# 数据库新技术综述报告——矢量数据库

Koorye

2023 年 10 月 12 日

# 目录

**摘要**

矢量数据库是一种新兴的数据库类型，专门用于存储和处理高维数据，在拥有对矢量的计算和分析能力的同时，也和传统数据库一样拥有对结构化数据的管理能力。本文对矢量数据库进行了详尽的介绍，从用户需求，即深度学习模型需要存储和比较海量数据的角度出发，介绍了矢量数据库的发展。之后介绍了矢量的定义、类型和算法，以及矢量数据库的工作原理，如索引编排、矢量查询、矢量后处理等，深入剖析了矢量数据库的底层逻辑。之后，本文介绍了矢量数据库系统的架构，存储管理、安全管理等，展示了现代的矢量数据库系统是如何在分布式场景下运作。最后，本文介绍了矢量数据库的一些应用场景，如图像检索、搜索引擎等，展示了矢量数据库的具体应用价值。

关键词：矢量数据库、矢量、深度学习、分布式。

# 1 介绍

本章节将从几个方面介绍矢量数据库，分别是矢量数据库的定义和矢量数据库的历史。

## 1.1 矢量数据库的定义

矢量数据库（或称向量数据库）是专门用于存储和处理高维数据的数据库。对于用户来说，它拥有高效的数据组织、检索和分析能力，可用来作为数据分析的辅助工具；同时也与传统数据库一样，拥有对结构化数据的管理能力，并向用户提供相应接口以进行高阶操作。

## 1.2 矢量数据库的历史

传统的关系型数据库被广泛用来存储结构化数据，在关系型数据库中，数据被组织成二维表格的形式，不同属性组织成若干列，每条数据则以字段形式存在表格行中。

然而，随着大数据、深度学习等领域的快速发展，许多领域越来越依赖高维数据。近几年计算机视觉领域出现巨大突破，继 AlexNet 将卷积神经网络引入图像分类之后，大量工作随后被发出，如 VGG/GoogleNet/ResNet 等。如图 1所示，这些工作使得图像分类的准确率快速增长，为深度学习领域注入崭新的活力。

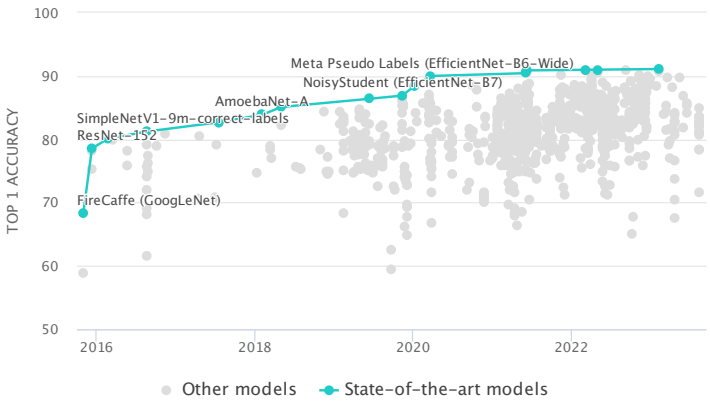



图 1: 近年来图像分类网络的发展

然而，深度学习的快速发展也带来了对数据崭新的需求。由于深度网络会将输入数据（如图像、文本、语音、视频等）编码为若干高维度的矢量，那么对于大量输入数据编码得到的矢量数据来说，如何有效存储它们就成为了一大难点。另一方面，人类难以理解这些矢量的含义，于是如何分析、比较这些矢量并输出结果又成为另一大难点。这些问题是传统关系型数据库无法解决的，关系型数据库只适合存储结构化数据而非多种数据格式的矢量，另外关系型数据库也不支持对矢量进行分析和计算。

基于上述原因，矢量数据库应运而生。矢量数据库以矢量的形式存储数据，其中每条矢量代表代表一个单独的样本，可能是一张图像或一句文本高维表示，并可能包含一些描述性的集合。在深度学习中，一条矢量也通常被叫做“特征”。为了实现高效的矢量存储和查找，矢量数据库实现了高效的矢量索引策略，便于通过比较矢量之间的相似度来进行相似性搜索。

目前，矢量数据库的发展还处于起步阶段，尚不存在统一的解决方案。国内一些代表性的厂商如 AI 领域的商汤、旷视，和互联网企业阿里巴巴、华为、京东等，这些企业各自形成了自己的矢量数据库系统。不过，当前的矢量数据库也开始出现统一规范的查询语言，并逐步与传统数据库一样实现了对数据的增删改查能力。另外，当前的矢量数据库也向着分布式、云原生的方向发展，未来的矢量数据库的扩展能力会进一步加强，单机的资源消耗和成本会进一步降低。

与传统数据库相比，矢量数据库有以下不同：

- 存储数据的类型和格式不同。关系数据库是为适合表的结构化数据而设计的，数据通常被组织成表格的形式，不同的属性组织成若干列，数据则以字段形式存在表格行中。矢量数据库则是为非结构化数据设计的，用于存储对象的高维信息表示。

- 数据规模更大。传统的关系型数据库管理 1 亿条数据已经是拥有很大的业务流量，而在矢量数据库中通常会需求更多数据；同时，一条矢量通常就有很大的规模，例如 ResNet-50 的输出就有 2048 个维度，进一步增加了空间需求。对于海量级别的数据来说，矢量数据库需要有支持分布式扩展的能力。

