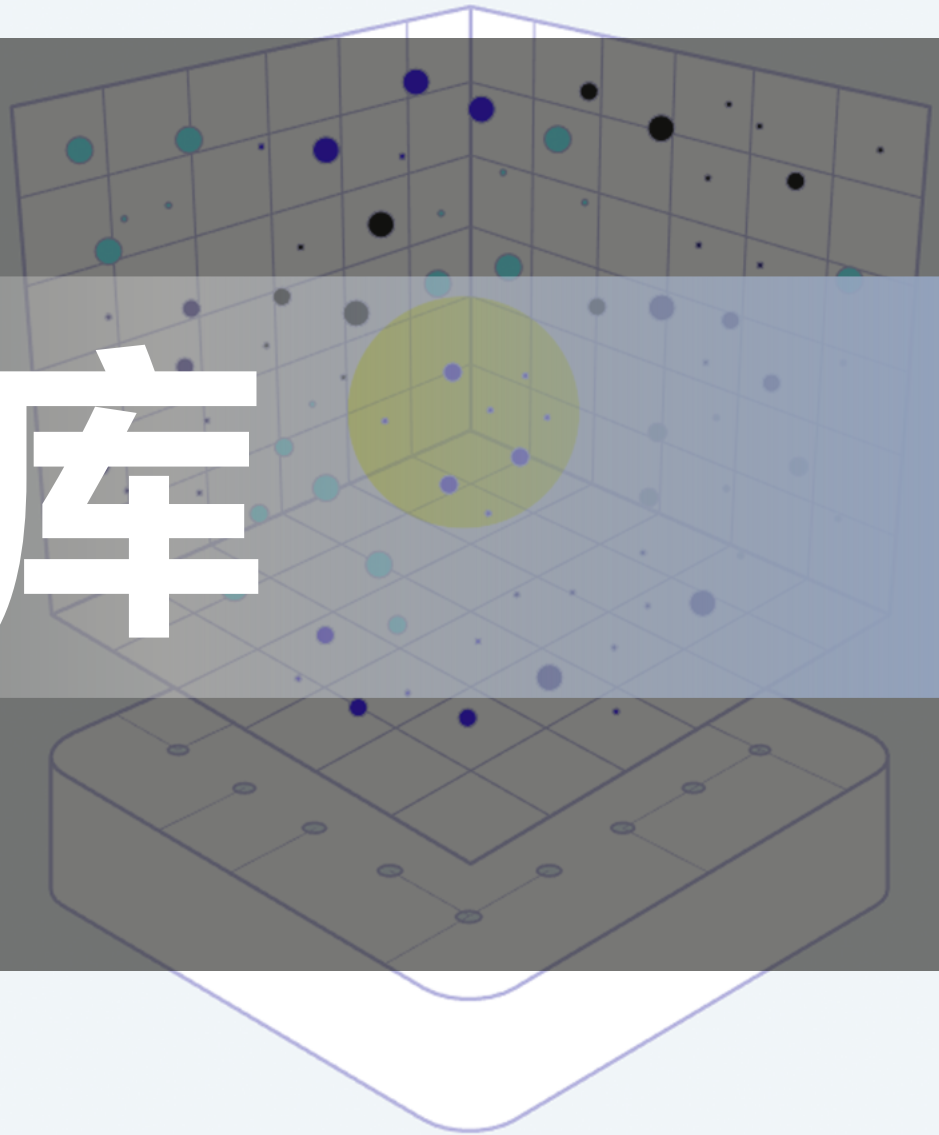
大模型系列 – 数据处理

# 向量数据库


LanceDB

ZOMI

Chroma



# 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

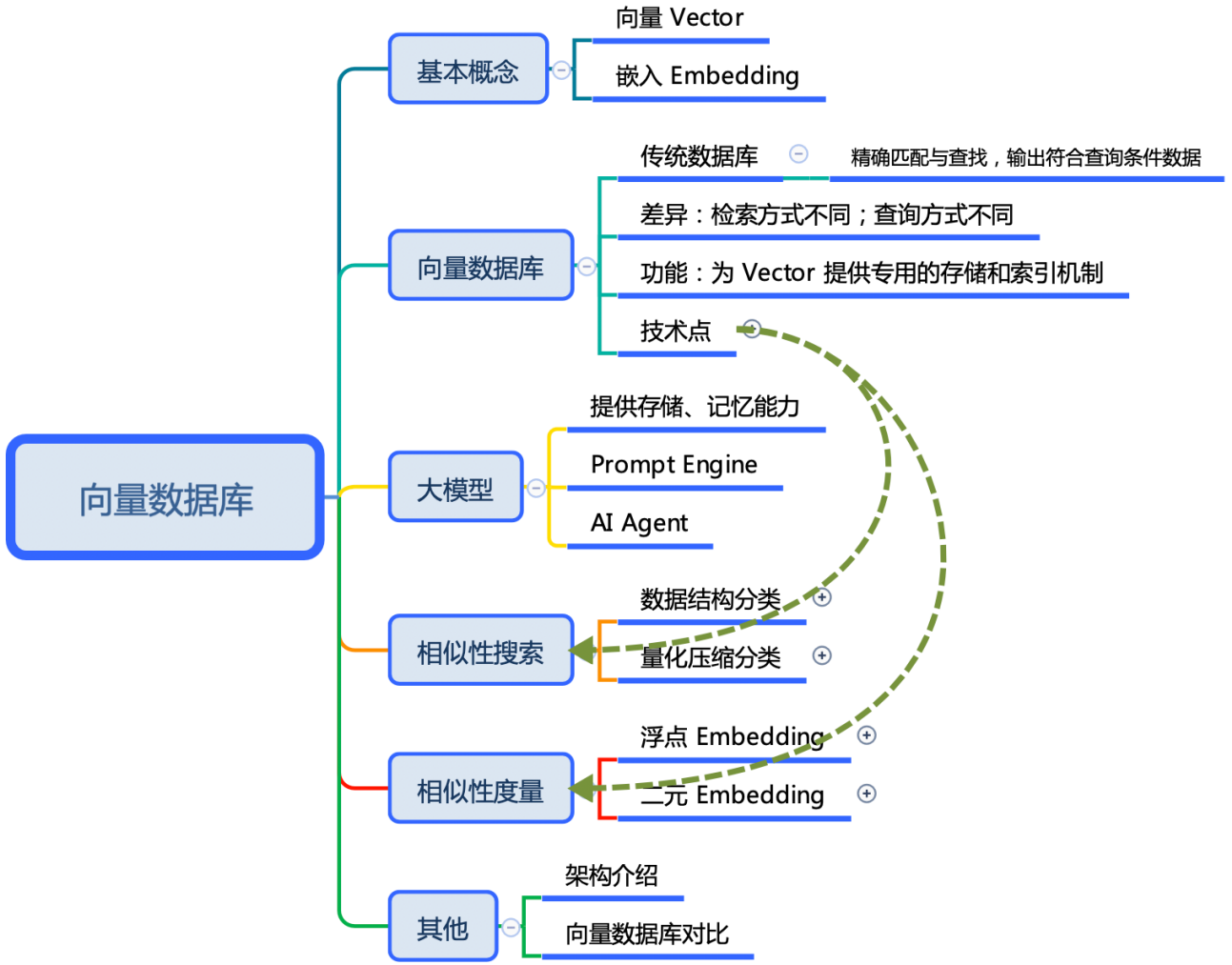
### 1. AI 集群建设

- 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

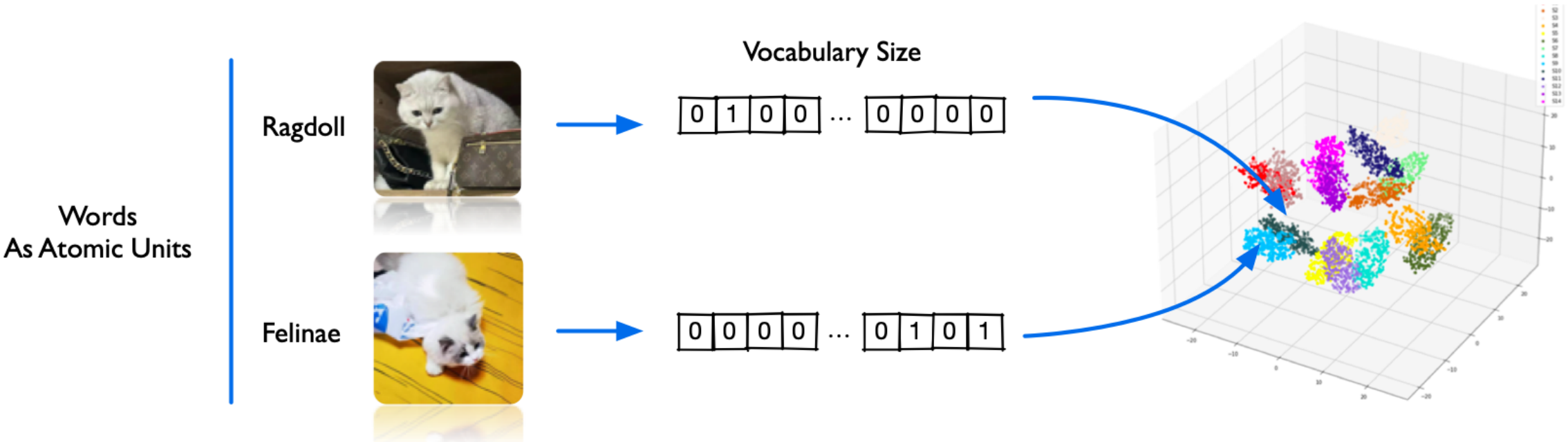
# 大模型系列 – 数据处理之向量数据库



# 1. 传统数据库

# 向量高维表示

- Vector Embedding 将 dog 和 cat 表示为两个不同向量 [0100…000] 和 [0000…0100]
- 两个向量包含了 dog 和 cat 特征，通过计算两个向量距离来判断其相似度
- 也可以将两个向量存入数据库，便于后续语义搜索



# 与传统数据库区别

- **传统数据库**：1）关系型数据库；2） 非关系型 NoSQL 数据库；3）分析型数据库。
- **搜索方式**：索引（B-Tree、倒排等） + 排序算法（BM25、TF-IDF）实现。
- **搜索本质**：基于文本的精确匹配， SQL 语言进行精确匹配与查找，输出符合查询条件数据
- **适合场景**：关键字搜索功能较优，对于语义搜索功能较差。

