Engineer

Búsqueda vectorial

Buscar datos con los vectores más próximos al de la consulta

Qué es

La búsqueda vectorial encuentra los elementos almacenados cuyos vectores están más cerca del vector de búsqueda. También se llama búsqueda por similitud vectorial. Aquí, un vector es una lista de números de longitud fija que un modelo de embeddings produce a partir de un texto, una imagen o un registro de producto. El modelo se entrena para que los elementos de contenido similar produzcan vectores cercanos; por eso una pregunta y el pasaje que la responde pueden quedar próximos aunque no compartan palabras.

«Más cerca» requiere una medida. pgvector, la extensión de PostgreSQL usada en los ejemplos del artículo, admite seis: distancia L2 o euclídea, producto escalar, distancia del coseno y distancia L1, además de las distancias de Hamming y Jaccard para vectores binarios. La documentación del propio modelo de embeddings indica para qué medida están pensados sus vectores. En vectores normalizados a longitud 1, la distancia del coseno y el producto escalar ordenan igual los vecinos; pgvector recomienda el producto escalar en ese caso para obtener el mejor rendimiento.

Hay dos formas de responder. La búsqueda exacta compara el vector de búsqueda con cada vector almacenado y devuelve los vecinos más próximos reales; pgvector lo hace de forma predeterminada y su documentación lo describe como exhaustividad perfecta. La búsqueda aproximada puede usar un índice que visite solo parte de los datos, sacrificando parte de la exhaustividad a cambio de velocidad. Cuántos vecinos reales devuelve depende de los datos, del índice y de sus ajustes: ni siquiera se garantiza que devuelva la mayoría. Por eso pgvector advierte que, a diferencia de un índice ordinario, uno aproximado cambia lo que devuelve una consulta.

pgvector ofrece dos tipos de índice aproximado. HNSW construye un grafo de varias capas, un método que Malkov y Yashunin publicaron por primera vez en 2016: cada capa contiene un subconjunto de la inferior; la búsqueda empieza en la pequeña capa superior y desciende hasta la completa. Su artículo informa de un crecimiento logarítmico del coste de búsqueda. IVFFlat divide los vectores en listas y busca solo en las más cercanas a la consulta.

Utilice el concepto en una decisión

Aplique «¿Qué es la búsqueda vectorial?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Sus registros al alcance de los agentes de IA, una única versión de cada cliente, acceso limitado a lo que corresponde y cada respuesta vinculada a su fuente.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Flujo de datos geográficos 50 veces más rápido — 50x más rapidez en los flujos de procesamiento de datos geográficos

Flujo de datos geográficos 50 veces más rápido Póngase en contacto →

Por qué importa

La búsqueda de texto completo encuentra palabras. Usando solo coincidencia de palabras y reducción a raíces, quien escriba «la pantalla sigue negra» no encuentra la sección del manual titulada «el monitor no se enciende», porque no comparten ninguna palabra que el algoritmo pueda reducir a una raíz común. Los sinónimos configurados pueden salvar algunas de esas diferencias. La búsqueda vectorial compara el significado tal como lo capturó el modelo de embeddings; así, la queja y la sección pueden quedar próximas. La idea también sirve fuera del texto: una imagen parecida a esta, una incidencia de soporte como otra ya resuelta o un producto como el que hay en la cesta.

También es una forma de hacer la recuperación de información en la generación aumentada por recuperación, donde un modelo de lenguaje grande responde a partir de pasajes obtenidos para la pregunta. Esta se convierte en un embedding con el mismo modelo que los pasajes y los más cercanos se incorporan al prompt. La respuesta solo puede ser tan buena como lo que haya recuperado esa búsqueda, por lo que merece el mismo cuidado que el modelo.