- 查询方式不同。传统的数据库基于关系模型进行查询，在关系模型中，通过关系表示实体与实体之间的联系，然后基于关系数据集合进行数据的查询、更新以及控制等操作查询。这通常是一种精确查找，即查

询完全符合某种或某些条件的结果。而矢量数据库基于矢量进行查询，其依赖的是矢量之间的度量函数，如余弦相似度等。这通常是一种近似查询，即查询与条件相近的结果，这对计算能力要求非常高。

# 2 相关背景

本章节将介绍矢量数据库的相关背景，包括矢量数据的类型和矢量的相关算法，为后续矢量数据库及其原理讲解作前置铺垫。

## 2.1 矢量的定义和表示

矢量（或称向量），是由数值组成的数据集合，一个矢量表示一个多维空间中的点或特征。每个矢量由一组有序的数值组成，这些数值可以是实数或离散值。在矢量数据中，每个维度代表了矢量的一个特征或属性。例如，如果考虑一个二维矢量数据集，每个矢量可以表示平面上的一个点，其中第一个维度表示横坐标，第二个维度表示纵坐标。一个矢量可以有如下表示：

$$V = \{v_1, v_2, \ldots, v_n\} \in \mathbb{R}^n, \tag{1}$$

如公式 1所示，$n$ 维矢量 $V$ 中包含若干元素 $v_1, v_2, \ldots, v_n$，其中每个元素 $v_i$ 都是一个数值，拥有其特定含义或属性。

在机器学习、深度学习等领域中，矢量数据拥有了更为高维的含义。机器学习通常通过人工标注的特征进行学习，例如可以将图像分解出颜色直方图、纹理特征、形状描述符等特征，之后将这些特征拼接组织成矢量后，就可以进行后续的分类、回归等任务。在深度学习中，矢量的含义则更加难以解释，由于深度学习依赖的特征数量极多，不可能逐一进行人工标注。因此深度学习模型得到的特征矢量往往是自学习的，每个特征拥有不同的含义，维度很多又难以解释。因此，矢量数据的组织和存储是很大的难题。

## 2.2 矢量的类型

从应用领域来看，矢量具有以下典型的类型：

- 图像特征矢量。在计算机视觉中，图像可以表示为一系列特征矢量。每个特征矢量可能代表图像中的某个区域或特定的视觉特征，这些特征

可能是人工的，如颜色直方图、纹理特征或形状描述符，也可能是深度模型自学习的。

- 文档矢量。在文本挖掘和自然语言处理中，文档可以表示为矢量。每个文档矢量可能表示文档中单词的出现频率、TF-IDF 值或其他文本特征，当然也可能是自学习的。
- 用户偏好矢量。在推荐系统中，可以使用用户的行为数据来构建用户偏好矢量。每个用户偏好矢量表示用户对不同项目或特征的偏好程度。
- 传感器数据矢量。在物联网和传感器网络中，传感器收集的数据可以表示为矢量。每个矢量可能包含不同传感器的测量值，如温度、湿度、压力等。
- 基因表达式矢量。在生物信息学中，基因表达数据可以表示为矢量。每个矢量表示基因在不同样本或实验条件下的表达水平。

目前，各种科研和工程领域都需要存储和分析大量矢量数据，因此矢量数据库对于各大领域来说都有很大的需求，前景广泛。

## 2.3 矢量的相关算法

在机器学习中，常见的算法可以分为分类、回归、聚类等任务，这些任务都需要依赖矢量才能进行。例如，分类和回归算法通常需要根据矢量中的特征，经过某种变换映射到标签上；而聚类算法通常需要通过度量矢量之间的关系，将矢量自适应的分为若干类簇。以经典算法 K 近邻分类 [**?**] 为例，对于一个待分类的样本来说，需要计算距离该样本最近的 K 个样本，之后统计这 K 个样本中出现最多的类别，从而认为该样本属于该类别。

图 2是一个典型的 K 近邻分类示例，图中不同的颜色/形状代表不同的类型，待分类对象是绿色圆点。当 $K = 3$ 时，最近的 3 个样本如实线圆圈所示，出现最多的样本为红色三角形，因此待分类对象也被认为是红色三角形。当 $K = 5$ 时情况发生了改变，出现最多的样本是蓝色正方形，因此待分类对象也被认为是蓝色正方形。算法的效果与参数 K 有很大的关系。

那么如何度量两个矢量之间的距离以找到最近的样本？一种常见方案是计算矢量的欧氏距离。

$$D(V^1, V^2) = \sqrt{(v_1^1 - v_1^2)^2 + (v_2^1 - v_2^2)^2 + \ldots + (v_n^1 + v_n^2)^2}, \qquad (2)$$

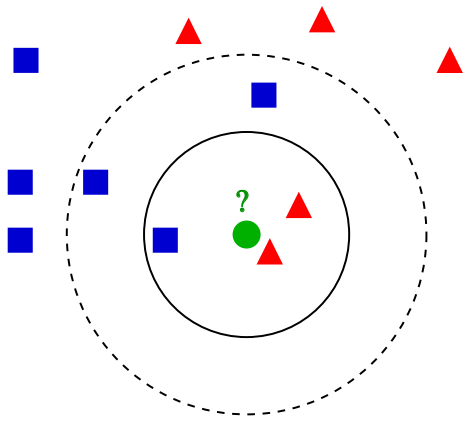



图 2: KNN 分类算法示例

公式 2展示了欧氏距离的计算方式，其中 $V^1, V^2$ 代表两个矢量，$v_i^j$ 代表第 $j$ 个矢量的第 $i$ 个元素。通过对该案例的分析，可以发现机器学习需要对矢量和矢量之间进行大量分析和统计，其余领域也是如此。因此，对于矢量的分析能力也是矢量数据库的一大需求。

