Bases de Datos Vectoriales

Santiago Máximo *Instituto de Computación*
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
santiagomaximoc@gmail.com

Resumen

En este documento se investiga acerca las Bases de Datos Vectoriales, que se han vuelto populares debido al creciente uso de vectores de alta dimensionalidad en aplicaciones que requieren el procesamiento de datos no estructurados. El objetivo de este estudio es examinar los componentes claves de las Bases de Datos Vectoriales, como las funciones de similaridad y las técnicas de indexación. Además, se examinarán las soluciones disponibles, evaluando las mejores opciones en diferentes contextos.

I. Introducción

Cada vez más se desarrollan aplicaciones que requieren la gestión de vectores de alta dimensionalidad (de hasta miles de dimensiones). Esto se debe a dos factores principales. Por un lado, los avances tecnológicos han provocado un aumento en la generación de datos no estructurados, como ser imágenes, videos, textos, audios o datos médicos. Por otra parte, son cada vez más frecuentes los modelos de Aprendizaje Automático que transforman datos no estructurados en vectores.

Un vector o *embedding* es una lista de números que se obtiene mediante un modelo de IA (Inteligencia Artificial), como los grandes modelos de lenguaje. La dimensión de estos vectores está relacionada con la cantidad de significado semántico que pueden capturar, lo que permite comprender patrones o relaciones subyacentes entre diferentes elementos. Como consecuencia de utilizar los modelos de *embeddings* se obtiene un espacio vectorial, en cual la proximidad entre dos vectores está relacionada con la similaridad semántica de los datos que representan.

En este nuevo contexto surge la posibilidad de realizar búsquedas vectoriales en lugar de las búsquedas tradicionales basadas en palabras claves. Las búsquedas por palabras claves se basan en la coincidencia de términos de una consulta con un texto, utilizando un índice invertido, lo cual dificulta encontrar términos con significados similares pero con palabras diferentes, y no son útiles para realizar búsquedas multimodales o multilingües. En cambio, en las búsquedas vectoriales hay una noción de similaridad geométrica que posibilita encontrar datos con significados similares y de diferente modalidad. [Pinecone, 2022], [Kan, 2021]

Para poder ejecutar búsquedas vectoriales eficientemente, es necesario contar con soluciones que permitan la gestión de datos vectoriales que sean escalables,y que permitan no solamente un procesamiento rápido de consultas sino que también un manejo eficiente de datos dinámicos (inserciones y eliminaciones). Las soluciones existentes pueden ser clasificado en tres categorías diferentes: bibliotecas, bases de datos extendidas o bases de datos vectoriales.

La primera alternativa es utilizar bibliotecas, como Faiss de Meta, que almacenan vectores en memoria permitiendo realizar búsquedas de similaridad. Estas implementaciones generalmente presentan las siguientes desventajas: almacenan los vectores en memoria lo que dificulta la distribución de datos; almacenan únicamente vectores, siendo necesario un almacenamiento secundario para los datos; no soportan datos dinámicos ni tampoco consultas complejas. [Wang et al., 2021]

Otra opción es utilizar bases de datos SQL o noSQL existentes, que pueden ser adaptadas agregando por ejemplo columnas dónde se puedan almacenar vectores. Sin embargo, como se analizará en el sigueinte informe, al no ser sistemas especializados para gestionar vectores y al no estar enfocados en consultas de tipo “exact-match”, presentan dificultades en lo que respecta a escalabilidad o alta dimensionalidad.

La tercera categoría se enfoca en generar sistemas especializados en vectores siguiendo las prácticas de diseño expresadas en [Stonebraker and Cetintemel, 2005], en oposición al enfoque “One Size Fits All” de generalizar las bases relacionales para que soporten vectores. Como consecuencia surgen las Bases de Datos Vectoriales diseñadas para almacenar y administrar vectores de alta dimensionalidad.

Las bases de datos vectoriales almacenan *embeddings* con el objetivo de permitir búsquedas de similaridad eficientes. Algunas de las capacidades que ofrece este tipo de base de datos son: escalabilidad, operaciones de inserción y borrado, almacenamiento y filtrado de metadatos.

Entre las aplicaciones de búsquedas vectoriales se encuentran las búsquedas de similaridad (entre imágenes, videos, textos o audios), clasificación de secuencias de ADN comparando secuencias similares, sistemas de recomendación, y sistemas de preguntas y respuestas.

El objetivo de este trabajo es investigar acerca de los principales componentes de las Bases de Datos Vectoriales, entre los que se destacan las funciones de similaridad y las técnicas de indexación. Cuanto más preciso sea el resultado, más lenta será la consulta. Un buen sistema debería proporcionar una búsqueda ultrarrápida con una precisión cercana a la perfección.

También se analizarán las soluciones disponibles, destacando su ventajas comparativas y evaluando la mejores opciones según diferentes escenarios.

