VECTOR DATABASE.

Trước đây các cơ sở dữ liệu thường được lưu trữ dưới dạng sql và nosql, tuy nhiên các dữ liệu được lưu trữ dưới dạng này không phù hợp để hỗ trợ cho sự phát triển của học máy.

Ví dụ: Cơ sở dữ liệu SQL (DB) hoạt động với dữ liệu có cấu trúc, sử dụng các tính năng như khóa chính, bảng (hàng và cột) cũng như các thuộc tính để mô tả, lưu trữ và sắp xếp dữ liệu. Một trong những điểm yếu của cơ sở dữ liệu SQL là không có khả năng mở rộng quy mô theo chiều ngang để hỗ trợ khối lượng công việc ngày càng tăng.

Còn với cơ sở dữ liệu NoSQL mặc dù chúng hoạt động với dữ liệu bán cấu trúc và phi cấu trúc, ở dạng kho tài liệu, kho khóa-giá trị, cơ sở dữ liệu đồ thị và cơ sở dữ liệu hướng cột. Điểm yếu về khả năng mở rộng quy mô theo chiều ngang, cũng như hỗ trợ công việc theo nhiều hướng của cơ sở dữ liệu SQL đã được khắc phục.

Xong dù cơ sở dữ liệu SQL và NoSQL có thể hoạt động trong một số trường hợp sử dụng ML, nhưng cơ sở dữ liệu vectơ phù hợp hơn với các trường hợp sử dụng liên quan đến văn bản, tìm kiếm, đề xuất, âm thanh và NLP.

Ví dụ ta có cơ sở dữ liệu về thông tin các bài hát trên youtube, khi ta có một bài hát và muốn tìm bài hát tương tự như thế. Nếu chúng ta sử dụng dữ liệu SQL, NoSQL để tìm kiếm sự tương đồng thì khá là khó khi chúng ta chỉ có thể tìm sự tương đồng thông qua các mục như thể loại, đạo diễn, … Mà điều này cho kết quả cũng không thực sự tốt.

Thông qua câu lệnh so sánh của SQL thì ta còn cần phải biết rõ nội dung được lưu trữ để tìm kiếm. Nếu lưu trữ thể loại phim là hài hước thì khi tìm kiếm ta phải sử dụng lệnh Where theloai=”hai huoc”. Chứ không thể sử dụng các từ đồng nghĩa của hài hước như vui nhộn, dí dỏm,…

Hơn nữa, SQL và NoSQL không thể hỗ trợ cho các kiểu dữ liệu phi cấu trúc như văn bản, âm thanh, hình ảnh, video,…

Vector database sẽ khắc phục được những nhược điểm này. Để tìm hiểu về cơ sở dữ liệu vector (vector database) trước tiên chúng ta cần biết về dữ liệu dạng vector (vector embedding) cũng như công dụng của nó.

1. Vector embedding

Với sự phát triển của AI ngày nay, các mô hình AI có thể trả lời thông tin chuyển tiếp qua lại giữa các dữ liệu phi cấu trúc như ảnh, âm thanh, văn bản, … Để thực hiện điều này, trước tiên cần chuyển đổi các dạng dữ liệu khác nhau đó về cùng một kiểu dữ liệu chung. Cụ thể phương pháp hiện tại của các mô hình AI đó là chuyển tất cả các dữ liệu trên về dạng vector thông qua một quá trình nhúng (embedding).