# 3 矢量数据库的原理

本章节将介绍矢量数据库的原理，包括矢量索引编排、矢量查询和矢量后处理，从而详细介绍矢量数据库是如何工作的，揭开其神秘面纱。

## 3.1 总览

矢量数据库通过若干算法的组合实现了数据库功能，这些算法组合成一个管道进行运作。如图 3所示，矢量进行索引编排后存入矢量数据库中，之后通过最近邻等方式进行查询，并进行后处理以得到最终结果。


图 3: 矢量数据库的算法管道

其中各个步骤的解释如下：

1. 索引编排。使用特定算法对矢量进行索引，将矢量映射到一个数据结构中，以实现更快的搜索。

2. 矢量查询。矢量数据库将索引的查询矢量与数据集中的索引矢量进行比较，以找到最近的邻居。

3. 矢量后处理。对最近邻矢量进行后处理以返回最终结果，这一步可以包括使用不同的相似性度量对最近的邻居进行重新排序。

### 3.2 矢量索引编排

矢量索引编排用于将矢量映射到一个简化表示，并尽可能保留矢量中的信息，从而使得矢量可以在一个较低的维度进行快速比较。下面介绍几种索引编排算法。

随机投影 [?]。随机投影的思想是通过一个随机数矩阵将矢量映射到目标维度，由于所有矢量通过相同的矩阵进行映射，只要矩阵设计合理，矢量就可以在低维空间中保持有效的关系。该算法可以表示为如下。

$$V \times M \to V', \tag{3}$$

其中 $V \in \mathbb{R}^{K \times D}$ 表示原始矢量集合，$V' \in \mathbb{R}^{K \times D'}$ 表示映射后的矢量集合，$D, D'$ 表示映射前后的维度，$D \gg D'$。通过随机数矩阵 $M \in \mathbb{R}^{D \times D'}$ 与原始矢量集合进行点乘，就可以得到映射后的矢量集合。

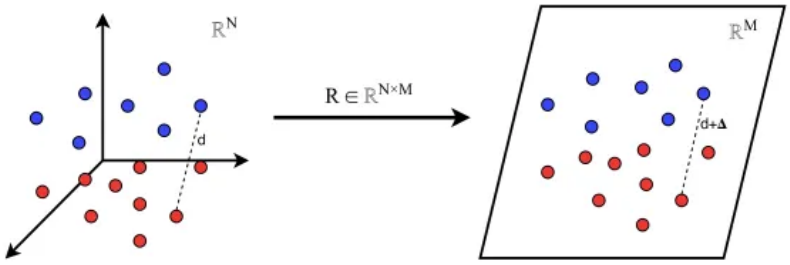


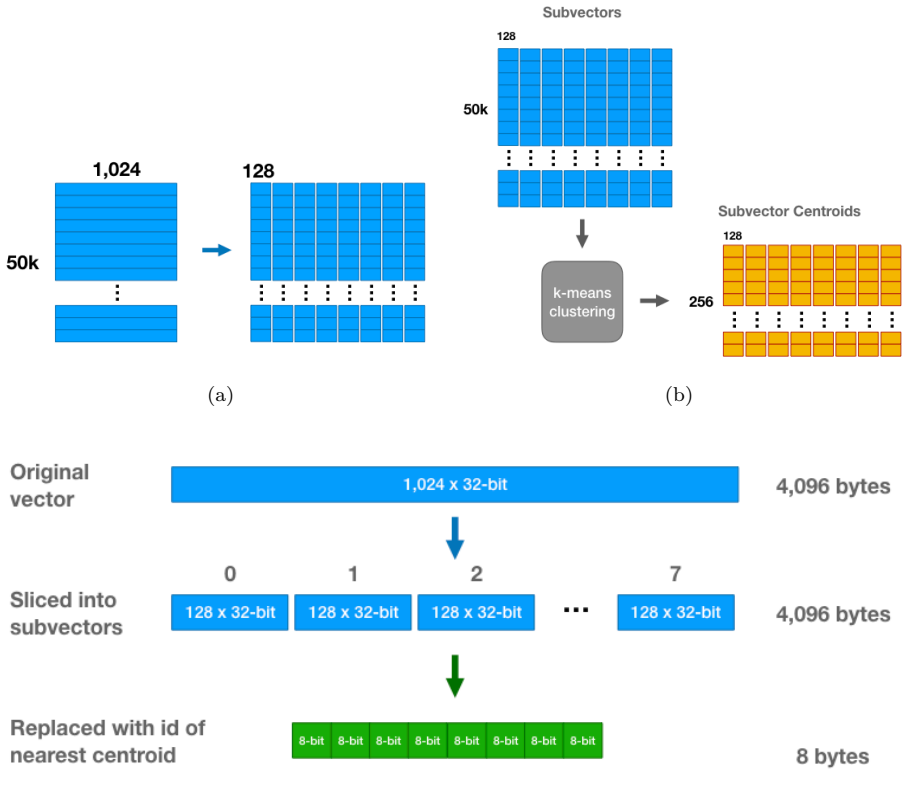

图 4: 随机投影算法

图 4展示了一个随机投影算法的示例，对于一个 $N = 3$ 维的立体数据集合而言，通过矩阵 $R \in \mathbb{R}^{M \times N}$，就可以将数据映射到 $M = 2$ 维的平面数据集合。