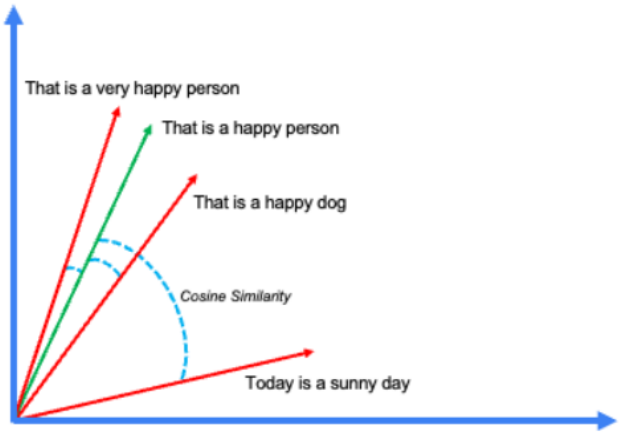

Figura 1: Representación vectorial de oraciones [Partee,]

II. Funciones de similaridad

Las Base de Datos Vectoriales ofrecen diferentes métricas de distancia, que son funciones que toman dos vectores como entrada y calculan un valor de distancia entre ellos. La distancia puede adoptar muchas formas, puede ser la distancia geométrica entre dos puntos, puede ser un ángulo entre los vectores, puede ser un recuento de diferencias en los componentes del vector, etc [Cardenas, 2021].

Cada medida de similitud tiene sus propias ventajas y desventajas. La elección de la medida a utilizar tendrá efectos en los resultados obtenidos, por lo tanto es importante elegirlas según el caso de uso y los requisitos [Schwaber-Cohen, 2023b]. En el contexto del procesamiento del lenguaje natural (PLN), dos vectores que representan significados de palabras, pueden considerarse cercanos entre sí están relacionados con ideas similares. En cambio, en sistemas de recomendación, dos vectores que representan preferencias de usuarios pueden ser considerados similares, si los usuarios han realizado elecciones similares en el pasado.

A continuación se describirán las tres métricas más usadas: similitud coseno, distancia euclidiana y producto escalar. Sin embargo es necesario destacar que hay otras distancias utilizadas, por ejemplo la Base de Datos Milvus también ofrece el índice de Jaccard, distancia de hamming e inclusive métricas específicas que permiten medir la distancia de una estructura química con su superestructura o subestructura.

II-A. Producto escalar

El producto escalar entre dos vectores es un número real que se obtiene multiplicando las respectivas componentes y sumándolas:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + a_3 + b_3 + \ldots + a_n b_n \tag{1}$$

Mide el producto de las magnitudes de los vectores y el coseno del ángulo entre ellos. Tiene un rango que abarca desde $-\infty$ hasta ∞, donde un valor positivo representa vectores que apuntan en la misma dirección, 0 representa vectores ortogonales y un valor negativo representa vectores que apuntan en direcciones opuestas. El producto escalar puede verse afectado tanto por la longitud como por la dirección de los vectores. Es adecuado para *ratings* o recomendaciones. Por ejemplo, dos películas pueden estar representadas con vectores en una misma dirección pero con magnitudes diferentes, lo que puede significar que tratan sobre la misma temática pero una es más popular que la otra.

II-B. Distancia euclidiana

La distancia euclidiana mide la distancia en línea recta entre dos vectores en un espacio vectorial. Tiene un rango que abarca de 0 a ∞, donde un valor igual a 0 representa vectores idénticos y valores más grandes representan vectores más distantes. Se calcula como la raíz cuadrada de la suma de los cuadrados de las diferencias entre los componentes correspondientes de los vectores:

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{(\mathbf{a_1} - \mathbf{b_1})^2 + (\mathbf{a_2} - \mathbf{b_2})^2 + \ldots + (\mathbf{a_n} - \mathbf{b_n})^2} \tag{2}$$

La distancia euclidiana es sensible a la escala, así como a la ubicación de los vectores en el espacio. Esto significa que los vectores con valores grandes tendrán una distancia euclidiana mayor que los vectores con valores pequeños, a pesar de que las distancias puedan ser similares en otros aspectos.

No suele utilizarse en modelos de Aprendizaje Profundo (*Deep Learning*) sino en técnicas de codificación de vectores más básicas. En general, la distancia euclidiana es una elección natural cuando el modelo no se entrenó con una función de pérdida específica.

Como es sensible a la magnitud, se adapta mejor a vectores que contienen información de mediciones o recuentos, siendo también útil, al igual que el producto escalar, para sistemas de recomendación.

II-C. Similitud coseno

La similitud coseno es una medida del ángulo entre dos vectores. Se calcula tomando el producto escalar de los vectores y dividiéndolo por el producto de sus magnitudes:

$$\text{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|} \tag{3}$$

Esta métrica no se ve afectada por el tamaño del vector, sino únicamente por el ángulo entre ellos, lo significa que los vectores con valores grandes o pequeños tendrán la misma similitud del coseno siempre que apunten en la misma dirección.

La similitud coseno varía entre -1 y 1. Dos vectores con la misma orientación tienen una similitud de coseno de 1 , dos vectores ortogonales tienen una similitud de 0, y dos vectores diametralmente opuestos tienen una similitud de -1.

La similitud coseno se utiliza comúnmente en el Procesamiento del Lenguaje Natural en tareas de clasificación o búsquedas semánticas, debido que puede comparar el contenido semántico de documentos sin tener en cuenta su tamaño. No es una medida adecuada para casos en los que la magnitud de los vectores es importante para determinar la similitud, por ejemplo, no es apropiada para comparar la similitud de vectores de imágenes generados en base a las intensidades de los píxeles. [OC, 2023], [Schwaber-Cohen, 2023a]

II-D. Recomendaciones

Una de las reglas básicas es seleccionar como métrica a la utilizada para entrenar el modelo con el cual se generaron los vectores.