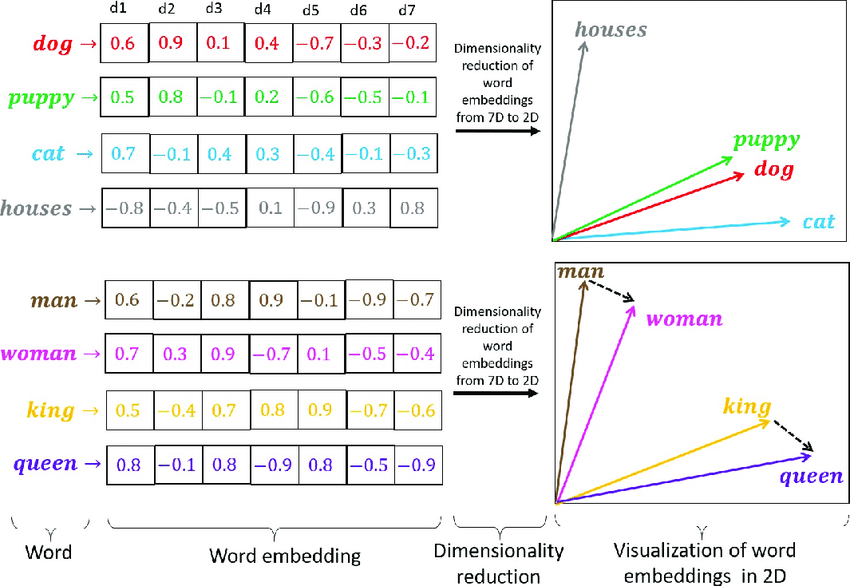
Embedding vector là vector thường có nhiều chiều giúp biểu diễn thông tin đặc trưng cho một đối tượng đầu vào như ảnh, một câu, một đoạn âm thanh,.... Chúng được sinh ra bởi các mô hình AI hay còn gọi là embedding model được huấn luyện trên rất nhiều dữ liệu. Thường thì các model này sẽ là một mạng thần kinh.

A diagram of a deep learning model

Description automatically generated

Cụ thể với các dữ liệu dạng text ta cần text embedding model, dạng videos ta cần video embedding model,….

Do embedding vector biểu diễn đặc trưng của một đối tượng đầu vào nên các đối tượng giống nhau hoặc cùng thể loại, có mối liên hệ gần với nhau sẽ có khoảng cách giữa các vector gần nhau.



Vì tính chất đó của embedding vector, khi sử dụng tìm kiếm đầu tiên chúng ta sẽ chuyển đổi tất cả dữ liệu thông qua embedding model về dạng embedding vector sau đó thực hiện một thuật toán khoảng cách (có thể là Euclidean, Cosine,…) để chọn ra các vector có khoảng cách gần nhất với dữ liệu đầu vào.