乘积量化 [?]。乘积量化是一种矢量有损压缩技术，通过将原始矢量分割为若干块，并为每个块生成代表性的代码来简化表示。其步骤如下：

1. 矢量分割。将矢量分割为若干段。
2. 代码生成。对每个段通过聚类算法得到若干簇，所有簇的聚类中心组成每个段的代码，每个代码拥有一个 ID 号。
3. 矢量查询。将矢量转换为若干代码的表示后，通过代码的 ID 号集合找到最相近的矢量。

该算法的准确率和性能视聚类中心数量而定。聚类中心越多，即聚类越细致时，该算法的查询越准确，然而速度也越慢；反之，虽然速度更快，查询却会更不准确。因此该算法需要根据实际情况进行合适的权衡。




(a) (b)

(c)

图 5: 乘积量化算法

图 5展示了乘积量化算法的具体步骤。子图 $a$ 表示矢量分割，1024 维矢量被分割为 8 个 128 维的矢量段。子图 $b$ 表示代码生成，通过 KMeans 聚类算法将每个段分为 256 个簇，并将簇中心的集合作为代码集合。子图 $c$ 表示矢量查询，原始矢量通过分割和寻找最近邻代码之后得到一个低维表示，256 个簇 ID 只需要 8bit 即可表示，而 8 个段总共只需要 $8 \times 8bit = 64bit$，最后只需要在压缩后的表示空间中寻找最近邻即可。基于乘积量化算法，矢量被大大压缩了。

局部敏感哈希 [?]。局部敏感哈希是一种近似最近邻的算法，对于一般的哈希函数来说，当内容发生微小变化后，哈希值就会发生无法预估的变化。而对于局部敏感哈希来说，内容发生微小变化后，内容也只发生微小变化甚至不变。于是，矢量在哈希空间中被映射到若干桶中，很容易找到最近邻。

一个简单的符合上述条件的哈希函数是：

$$H(V) = |V \cdot R + b|/a, \tag{4}$$

公式 4展示了一个简单的局部敏感哈希函数，其中 $R$ 是一个随机数矩阵，$b$ 是 $[0, a]$ 之间均匀分布的随机变量，$a$ 是桶宽。当 $R \in \mathbb{R}^{D \times 1}$ 时，所有矢量被映射到一条直线上，该直线被划分为若干长度为 $a$ 的线段，每个向量会随机映射到不同的线段上。

还有一种局部敏感哈希函数是基于 bit 采样的哈希函数：

$$H(V) = V[i], \tag{5}$$

公式 5中 $V[i]$ 表示矢量 $V$ 在第 $i$ 位的值，将矢量二值化为元素 0 和 1 组成之后，就可以通过该函数在某个位上比较两者是否相同。

此外还有基于聚类、汉明距离等思想的局部敏感哈希函数，这里就不一一列举了。

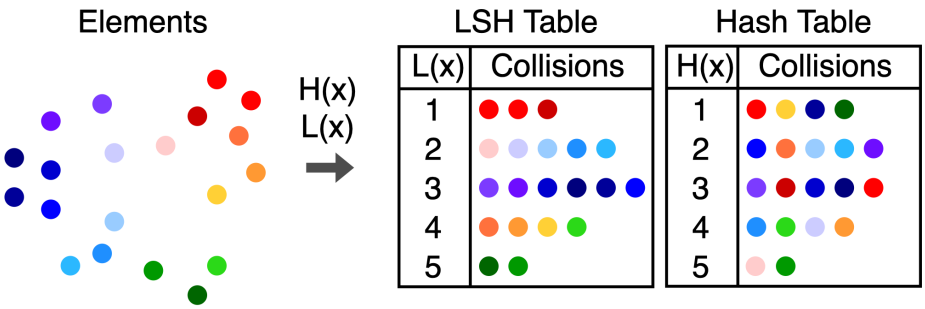



图 6: 局部敏感哈希算法

图 6展示了局部敏感哈希的思想。图中 $H(x)$ 表示普通的哈希函数，经过哈希后，不同哈希值对应的矢量集合很难有规律可循。而 $L(x)$ 表示局部敏感哈希函数，经过哈希后，相似的矢量集合被分配到一起。

### 3.3 矢量查询

矢量查询基于对编排后的索引计算相似度来找到最近邻，一些常见的相似度度量方式有：

- 余弦相似度。测量矢量空间中两个矢量之间的角度的余弦值。它的范围是 $-1$ 到 1，其中 1 代表矢量方向完全相同，0 代表矢量方向正交，$-1$ 代表矢量方向完全相反。

$$\cos(V^1, V^2) = \frac{V^1 \cdot V^2}{\|V^1\|\|V_2\|} = \frac{\sum_{i=1}^{N} v_i^1 v_i^2}{\sqrt{\sum_{i=1}^{N}(v_i^1)^2} \cdot \sqrt{\sum_{i=1}^{N}(v_i^2)^2}}, \tag{6}$$