Existen modelos que normalizan los vectores, como ser el caso del modelo *text-embedding-ada-002* de OpenAI con *embeddings* normalizados a 1. En este caso, el producto escalar de dos vectores coincide con la similitud del coseno, y por lo tanto, a pesar de que el modelo fue entrenado con la similitud del coseno, el producto escalar puede calcularse de forma más rápida.

La normalización de vectores también provoca que la similitud coseno y la distancia euclidiana generen *rankings* idénticos.

En resumen, no existe una métrica de distancia universal que se adapte a todas las situaciones, sino que depende de los datos, del modelo, de los recursos y de la aplicación específica.

III. Búsquedas de similitud e Índices

Las búsquedas de similitud generalmente incluyen dos categorías: la búsqueda de los k-vecinos más cercanos (KNN, por sus siglas en inglés) y la búsqueda por rango. La búsqueda de los k-vecinos más cercanos.se define de la siguiente manera: dada una consulta y una métrica de distancia, se buscan los k datos más cercanos a la consulta en todo el conjunto de datos o espacio de búsqueda. Y la búsqueda por rango se define como la búsqueda de todos los datos cuya distancia a la consulta sea menor que un umbral dado bajo alguna métrica de distancia.

Para los vectores de alta dimensionalidad provistos por los modelos de Aprendizaje Profundo utilizados actualmente, es difícil determinar el umbral de distancia sin una inspección exhaustiva del conjunto de datos. Por ejemplo, si se utiliza la distancia euclidiana como métrica de similitud, el rango puede variar de 0 a ∞, mientras que si se utiliza la similitud coseno, aunque el rango varía entre -1 y 1, sigue siendo difícil determinar un umbral preciso para diferenciar lo que es similar de lo que no lo es. Si bien recientemente ha habido algunas investigaciones enfocadas en las búsquedas por rango, el progreso no es tan significativo en el campo de la búsqueda de los k-vecinos más cercanos.

La búsqueda de los k-vecinos más cercanos devuelve exactamente los k puntos más cercanos a la consulta. Sin embargo, en aplicaciones del mundo real, especialmente en escenarios de grandes volúmenes de datos, una búsqueda exacta requiere un tiempo de cómputo inasequible. Un área activa de investigación es la búsqueda aproximada de los k-vecinos más cercanos (ANN, por sus siglas en inglés) que tiene como objetivo equilibrar el tiempo de cómputo y la precisión. La búsqueda ANN devuelve k puntos que no necesariamente son los k-puntos más cercanos, y por lo tanto, la precisión de búsqueda de los algoritmos ANN normalmente se mide mediante alguna relación entre los resultados devueltos y las respuestas correctas.

Las búsquedas ANN suelen implementarse utilizando índices específicos para recuperar la información de manera eficiente y efectiva. Los índices más adecuados para el contexto de vectores obtenidos por modelos de Aprendizaje Profundo se pueden agrupar en cuatro categorías: basados en *hashing*, basados en grafos, basados en árboles y basados en *quantization* [Wang, 2022].

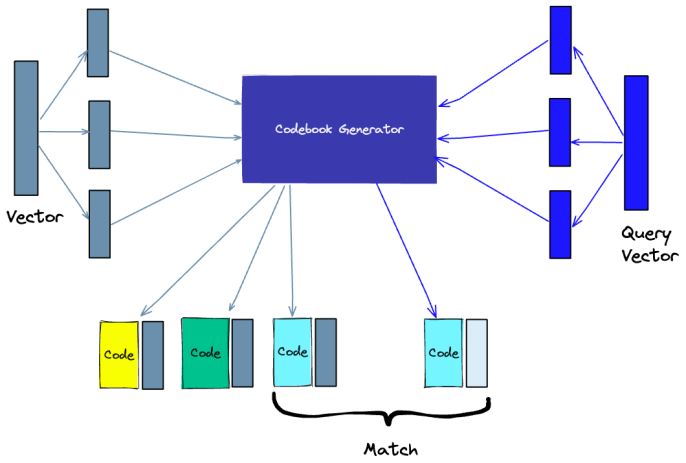

Figura 2: Diagrama de PQ [Schwaber-Cohen, 2023b]

III-A. Índices basados en Quantization

Vector Quantization (VQ) es una herramienta potente que permite reducir el cómputo en la búsqueda del vecino más cercano cuando se cuenta con vectores de alta dimensionalidad. Esta técnica consiste en utilizar una función z de tipo *quantizer* que mapea un vector v a un *codeword* $z(v)$ elegido de un *codebook* C. El algoritmo de *clustering* K-medias es utilizado usualmente para construir el *codebook* C, donde cada *codeword* es un centroide y $z(v)$ es el centroide más cercano a v [Wang et al., 2021].

Cuando se realiza una búsqueda de similitud, la distancia entre dos vectores se puede aproximar a la distancia entre sus *codewords* correspondientes. Los *codewords* se representan utilizando considerablemente menos bits/dimensiones que los vectores originales, lo que no solo ahorra el uso de memoria, sino que también acelera el cálculo de distancias.