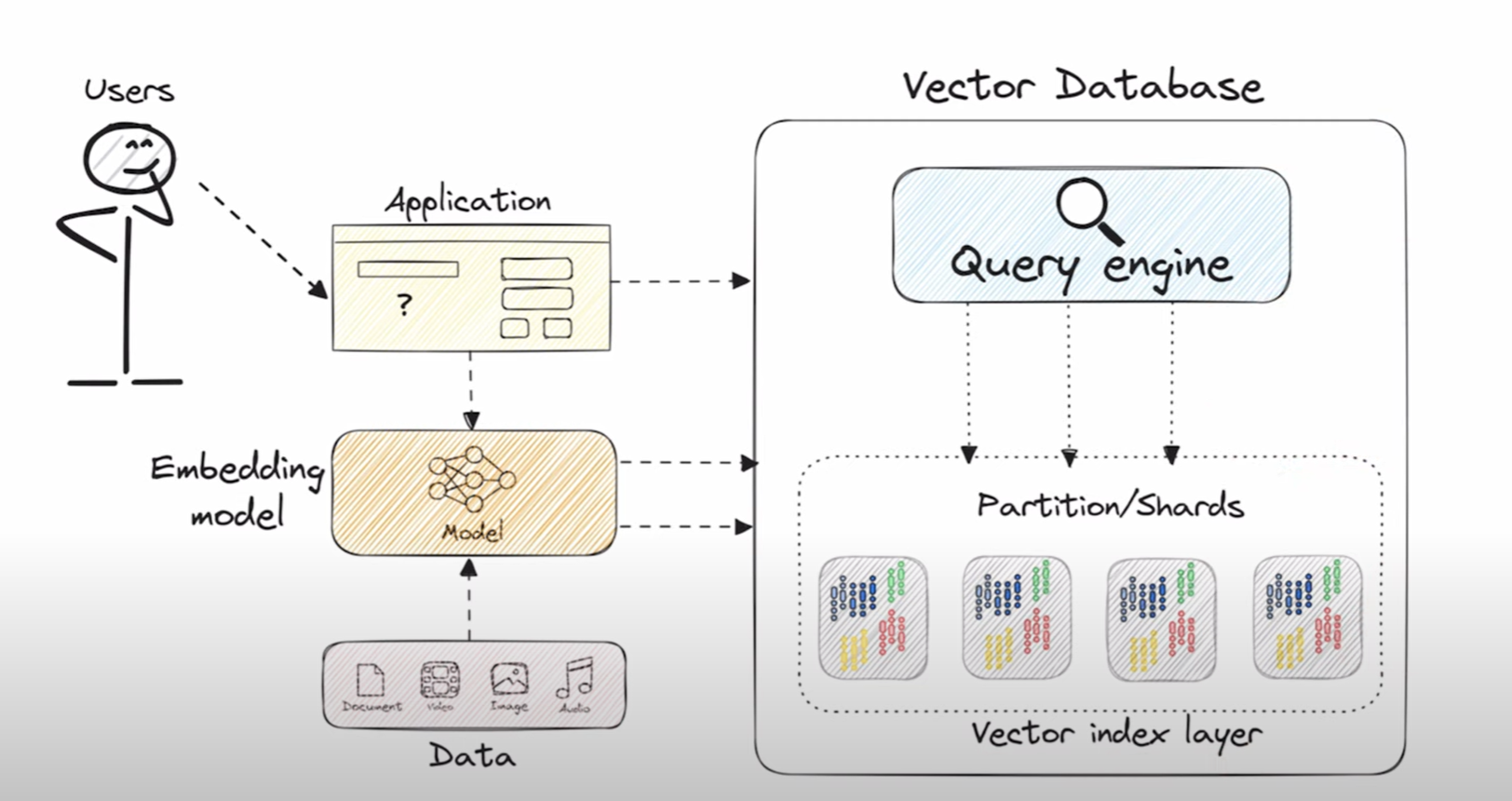
A diagram of a diagram

Description automatically generated with medium confidence

1. Vector database.

Vector database chứa các vector được chuyển đổi từ dữ liệu ảnh, video, text và quy định cách tính khoảng cách giữa 2 vector.

Cách thức hoạt động cơ bản của vector database



Đầu tiên, có 1 lượng lớn dữ liệu data được thông qua embedding model để chuyển về dạng vector và tạo thành các vector thêm vào vector database.

Để có thể thực hiện được việc tìm kiếm và lấy ra các dữ liệu thì chúng ta phải thực hiện index các vector. Việc index sẽ giúp chúng ta so sánh được khoảng cách một cách nhanh nhất sau này.

Sau khi có vector database hoàn chỉnh, người dùng sẽ sử dụng một ứng dụng nào đó để tìm kiếm thông qua ngôn ngữ tự nhiên bằng một câu văn hỏi.

Và câu hỏi này cũng sẽ được chuyển đổi về dạng vector thông qua embedding model. Sau đó dựa vào vector của câu hỏi mô hình AI sẽ tìm kiếm các vector có khoảng cách gần nhất với nó có trong tất cả các vector có trong database và trả về câu trả lời.

1. Ứng dụng của vector database.

* Hỗ trợ việc tìm kiếm tương tự hiệu quả. Ví dụ như tìm các sản phẩm tương tự trong ứng dụng mua sắm.
* Và ứng dụng đặc biệt khiến vector database thu hút được sự chú ý và đầu tư lớn đó là khả năng tăng cường dữ liệu cho các mô hình ngôn ngữ lớn. A screenshot of a black background with white text