# 与传统数据库区别

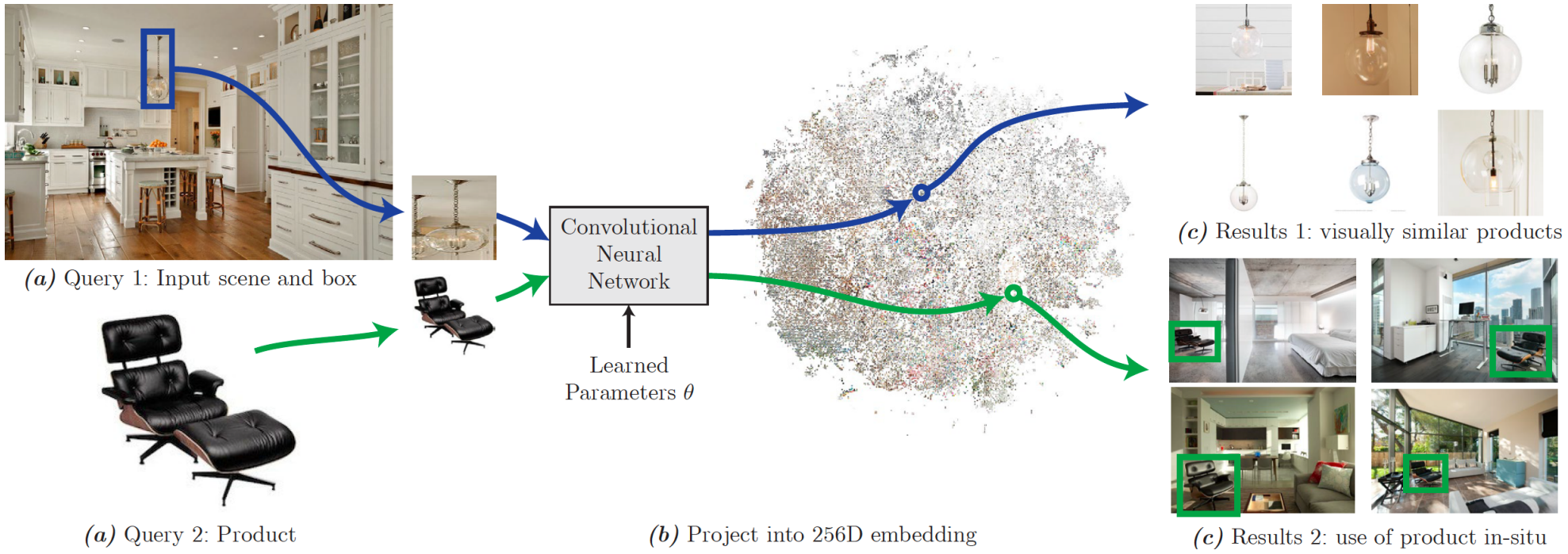
- **e.g.**：搜索“猫咪”，得到“猫咪”关键字结果，而无法得到“银渐层” “布偶”
- **问题**：传统数据库无法识别语义关系，需要打上标签进行关联，才能实现语义搜索

```
1
2   SELECT * FROM anmials WHERE name = "小猫";
3
```

```
4
5   SELECT * FROM anmials WHERE name = "小猫" and feature = "银渐*";
6
```

# 差异 1：检索方式不同

- **传统数据库**：归结为点查和范围查（精确查找），查询得到结果为符合条件/不符合条件
- **向量数据库**：针对向量近似查找，查询得到结果是与输入条件相似 TOP-K 向量




*(a)* Query 1: Input scene and box

*(a)* Query 2: Product

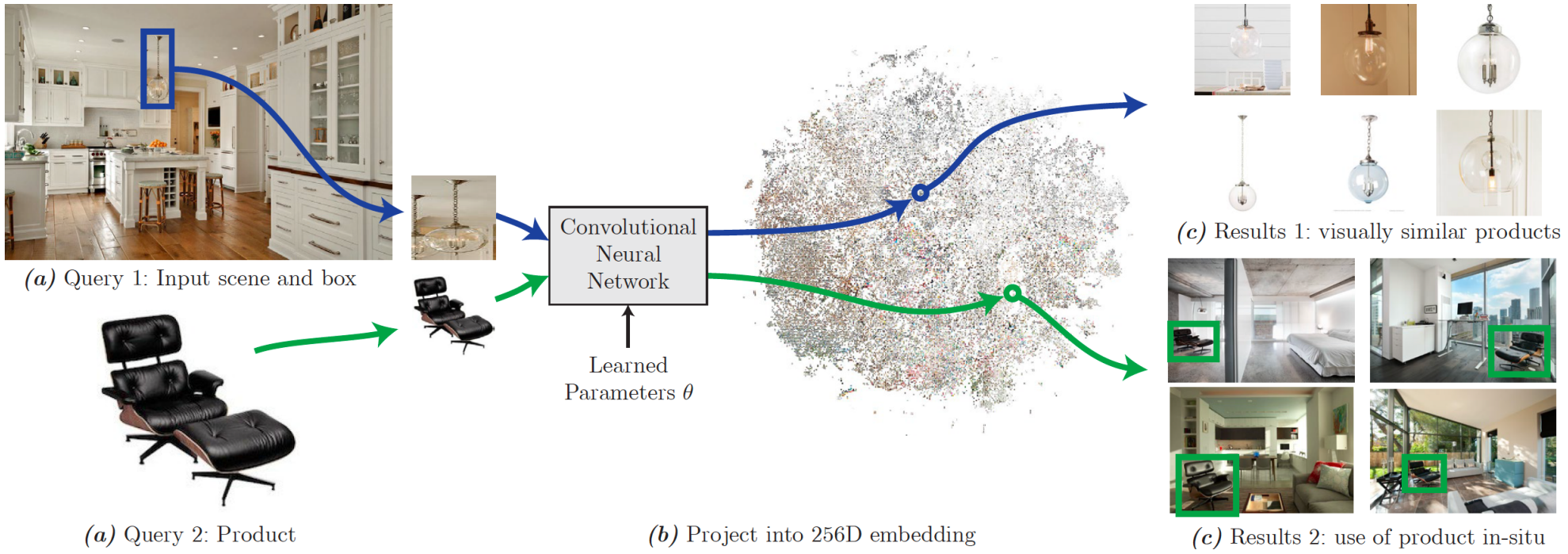
*(b)* Project into 256D embedding

*(c)* Results 1: visually similar products

*(c)* Results 2: use of product in-situ

# 差异 2：查询方式不同

- **传统数据库**：直接处理数据，即使用 SQL 语言对文本精确匹配与查找，输出符合查询条件数据
- **向量数据库**：将非结构化数据转化为向量，再基于向量数据库中进行存储、计算和建立索引