Sin embargo, cuando se trabaja con grandes volúmenes de datos, VQ igualmente requiere una cantidad significativa de memoria. Para abordar este problema, se propone la técnica *Product Quantization* (PQ). PQ divide cada vector original en M subvectores de dimensiones más bajas, luego se aplica VQ a cada subvector individualmente, mediante lo cual PQ descompone el espacio vectorial original en el producto cartesiano de M subespacios. Finalmente, el espacio original se representa mediante el conjunto de *codebooks* de los subespacios reduciendo los requerimientos de memoria en varios órdenes de magnitud [Wang, 2022].

Cuantos más vectores representativos haya en el *codebook*, más precisa será la representación de los vectores en el subespacio, pero mayor será el costo computacional para buscar en el *codebook*.

III-B. Índices basados en Grafos

Los índices basados en grafos están emergiendo en los últimos años con el rápido desarrollo de las GPUs, las cuales son muy adecuadas para acelerar el cómputo de grafos. Formalmente, un índice de similitud basado en grafos es un grafo $G(V, E)$, ya sea dirigido o no dirigido, donde los vértices V son el conjunto de todos los puntos de datos en el espacio de búsqueda y el conjunto de aristas E está determinado por las distancias entre los puntos de datos. Específicamente, si dos puntos están lo suficientemente cerca como para considerarse vecinos según alguna métrica de distancia, habrá una arista que conecte sus vértices correspondientes en G, de lo contrario, esa arista no existirá en E [Wang, 2022].

Navigable Small World (NSW) es una búsqueda de vectores utilizando grafos que se basa en la idea de que si se construye un grafo de tal manera que existan enlaces tanto de corto alcance como de largo alcance, los tiempos de búsqueda se reducen a una complejidad (poli/)logarítmica. Los enlaces de corto alcance capturan las relaciones y similitudes entre conceptos o palabras estrechamente relacionados, mientras que los enlaces de largo alcance capturan las relaciones y similitudes entre conceptos más distantes. Cada vértice en el grafo está conectado a varios otros vértices, es decir que cada vértice mantiene una lista de "amigos" [Pinecone, 2023a].

Al buscar en un grafo NSW, se comienza en un punto de entrada predefinido. Este punto de entrada se conecta a varios vértices cercanos. Identificamos cuál de estos vértices es el más cercano al vector de la consulta y se procede a desplazarse hacia él. El proceso de búsqueda se repite, mediante la identificación de los vértices vecinos más cercanos en cada lista de amigos, lo que permite el movimiento de vértice a vértice. Eventualmente, se llega a un punto donde no se encuentran vértices más cercanos al vértice en curso, el cual actúa como un mínimo local y sirve como condición de parada.

Hierarchical Navigable Small Worlds (HNSW) es esencialmente un grafo NSW de múltiples capas, donde los enlaces se separan según su escala de longitud en diferentes capas, de modo que las capas superiores incluyen enlaces de mayor alcance mientras que las capas inferiores incluyen enlaces de menor alcance.

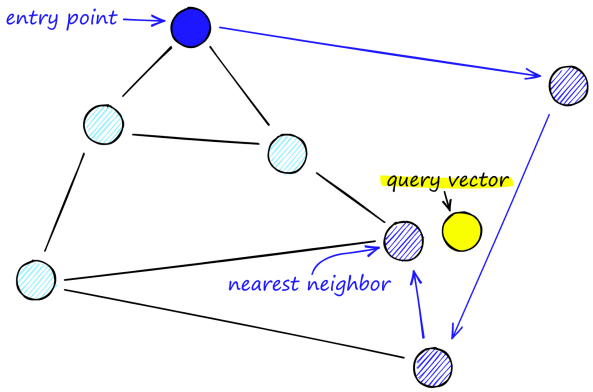

Figura 3: Diagrama de NSW [Pinecone, 2023a]

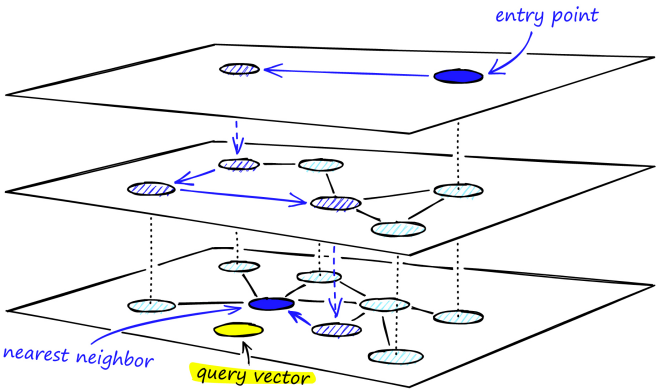

Figura 4: Diagrama de HNSW [Pinecone, 2023a]

La búsqueda en HNSW comienza en la capa superior, donde se ejecuta el algoritmo NSW para alcanzar el vértice óptimo local en la capa actual. Luego, la búsqueda desciende y comienza en la siguiente capa inferior desde el vértice óptimo local de la capa anterior. Dado que la capa inferior incluye enlaces de menor alcance, la búsqueda se vuelve más precisa. El algoritmo repite este proceso hasta que se haya explorado la capa más baja, donde se obtienen los resultados de búsqueda más precisos. El grafo HNSW proporciona la capacidad de una búsqueda más refinada y eficiente que NSW, logrando un mayor *recall* [Wang, 2022].