Guardar los vectores en la base de datos ya operativa tiene una ventaja práctica, argumento con el que pgvector abre su documentación. Los vectores reciben las mismas transacciones, recuperación a un momento dado y combinaciones de tablas que cualquier otra columna; como pgvector escribe en el registro de

escritura anticipada, la replicación también los transporta. Filtrar por cliente, fecha o permiso es una cláusula `WHERE` ordinaria sobre la misma fila. Una instantánea tomada después de confirmarse una eliminación ya no ve ese documento, pero una consulta que use una instantánea anterior aún puede verlo. Un almacén vectorial separado es otro sistema que hay que llenar y mantener sincronizado con el primero.

El equipo también debe saber qué no hace la búsqueda vectorial. Ordena por proximidad, no por veracidad. El pasaje más cercano es el más parecido a la pregunta; aun así puede estar desactualizado, ser erróneo o proceder de un documento que el lector no pueda consultar. Algunas preguntas dependen de una palabra exacta, como un nombre, una referencia de pieza o un código de error, y la documentación de pgvector propone combinar la búsqueda vectorial con la de texto completo de PostgreSQL. Esa combinación se llama búsqueda híbrida.

Cómo se aplica en la práctica

pgvector, en su versión 0.8.7 de octubre de 2026, admite PostgreSQL 13 y posteriores. Active la extensión una vez en cada base de datos y añada a la tabla una columna de tipo `vector` con el número de dimensiones que produzca el modelo:

```
CREATE EXTENSION vector;

CREATE TABLE passages (
    id        bigserial PRIMARY KEY,
    source    text NOT NULL,
    body      text NOT NULL,
    embedding vector(3)
);

CREATE INDEX ON passages USING hnsw (embedding vector_cosine_ops);
```

Las tres dimensiones permiten leer el ejemplo con facilidad; un modelo real produce muchas más y la columna declara cuántas produce. La aplicación calcula cada embedding con el modelo y lo inserta junto al texto del que procede. El índice especifica una distancia: `vector_cosine_ops` sirve a consultas ordenadas por distancia del coseno; una consulta ordenada por distancia L2 necesita un segundo índice construido con `vector_l2_ops`. HNSW no tiene fase de entrenamiento, por lo que el índice puede crearse sobre una tabla vacía.

La consulta convierte la pregunta en un embedding con el mismo modelo y pide los pasajes más cercanos:

```
SELECT id, source, body
FROM   passages
ORDER  BY embedding <=> $1
LIMIT  5;
```

`<=>` es la distancia del coseno y `$1` es el vector de la consulta, enviado como parámetro. Dos detalles permiten que el planificador use el índice: la consulta incluye tanto `ORDER BY` como `LIMIT` y su `ORDER BY` es el operador de distancia sin más, en orden ascendente. Una puntuación de similitud para mostrar, `1 - (embedding <=> $1)`, debe ir en la lista de selección. Los demás operadores son `<->` para la distancia L2, `<+>` para L1 y `<#>` para el producto escalar. El último devuelve el producto escalar negativo, porque PostgreSQL solo recorre un índice por un operador en orden ascendente.

El índice HNSW tiene dos parámetros de construcción: `m`, el número de vecinos con los que enlaza cada nodo en cada capa, 16 salvo que se cambie, y `ef_construction`, el tamaño de la lista de candidatos durante la construcción, 64 de forma predeterminada. Al consultar, `hnsw.ef_search` fija el tamaño de esa lista, 40 de forma predeterminada. Aumentarlo encuentra más vecinos reales y ralentiza cada consulta; `SET LOCAL` lo aumenta hasta que termina la transacción actual. Para limitar el ajuste a una consulta, incluya solo esa consulta en la transacción. Para medir el coste del índice en exhaustividad, ejecute la misma consulta dentro de una transacción con `SET LOCAL enable_indexscan = off`, que da la respuesta exacta, y compare las dos listas.

El trabajo de la empresa en agentes de IA personalizados guarda la memoria de casos anteriores del agente en la base de datos del propio cliente, en lugar del historial de conversación de un proveedor. Una tabla como `passages` es una posible forma de esa memoria: cuando llega un caso nuevo, los anteriores más parecidos están a una consulta de distancia, en la misma base de datos y con los mismos permisos que el resto de la información.