*(a)* Query 1: Input scene and box

*(a)* Query 2: Product

*(b)* Project into 256D embedding

*(c)* Results 1: visually similar products

*(c)* Results 2: use of product in-situ

# DEMO

- **Pinecone**：https://colab.research.google.com/github/pinecone-io/examples/blob/master/docs/semantic-search.ipynb#scrollTo=JWcO7jAK-N_I
- **MySQL**：https://dev.mysql.com/doc/connector-python/en/connector-python-example-cursor-select.html

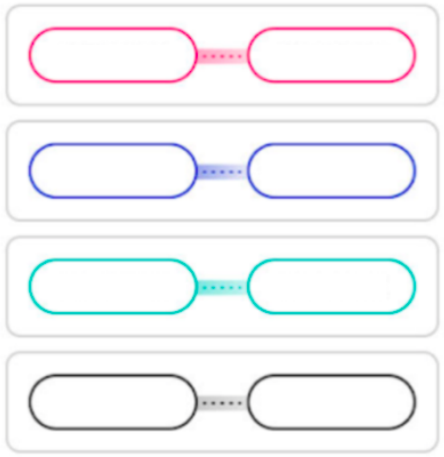
# 2. Vector DB

# 原理、功能、特点

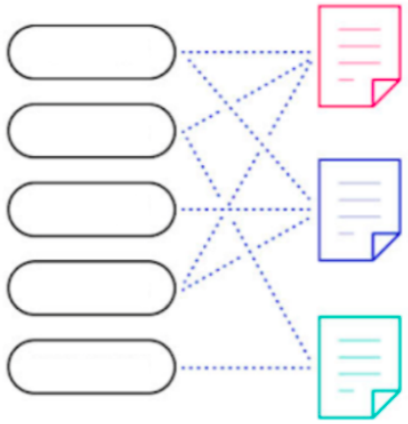
# 向量存储需要新的数据库

- 随移动互联网发展，非结构化数据越来越常见。为让计算机理解和处理非结构化数据，Embedding 将各式数据转换为 Vector 数据。通过数据库存储和索引 Vector 数据，分析 Vector 间相关性。