La idea clave detrás de HNSW es crear una estructura de «mundo pequeño» donde los nodos tienen conexiones tanto de corto alcance con nodos cercanos como de largo alcance con nodos más distantes. El algoritmo incorpora nuevos vectores al grafo agregándolos a los niveles correspondientes y ajustando las conexiones para mantener las propiedades jerárquicas y de «mundo pequeño».

III-C. Índices basados en hashing

Un índice de similitud basado en *hashing* se construye al codificar todos los objetos de datos en el espacio de búsqueda. Para responder una consulta, se aplica una función *hash* a la consulta, luego se busca a los candidatos dentro del contenedor o *bucket* donde se encuentran los objetos codificados con el mismo código *hash*. Una función *hash* mapea datos de alta dimensión a espacios de baja dimensión, lo que reduce significativamente el cómputo. Además, en la mayoría de los casos, la similitud entre códigos *hash* puede calcularse mediante operaciones livianas (por ejemplo, calcular la distancia de Hamming de dos códigos *hash* binarios mediante XOR), lo que aumenta aún más la velocidad de procesamiento [Wang, 2022].

Locality Sensitive Hashing (LSH) es una familia de algoritmos de *hashing* diseñados específicamente para preservar la propiedad de ”localidad”, lo que significa que vectores similares tienen más probabilidades de ser asignados a los mismos contenedores. LSH logra esto utilizando múltiples funciones *hash*, cada una de las cuales produce un código *hash* diferente para el vector de entrada. Estos códigos *hash* se combinan para formar un *hash* compuesto, que se utiliza para indexar y recuperar vectores de manera eficiente. Los algoritmos LSH buscan maximizar las colisiones, a diferencia de la mayoría de las funciones *hash* que buscan minimizarlas.

Random Projection o Proyección Aleatoria es una de las técnicas dentro de la familia LSH, que tiene como objetivo proyectar vectores de alta dimensión a un espacio de menor dimensión utilizando una matriz de proyección aleatoria.

El primer paso en la técnica *Random Projection* es generar una matriz aleatoria según la dimensión deseada. Luego se calcula el producto escalar de los vectores de la base de datos y la matriz, lo que resulta en vectores de menor dimensión que

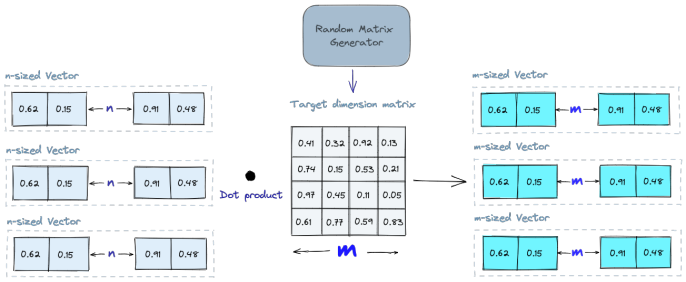

Figura 5: Diagrama de Proyección Aleatoria [Schwaber-Cohen, 2023b]

los vectores originales pero que conservan relaciones de similitud. Al momento de una consulta, se utiliza la misma matriz de proyección para proyectar el vector de consulta en el espacio de menor dimensión. Luego, se compara el vector de consulta proyectado con los vectores proyectados en la base de datos para encontrar los vecinos más cercanos [Schwaber-Cohen, 2023b].

III-D. Índices basados en Árboles

Los índices basados en árboles son muy utilizados en las búsquedas de similitud, por ejemplo, el *KD-tree*. Para datos de baja dimensionalidad, son capaces de lograr una complejidad temporal logarítmica, sin embargo, su rendimiento se ve disminuido y no es mejor que una búsqueda exhaustiva en un espacio de alta dimensionalidad. Debido a lo mencionado anteriormente no se analizará en detalle ningún algoritmo de esta familia. No obstante, es necesario destacar que hay estudios que proponen variantes a los índices basados en árboles clásicos, que se ajustan a datos de alta dimensionalidad.

IV. Consultas avanzadas

Para cumplir con requerimientos de las exigencias y sistemas actuales, las bases de datos vectoriales deben ofrecer consultas más sofisticadas que una simple búsqueda vectorial. Esto implica la capacidad de aplicar filtros basados en los metadatos asociados a los vectores, así como de ofrecer resultados de calidad en escenarios en los que los objetos se representan mediante más de un vector.

IV-A. Consultas híbridas

Existen Bases de Datos Vectoriales que ofrecen la posibilidad de almacenar metadatos asociados a los vectores y realizar consultas híbridas que incluyen búsquedas de similitud vectorial pero que además filtran datos en base a los metadatos (por ejemplo, buscar imágenes en un repositorio que sean similares a una imagen de entrada y que hayan sido generadas en la última semana.). Este tipo de consultas pueden resultar lentas y surgen ciertas dificultades que deben ser analizadas al implementar las Bases de Datos Vectoriales.

Generalmente se mantienen dos índices: un índice de vectores y un índice de metadatos, y se plantean diferentes estrategias dependiendo de si el filtrado de metadatos se realiza antes o después de la búsqueda vectorial. La eficiencia de una u otra alternativa estará vinculada también a la selectividad tanto de la búsqueda vectorial como de las restricciones vinculadas a los metadatos [Rockset, 2023].

En las estrategias de pre-filtrado, primero se realiza un filtrado de metadatos utilizando el índice correspondiente, y luego una búsqueda exhaustiva de vectores. Este enfoque es adecuado cuando las restricciones por metadatos son altamente selectivas. En este caso se estaría realizando una búsqueda vectorial exacta y no aproximada de los vecinos más cercanos.