  公式 6表示了余弦相似度的计算，其中 $V^1, V^2$ 代表两个长度为 $N$ 的矢量，$v_i^j$ 表示第 $j$ 个矢量的第 $i$ 个元素。

- 欧氏距离。测量矢量空间中两个矢量之间的直线距离。它的范围从 0 到无穷大，其中 0 代表矢量完全相同，较大的数值代表矢量越不相似。公式见 2。

### 3.4 矢量后处理

矢量后处理即根据用户需求来过滤需要的结果，用户的需求通常以元数据的形式表示。元数据可能是矢量的一些描述性信息，对于图像数据来说，

元数据可能是照片拍摄时间、地点；对于文本数据来说，元数据则可能是文档来源、作者信息等。总之元数据是由用户需求指定的一些描述性的辅助信息。

为了实现基于元数据的过滤，在维护一组矢量索引以外，矢量数据库还需要维护一组元数据索引，并根据元数据执行过滤。矢量数据库的过滤方法在步骤上可以分为两种：预过滤和后过滤。

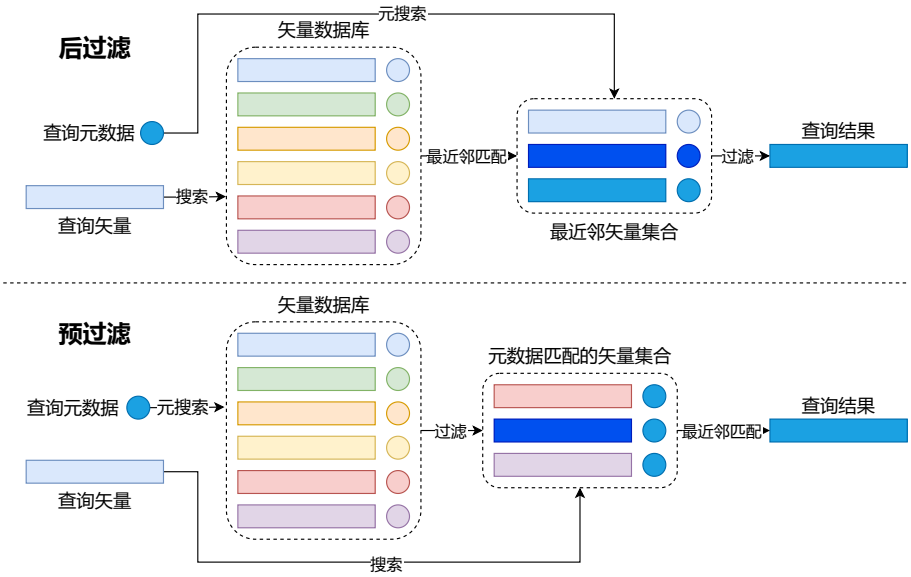



图 7: 矢量后处理流程

图 7展示了矢量的后处理流程，其方法有两种：

- 预过滤。元数据过滤在矢量搜索之前进行。虽然这有助于减少搜索空间，但它也可能导致系统忽略那些不符合元数据过滤标准的相关结果。
- 后过滤。元数据过滤在矢量搜索之后进行。这有助于确保所有相关的结果都被考虑在内，但它也可能引入额外的开销并减慢查询过程，因为不相关的结果需要在搜索完成后被过滤掉。

预过滤和后过滤策略都有各自的优缺点，需要视实际情况而定。

# 4 矢量数据库系统

这一章将介绍矢量数据库系统，包括矢量数据库的架构、存储管理、安全管理和一些常见的矢量数据库系统。

## 4.1 矢量数据库的架构

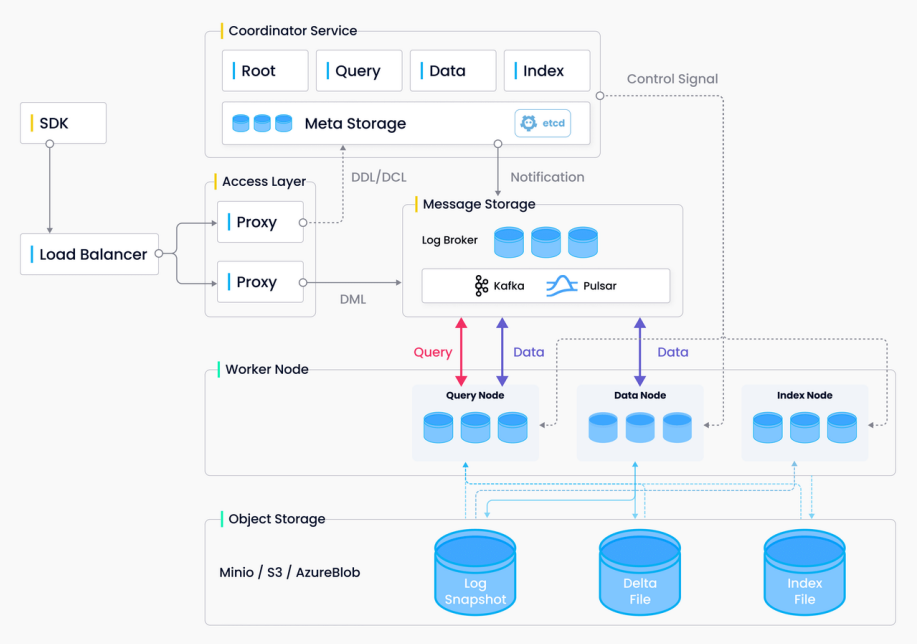



图 8: Milvus 矢量数据库的架构

图 8展示了 Milvus[?]——一个现代的矢量数据库的完整架构。Milvus 是一个支持数据分片、数据持久性、向量与标量混合搜索等功能的矢量数据库，采用了共享存储架构，对计算节点实现了存储和计算的分离和水平可扩展性。由于不同矢量数据库可能存在完全不同的架构，这里就以 Milvus 为例，介绍矢量数据库的基本架构。