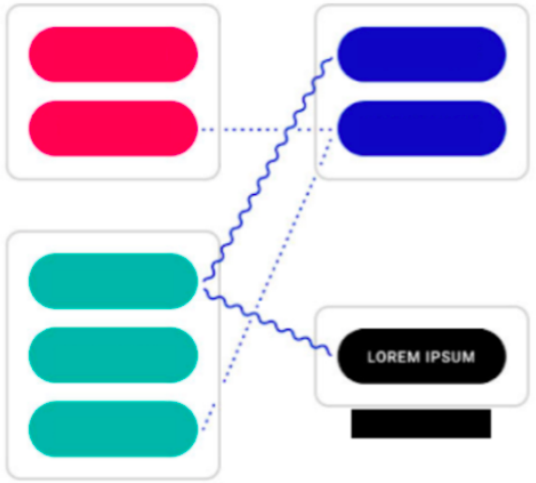
Key-Value



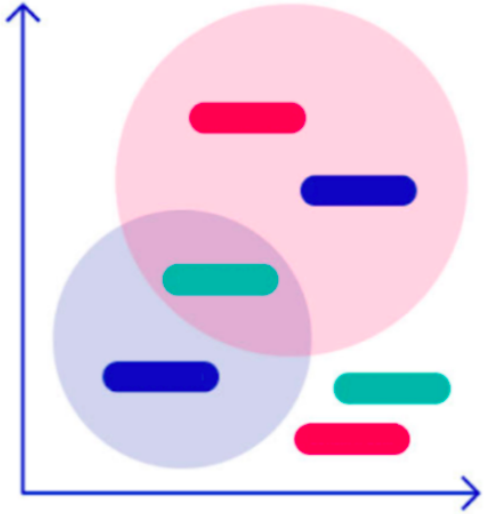
Document



Graph

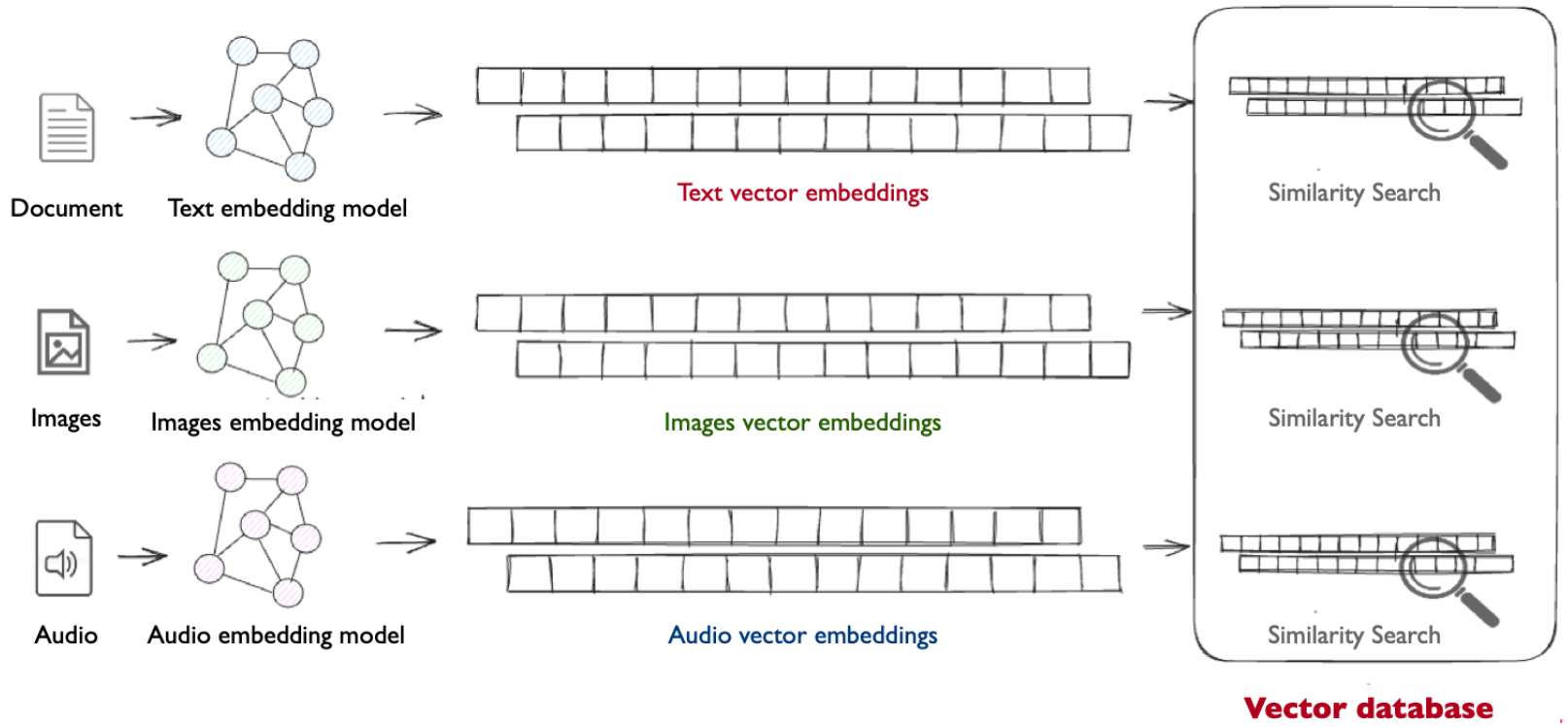



Vector



# Vector DB：出现

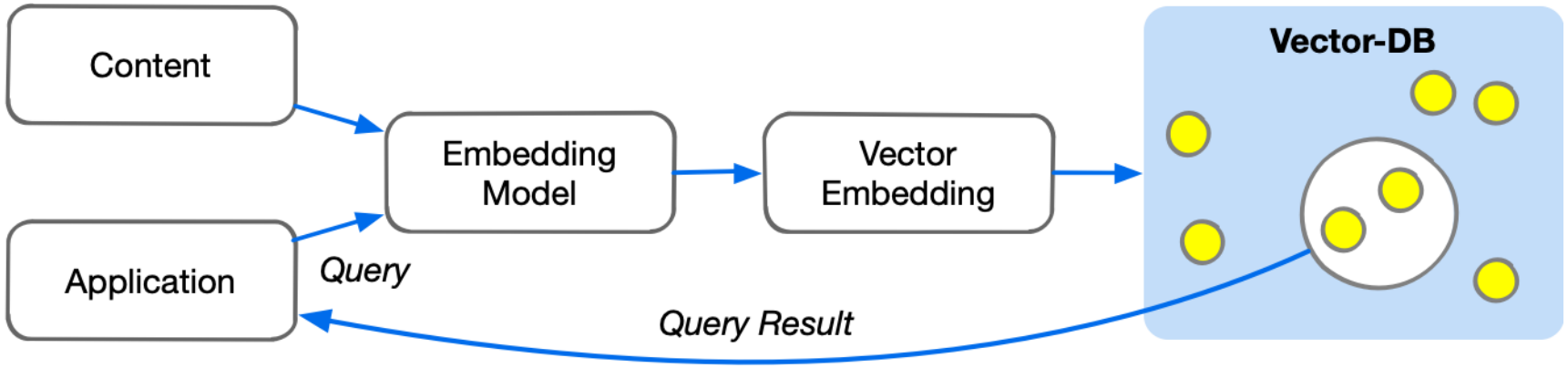
- 用户输入查询（NLP、Image、Audio）被转换成 Vector，与源数据本身位于相同 Embedding 空间中，Vector 通过向量数据进行语义近似搜索，返回与输入查询最相似 TOP-K 个结果。



# Vector DB：定义

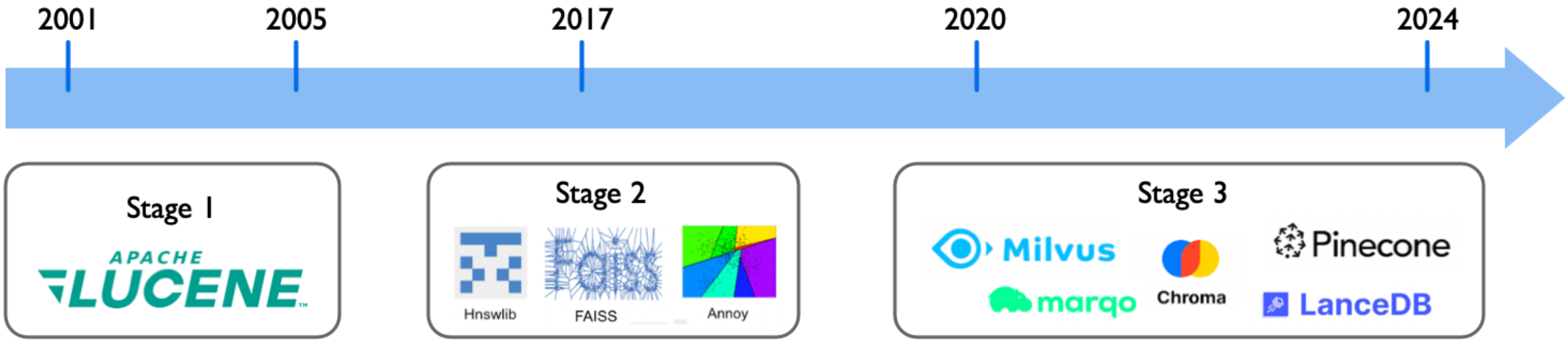
**向量数据库为向量数据提供专用的存储和索引机制。**

- 向量数据被存储为高维空间中点，DB 为点建立索引（KD-Tree、BB-Tree、HNSW等）。
- 索引结构使得 Vector DB 高效地进行向量间相似度查询，从而提升处理向量数据效率。



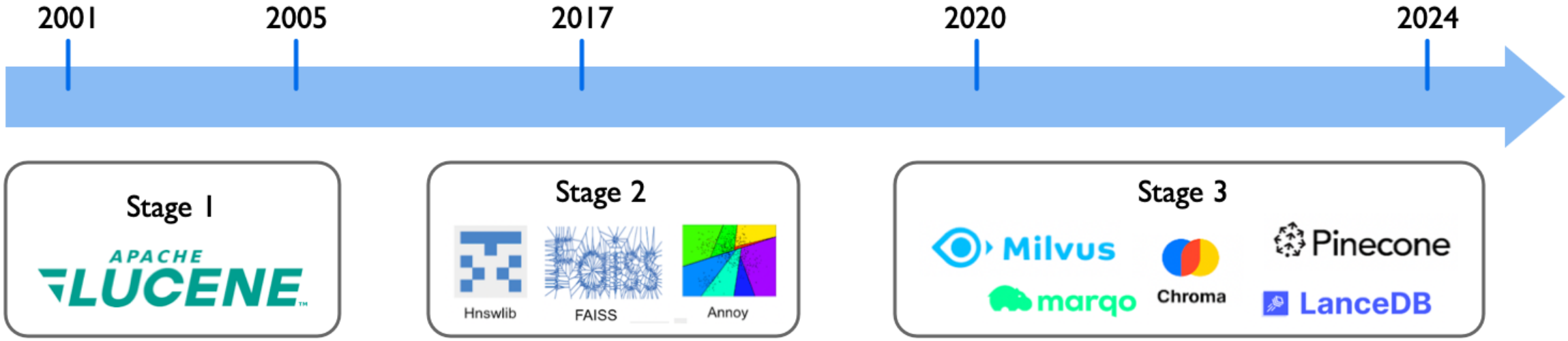
# Vector DB：发展历程

- **第一阶段（初级阶段）：**
  - 主要是以文件形式存储向量数据，缺乏有效索引和查询能力，典型产品如 Lucene 等。



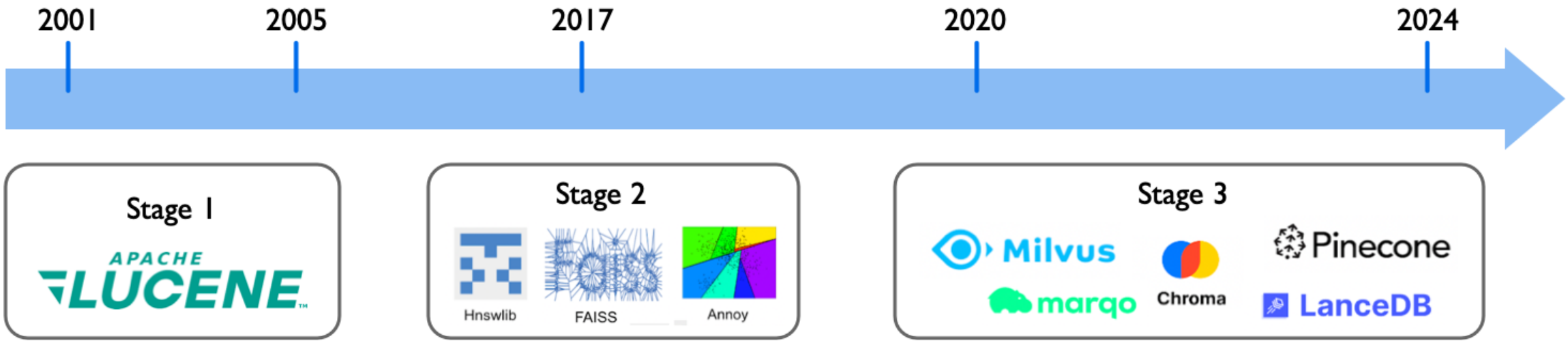
# Vector DB：发展历程

- **第二阶段（发展阶段）：**
  - 使用 KD 树等索引结构，可实现一定查询性能，但在高维空间的查询效率还不高，典型产品有 FAISS、Annoy 等。



# Vector DB：发展历程

- **第三阶段（成熟阶段）：**
  - 通过复杂 Index 算法实现高效向量索引和查询，处理高维向量数据，典型产品有 Milvus、Elasticsearch 等。