En las estrategias de post-filtrado primero se realiza la búsqueda aproximada de vecinos más cercanos con el índice correspondiente y luego filtrado exhaustivo en base a las restricciones de metadata. Este enfoque es adecuado para casos en los cuales la selectividad de la búsquedas vectoriales son altamente selectivas. Una desventaja inevitable de esta estrategia es que hay escenarios en los cuales el post-filtrado genera menos resultados que los previstos o inclusive un resultado vacío, por lo tanto es conveniente seleccionar un k mayor al realizar la búsqueda vectorial.

Existen estrategias que toman decisiones en base a estimaciones de selectividad, en base a estimaciones de los costos de pre-filtrado o post-filtrado, y en base a la frecuencia de cada tipo de búsqueda [Wang et al., 2021]. Inclusive hay enfoques que fusionan los índices de metadatos y de vectores en un solo índice [Pinecone, 2023b].

IV-B. Consultas multi-vector

Hay situaciones donde una misma entidad puede estar asociada a múltiples vectores. Por ejemplo, aplicaciones donde se representa a una persona con un vector correspondiente a la imagen de la cara y otro vector de su postura. Otra fuente de vectores múltiples son aplicaciones que utilizan más de un modelo de aprendizaje automático para obtener diferentes vectores de un mismo objeto, para describirlo de la mejor manera posible. Cada entidad estará asociada a μ vectores $\mathrm{v}0, \mathrm{v}1, \ldots, \mathrm{v}\mu - 1$.

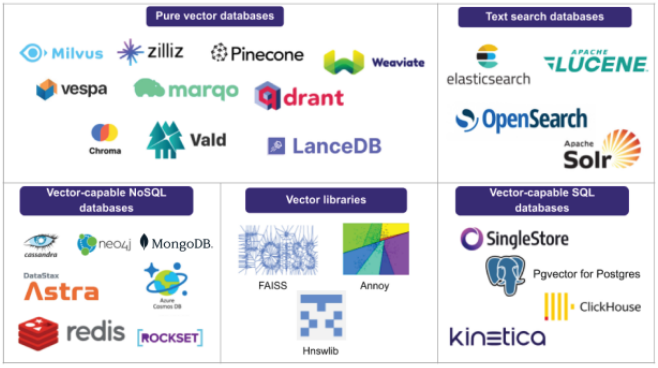

Figura 6: Soluciones disponibles [Srivastava,]

Una consulta *mult-vector* encuentra los k vecinos más cercanos utilizando una función de agregación g sobre una función de similaridad f (por ejemplo, producto interno) de cada vector individual vi. Específicamente, la similitud entre dos entidades X y Y se calcula como $g(f(X.v0, Y.v0), \ldots, f(X \cdot \mathrm{v}\mu - 1, Y \cdot \mathrm{v}\mu - 1))$, donde $X.vi$ significa el vector vi de la entidad X.

V. Productos disponibles

Actualmente existen múltiples opciones de sistemas que permiten gestionar las búsquedas vectoriales. Cómo se puede observar en la Figura 6, existen Bases de Datos Vectoriales puras pero también productos adaptados para soportar búsquedas vectoriales, ya sean Bases de Datos SQL como PostgreSql, Bases de Datos NoSQL como MongoDB, bibliotecas como FAISS, o motores de búsqueda como Elasticsearch.

Entre las Bases de Datos Vectoriales más populares se encuentran Milvus, Pinecone, Qdrant, Weavite. Estas Bases de Datos fueron diseñadas específicamente para almacenar vectores y por lo tanto cuentan con ciertas capacidades en común, como ser la implementación de índices que favorecen las búsquedas vectoriales, el almacenamiento de metadatos asociados a los vectores, la posibilidad agregar consultas por rangos a las consultas vectoriales, y la implementación de *sharding*. Sin embargo alguna de ellas son *self-hosted* y otras *managed*, algunas son de código abierto y otras de código cerrado. En la tabla I visualizan estas diferencias así como también se especifican los índices soportados por cada opción, donde se evidencia que HNSW es la técnica más utilizada.

Cuadro I: Comparación soluciones

Base de Datos	Weaviate	Qdrant	Milvus	Pinecone
Tipo	Managed / Self-hosted	Managed/Self-hosted	Managed(Zillz Cloud)/Self-hosted	Managed
Código Abierto?	Si	Si	Si	Si
Lanzamiento	2019	2021	2019	2019
Técnicas ANN	HNSW personalizado	HNSW personalizado	HNSW, Quantization, Árboles	Propietario

A continuación de describen algunas características particulares de las cuatro Bases de Datos:

- Qdrant agrega aristas adicionales al grafo HNSW basados en valores asociados a los metadatos, lo que permite buscar eficientemente vectores aplicando filtros al mismo tiempo. Además permite almacenar múltiples tipos de metadatos como ser *String*, *Boolean* y coordenadas geográficas
- Weavite permite agregar *Product Quantization* cuando se utiliza HNSW, y permite agregar otros algoritmos además de HNSW siempre que admitan operaciones CRUD. Ofrece una interfaz similar a GraphQL que permite consultar y explorar datos fácilmente.
- Pinecone en sus versiones más recientes incluye *Single-Stage Filtering* que combina los índices de vectores y metadatos en uno solo. A su vez al ser una una Base de Datos *fully-managed* permite permite una rápida integración sin necesidad de destinar recursos a la administración del producto.
- Milvus ofrece métricas específicas para vectores binarios, como la distancias de Hamming o Jaccard, o métricas para medir similitudes vinculadas a estructuras químicas. A su vez Milvus admite índices tanto para vectores binarios como de punto flotante, basados en grafos, en *hashing*, y en árboles.