Milvus 采用分层架构，将整个数据库系统划分了 4 个层：

- 访问层。访问层包含一系列无状态代理服务，是系统与用户访问之间的接口。访问层采用负载均衡组件（如 Nginx、Kubernetes Ingress 等）为用户提供一个统一的服务地址，并将请求分配到各个节点上。

- 协调服务层。协调服务层是系统的大脑，负责集群拓扑管理、负载均衡、时间戳生成、数据管理等，负责分配任务给系统节点。协调服务层中包括 4 个协调器：
  - 根协调器，负责创建和删除节点等。
  - 查询协调器，负责查询节点的拓扑结构和负载均衡。
  - 数据协调器，负责维护元数据，控制后台数据的刷写、合并等操作。
  - 索引协调器，负责构建和维护索引。
- 工作节点层。工作节点层是系统中负责执行的部件，专注进行计算工作，包括 3 个节点：
  - 查询节点，负责加载数据并执行向量/标量混合搜索。
  - 数据节点，负责获取日志，打包快照并进行持久化。
  - 索引节点，负责建立索引。
- 持久化层。持久化层负责数据的持久化，包含元数据存储、日志代理和对象存储。

总之，现代的矢量数据库系统基本都支持分布式架构，并与分布式文件系统、对象存储等云技术结合。这是因为深度学习、大数据等领域等对大规模矢量的存储需求，单机难以承担昂贵的需求，必须要采用分布式的架构进行存储。此外，现代的矢量数据库系统通常支持多样化的功能，如矢量/标量混合查询、不同度量方式的相似度搜索等，满足用户的多样化需求。

## 4.2 矢量数据库的存储管理

基于分布式架构，数据的存储也出现各种各样的问题。可以想象的是，节点越多时，部分节点出现错误和故障的可能性就越大。因此，现代的矢量数据库必须能在各个节点随时可能宕机的情况下正常工作，满足包括查询需要尽快返回、数据不能丢失、部分节点宕机时也能正常工作等复杂需求，这对现代的矢量数据库系统提出非常高的要求。为了实现上述功能，下面将介绍现代的矢量数据库系统的存储管理策略。

Milvus1.x 版本采用共享存储策略，即通过多台硬盘驱动器组成阵列来提高容量和数据安全性。一个经典的共享存储技术是 RAID[?]，该技术通过分片和校验等技术实现高效的读写和安全管理。

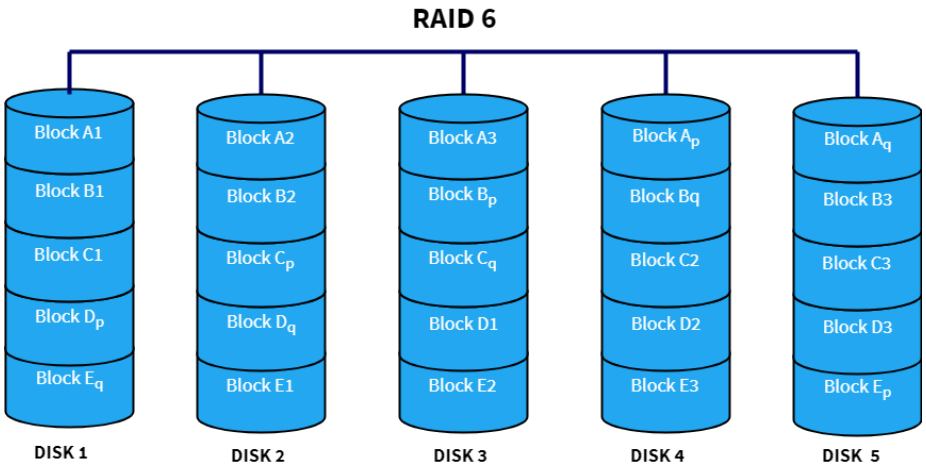



图 9: RAID 阵列技术

如图 9所示，RAID 技术将若干硬盘组成阵列，对于一条数据来说，它会被拆分为若干片，存入不同的硬盘中，这样读写数据时就可以同时读写多个硬盘，大大提高速度。另一方面，如果部分硬盘出现故障，RAID 还可以通过奇偶校验 [?] 或异或校验等技术来恢复故障硬盘的数据。RAID 6 阵列中最多支持 2 个硬盘发生故障，这大大提升了分布式系统的可用性。

Milvus2.0 版本则直接采用云原生架构，采用云对象存储和分布式文件系统服务，由云服务厂商保障数据的安全性。

## 4.3 矢量数据库的安全管理

为了有效地管理和维护一个矢量数据库，我们需要一个强大的监控系统来跟踪数据库的性能、健康和整体状态的重要方面。监测对于检测潜在的问题、优化性能和确保顺利的生产运营至关重要。通常一个矢量数据库需要支持如下数据的监控：

- 资源使用情况。监测资源使用情况，如 CPU、内存、磁盘空间和网络活动，可以识别可能影响数据库性能的潜在问题或资源限制。

- 查询性能。查询延迟、吞吐量和错误率可能表明需要解决的潜在系统性问题。
- 系统健康。整体系统健康监测包括单个节点、复制过程和其他关键组件的状态。

此外，为了保证用户安全，矢量数据库需要支持用户权限的访问控制，确保只有授权用户才有能力查看、修改、存储库中的部分敏感数据，避免数据丢失和财产损失。

最后，为了保证数据安全，矢量数据库需要提供定期备份的功能，备份数据存储在外部系统中，与数据库存储分离，确保数据的安全性和可恢复性。当数据发生丢失和损坏时，这些备份可以用来将数据库恢复到以前的状态，最大限度地减少停机时间和对整个系统的影响。

