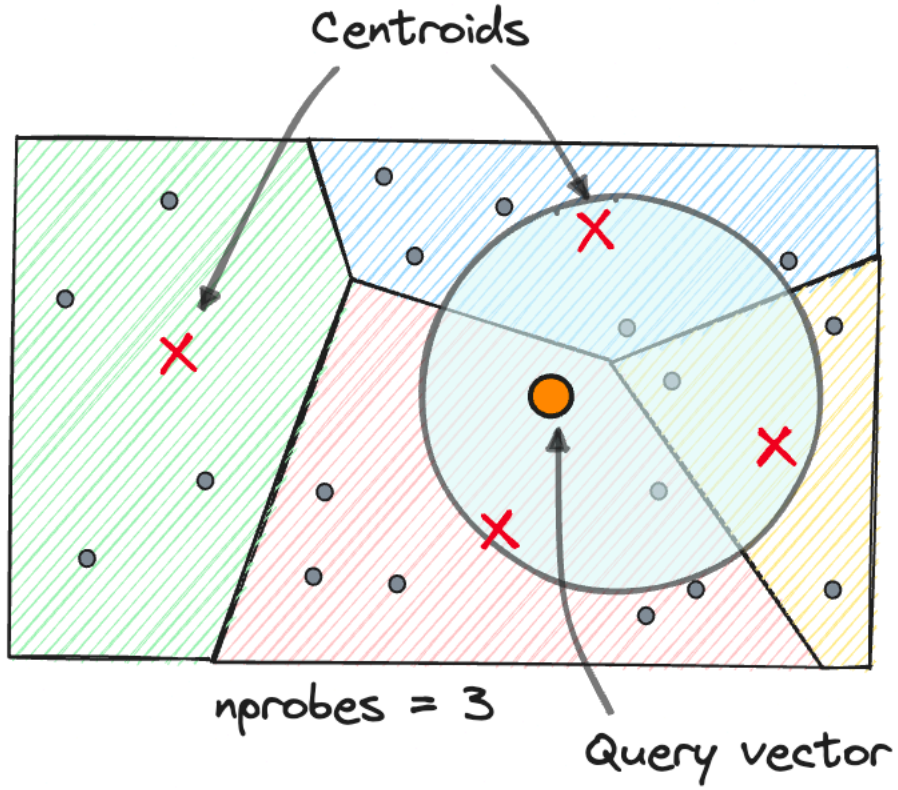
- IVF 倒排索引，用于缩小搜索空间，提高索引召回率
- PQ 加速查询向量与 DB 向量间距离计算，量化减少内存需求





Question

- **边缘问题**：根据查询向量所在位置，可能靠近多个 Voronoi 单元边界，使得哪个单元返回最近邻居变得不明确，从而导致边缘问题。



IVF-PQ

- **IVF-PQ 索引涉及附加参数：**
  1. 搜索算法向外扩展至参数指定中心数量 n_probes

- **PQ 子向量数量和 IVF 分区数量的平衡，才能有效构建 IVF-PQ 索引：**
  1. 1）PQ 子向量越多，子空间越小，导致信息失真；2）子向量多会导致 PQ 步骤中 I/O 和计算量增加。
  2. 控制 IVF 分区数以平衡召回和搜索速度。分区数量 = 向量数量 >>> 暴力搜索，沦为 IVF-Flat 索引。

5. Summary 总结

支持索引总结

Pinecone	Proprietary composite index
milvus / zilliz	Flat, Annoy, IVF, HNSW/RHNSW (Flat/PQ), DiskANN
Weaviate	Customized HNSW, HNSW (PQ), DiskANN (in progress...)
qdrant	Customized HNSW
chroma	HNSW
LanceDB	IVF (PQ), DiskANN (in progress...)
vespa	HNSW + BM25 hybrid
Vald	NGT
elasticsearch	Flat (brute force), HNSW
redis	Flat (brute force), HNSW
pgvector	IVF (Flat), IVF (PQ) in progress...

检索算法小结

- **算法比较：**
  - **聚类检索：**结果准确度高，消耗算力、耗时长（ANN）
  - **基于图结构：**结果比较准确，速度快、消耗内存（HNSW）
  - **组合检索：**准确度取决于模型训练，速度快、节省资源（IVF-PQ）

支持索引总结

- 在搜索或查询插入的向量前，必须声明索引类型和相似度度量。**索引类型：**

1. **FLAT：**适合在小规模，百万级数据集上寻求完全准确和精确的搜索结果的场景。
2. **IVF_FLAT**：适合于在精度和查询速度之间寻求理想平衡的场景。
3. **IVF_SQ8**：适合于在磁盘、CPU和GPU内存消耗非常有限的场景中显著减少资源消耗。
4. **IVF_PQ**：适合于在高查询速度的情况下以牺牲精度为代价的场景。
5. **HNSW**：基于图形的索引，最适合于对搜索效率有很高需求的场景。