En cuanto a las adaptaciones de Base de Datos SQL, se destaca la extensión de código abierto PgVector que agrega las búsquedas vectoriales a PostgreSQL. PgVector permite realizar búsquedas exactas y aproximadas de vecinos más cercanos, y ofrece también la posibilidad de agregar filtros sobre atributos a las búsquedas vectoriales. Permite indexar vectores de hasta 2000 dimensiones.

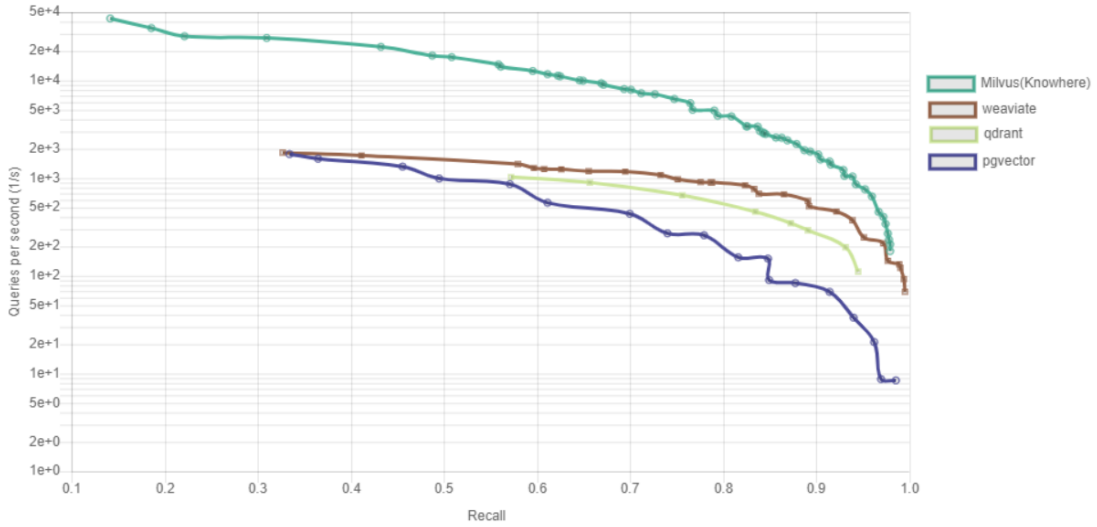

Figura 7: ANN Benchmark para k=10, métrica = similitud del coseno, dataset = glove-100-angular

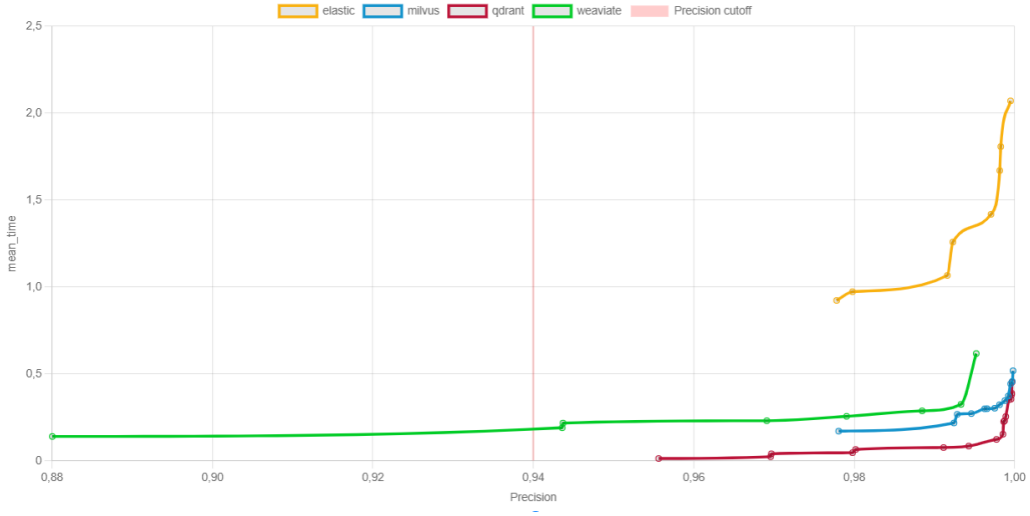

Figura 8: Resultados de latencia para Qdrant Benchmark

V-A. ANN Benchmarks

Al día de hoy no existen *benchmarks* estandarizados para evaluar algoritmos y Bases de Datos vectoriales, en parte debido a la amplia gama de dominios (imágenes, texto, audio, etc), la variedad de familia de algoritmos existentes, y el rápido avance del área lo que hace complejo definir conjuntos de datos y métricas apropiadas.

