Engineer

Векторний пошук

пошук записів, чиї вектори найближчі до вектора запиту

Що це таке

Векторний пошук знаходить ті збережені об’єкти, чиї вектори лежать найближче до вектора запиту. Англійською його називають vector search. Вектор тут — це список чисел фіксованої довжини, який модель ембедингів обчислює з фрагмента тексту, зображення чи картки товару. Модель навчено так, щоб об’єкти зі схожим змістом отримували близькі вектори, тому питання і фрагмент, що на нього відповідає, можуть опинитися поруч, навіть якщо спільних слів у них немає.

«Найближче» потребує міри. pgvector, розширення PostgreSQL, на якому побудовано приклади цієї статті, підтримує шість: відстань L2 (евклідову), скалярний добуток, косинусну відстань і відстань L1, а для бінарних векторів ще відстань Геммінга та відстань Жаккара. Для якої міри розраховано вектори, вказує документація самої моделі. Для векторів, нормалізованих до довжини 1, косинусна відстань і скалярний добуток розставляють сусідів в однаковому порядку, і тут pgvector радить скалярний добуток, бо він дає найкращу швидкодію.

Відповісти на запит можна двома способами. Точний пошук порівнює запит із кожним збереженим вектором і повертає справжніх найближчих сусідів. Так pgvector працює за замовчуванням, і документація називає це ідеальною повнотою (recall): до результату потрапляють усі справжні сусіди. Наближений пошук може використовувати індекс, який переглядає лише частину даних, і жертвує частиною повноти заради швидкості. Скільки справжніх сусідів повернеться, залежить від даних, індексу та його налаштувань; навіть більшість не гарантована. Тому pgvector попереджає, що наближений індекс, на відміну від звичайного, змінює те, що повертає запит.

pgvector має два типи наближених індексів. HNSW будує граф із кількох шарів; цей метод уперше описано в статті 2016 року. Кожен шар містить підмножину вузлів шару під ним, пошук починається в невеликому верхньому шарі й спускається до повного, а за даними статті вартість пошуку зростає логарифмічно. IVFFlat ділить вектори на списки й шукає лише в тих списках, що лежать найближче до запиту.

Застосуйте поняття до рішення

Застосуйте «Що таке векторний пошук?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Ваші дані зібрано там, де їх дістануть ШІ-агенти: одна версія кожного клієнта, ніхто не бачить більше, ніж йому можна, а кожну відповідь можна простежити до джерела.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Прискорила пайплайн географічних даних у 50 разів — 50x швидші пайплайни географічних даних

Прискорила пайплайн географічних даних у 50 разів Зв'язатися з нами →

Чому це важливо

Повнотекстовий пошук порівнює слова. За самої лише відповідності слів і стемінгу клієнт, який пише «екран залишається чорним», не побачить розділу інструкції із заголовком «дисплей не вмикається», бо спільного слова в цих фразах немає навіть після відкидання закінчень. Налаштовані синоніми можуть подолати частину таких розбіжностей. Векторний пошук порівнює зміст так, як його вловила модель ембедингів, тож скарга й розділ можуть опинитися поруч. Та сама ідея працює не лише з текстом: зображення, схоже на це, звернення до підтримки, схоже на вже розв’язане, товар, схожий на той, що лежить у кошику.

Це також один зі способів виконати пошукову частину генерації, доповненої пошуком: велика мовна модель відповідає на питання, спираючись на знайдені для нього фрагменти. Питання перетворюють на вектор тією самою моделлю, що й фрагменти, а найближчі фрагменти додають до промпту. Відповідь не буде кращою за те, що повернув пошук, тож пошук заслуговує на таку саму увагу, як і модель.

Зберігати вектори в базі даних, яка у вас уже працює, вигідно з практичного погляду, і pgvector відкриває свою документацію саме цим аргументом. Вектори отримують ті самі транзакції, відновлення на момент часу (point-in-time recovery) і з’єднання таблиць, що й будь-який інший стовпець, а оскільки

pgvector пише в журнал попереднього запису (WAL), реплікація переносить і їх. Фільтр за клієнтом, датою чи правами доступу — звичайна умова `WHERE` на тому самому рядку. Знімок, зроблений після фіксації видалення, уже не бачить документа, але запит зі старішим знімком усе ще може його бачити. Окреме векторне сховище — це ще одна система, яку треба наповнювати й тримати в узгодженому стані з першою.

Команді варто знати й те, чого векторний пошук не робить. Він ранжує за близькістю, а не за істинністю. Найближчий фрагмент — той, що найбільше схожий на питання; він усе одно може бути застарілим, хибним або взятим із документа, який читачеві бачити не можна. Деякі питання тримаються на одному точному слові, як-от імені, артикулі чи коді помилки, і документація pgvector пропонує поєднувати векторний пошук із повнотекстовим пошуком PostgreSQL. Таке поєднання називають гібридним пошуком.