  Description automatically generated

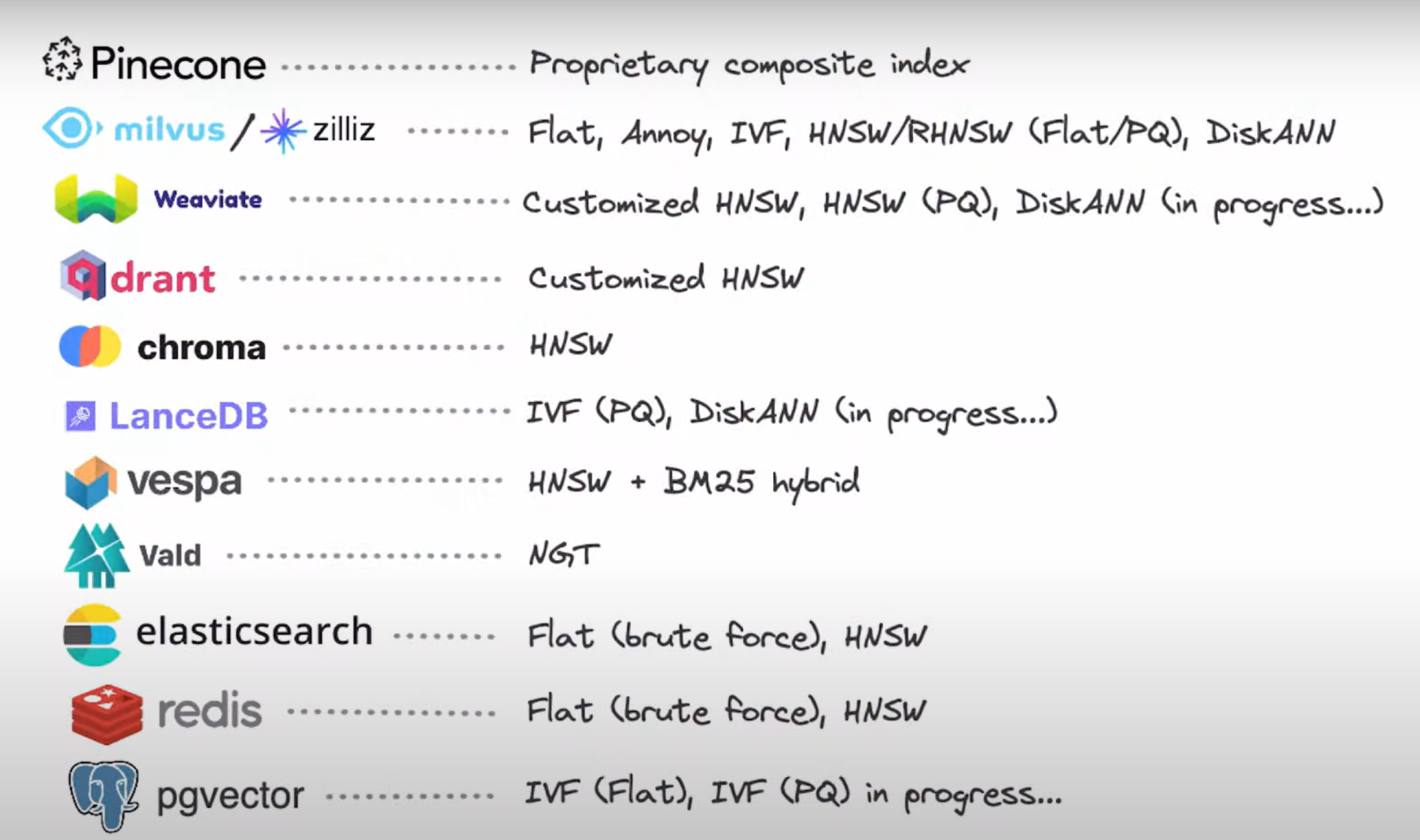
Ví dụ ChatGPT chỉ có dữ liệu đến năm 2021, nó sẽ không thể trả lời các dữ liệu ở thời điểm mới, từ 2022 đến nay. Và vector database sẽ giúp chatgpt khắc phục được dữ liệu này, thông qua việc thêm những dữ liệu mới vào trong mô hình.

Ví dụ về vector database trong mô hình ngôn ngữ lớn LLM.

A diagram of a software development process

Description automatically generated with medium confidence

1. Các loại vector database.



Đây là các loại vector database và giải thuật tìm khoảng cách giữa vector đầu vào và vector trong database.

Các giải thuật này sẽ quyết định thời gian đưa ra kết quả.

1. Các thuật toán tìm kiếm vector (search vector)

Tìm kiếm vectơ, còn được gọi là tìm kiếm tương tự vectơ hoặc tìm kiếm lân cận gần nhất, là một kỹ thuật được sử dụng trong hệ thống truy xuất dữ liệu và truy xuất thông tin để tìm các mục hoặc điểm dữ liệu tương tự hoặc liên quan chặt chẽ với một vectơ truy vấn nhất định.

Trong tìm kiếm vectơ, chúng ta biểu thị các điểm dữ liệu, chẳng hạn như hình ảnh, văn bản và âm thanh, dưới dạng vectơ trong không gian nhiều chiều. Mục tiêu của tìm kiếm vectơ là tìm kiếm và truy xuất một cách hiệu quả các vectơ phù hợp nhất tương tự hoặc gần nhất với vectơ truy vấn. Thông thường, các số liệu khoảng cách như khoảng cách Euclide hoặc độ tương tự cosin đo lường độ tương tự giữa các vectơ. Độ gần của vectơ trong không gian vectơ quyết định mức độ giống nhau của nó.

