Engineer

Recherche vectorielle

trouver les entrées les plus proches du vecteur de requête

Ce que c’est

La recherche vectorielle trouve les éléments stockés dont les vecteurs sont les plus proches du vecteur de recherche. Elle est aussi appelée recherche par similarité vectorielle. Un vecteur est ici une liste de nombres de longueur fixe produite par un modèle d’embedding à partir d’un passage de texte, d’une image ou d’une fiche produit. Le modèle est entraîné pour que des contenus semblables produisent des vecteurs voisins : une question et le passage qui y répond peuvent donc être proches même sans avoir de mots en commun.

« Le plus proche » exige une mesure. pgvector, l’extension PostgreSQL utilisée dans les exemples de cet article, en propose six : distance L2, ou euclidienne, produit scalaire, distance cosinus et distance L1, plus les distances de Hamming et de Jaccard pour les vecteurs binaires. La documentation du modèle d’embedding indique pour quelle mesure ses vecteurs sont conçus. Pour des vecteurs normalisés à une longueur de 1, la distance cosinus et le produit scalaire classent les voisins dans le même ordre, et pgvector recommande le produit scalaire dans ce cas pour les meilleures performances.

Il existe deux façons de répondre à la question. La recherche exacte compare le vecteur de recherche à chaque vecteur stocké et renvoie les véritables plus proches voisins ; pgvector le fait par défaut, et sa documentation parle de rappel parfait. La recherche approximative peut utiliser un index qui ne visite qu’une partie des données, échangeant une part de rappel contre de la vitesse. Le nombre de véritables voisins retrouvés dépend des données, de l’index et de ses réglages ; même une majorité n’est pas garantie. C’est pourquoi pgvector avertit que, contrairement à un index ordinaire, un index approximatif modifie les résultats d’une requête.

pgvector propose deux types d'index approximatifs. HNSW construit un graphe en plusieurs couches, une méthode publiée pour la première fois par Malkov et Yashunin en 2016 : chaque couche contient un sous-ensemble de celle située dessous. La recherche part de la petite couche supérieure et descend jusqu'à la couche complète, et leur article rapporte une croissance logarithmique du coût de recherche. IVFFlat répartit les vecteurs en listes et ne cherche que dans les listes les plus proches de la requête.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que la recherche vectorielle ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des données accessibles aux agents d'IA, une seule fiche par client, des accès limités au nécessaire et chaque réponse reliée à sa source.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Pipeline géographique accéléré de 50 fois — 50x d'accélération des pipelines de données géographiques

Pipeline géographique accéléré de 50 fois Contactez-nous →

Pourquoi c'est important

La recherche plein texte compare des mots. Avec la seule correspondance de mots et leur réduction à un radical, un client écrivant « l'écran reste noir » ne trouve pas la section du manuel intitulée « l'affichage ne s'allume pas », car les deux ne partagent aucun mot qu'un algorithme de racinisation puisse rapprocher. Des synonymes configurés peuvent combler certaines lacunes. La recherche vectorielle compare le sens tel que le modèle d'embedding l'a capturé : la plainte et la section peuvent donc se retrouver proches. La même idée dépasse le texte : une image semblable à celle-ci, un ticket d'assistance semblable à un cas déjà résolu, un produit comme celui du panier.

C'est aussi une façon d'effectuer la recherche dans la génération augmentée par récupération, où un grand modèle de langage répond à partir de passages récupérés pour la question. La question est vectorisée avec le même modèle que les passages, et les plus proches sont inclus dans le prompt. La réponse ne peut être meilleure que les éléments retrouvés : la recherche mérite donc autant de soin que le modèle.

Conserver les vecteurs dans la base de données déjà exploitée présente un intérêt pratique, et la documentation de pgvector commence par cet argument. Les vecteurs bénéficient des mêmes transactions, restaurations à un instant

donné et jointures que toute autre colonne, et pgvector écrivant dans le journal d'écriture anticipée, la réplication les transporte aussi. Un filtre sur le client, la date ou la permission est une clause `WHERE` ordinaire sur la même ligne. Un instantané pris après la validation d'une suppression ne voit plus le document, mais une requête utilisant un instantané plus ancien peut encore le voir. Un stockage vectoriel distinct est un deuxième système à alimenter et à maintenir synchronisé avec le premier.

Une équipe doit aussi savoir ce que la recherche vectorielle ne fait pas. Elle classe par proximité, pas par vérité. Le passage le plus proche est celui qui ressemble le plus à la question ; il peut encore être obsolète, faux ou provenir d'un document auquel le lecteur n'a pas accès. Certaines questions dépendent d'un mot précis, comme un nom, une référence de pièce ou un code d'erreur, et la documentation de pgvector suggère d'associer la recherche vectorielle à la recherche plein texte de PostgreSQL. Cette combinaison est appelée recherche hybride.