问题考虑

- **实际数据库还需要考虑很多工程化问题：**

  1. 数据如何高效存储
  2. 数据如何分布式计算
  3. 数据如何横向资源扩展
  4. 如何实现容错容灾、资源备份

Reference 引用&参考

1. Maximizing the Potential of LLMs: Using Vector Databases (ruxu.dev)
2. Maglott D， Ostell J， Pruitt KD， Tatusova T. Entrez Gene: gene-centered information at NCBI. Nucleic Acids Res. 2005 Jan 1;33 (Database issue):D54-8.
3. Wang Y， Xiao J， Suzek TO， Zhang J， Wang J， Zhou Z， Han L， Karapetyan K， Dracheva S， Shoemaker BA， Bolton E， Gindulyte A， Bryant SH. PubChe m's BioAssay Database. Nucleic Acids Res. 2012 Jan;40 (Database issue):D400-12.
4. Torng W， Altman RB. 3D deep convolutional neural networks for amino acid environment similarity analysis. BMC Bioinformatics. 2017 Mar 16;18 (1):302.
5. Zheng S， Shao W， Chen L. UniVec: a database of gene expression vectors for PCA based gene similarity search. BMC Genomics. 2017 Dec 6;18 (Suppl 10):918.
6. Manning CD， Raghavan P， Schütze H. Introduction to Information Retrieval. Cambridge: Cambridge University Press， 2008.
7. Mikolov， Tomas， et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).
8. Andoni， Alexandr， and Piotr Indyk. "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions." Communications of the ACM 51.1 (2008): 117-122.
9. Jégou， Hervé， et al. "Product quantization for nearest neighbor search." IEEE transactions on pattern analysis and machine intelligence 33.1 (2010): 117-128.
10. Ge， Tiezheng， et al. "Optimized product quantization." IEEE transactions on pattern analysis and machine intelligence 36.4 (2013): 744-755.
11. Babenko， Artem， and Victor Lempitsky. "The inverted multi-index." IEEE transactions on pattern analysis and machine intelligence 37.6 (2014): 1247-1260.
12. Datar M， Immorlica N， Indyk P， Mirrokni VS. Locality-sensitive hashing scheme based on p-stable distributions. Proceedings of the twentieth annual symposium on Co mputational geometry. 2004 Jun 8:253-62.
13. Muja M， Lowe DG. Fast approximate nearest neighbors with automatic algorithm configuration. VISAPP (1). 2009 Feb 4:331-40.
14. Jégou， Hervé， et al. "Product quantization for nearest neighbor search." IEEE transactions on pattern analysis and machine intelligence 33.1 (2011): 117-128.
15. Chen， Zhenjie， and Jingqi Yan. "Fast KNN search for big data with set compression tree and best bin first." 2016 2nd International Conference on Cloud Computing an d Internet of Things (CCIOT). IEEE， 2016.
16. Dehmamy， Nima， Albert-László Barabási， and Rose Yu. "Understanding the representation power of graph neural networks in learning graph topology." Advances in Neural Information Processing Systems 32 (2019).
17. Babenko A， Lempitsky V. The inverted multi-index. IEEE transactions on pattern analysis and machine intelligence. 2014 Jun 7;37 (6):1247-60.

Reference 引用&参考

1. https://www.bilibili.com/video/BV11a4y1c7SW
2. https://www.bilibili.com/video/BV1BM4y177Dk
3. https://www.pinecone.io/learn/vector-database/
4. https://github.com/guangzhengli/ChatFiles
5. https://github.com/guangzhengli/vectorhub
6. https://www.anthropic.com/index/100k-context-windows
7. https://js.langchain.com/docs/
8. https://www.pinecone.io/learn/series/faiss/locality-sensitive-hashing/
9. https://www.pinecone.io/learn/series/faiss/product-quantization/
10. https://www.pinecone.io/learn/series/faiss/locality-sensitive-hashing-random-projection/
11. https://www.youtube.com/watch?v=QvKMwLjdK-s&t=168s&ab_channel=JamesBriggs
12. https://www.pinecone.io/learn/series/faiss/faiss-tutorial/
13. https://www.youtube.com/watch?v=sKyvsdEv6rk&ab_channel=JamesBriggs
14. https://www.pinecone.io/learn/vector-similarity/
15. https://github.com/chroma-core/chroma
16. https://github.com/milvus-io/milvus
17. https://www.pinecone.io/
18. https://github.com/qdrant/qdrant
19. https://github.com/typesense/typesense
20. https://github.com/weaviate/weaviate
21. https://redis.io/docs/interact/search-and-query/
22. https://github.com/pgvector/pgvector

Thank you

把AI系统带入每个开发者、每个家庭、每个组织，构建万物互联的智能世界

Bring AI System to every person, home and organization for a fully connected, intelligent world.

ZOMI

Course chenzomi12.github.io

GitHub github.com/chenzomi12/DeepLearningSystem