Để tổ chức và tìm kiếm vectơ một cách hiệu quả, thuật toán tìm kiếm vectơ sử dụng các cấu trúc lập index như cấu trúc dựa trên cây hoặc kỹ thuật băm. Tìm kiếm vectơ có nhiều ứng dụng khác nhau, bao gồm hệ thống đề xuất, truy xuất hình ảnh và video, xử lý ngôn ngữ tự nhiên, phát hiện bất thường cũng như chatbot hỏi đáp. Sử dụng tìm kiếm vectơ giúp bạn có thể tìm thấy các mục, mẫu hoặc mối quan hệ có liên quan trong dữ liệu nhiều chiều, cho phép truy xuất thông tin chính xác và hiệu quả hơn.

* Linear search

Thuật toán tìm kiếm lân cận gần nhất đơn giản nhất nhưng hiệu quả kém nhất là tìm kiếm tuyến tính: tính khoảng cách từ vectơ truy vấn đến tất cả các vectơ khác trong cơ sở dữ liệu vectơ.

Vì lí do dữ liệu, linear search không hoạt động khi cố gắng mở rộng cơ sở dữ liệu vectơ lên hàng chục hoặc hàng trăm triệu vectơ. Nhưng khi tổng số phần tử trong cơ sở dữ liệu nhỏ, đây thực sự có thể là cách hiệu quả nhất để thực hiện tìm kiếm vectơ vì không cần cấu trúc dữ liệu riêng cho chỉ mục, trong khi việc chèn và xóa có thể được thực hiện khá dễ dàng.

Do không cần độ phức tạp về việc phân vùng không gian cũng như chi phí không gian không đổi liên quan đến tìm kiếm đơn giản, phương pháp này thường có thể hoạt động tốt hơn phân vùng không gian ngay cả khi truy vấn trên số lượng vectơ vừa phải.

* Space partitioning (Phân vùng không gian)

Space partitioning là một nhóm thuật toán sử dụng cùng một khái niệm.

K-dimensional trees (kd-trees) có lẽ là cây nổi tiếng nhất trong họ này và hoạt động bằng cách liên tục chia đôi không gian tìm kiếm (chia vectơ thành các nhóm “trái” và “phải”) theo cách tương tự như cây tìm kiếm nhị phân .

* Quantization (Lượng tử hóa)

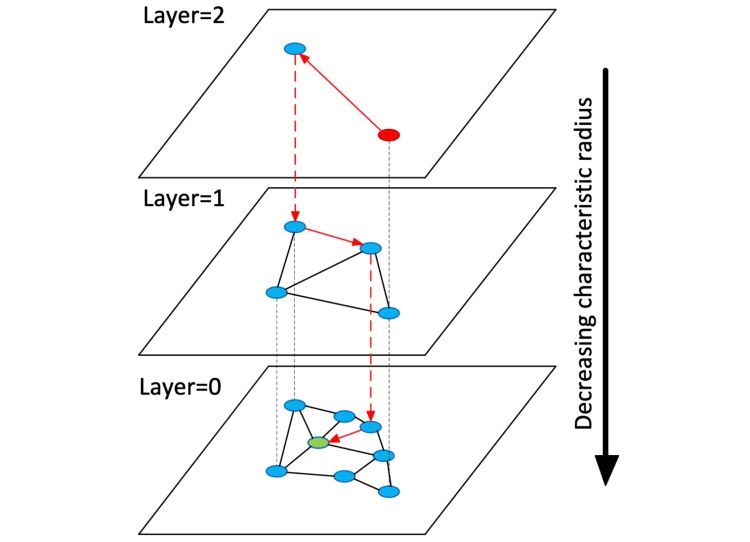
Quantization là một kỹ thuật nhằm giảm tổng kích thước của cơ sở dữ liệu bằng cách giảm độ chính xác của vectơ.

Ví dụ: Một kĩ thuật Quantization là sử dụng việc nhân vector với đại lượng vô hướng để chuyển các phần tử của vector sang số nguyên gần nhất chúng. Điều này giúp giảm kích thước của cơ sở dữ liệu (ví dụ chuyển từ kiểu float sang kiểu int), đồng thời giúp tăng tốc độ tính toán khoảng cách vector.