La búsqueda híbrida ejecuta ambas búsquedas y fusiona sus listas. La documentación de pgvector menciona Reciprocal Rank Fusion, de un artículo de 2009 de Cormack, Clarke y Büttcher: cada documento recibe 1 / (k + rank) por cada lista en la que aparece y se suman las puntuaciones. Los autores fijaron k en

60 y sus pruebas preliminares mostraron que 60 era «casi óptimo, aunque la elección no era decisiva». Con una columna `tsvector` llamada `search_vec` en la misma tabla, construida como para la búsqueda de texto completo, la consulta fusionada es:

```
WITH semantic AS (
    SELECT id, row_number() OVER (ORDER BY dist) AS r
    FROM  (SELECT id, embedding <=> $1 AS dist
           FROM   passages
           ORDER  BY dist
           LIMIT  20) AS nearest
),
keyword AS (
    SELECT id, row_number() OVER (ORDER BY score DESC) AS r
    FROM  (SELECT id, ts_rank_cd(search_vec, query) AS score
           FROM   passages, plainto_tsquery('english', $2) AS query
           WHERE  search_vec @@ query
           ORDER  BY score DESC
           LIMIT  20) AS matches
)
SELECT id,
       coalesce(1.0 / (60 + s.r), 0)
     + coalesce(1.0 / (60 + k.r), 0) AS rrf
FROM   semantic AS s
FULL   JOIN keyword AS k USING (id)
ORDER  BY rrf DESC
LIMIT  5;
```

Cada búsqueda conserva sus primeros 20 resultados y los numera desde 1. Un pasaje al que ambas sitúan arriba acumula dos aportaciones y asciende; uno encontrado por una sola conserva una aportación. La fusión usa posiciones en lugar de puntuaciones porque la distancia del coseno y la puntuación textual usan escalas incomparables. pgvector menciona un cross-encoder como la otra forma de combinar las listas.

Dónde surgen los problemas

Llegan menos filas de las que pidió `LIMIT` **.** Con un índice aproximado, la cláusula `WHERE` se aplica después de recorrer el índice. pgvector aporta el cálculo: si una condición coincide con el 10 % de las filas, un índice HNSW con el valor predeterminado de `hnsw.ef_search` , 40, deja unas 4 filas de media. Desde la versión 0.8.0, los recorridos iterativos examinan más partes del índice hasta que pasan el filtro suficientes filas o se alcanzan sus límites. Actívelos con `SET hnsw.iterative_scan = strict_order` , o con `relaxed_order` para mejorar la

exhaustividad con resultados ligeramente desordenados. En HNSW, `hnsw.max_scan_tuples`, con valor predeterminado de 20.000, es un límite aproximado de tuplas adicionales visitadas y no afecta al recorrido inicial. Un límite de memoria independiente también puede detener el recorrido adicional. Si el filtro solo tiene unos pocos valores distintos, también sirve un índice parcial por valor.

Una consulta no usa el índice de vecinos más cercanos. `ORDER BY 1 - (embedding <=> $1) DESC` ordena por una expresión; una consulta sin `LIMIT` no pide un resultado acotado de vecinos más cercanos. Ninguna cumple los requisitos de pgvector para usar ese índice. El número de filas leídas por el plan elegido también depende de los filtros y de otras decisiones del planificador. Un índice construido con `vector_l2_ops` no sirve a una consulta ordenada por `<=>`. Ordene por el operador sin más, en sentido ascendente y con un `LIMIT`; construya un índice por cada distancia que consulte y compruebe el plan con `EXPLAIN (ANALYZE, BUFFERS)`.

Se construye un índice IVFFlat demasiado pronto. IVFFlat ejecuta k-means sobre las filas presentes al construir el índice y reparte los vectores en listas a partir del resultado. Un índice creado con pocos datos para su número de listas devuelve menos filas de las pedidas, y pgvector aconseja eliminarlo hasta que la tabla tenga más datos. Cargue primero los datos y construya el índice después. pgvector propone empezar con rows / 1000 listas hasta un millón de filas y, por encima, con la raíz cuadrada del número de filas; para probes, cuyo valor predeterminado es 1, propone la raíz cuadrada del número de listas. HNSW no tiene esa fase de entrenamiento.