Milvus 支持了上述提到了监控、权限管理、数据备份等功能，拥有丰富的安全管理机制，能够保证安全性。

## 4.4 一些常见的矢量数据库系统

以下是一些流行的矢量数据库系统：

- Milvus。Milvus 是一个开源矢量数据库，可以管理万亿矢量数据集，支持多种矢量搜索索引和内置过滤。
- Pinecone。Pinecone 是一个专为机器学习应用程序设计的矢量数据库。它速度快、可扩展，并支持多种机器学习算法。Pinecone 建立在 Faiss 之上，Faiss 是一个用于密集向量高效相似性搜索的库。
- Faiss。Faiss 库是由 Facebook 开发的适用于稠密向量匹配的开源库，支持多种向量检索方式，包括内积、欧氏距离等，同时支持精确检索与模糊搜索.
- Annoy。Annoy 是一个 C++ 库，用于在高维空间中搜索最近邻居。它支持欧几里得距离和曼哈顿距离，并且可以使用多个 CPU 核心进行计算。
- Hnswlib。Hnswlib 是一种快速、可扩展和高效的近似最近邻居搜索库。它支持多线程计算，并且可以在大型数据集上进行分布式计算。

# 5 矢量数据库的应用场景

矢量数据库的最典型应用场景，就是与深度学习模型相结合，基于深度学习模型将数据变换为矢量后，存储在数据库中，之后利用数据库系统的内置函数进行匹配。

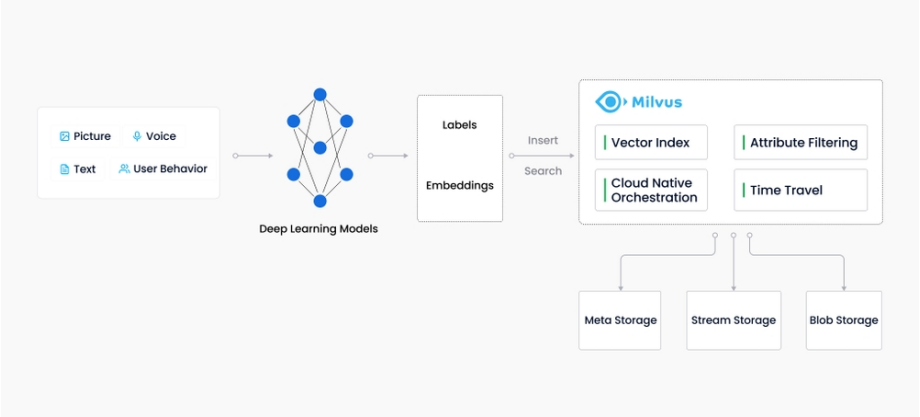



图 10: 矢量数据库 Milvus 的应用

如图 10所示，图像、语音、文本甚至用户行为等信息都可以通过深度学习模型编码成矢量，并与标签（即元数据）附加在一起送入矢量数据库 Milvus 中进行存储。之后，就可以调用 Milvus 的一系列方法进行任务。

这里将介绍矢量数据库的 2 种典型应用场景，图像检索和搜索引擎。

## 5.1 图像检索

图像检索指的是通过指定的一些信息，从大量图像种找到最适合的一系列图像。指定的信息可以是文本、图像或其他信息，这里介绍一个以图搜图的案例。

使用现有模型 VGG[?]/ResNet[?] 等深度学习模型，可以将图像编码成矢量的形式。图 11展示了 VGG 的模型结构，输入尺寸为 $224 \times 224$ 的图像，VGG 通过若干层卷积、池化和线性变换操作将其映射到一个长度为 4096 的矢量。

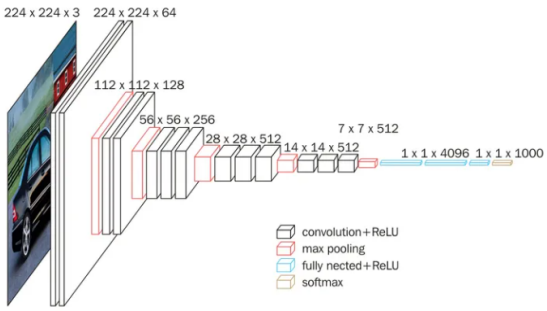



图 11: VGG 模型

我们可以将大量图像通过 VGG 转换为矢量之后存入矢量数据库中，之后对于一张新的图像，只需要也将其转换为矢量，就可以利用矢量数据库找到其中最相近的图像。图 12展示了这一详细流程。

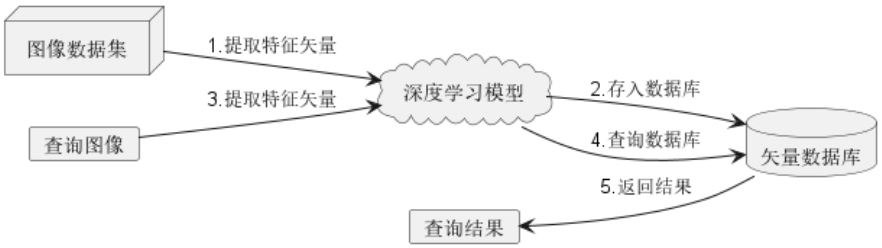



图 12: 基于矢量数据库的图像检索工作流程

基于 Milvus，可以很容易实现上述功能。Milvus 提供了丰富的接口，供用户上传和查询矢量，下面是一个简单的以图搜图案例展示。如图 13所示，上传一张猫的图像，该系统可以帮助我们找到相似的猫的图像，并返回相似度信息。