# Vector DB：存储向量类型

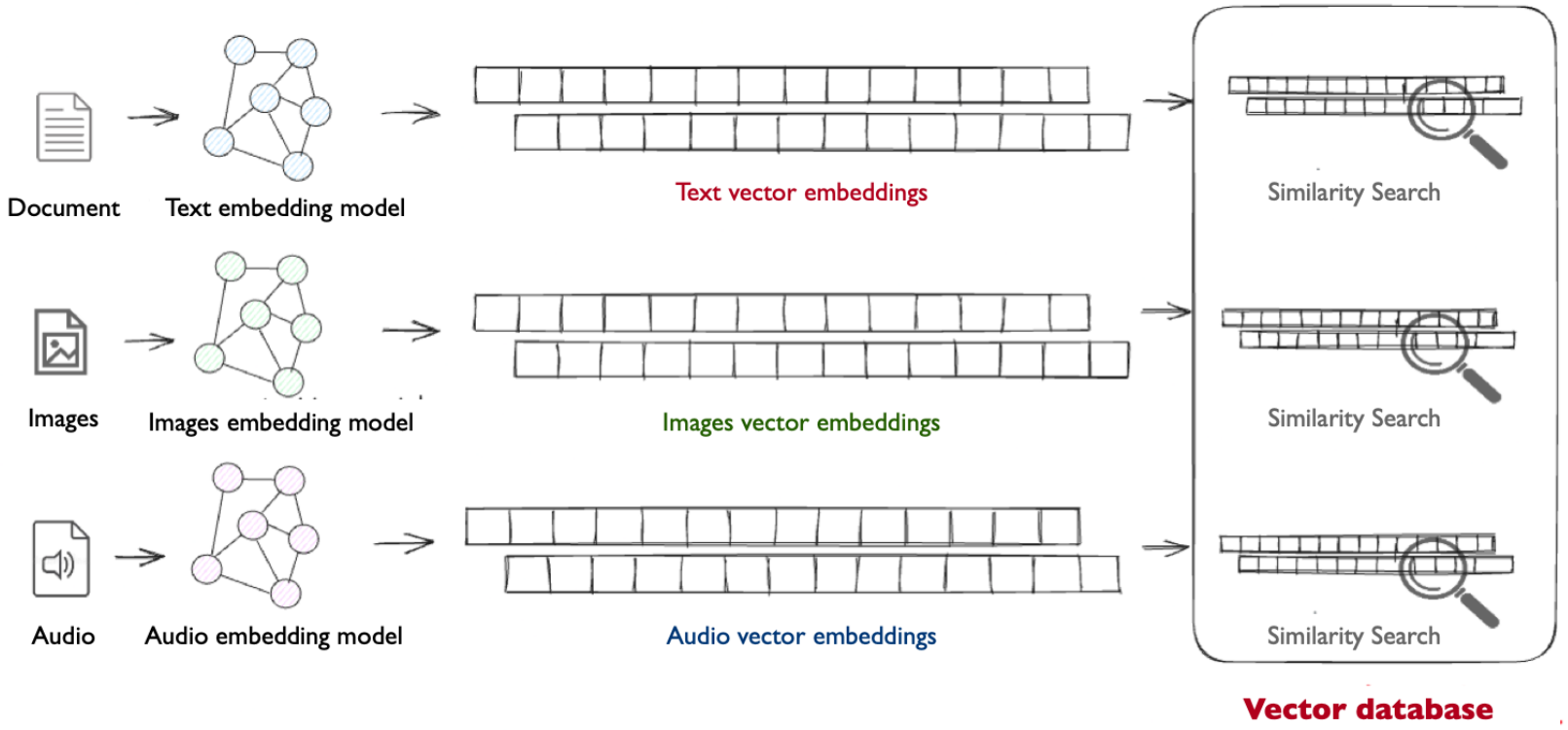
1. **私域知识 Domain Knowledge**
   - 私域知识是指可以把向量数据库作为大模型的外部知识库。不需要去训练模型，比常见的大模型微调地方法成本更低、速度更快也能通过更新数据库保证AI大模型知识的实时更新。
2. **本地存储 Local Storage**
   - 将向量数据存储在本地。通过向量的相似关系保证隐私信息不会提供给大模型进行训练。
3. **长期记忆 Long Time Memory**
   - 长期记忆是相比大模型的短期记忆，大模型如ChatGPT其上下文信息有数量限制。Vector DB 作为外部数据库，存储单次上传的超大文本、对话内容等信息，为大模型提供理论上没有上限的长期记忆。

# Vector DB：解决的问题

- 主要解决2个问题，高效的检索与高效的分析：
  1. **检索**：以图搜图场景，如人脸检索、人脸支付、车牌号码检索、相似商品检索等；
  2. **分析**：以图分析行为，人脸撞库、人脸对比、场景再现等；

# 作用 1：相似性搜索

- 允许根据向量距离或相似性对向量数据 Vector 进行快速准确的相似性搜索和检索。这意味着可以使用 Vector DB 根据语义或上下文含义查找最相似或相关的数据，而非精确匹配或预定义标准查询数据库的传统方法。



# 作用 II：提升性能

- 针对大量向量数据 Vector 存储和检索操作进行优化，每次查询通常处理数亿个向量，并且比传统数据库的处理速度快得多。以下主要介绍向量数据库最核心的几种技术和能力：

1. **相似度计算**：使用相似性度量等算法过滤和定位相似的向量
2. **相似性搜索**：提供 Vector检索算法，使向量检索速度显着加快并更精确
3. **高效存储**：以更紧凑方式存储向量，如压缩和量化向量数据，尽可能在内存中查询数据
4. **分布式**：跨多台机器对向量数据机型分片和分片检索

# 常用评价标准

1. **准确率（Precision）**
   - 准确率 = 检索相关的向量 / 检索出的向量总数
2. **召回率（Recall）**
   - 召回率 = 检索相关的向量 / 向量数据库中相关的向量总数
3. **每秒平均吞吐（QPS）**
   - Query Per Second，QPS 是每秒查询数，每秒向量数据库能够处理的查询请求次数
4. **平均响应延迟（Latency）**
   - 向量数据库的请求平均响应时间

# Vector DB：功能与特点

| 功能 | 描述 |
| --- | --- |
| **Vector Embedding**<br>**向量嵌入** | 数学上，vector embedding 是一个浮点数或二进制数的数组，通过对非结构化数据转换为具体的 vector（对非结构化数据的特征抽象）。Vector DB 提供数据写入/检索自动向量化，对齐传统数据库的使用体验，用户无需关注向量生成过程。 |
| **Similar Search**<br>**相似性搜索** | 使用 Index 算法和特殊数据结构对 Embedding 进行相似性搜索。用户可以快速找到与给定查询最接近的 vector，因此适合对图像或文本的相似性搜索任务。其中向量相似度搜索是将输入 Vector 与数据库进行比较，以找到与查询向量最相似的向量过程。 |
| **High Performance**<br>**提升性能** | Vector DB 单索引支持 > 亿级向量数据规模，可支持百万级 QPS 及毫秒级查询延迟。另外对高维向量的高效存储，能够在最小存储空间下处理更大量数据。 |
| **可扩展** | 具备良好的可扩展性，能够轻松处理大规模数据集，支持对 Vector 数据进行分片，良好支持广泛依赖嵌入的大语言模型和其他 AI 应用。 |
| **灵活性** | 能够处理不同数据（非机构化数据）类型，这些信息不遵循预定义的模型或组织方式，包括文本、图像、音频和视频。 |

# 向Vector DB：管理挑战

1. **有效存储**：向量数据多为浮点或者二进制数据，数据压缩率低，存储成本高；
2. **高效存储**：向量数据库计算复杂度高，需要集群&分布式计算能力；
3. **索引复杂**：有tree、graph、hash等多种向量索引方式，索引管理和使用成本高；
4. **扩展性**：向量数据库的增长对系统的扩展性要去越来越高；
5. **低延迟**：查询以ms级别响应业务需求；