À quoi cela ressemble en pratique

pgvector, en version 0.8.7 en octobre 2026, prend en charge PostgreSQL 13 et les versions ultérieures. Il faut activer l'extension une fois dans chaque base de données, puis ajouter à la table une colonne de type `vector` ayant le nombre de dimensions produit par le modèle :

```
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);
```

L'exemple est limité à trois dimensions pour rester lisible ; un modèle réel en produit bien davantage, et la colonne déclare le nombre qu'il produit. L'application calcule chaque embedding avec le modèle et l'insère à côté du texte d'origine. L'index désigne une distance : `vector_cosine_ops` sert les requêtes

triées par distance cosinus, et une requête triée par distance L2 nécessite un deuxième index construit avec `vector_l2_ops`. HNSW n'a pas d'étape d'entraînement, l'index peut donc être créé sur une table vide.

La requête vectorise la question avec le même modèle et demande les passages les plus proches :

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

`<=>` est la distance cosinus et `$1` le vecteur de requête passé comme paramètre. Deux détails permettent au planificateur d'utiliser l'index : la requête contient à la fois `ORDER BY` et `LIMIT`, et `ORDER BY` porte directement sur l'opérateur de distance, en ordre croissant. Un score de similarité à afficher, `1 - (embedding <=> $1)`, a sa place dans la liste de sélection. Les autres opérateurs sont `<->` pour la distance L2, `<+>` pour L1 et `<#>` pour le produit scalaire. Ce dernier renvoie le produit scalaire négatif, car PostgreSQL ne parcourt un index sur un opérateur qu'en ordre croissant.

L'index HNSW a deux paramètres de construction : `m`, le nombre de voisins auxquels un nœud se relie sur chaque couche, 16 sauf réglage contraire, et `ef_construction`, la taille de la liste de candidats pendant la construction, 64 par défaut. À l'exécution d'une requête, `hnsw.ef_search` définit la taille de la liste de candidats, 40 par défaut. L'augmenter retrouve davantage de véritables plus proches voisins et ralentit chaque requête, et `SET LOCAL` l'augmente jusqu'à la fin de la transaction en cours. Pour limiter le réglage à une requête, il ne faut mettre que cette requête dans la transaction. Pour connaître le coût de l'index en rappel, il faut exécuter la même requête dans une transaction avec `SET LOCAL enable_indexscan = off`, ce qui donne la réponse exacte, puis comparer les deux listes.

Le travail de l'entreprise sur des agents IA sur mesure conserve la mémoire des cas passés d'un agent dans la base de données du client plutôt que dans l'historique de conversation d'un fournisseur. Une table comme `passages` est une forme que peut prendre cette mémoire : à l'arrivée d'un nouveau cas, les cas passés les plus semblables sont accessibles en une requête, dans la même base et avec les mêmes permissions que les autres données.

La recherche hybride exécute les deux recherches et fusionne leurs listes. La documentation de pgvector cite Reciprocal Rank Fusion, issu d'un article de 2009 de Cormack, Clarke et Büttcher : chaque document reçoit 1 / (k + rank) pour chaque liste où il apparaît, et ces valeurs sont additionnées. Les auteurs ont fixé k à 60, et leurs essais préliminaires ont montré que 60 était « quasi optimal, mais que le choix n'était pas déterminant ». Avec une colonne `tsvector` nommée `search_vec` dans la même table, construite comme pour la recherche plein texte, la requête fusionnée s'écrit :

```
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;
```

Chaque recherche conserve ses 20 premiers résultats et les numérote à partir de 1. Un passage bien classé par les deux recherches reçoit deux contributions et remonte ; un passage trouvé par une seule en garde une. La fusion utilise les rangs plutôt que les scores, car la distance cosinus et le score de pertinence textuelle sont sur des échelles non comparables. pgvector cite un cross-encoder comme autre méthode pour combiner les listes.

Où les problèmes surviennent

Moins de lignes que le nombre demandé par `LIMIT` . Avec un index approximatif, la clause `WHERE` est appliquée après le parcours de l'index. pgvector donne le calcul : si une condition correspond à 10 % des lignes, un index

HNSW avec la valeur par défaut de 40 pour `hnsw.ef_search` laisse environ 4 lignes en moyenne. Depuis la version 0.8.0, les parcours d'index itératifs explorent davantage l'index jusqu'à ce qu'assez de lignes passent le filtre ou que les limites de parcours soient atteintes. Il faut les activer avec `SET hnsw.iterative_scan = strict_order`, ou `relaxed_order` pour un meilleur rappel avec des résultats légèrement désordonnés. Pour HNSW, `hnsw.max_scan_tuples`, fixé à 20 000 par défaut, est une limite approximative des tuples supplémentaires visités et n'affecte pas le parcours initial. Une limite de mémoire distincte peut aussi arrêter la poursuite du parcours. Pour un filtre portant sur peu de valeurs distinctes, un index partiel par valeur fonctionne aussi.

Une requête n'utilise pas l'index des plus proches voisins. `ORDER BY 1 - (embedding <=> $1) DESC` trie selon une expression, et une requête sans `LIMIT` ne demande pas un résultat borné de plus proches voisins. Ni l'un ni l'autre ne satisfait les exigences de pgvector pour utiliser cet index. Le nombre de lignes lues par le plan choisi dépend aussi des filtres et d'autres choix du planificateur. Un index construit avec `vector_l2_ops` ne sert pas à une requête triée par `<=>`. Il faut trier directement par l'opérateur, en ordre croissant, avec un `LIMIT`, construire un index par distance interrogée et vérifier le plan avec `EXPLAIN (ANALYZE, BUFFERS)`.