* Hierarchical Navigable Small Worlds (HNSW)

HNSW là một thuật toán truy xuất và lập index dựa trên đồ thị.

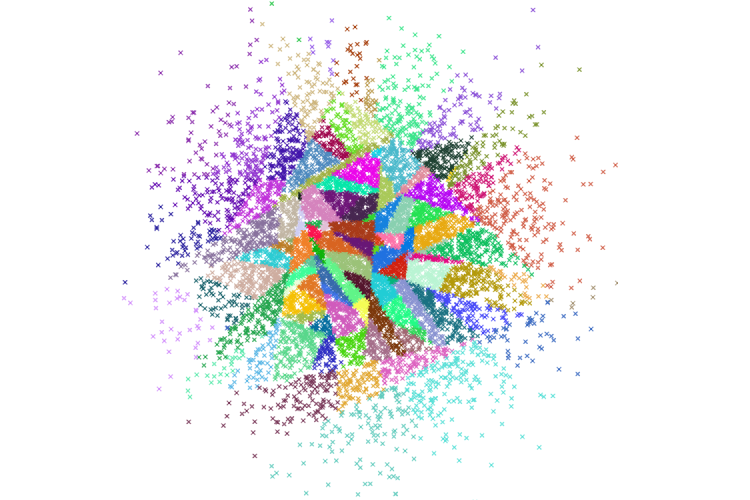
HNSW tạo biểu đồ nhiều lớp từ dữ liệu gốc. Các lớp trên chỉ chứa "kết nối dài" trong khi các lớp dưới chỉ chứa "kết nối ngắn" giữa các vectơ trong cơ sở dữ liệu. Các kết nối biểu đồ riêng lẻ được tạo theo danh sách bỏ qua. Với kiến ​​trúc này, việc tìm kiếm trở nên khá đơn giản – chúng ta nhanh chóng duyệt qua biểu đồ trên cùng (biểu đồ có kết nối giữa các vectơ dài nhất) để tìm vectơ gần nhất với vectơ truy vấn của chúng ta. Sau đó, chúng tôi thực hiện tương tự cho lớp thứ hai, sử dụng kết quả tìm kiếm của lớp đầu tiên làm điểm bắt đầu. Điều này tiếp tục cho đến khi chúng tôi hoàn thành việc tìm kiếm ở lớp dưới cùng, kết quả của nó sẽ trở thành lân cận gần nhất của vectơ truy vấn.



* Approximate Nearest Neighbors Oh Yeah (ANNOY)

ANNOY thực ra khá đơn giản – cây nhị phân.

ANNOY hoạt động bằng cách trước tiên chọn ngẫu nhiên hai vectơ trong cơ sở dữ liệu và chia đôi không gian tìm kiếm dọc theo siêu phẳng ngăn cách hai vectơ đó. Việc này được thực hiện lặp đi lặp lại cho đến khi có ít hơn một số tham số được xác định trước số vector quy định. Vì chỉ mục kết quả về cơ bản là một cây nhị phân, điều này cho phép chúng ta thực hiện tìm kiếm theo độ phức tạp O(log n).



1. Một vài loại vector database hàng đầu hiện nay có mã nguồn mở.

* Milvus

Milvus là một cơ sở dữ liệu vectơ mã nguồn mở được xây dựng để hỗ trợ việc nhúng các ứng dụng AI và tìm kiếm tương tự. Milvus giúp việc tìm kiếm dữ liệu phi cấu trúc trở nên dễ tiếp cận hơn và cung cấp trải nghiệm người dùng nhất quán bất kể môi trường triển khai.

Được xây dựng dựa trên các thư viện tìm kiếm vectơ phổ biến bao gồm Faiss, Annoy, HNSW, v.v.,

Milvus được thiết kế để tìm kiếm sự tương tự trên các tập dữ liệu vectơ dày đặc chứa hàng triệu, hàng tỷ hoặc thậm chí hàng nghìn tỷ vectơ.

Milvus cũng hỗ trợ phân chia dữ liệu, lưu trữ dữ liệu, nhập dữ liệu trực tuyến, tìm kiếm kết hợp giữa dữ liệu vectơ và vô hướng cũng như nhiều chức năng nâng cao khác. Nền tảng này cung cấp hiệu suất theo yêu cầu và có thể được tối ưu hóa để phù hợp với bất kỳ loại truy xuất nhúng nào.