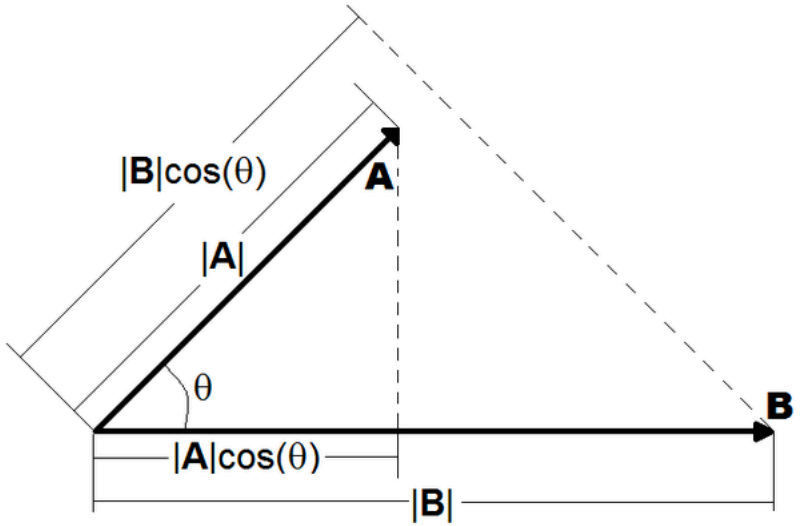
# 3. Vector-DB

# 技术点梳理

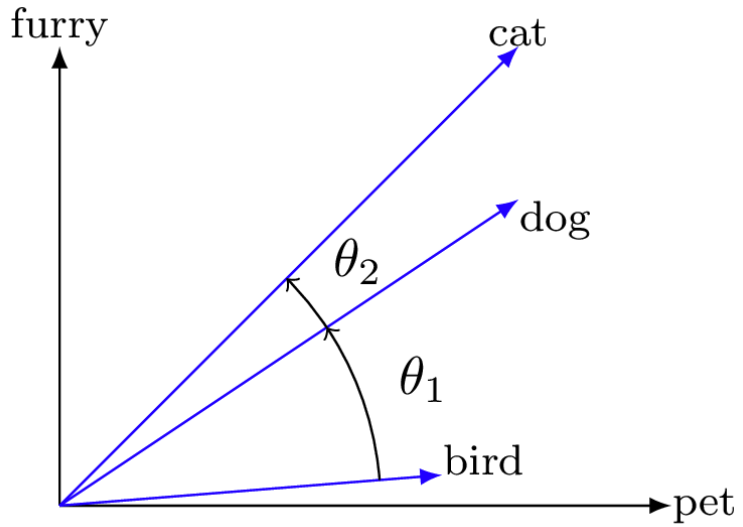
# 相似度是如何计算

- Vector-DB 提供不同的度量算法计算向量相似性。e.g.，最常用NLP 度量算法：
  - **点积**： $a * b = |a||b|\cos\theta$ ，输出任意大小的值
  - **余弦距离**： $cosines = (A \cdot B)/(||A||\ ||B||)$ ，产生一个标准化值（-1 和 1 之间）

dot product



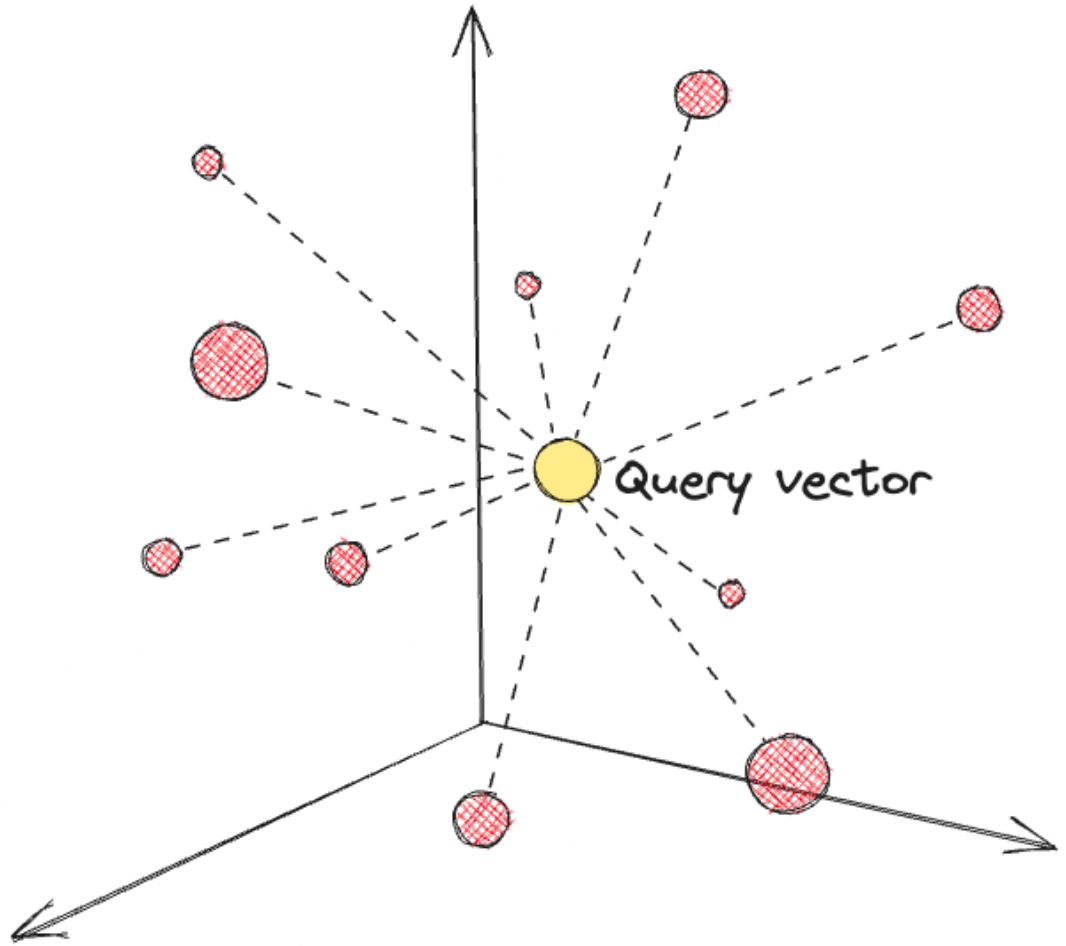

Cosine Distance



# 相似性搜索是什么

- 用户查询时，相似性搜索提供与输入向量相似 TOP-K 向量。

- e.g.，使用 k 最近邻（KNN）算法将查询向量与 Vector-DB 中每个向量进行比较。



# Question

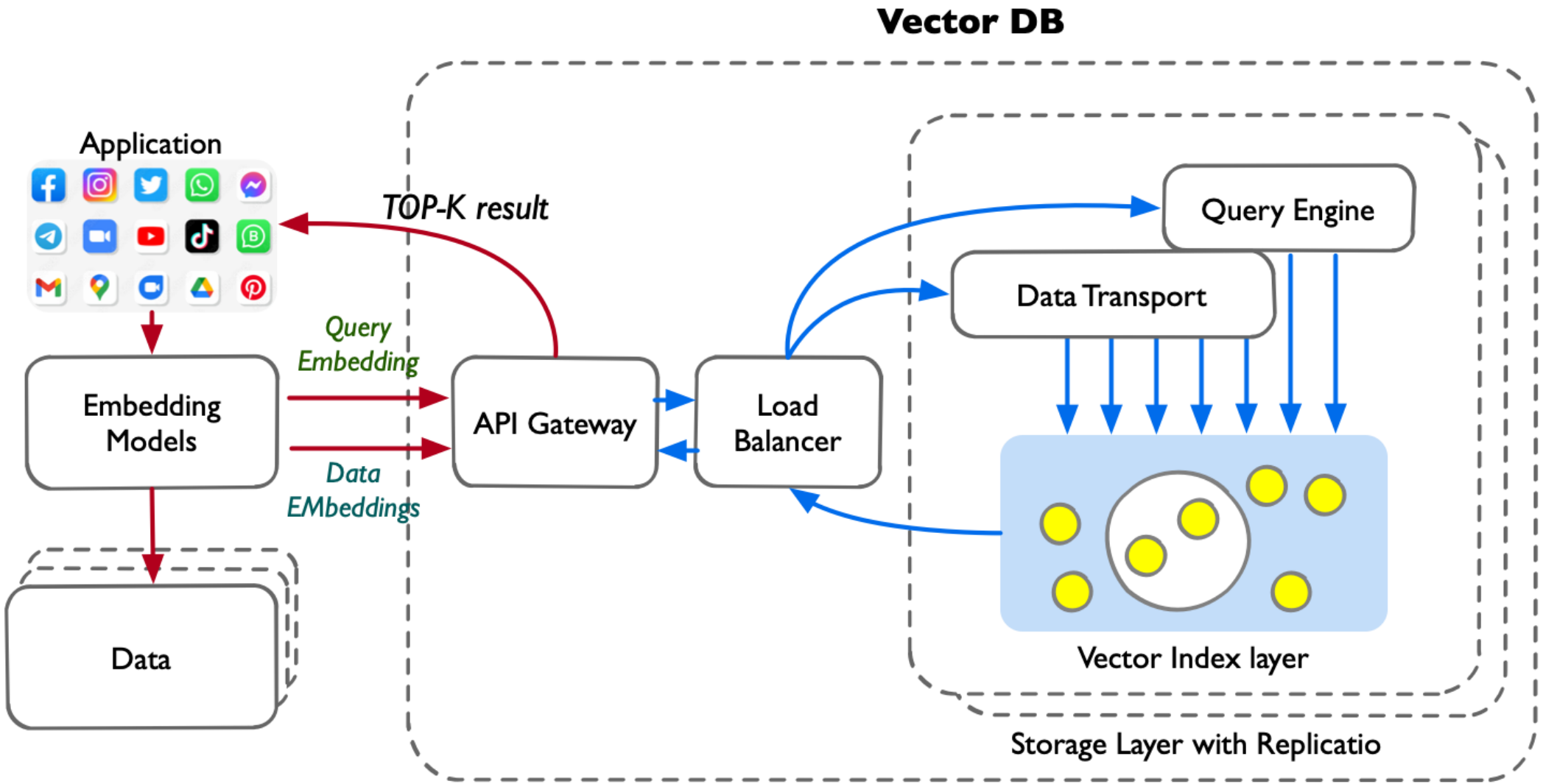
- 搜索空间扩展到百万/十亿个数据点时，所需比较数量随着数据增加而线性增加，分布式的线性度下降。查询变慢，如何解决？