Як це виглядає на практиці

pgvector у версії 0.8.7, актуальній у жовтні 2026 року, підтримує PostgreSQL 13 і новіші. Розширення вмикають один раз у кожній базі даних, після чого таблиця отримує стовпець типу `vector` з тією кількістю вимірів, яку видає модель:

```
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);
```

Три виміри лише роблять приклад читабельним; справжня модель видає їх значно більше, і стовпець оголошує саме цю кількість. Застосунок обчислює кожен ембединг моделлю і вставляє його поруч із текстом, з якого той походить. Індекс прив’язаний до однієї відстані: `vector_cosine_ops` обслуговує запити, упорядковані за косинусною відстанню, а запитові, упорядкованому за відстанню L2, потрібен ще один індекс, побудований із `vector_l2_ops`. HNSW не потребує етапу навчання, тому індекс можна створити й на таблиці без рядків.

Запит перетворює питання на вектор тією самою моделлю і просить найближчі фрагменти:

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

`<=>` — це косинусна відстань, а `$1` — вектор запиту, переданий як параметр. Дві деталі дають PostgreSQL змогу скористатися індексом: у запиті є і `ORDER BY`, і `LIMIT`, а `ORDER BY` стоїть на самому операторі відстані в порядку зростання. Оцінка схожості для показу, `1 - (embedding <=> $1)`, належить до списку `SELECT`. Інші оператори — `<->` для відстані L2, `<+>` для L1 і `<#>` для скалярного добутку. Останній повертає скалярний добуток зі знаком мінус, бо PostgreSQL сканує індекс за оператором лише в порядку зростання.

HNSW-індекс має два параметри побудови: `m`, найбільшу кількість зв'язків на шар, за замовчуванням 16, і `ef_construction`, розмір списку кандидатів під час побудови, за замовчуванням 64. Під час пошуку розмір списку кандидатів задає `hnsw.ef_search`, за замовчуванням 40. Більше значення дає кращу повноту ціною швидкості, а `SET LOCAL` підвищує його до кінця поточної транзакції. Щоб обмежити налаштування одним запитом, помістіть у транзакцію лише цей запит. Щоб дізнатися, скільки повноти забирає індекс, виконайте той самий запит у транзакції з `SET LOCAL enable_indexscan = off`, що дає точну відповідь, і порівняйте два списки.

У роботі компанії з ШІ-агентами пам'ять агента про минулі справи зберігається у власній базі даних клієнта, а не в історії чату постачальника. Таблиця на кшталт `passages` — одна з форм, яких може набути така пам'ять: коли надходить нова справа, найсхоліші минулі справи лежать за один запит від неї, у тій самій базі даних і під тими самими правами доступу, що й решта даних.

Гібридний пошук виконує обидва пошуки й зводить два списки в один. Документація pgvector називає для цього Reciprocal Rank Fusion зі статті Кормака, Кларка й Бюттхера 2009 року: кожен документ отримує 1 / (k + ранг) за кожен список, у якому він трапляється, і ці частки додаються. Автори взяли k рівним 60, а їхні попередні прогони показали, що це значення

близьке до оптимального, але сам вибір не критичний: «near-optimal, but that the choice was not critical». Якщо в тій самій таблиці є стовпець `search_vec` типу `tsvector`, побудований так само, як для повнотекстового пошуку, об'єднаний запит має такий вигляд:

```
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;
```

Кожен пошук залишає власні 20 найкращих результатів і нумерує їх від 1. Фрагмент, який обидва пошуки ставлять високо, отримує дві частки й піднімається нагору; фрагмент, який знайшов лише один пошук, зберігає одну. Злиття спирається на ранги, а не на оцінки, бо косинусна відстань і текстовий ранг лежать на шкалах, які неможливо порівняти. У `plainto_tsquery` має стояти та сама назва, що й у стовпці, тут `english`. Іншим способом поєднати списки pgvector називає крос-енкодер (cross-encoder).

Де виникають проблеми

Менше рядків, ніж просив `LIMIT`. З наближеним індексом умова `WHERE` застосовується лише після сканування індексу. pgvector наводить розрахунок: якщо умова відповідає 10% рядків, HNSW-індекс зі стандартним значенням `hnsw.ef_search`, що дорівнює 40, у середньому залишає лише близько 4 рядків. Починаючи з версії 0.8.0, ітеративне сканування переглядає дедалі більшу частину індексу, доки фільтр не пропустить

достатньо рядків або не буде досягнуто меж сканування. Увімкніть його через `SET hnsw.iterative_scan = strict_order` або через `relaxed_order`, якщо потрібна краща повнота, а результати можуть трохи порушувати порядок. Для HNSW `hnsw.max_scan_tuples`, за замовчуванням 20 тисяч, — приблизна межа кількості додатково переглянутих кортежів, яка не впливає на початкове сканування. Окрема межа пам'яті також може зупинити подальше сканування. Якщо фільтр має лише кілька різних значень, допоможе й частковий індекс на кожне значення.