Un index IVFFlat construit trop tôt. IVFFlat exécute k-means sur les lignes présentes lors de la construction de l'index et en répartit les vecteurs en listes. Un index créé avec trop peu de données pour son nombre de listes renvoie moins de lignes que demandé, et pgvector conseille de le supprimer jusqu'à ce que la table contienne plus de données. Il faut charger les données avant de construire l'index. Pour commencer, pgvector suggère un nombre de listes égal au nombre de lignes divisé par 1 000 jusqu'à un million de lignes, puis à la racine carrée du nombre de lignes au-delà. Pour probes (le nombre de listes sondées, 1 par défaut), il propose la racine carrée du nombre de listes. HNSW n'a pas cette étape d'entraînement.

Des lignes jamais trouvées. Les vecteurs NULL ne sont pas indexés, pas plus que les vecteurs nuls pour la distance cosinus, qui n'a pas de direction à comparer. Une ligne dont l'embedding manque encore est invisible pour une recherche par index. Il faut ajouter `NOT NULL` une fois le chargement fiable et compter régulièrement les lignes encore dépourvues d'embedding.

Des vecteurs de deux modèles dans une colonne. Deux modèles d'embedding, ou deux versions d'un même modèle, placent le même texte à des endroits différents ; leurs vecteurs ne sont pas comparables même si le nombre de dimensions concorde. Il faut vectoriser la question avec le modèle qui a vectorisé les passages, stocker le modèle dans une colonne et tout vectoriser à nouveau lorsque le modèle change. Pour une colonne contenant des vecteurs de plusieurs tailles, pgvector montre un index d'expression avec une condition par modèle, `WHERE (model_id = 123)` .

Trop de dimensions pour l'index. Le type `vector` stocke jusqu'à 16 000 dimensions, mais HNSW et IVFFlat en indexent au plus 2 000. Il faut indexer une conversion en `halfvec` pour aller jusqu'à 4 000, utiliser la quantification binaire jusqu'à 64 000 puis reclasser la présélection selon les vecteurs complets, ou indexer un sous-vecteur si le modèle le permet.

Les vecteurs d'un client ralentissent les recherches d'un autre. pgvector avertit qu'un index approximatif partagé entre organisations clientes permet aux vecteurs de l'une d'influencer le rappel et la vitesse pour les autres. Il faut partitionner la table par client avec un partitionnement par liste, ou donner sa propre table à chaque client.

Une construction d'index bloque l'application. HNSW se construit bien plus vite lorsque le graphe tient dans `maintenance_work_mem` , et pgvector émet un avis quand ce n'est plus le cas. Allouer tellement de mémoire à la construction que le serveur en manque provoque une panne à part entière. Sur un système en fonctionnement, il faut construire l'index avec `CREATE INDEX CONCURRENTLY` , qui ne bloque pas les écritures.

Des passages que le lecteur n'a pas le droit de voir. Une recherche sur tous les passages peut transmettre à un modèle de langage le texte d'un document que l'utilisateur n'a pas le droit de lire, et le modèle peut le citer. La condition de permission doit figurer dans la même clause `WHERE` que la recherche, pour qu'une telle ligne ne quitte jamais la base de données. Il faut activer les parcours itératifs pour chercher davantage de passages autorisés lorsque le filtrage élimine des candidats, sous réserve des limites de parcours.

Une distance interprétée comme une certitude. Une distance cosinus de 0,2 dit qu'un passage est plus proche qu'un autre à 0,4, pas qu'il répond à la question. Un seuil réglé pour un modèle ne vaut pas pour le suivant. Il faut trier par

distance, borner avec `LIMIT` et évaluer la recherche sur un ensemble de vraies questions dont les bons passages sont connus.

Pour aller plus loin

- pgvector — la documentation de l'extension : types, opérateurs de distance, options HNSW et IVFFlat, filtrage, parcours d'index itératifs, recherche hybride et dépannage.
- Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs — l'article de Malkov et Yashunin qui a présenté HNSW.
- Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods — l'article SIGIR de 2009 avec la formule de fusion et les essais préliminaires à l'origine de sa constante.
- Introduction to full text search — la recherche lexicale de PostgreSQL, l'autre moitié d'une requête hybride.
- Controlling Text Search — `plainto_tsquery` et `ts_rank_cd`, employés ci-dessus dans la partie lexicale.
- Transaction Isolation — quand les instantanés PostgreSQL voient les modifications validées, notamment pourquoi un instantané plus ancien peut encore voir un document après la validation de sa suppression.

Glossaire

Choisissez selon vos contraintes réelles

Présentez un jeu de données représentatif, les questions auxquelles il doit répondre et le goulot d'étranglement actuel. Nous pouvons convenir de la manière de mesurer la qualité, le débit et la reprise.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que la recherche vectorielle ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/vector-search/