ANN-Benchmarks [Bernhardsson, 2022] es un proyecto que ofrece herramientas para generar *benchmarks*, basados en [Aumüller et al., 2018] , y conjuntos de datos para búsquedas de hasta 100 vecinos más cercanos. Los resultados publicados incluyen información del parámetro k, del conjunto de datos y de la métrica utilizados. Hay gráficos comparativos que evalúan, para algoritmos de diferentes productos, el *recall* sobre la cantidad de consultas por segundo. Como también hay gráficos que evalúan para un mismo algoritmo los resultados para los distintos productos. De estos resultados se observa que Pgvector muestra los rendimientos más bajos, pero para el resto de las Bases de Datos no se pueden deducir conclusiones firmes.

Por otra parte Qdrant ha implementado sus propios *benchmarks* [qdrant, 2023] basados en ANN-Benchmarks. Se realizaron las pruebas sobre idéntico *hardware*. Los servidores de las Bases de Datos (*self-hosted*) contaban con CPUs y 32GB de RAM con memoria adicional limitada 25 GB. Se utilizaron máquinas clientes (con Python) con 8 CPUs y 16GB de RAM. Los resultados indicaron que:

- Qdrant y Milvus son los motores más rápidos en cuanto a tiempo de indexación.
- Qdrant alcanzó los valores de latencia más bajos.
- Redis obtuvo mejor rendimiento en pruebas con 1 hilo, lo que implica que puede ser útil para casos en los que el cuello de botella.

VI. Conclusiones

Se puede concluir que las bases de Datos Vectoriales son uno de los tópicos del momento en lo que respecta a Tecnología e Inteligencia Artificial, y por lo tanto constantemente surgen nuevos productos, y las soluciones existentes están en constante evolución. Elegir la Base de Datos más apropiada es una tarea difícil ya que se deben tener en consideración diversos factores.

Aquellas organizaciones con experiencia previa en herramientas como PostgreSQL o ElasticSearch pueden empezar utilizando estas opciones, ya que ofrecen funcionalidades básicas, y pueden ser útiles para el diseño de prototipos de aplicaciones o en escenarios donde la cantidad de vectores almacenados no supere los 100k aproximadamente. En cambio si la prioridad son las búsquedas vectoriales sin necesidad de persistencia ni actualización, la mejor opción puede ser utilizar una biblioteca como FAISS.

Sin embargo, cuando se trabaja con un gran número de vectores y el rendimiento es crucial, es recomendable recurrir a bases de datos puras como Milvus o Qdrant. Si no se cuenta con recursos humanos e infraestructura adecuada , se debe optar por una opción managed como Pinecone.

Es importante tener conocimiento sobre los algoritmos de indexación disponibles, siendo HNSW uno de los más eficaces actualmente. En caso de que la aplicación requiera consultas complejas que involucren filtros por metadatos, es necesario buscar soluciones que permitan consultas híbridas y las resuelvan eficientemente, ya sea mediante un enfoque dinámico que considere la selectividad de las consultas o mediante la implementación de índices compuestos que incluyan información tanto vectorial como de metadatos.

Aunque opciones como Pgvector pueden no ser las más adecuadas en la actualidad, se espera que mejoren en un futuro cercano, por ejemplo, incorporando HNSW y equiparándose en ciertos aspectos a las soluciones mejor calificadas. Además, es previsible que surja nueva literatura relacionada con los algoritmos de indexación y se optimicen los tiempos y las latencias de las consultas.

Referencias

[Aumüller et al., 2018] Aumüller, M., Bernhardsson, E., and Faithfull, A. (2018). ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms.

[Bernhardsson, 2022] Bernhardsson, E. (2022). Ann Benchmarks.

[Cardenas, 2021] Cardenas, E. (2021). What are Distance Metrics in Vector Search?

[Kan, 2021] Kan, D. (2021). Not All Vector Databases Are Made Equal.

[OC, 2023] OC, S. (2023). Hands-on Vector Databases: From Embeddings to Similarities & Applications.

[Partee,] Partee, S. Vector similarity search from basics to production.

[Pinecone, 2022] Pinecone (2022). How to Choose a Vector Database.

[Pinecone, 2023a] Pinecone (2023a). Hierarchical Navigable Small Worlds (HNSW).

[Pinecone, 2023b] Pinecone (2023b). The Missing WHERE Clause in Vector Search.

[qdrant, 2023] qdrant (2023). Vector Database Benchmarks.

[Rockset, 2023] Rockset (2023). From Spam Fighting at Facebook to Vector Search at Rockset: How to Build Real-Time ML at Scale.

[Schwaber-Cohen, 2023a] Schwaber-Cohen, R. (2023a). Vector Similarity Explained.

[Schwaber-Cohen, 2023b] Schwaber-Cohen, R. (2023b). What is a Vector Database?

[Srivastava,] Srivastava, A. Choosing a Vector Database for Your Gen AI Stack.

[Stonebraker and Cetintemel, 2005] Stonebraker, M. and Cetintemel, U. (2005). One size fits all": an idea whose time has come and gone.

[Wang et al., 2021] Wang, J., Yi, X., Guo, R., Jin, H., Xu, P., Li, S., Wang, X., Guo, X., Li, C., Xu, X., Yu, K., Yuan, Y., Zou, Y., Long, J., Cai, Y., Li, Z., Zhang, Z., Mo, Y., Gu, J., Jiang, R., Wei, Y., and Xie, C. (2021). Milvus: A Purpose-Built Vector Data Management System.

[Wang, 2022] Wang, Y. (2022). A survey on efficient processing of similarity queries over neural embeddings.