Запит, який оминає індекс найближчих сусідів. `ORDER BY 1 - (embedding <=> $1) DESC` сортує за виразом, а запит без `LIMIT` не запитує обмеженого результату найближчих сусідів. Жоден із цих варіантів не відповідає вимогам pgvector для використання такого індексу. Кількість рядків, які прочитає обраний план, залежить також від фільтрів та інших рішень планувальника. Індекс, побудований із `vector_l2_ops`, не допомагає запитові, упорядкованому за `<=>`. Сортуйте за самим оператором, за зростанням і з `LIMIT`, будуйте окремий індекс на кожну відстань, за якою шукаєте, і перевіряйте план запиту через `EXPLAIN (ANALYZE, BUFFERS)`.

IVFFlat-індекс, побудований зарано. IVFFlat запускає k-means на рядках, наявних у момент побудови індексу, і за результатом ділить вектори на списки. Індекс, створений на замалому обсязі даних для свого числа списків, повертає менше рядків, ніж просив запит, і pgvector радить видалити його, доки в таблиці не стане більше даних. Спершу завантажте дані, потім будуйте індекс. pgvector пропонує починати з кількості рядків / 1000 списків, якщо рядків до мільйона, і з квадратного кореня з кількості рядків, якщо більше, а probes, що за замовчуванням дорівнює 1, ставити на рівні квадратного кореня з кількості списків. HNSW такого етапу навчання не має.

Рядки, невидимі для пошуку. NULL-вектори не потрапляють до індексу, як і нульові вектори за косинусної відстані, бо в них немає напрямку, який можна порівняти. Рядок без ембедингу невидимий для пошуку за індексом. Додайте `NOT NULL`, щойно завантаження стане надійним, і налаштуйте регулярний підрахунок рядків, яким досі бракує ембедингу.

Вектори двох моделей в одному стовпці. Дві моделі ембедингів або дві версії однієї розміщують той самий текст у різних місцях, і їхні вектори не можна порівнювати, навіть коли кількість вимірів однакова. Перетворюйте питання на вектор тією моделлю, якою обробили фрагменти, зберігайте

назву моделі в окремому стовпці й перераховуйте всі вектори, коли модель змінюється. Для стовпця з векторами різних розмірів pgvector показує індекс за виразом з умовою для кожної моделі, `WHERE (model_id = 123)` .

Забагато вимірів для індексу. Тип `vector` зберігає до 16 тисяч вимірів, але HNSW та IVFFlat індексують щонайбільше 2000. Індексуйте приведення до `halfvec` , щоб охопити до 4000, застосуйте бінарне квантування, щоб охопити до 64 тисяч, і переранжуйте короткий список за повними векторами, або індексуйте підвектор, якщо модель це підтримує.

Вектори одного клієнта гальмують іншого. pgvector попереджає, що коли кілька клієнтів ділять один наближений індекс, вектори одного з них можуть впливати на повноту й швидкість для інших. Розбийте таблицю за клієнтами за допомогою партиціонування за списком або дайте кожному клієнтові власну таблицю.

Побудова індексу, що блокує застосунок. HNSW будується значно швидше, коли граф уміщується в `maintenance_work_mem` , і pgvector виводить повідомлення, коли він уже не вміщується. Якщо ж задати цю пам'ять настільки великою, що серверу забракне пам'яті, це спричинить окремий простій. На робочій системі будуйте індекс через `CREATE INDEX CONCURRENTLY` , що не блокує запис.

Фрагменти, яких читач не має бачити. Пошук по всіх фрагментах може передати мовній моделі текст із документа, який користувач не має права читати, і модель може його процитувати. Додайте умову прав доступу в той самий `WHERE` , що й пошук, щоб такий рядок ніколи не залишав базу даних, і ввімкніть ітеративне сканування для пошуку додаткових дозволених фрагментів, коли фільтрування відкидає кандидатів, у межах обмежень сканування.

Відстань, яку сприймають як упевненість. Косинусна відстань 0,2 означає, що фрагмент ближчий за той, що має 0,4, а не те, що він відповідає на питання. Поріг, підібраний для однієї моделі, для наступної втрачає сенс. Сортуйте за відстанню, обмежуйте через `LIMIT` і оцінюйте пошук на наборі реальних питань, для яких правильні фрагменти відомі заздалегідь.

Що почитати далі

- pgvector — власна документація розширення: типи, оператори відстані, параметри HNSW та IVFFlat, фільтрування, ітеративне сканування індексу, гібридний пошук і розв'язання проблем.
- Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs — стаття, у якій уперше описано HNSW.
- Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods — доповідь на SIGIR 2009 з формулою злиття та попередніми прогонами, на яких тримається її константа.
- Introduction to full text search — пошук за словами в PostgreSQL, друга половина гібридного запиту.
- Controlling Text Search — `plainto_tsquery` і `ts_rank_cd`, як їх використано в пошуку за словами вище.
- Transaction Isolation — коли знімки PostgreSQL бачать зафіксовані зміни, зокрема чому старіший знімок може й далі бачити документ після фіксації його видалення.

Словник

Обирайте за фактичними умовами

Надайте репрезентативний набір даних, запитання до нього та поточне вузьке місце. Узгодимо вимірювання якості, пропускної здатності та відновлення.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке векторний пошук?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/vector-search/