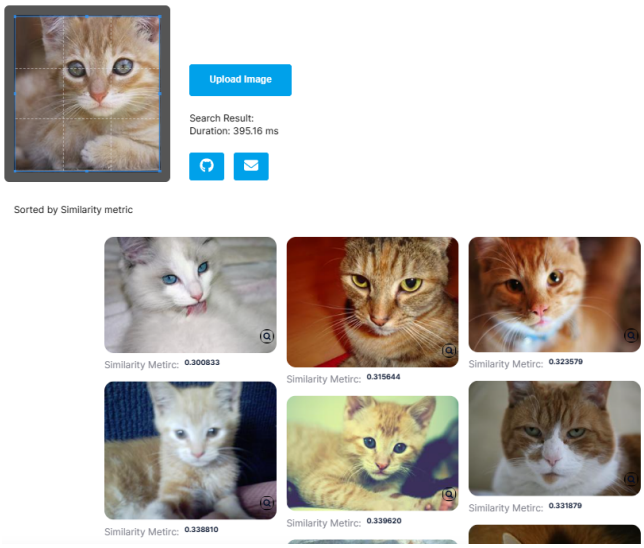



图 13: 以图搜图案例

## 5.2 搜索引擎

同样的，矢量数据库可以用作搜索引擎，只需将大量文本内容编码为矢量后存入数据库，再将搜索内容同样编码为矢量之后，与库中进行匹配，找到最相似的矢量对应的文本内容即可。

为了实现文本内容的编码，这里介绍一个经典的自然语言处理模型 BERT[?]。如图 14所示，BERT 模型能够将输入的文本序列编码为一组深度的 token 序列。

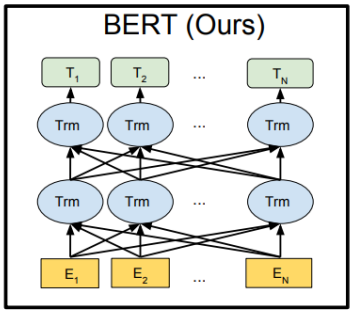



图 14: BERT

于是，基于矢量数据库的搜索引擎可以以如下形式构建。如图 15所示，大量文档的标题通过 BERT 模型编码为矢量后，存入 Milvus 矢量数据库中（蓝色线条）。之后待搜索文本同样通过 BERT 编码为矢量后，就可以在 Milvus 中寻找匹配矢量（橙色线条）。需要注意的是，这里还引入了关系型数据库 PostgreSQL 来保存文档本身的内容，Milvus 会返回文档号，之后就可以通过文档号在关系型数据库中获取文档本身的内容。

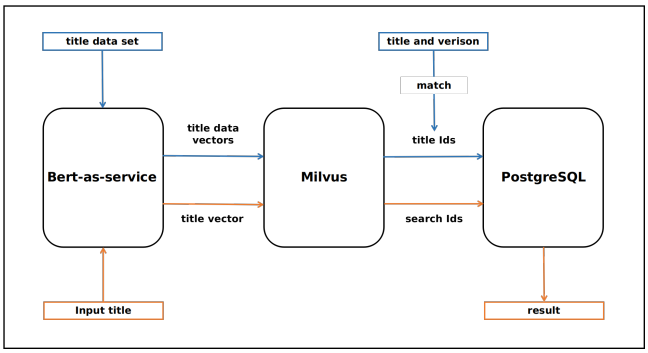



图 15: 基于矢量数据库的搜索引擎工作流程

这里同样展示了一个搜索引擎的简单案例，如图 16所示，输入文本内，该案例可以返回相似的一些文档内容。


图 16: 搜索引擎案例

# 6 结语

本文对矢量数据库做了详尽的介绍，包括其发展历史、矢量的含义、矢量数据库的工作原理、矢量数据库系统和矢量数据库的应用场景。矢量数据库是一种存储和检索向量数据的数据库，目前，主流的矢量数据库都支持向量存储和检索，具有高速、准确、可扩展等特性。总体来说，矢量数据库在人工智能领域中应用广泛，可以实现快速、直观、无缝的信息检索。

可以预见的是，矢量数据库未来会有如下发展：

- 更好的可扩展性和容错性。目前主流的矢量数据库支持通过水平扩展来提高性能，但是在大规模数据集上的性能仍然需要改进。因此，未来的版本可能会更加注重可扩展性和容错性。
- 更多的功能。目前主流的矢量数据库支持基本的向量存储和检索功能，但是未来的版本可能会增加更多的功能，例如支持图像、文本等非向量数据类型。
- 更好的用户体验。目前主流的矢量数据库还有一些使用上的限制，例如需要手动调整参数等。未来的版本可能会更加注重用户体验，提供更加友好的界面和自动化工具。

总体来说，矢量数据库在人工智能领域中应用广泛，可以实现快速、直观、无缝的信息检索。未来，随着人工智能技术的不断发展，矢量数据库将会更加智能化和高效化，为人工智能应用提供更加优质的服务。