Các tính năng chính:

* Tìm kiếm mili giây trên tập dữ liệu vectơ nghìn tỷ: Độ trễ trung bình được đo bằng mili giây trên bộ dữ liệu vectơ nghìn tỷ.
* Quản lý dữ liệu phi cấu trúc được đơn giản hóa:

+ API phong phú được thiết kế cho quy trình làm việc khoa học dữ liệu.

+ Trải nghiệm người dùng nhất quán trên máy tính xách tay, cụm cục bộ và đám mây.

+ Tìm kiếm và phân tích theo thời gian thực với hầu hết mọi ứng dụng.

* Đáng tin cậy: Các tính năng sao chép và chuyển đổi dự phòng/dự phòng tích hợp của Milvus đảm bảo dữ liệu và ứng dụng có thể duy trì hoạt động kinh doanh liên tục trong trường hợp bị gián đoạn.
* Khả năng mở rộng và mềm dẻo cao:

+ Khả năng mở rộng ở cấp độ thành phần giúp có thể tăng và giảm quy mô theo yêu cầu.

+ Milvus có thể tự động điều chỉnh quy mô ở cấp thành phần tùy theo loại tải, giúp việc lập kế hoạch tài nguyên hiệu quả hơn nhiều.

* Hỗ trợ tìm kiếm các kiểu ngoài dạng vector:

+ Milvus còn hỗ trợ các loại dữ liệu như Boolean, số nguyên, số dấu phẩy động, v.v. Một bộ sưu tập trong Milvus có thể chứa nhiều trường để cung cấp các tính năng hoặc thuộc tính dữ liệu khác nhau.

* Sử dụng cấu trúc Lambda hợp nhất:

+ Milvus kết hợp xử lý luồng và hàng loạt để lưu trữ dữ liệu nhằm cân bằng tính kịp thời và hiệu quả.

=> Giao diện hợp nhất của nó làm cho việc tìm kiếm vector tương tự trở nên dễ dàng.

* Chroma.

Chroma là cơ sở dữ liệu embedding mã nguồn mở. Chroma giúp việc xây dựng ứng dụng LLM trở nên dễ dàng bằng cách cập nhật kiến thức, sự kiên, kết nối qua LLMs.

* Chroma cung cấp cho bạn các công cụ để:

+ lưu trữ các embedding và bối cảnh của chúng

+ nhúng tài liệu và truy vấn

+ tìm kiếm nhúng

* Chroma ưu tiên:

+ Sự đơn giản và năng suất của nhà phát triển

+ Phân tích những kết quả tìm kiếm hàng đầu.

+ Tốc độ tìm kiếm nhanh.

* Có thể lưu trữ dữ liệu trên cloud hoặc tại chỗ.
* Hỗ trợ nhiều loại và định dạng dữ liệu.
* Hoạt động đặc biệt tốt trong dữ liệu âm thanh.
* Weaviate
* Là cơ sở dữ liệu vectơ nguồn mở có thể sử dụng làm giải pháp tự lưu trữ hoặc được quản lý hoàn toàn.
* Weaviate cho phép lưu trữ và truy xuất các đối tượng dữ liệu dựa trên các thuộc tính ngữ nghĩa của chúng bằng cách lập chỉ mục cho chúng bằng vectơ .
* Weaviate có thể được sử dụng độc lập (hay còn gọi là bring your vectors – tức là chúng ta thực hiện embedding rồi đưa vector vào Weaviate lưu trữ) hoặc với nhiều mô-đun khác nhau có thể thực hiện vectơ hóa cho và mở rộng các khả năng cốt lõi.
* Weaviate có GraphQL-API để truy cập dữ liệu một cách dễ dàng.
* Weaviate rất nhanh.
* Kết hợp tìm kiếm vectơ và vô hướng: Weaviate cho phép tìm kiếm vectơ và vô hướng kết hợp hiệu quả. Ví dụ: "các bài viết liên quan đến đại dịch COVID-19 được xuất bản trong vòng 7 ngày qua". Weaviate lưu trữ cả đối tượng và vectơ và đảm bảo việc truy xuất cả hai luôn hiệu quả. Không cần lưu trữ đối tượng của bên thứ ba.
* Sử dụng thời gian thực và liên tục:

Weaviate cho phép bạn tìm kiếm trong dữ liệu của mình ngay cả khi dữ liệu đó hiện đang được nhập hoặc cập nhật.