# 如何进行索引

- 数据通过**索引 Indexing** 存储在向量数据库中，索引通过缩小搜索空间来有效地查找相似向量；
- 但 Embedding 维度较高，Vector-DB 会这对 Vector 特征使用索引算法：

1. K-means and/or Faiss
2. Inverted File Index（ IVF ）
3. Hierarchical Navigable Small World（ HNSW ） graphs

# 整体回顾


Vector DB
Application
TOP-K result
Query Engine
Data Transport
Query
Embedding
Embedding
Models
API Gateway
Load
Balancer
Data
EMbeddings
Data
Vector Index layer
Storage Layer with Replicatio

# 整体回顾

- **存储和数据索引**
  - Embedding：数据输入 Embedding Model 转换为 Vector，通过 API 网关输入到 Vector-DB 的存储层
  - Indexing：对向量 Vector 数据进行分区/分片以实现可扩展性和快速查找
  - Searching：查询引擎与存储层紧密集成，通过 Vector-DB 的索引算法快速检索相似向量 Vector

- **应用层**
  - Query：用户通过 APP UI 将查询给 Embedding Model，将输入查询转换为与数据相同的 Embedding Space
  - Return：Vector 化查询通过 API 网关发送到查询引擎，可异步处理多个查询， TOP-K 结果返回用户

# 3.混合搜索系统

# DB-Engines Ranking of Vector DBMS

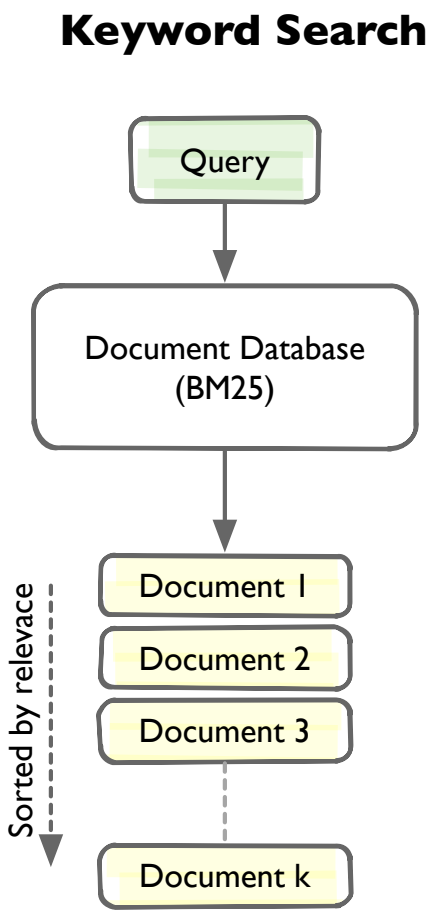
☐ include secondary database models  14 systems in ranking, January 2024

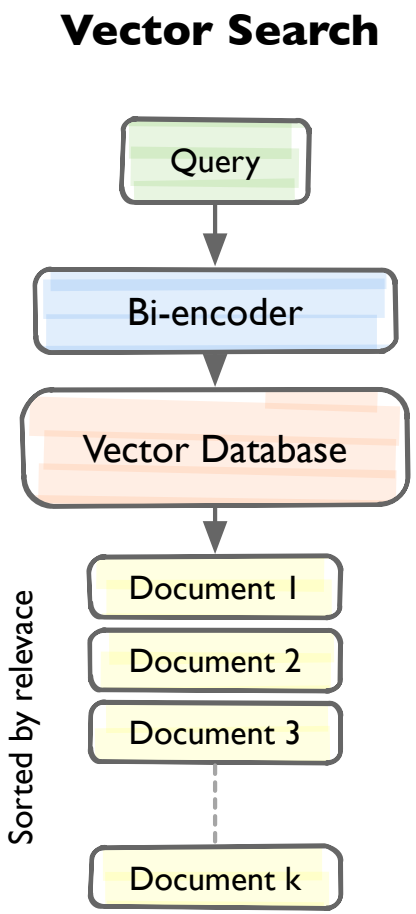
| Rank Jan 2024 | Rank Dec 2023 | Rank Jan 2023 | DBMS | Database Model | Score Jan 2024 | Score Dec 2023 | Score Jan 2023 |
|---|---|---|---|---|---|---|---|
| 1. | 1. | 1. | Kdb | Multi-model | 7.97 | -0.31 | +1.08 |
| 2. | 2. | ↑ 3. | Pinecone | Vector | 3.21 | -0.16 | +1.93 |
| 3. | 3. | | Chroma | Vector | 2.23 | -0.03 | |
| 4. | 4. | ↑ 6. | Weaviate | Vector | 2.03 | 0.00 | +1.61 |
| 5. | 5. | 5. | Milvus | Vector | 1.44 | -0.04 | +0.76 |
| 6. | ↑ 7. | ↓ 2. | AllegroGraph | Multi-model | 1.27 | +0.09 | -0.11 |
| 7. | ↓ 6. | 7. | Qdrant | Vector | 1.24 | -0.01 | +1.14 |
| 8. | 8. | ↓ 4. | CrateDB | Multi-model | 1.00 | -0.01 | +0.01 |
| 9. | 9. | | Vald | Vector | 0.80 | -0.04 | |
| 10. | 10. | | Vespa | Multi-model | 0.66 | -0.15 | |
| 11. | ↑ 12. | | MyScale | Multi-model | 0.38 | +0.06 | |
| 12. | ↓ 11. | | Deep Lake | Vector | 0.36 | -0.05 | |
| 13. | ↑ 14. | ↓ 8. | JaguarDB | Multi-model | 0.00 | 0.00 | -0.01 |
| 13. | 13. | | Transwarp Hippo | Vector | 0.00 | -0.05 | |