Hay filas que nunca se encuentran. Los vectores NULL no se indexan; tampoco los vectores cero con distancia del coseno, pues carecen de dirección que comparar. Una fila cuyo embedding aún falta es invisible para una búsqueda con índice. Añada `NOT NULL` cuando la carga sea fiable y programe un recuento de las filas que todavía no tengan embedding.

Una columna contiene vectores de dos modelos. Dos modelos de embeddings, o dos versiones de uno, colocan el mismo texto en puntos distintos; sus vectores no pueden compararse aunque coincida el número de dimensiones. Convierta la pregunta con el mismo modelo que convirtió los pasajes, guarde el modelo en una columna y regenere todos los embeddings cuando cambie. Para una columna que guarda vectores de varios tamaños, pgvector muestra un índice sobre una expresión con una condición por modelo, `WHERE (model_id = 123)`.

El índice no admite tantas dimensiones. El tipo `vector` guarda hasta 16.000 dimensiones, pero HNSW e IVFFlat indexan como máximo 2.000. Indexe una conversión a `halfvec` para llegar a 4.000, use cuantización binaria para llegar a 64.000 y vuelva a ordenar los candidatos con los vectores completos, o indexe un subvector cuando el modelo lo permita.

Los vectores de un cliente ralentizan las búsquedas de otro. pgvector advierte que un índice aproximado compartido entre clientes permite que los vectores de uno afecten a la exhaustividad y la velocidad de los demás. Particione la tabla por cliente mediante particionado por listas, o dé a cada cliente su propia tabla.

La construcción del índice bloquea la aplicación. HNSW se construye mucho más deprisa si el grafo cabe en `maintenance_work_mem`; pgvector muestra un aviso cuando deja de caber. Asignar tanta memoria que al servidor le falte para otras tareas provoca otra interrupción del servicio. En un sistema en uso, construya con `CREATE INDEX CONCURRENTLY`, que no bloquea las escrituras.

Se recuperan pasajes que el lector no puede consultar. Buscar en todos los pasajes puede entregar a un modelo de lenguaje texto de un documento que el usuario no tiene derecho a leer, y el modelo puede citarlo. Ponga la condición de permisos en la misma cláusula `WHERE` de la búsqueda para que esa fila no salga de la base de datos; active los recorridos iterativos para encontrar más pasajes permitidos cuando el filtro descarte candidatos, dentro de los límites de recorrido.

Se interpreta una distancia como confianza. Una distancia del coseno de 0,2 indica que un pasaje está más cerca que uno a 0,4, no que responda a la pregunta. Un umbral ajustado para un modelo no significa nada para el siguiente. Ordene por distancia, limite con `LIMIT` y evalúe la búsqueda mediante un conjunto de preguntas reales cuyos pasajes correctos se conozcan.

Lecturas adicionales

- pgvector — la documentación de la propia extensión: tipos, operadores de distancia, opciones HNSW e IVFFlat, filtros, recorridos iterativos de índices, búsqueda híbrida y resolución de problemas.
- Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs — el artículo de Malkov y Yashunin que presentó HNSW.

- Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods — el artículo de SIGIR 2009 con la fórmula de fusión y las pruebas preliminares que sustentan su constante.
- Introduction to full text search — la búsqueda por palabras de PostgreSQL, la otra mitad de una consulta híbrida.
- Controlling Text Search — `plainto_tsquery` y `ts_rank_cd`, usados en la búsqueda por palabras anterior.
- Transaction Isolation — cuándo las instantáneas de PostgreSQL ven los cambios confirmados, incluido por qué una instantánea anterior aún puede ver un documento tras confirmarse su eliminación.

Glosario

Elija según sus restricciones reales

Traiga un conjunto de datos representativo, las preguntas que debe responder y el cuello de botella actual. Podemos acordar cómo medir la calidad, la capacidad de procesamiento y la recuperación.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es la búsqueda vectorial?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/vector-search/