* Hiệu quả chi phí:

+ Các bộ dữ liệu rất lớn không cần phải được lưu giữ hoàn toàn trong bộ nhớ trong Weaviate.

+Đồng thời, bộ nhớ khả dụng có thể được sử dụng để tăng tốc độ truy vấn. => Điều này cho phép cân bằng giữa tốc độ/chi phí một cách có ý thức để phù hợp với mọi trường hợp sử dụng.

1. Cách thức lưu trữ dữ liệu với từng sinh viên.
2. Siri

Việc siri cải thiện dịch vụ người dùng cá nhân được thực hiện thông qua việc thu thập thông tin cá nhân của người dùng. Tuy nhiên quá trình này đòi hỏi, bộ nhớ lưu trữ lớn cũng như cần có cách lưu trữ hợp lí.

Sau đây là một vài nguồn dữ liệu cá nhân mà Siri thu thập cũng như cách lưu trữ của siri:

* Dữ liệu được lưu trong Icloud của người dùng.
* Các thông tin được lưu trong các ứng dụng mà hệ thống apple xây dựng (ví dụ: ảnh, sức khỏe, lời nhắc,…)
* Các yêu cầu mà người dùng gửi đến Siri.

Ví dụ: Khi bạn sử dụng Siri và Đọc chính tả, thiết bị của bạn sẽ gửi Dữ liệu Siri khác đến Apple, ví dụ như:

Tên, biệt danh và mối quan hệ (ví dụ như “bố tôi”) của liên hệ, nếu bạn thiết lập chúng trong danh bạ

Cách xưng hô, nếu được chỉ định trong cài đặt ngôn ngữ và vùng

Nhạc và podcast mà bạn thưởng thức

Tên của các thiết bị của bạn và các thành viên Chia sẻ trong gia đình

Tên của các phụ kiện, nhà, cảnh, thành viên của nhà được chia sẻ trong ứng dụng Nhà và hồ sơ người dùng Apple TV

Nhãn cho các mục, chẳng hạn như tên người trong Ảnh, tên báo thức và tên của danh sách Lời nhắc

Tên của các ứng dụng được cài đặt trên thiết bị của bạn và các phím tắt mà bạn đã thêm thông qua Siri.

Cách lưu trữ dữ liệu:

* Dữ liệu Siri, bao gồm các yêu cầu Siri của bạn, được liên kết với một mã định danh ngẫu nhiên do thiết bị tạo ra.
* Mã định danh ngẫu nhiên này không được liên kết với ID Apple, địa chỉ email của bạn hoặc dữ liệu khác mà Apple có thể có được từ việc bạn sử dụng các dịch vụ khác của Apple.
* Các dữ liệu sẽ bị xóa sau 6 tháng.

1. Ứng dụng cho bài toán quản lí sinh viên.

Nguồn dữ liệu mà bài toán sử dụng:

* Các thông tin chung mà sinh viên cần biết (quy chế trường, điều kiện học bổng,...).
* Các thông tin riêng của mỗi sinh viên: Dữ liệu trên slink, qldt.

Cách tổ chức dữ liệu:

* Mỗi sinh viên được cấp 1 lượng bộ nhớ riêng nhỏ, để chứa các index riêng 1 và index riêng 2 mà không lưu trữ các vector tương ứng của nó.
* Bộ nhớ chính sẽ chứa toàn bộ các vector embedding và index của nó.
* Những vector embedding có index chung và index riêng 1 sẽ không bị xóa. Những vector embedding có index riêng 2 sẽ bị xóa theo định kì.

Cách search vector:

* Từ câu hỏi của sinh viên thực hiện embedding để tạo vector yêu cầu.
* Vector yêu cầu của sinh viên đó sẽ được xét khoảng cách với các vector có index chung và những index được lưu trong bộ nhớ riêng của sinh viên.

Cách tăng cường dữ liệu:

* Thông qua những câu hỏi của học sinh, thực hiện lấy định kì dữ liệu của hội thoại của chatbot và học sinh.
* Thực hiện embedding dữ liệu và tạo index rồi lưu index vào bộ nhớ riêng của học sinh, lưu embedding vector và index của nó vào bộ nhớ chính.
* Tương tự mỗi kì học lấy những dữ liệu mới của qldt, slink để tạo thêm các index riêng.