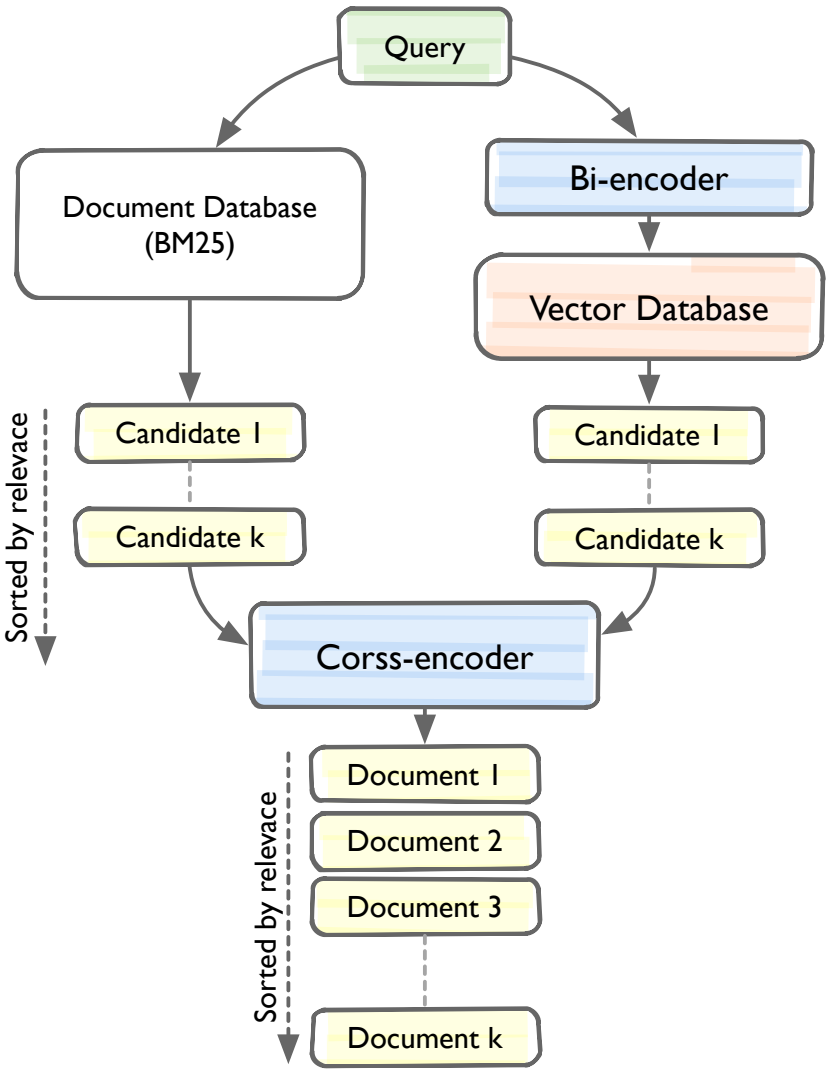
# 混合搜索系统

- **Key Word 搜索**：当用户知道期望搜索的结果或者与搜索词中的短语完全匹配时，找到明确相关、有用的结果，使用传统数据库，而非向量数据库。
- **向量搜索**：当用户不知道需要明确的搜索目标，而是期望搜索语义相关、特征相关结果。使用向量数据库，而非传统数据库。
- **混合关键字+向量搜索**：结合全文关键字和向量搜索的候选结果使用交叉编码器模型对结果重新排名。结合传统数据库和向量数据库。

# 混合搜索系统


Keyword Search
Query
Document Database
(BM25)
Document 1
Document 2
Document 3
Document k
Sorted by relevace

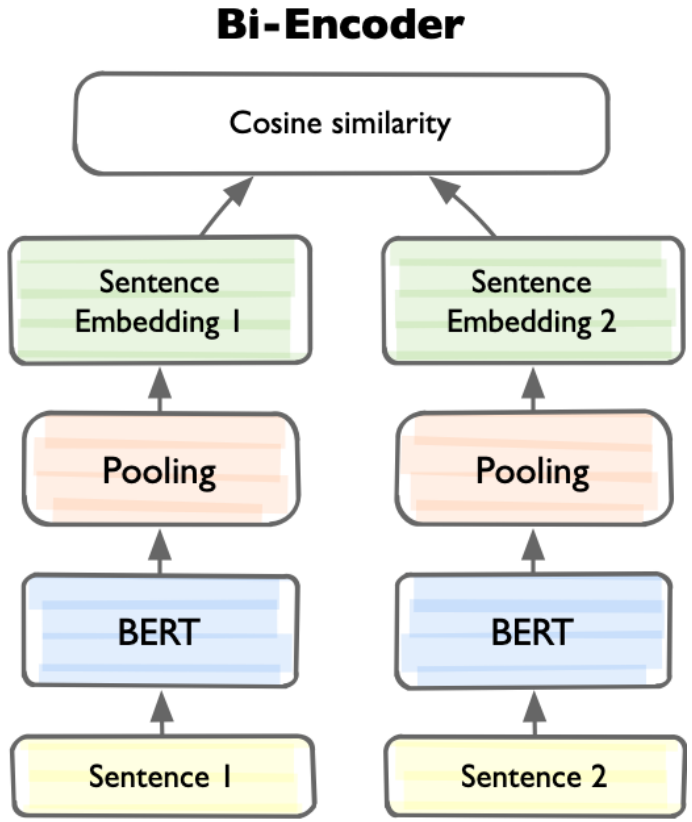

Vector Search
Query
Bi-encoder
Vector Database
Document 1
Document 2
Document 3
Document k
Sorted by relevace


Hybrid Search
Query
Document Database
(BM25)
Bi-encoder
Vector Database
Candidate 1
Candidate k
Candidate 1
Candidate k
Sorted by relevace
Corss-encoder
Document 1
Document 2
Document 3
Document k
Sorted by relevace

# 检索基本方式

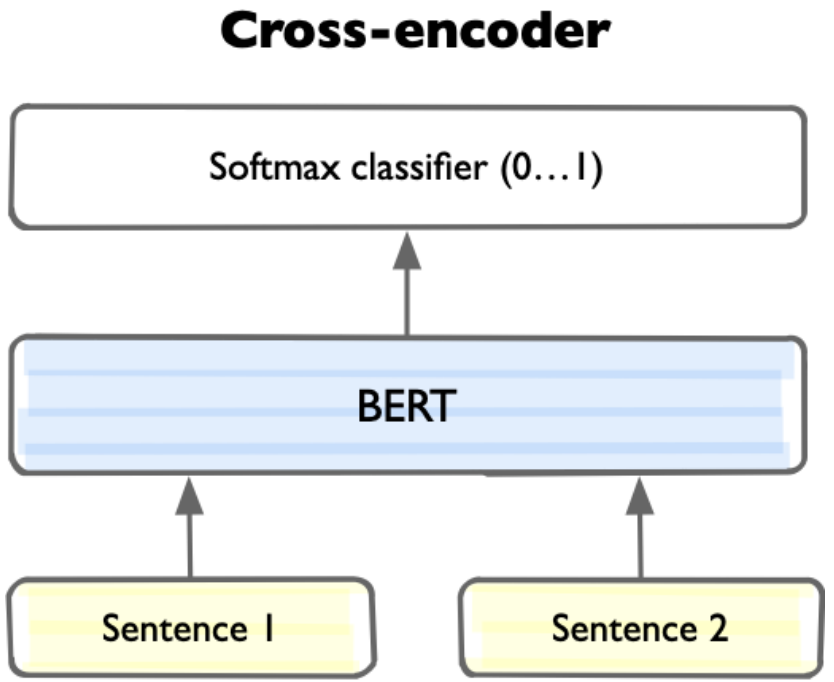
- **传统数据库**（Elasticsearch、Opensearch、MongoDB）：主要使用类 BM25 关键字索引算法。与逆文档频率（IDF）相关的关键字频率来生成稀疏向量。

- **向量数据库**：主要对文本进行编码或物体特征化为稠密Dense Vector，使用双编码器模型 Bi-Encoder 对生成 Embedding Vector Pair 进行比较以产生余弦相似度得分。



# 混合搜索系统与交叉编码器

- **混合搜索**：将关键字搜索（BM25）和 Vector 搜索（余弦相似度）获得的搜索结果结合，需要交叉编码器。
- **Cross Encoder**：将两个句子同时输入 Encoder 模型（BERT）。与 Bi-Encoder不同Cross-Encoder不生成 Embedding。
- **重新排名**：允许通过 softmax 为一对输入分配最大似然概率进行分类，获取结合关键字+矢量搜索的结果。



# Question ?

1. 不是大模型让向量数据库火起来的嘛，为什么没有大模型相关的场景？
2. 向量数据库的应用场景规模感觉比较有限，增量在哪里？

# 重点之重点

- 向量数据库本身并不支持语义近似，只支持位置 Position 近似。需要采用 Embedding 方式来使得位置接近于语义近似。

## Pure vector databases

Milvus
zilliz
Pinecone
Weaviate
vespa
marqo
qdrant
Chroma
Vald
LanceDB

## Text search databases

elasticsearch
APACHE LUCENE
OpenSearch
Apache Solr

## Vector-capable NoSQL databases

cassandra
neo4j
MongoDB.
DataStax Astra
Azure Cosmos DB
redis

## Vector libraries

FAISS
Annoy
Hnswlib

## Vector-capable SQL databases

SingleStore
pgvector for Postgres
ClickHouse
kinetica
ROCKSET

# 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.

# 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