Engineer

Vektorsøgning

søgning efter vektorer, der ligger nærmest en forespørgsel

Hvad det er

Vektorsøgning finder de lagrede elementer, hvis vektorer ligger tættest på vektoren for en forespørgsel. På engelsk hedder det vector search. En vektor er her en liste af tal med fast længde, som en embeddingmodel danner ud fra et stykke tekst, et billede eller en produktpost. Modellen er trænet, så elementer med beslægtet indhold får vektorer, der ligger tæt på hinanden, og derfor kan et spørgsmål og det afsnit, der besvarer det, ligge tæt sammen, selv om de ikke har et eneste ord til fælles.

»Tættest« kræver et mål. pgvector, den PostgreSQL-udvidelse som artiklens eksempler bygger på, understøtter seks: L2-afstand (euklidisk afstand), indre produkt, cosinusafstand og L1-afstand samt Hamming- og Jaccard-afstand til binære vektorer. Embeddingmodellens egen dokumentation oplyser, hvilket mål dens vektorer er beregnet til. For vektorer, der er normaliseret til længden 1, giver cosinusafstand og indre produkt naboerne i samme rækkefølge, og her anbefaler pgvector indre produkt, fordi det giver den bedste ydelse.

Der er to måder at besvare spørgsmålet på. Eksakt søgning sammenligner forespørgslen med hver eneste lagrede vektor og returnerer de sande nærmeste naboer. Det gør pgvector som standard, og dokumentationen beskriver det som perfekt recall, det vil sige at alle de sande naboer kommer med. Approksimativ søgning kan bruge et indeks, der kun besøger en del af dataene, og bytter noget recall for fart. Hvor mange sande naboer der kommer med, afhænger af dataene, indekset og dets indstillinger; selv et flertal er ikke garanteret. Derfor advarer pgvector om, at et approksimativt indeks i modsætning til et almindeligt indeks ændrer det, en forespørgsel returnerer.

pgvector har to typer approksimativt indeks. HNSW bygger en graf i flere lag, en metode som Malkov og Yashunin første gang offentliggjorde i 2016: hvert lag rummer en delmængde af laget nedenunder, en søgning begynder i det lille øverste lag og arbejder sig ned til det fulde, og deres artikel rapporterer, at søgningens omkostning vokser logaritmisk. IVFFlat deler vektorerne op i lister og søger kun i de lister, der ligger nærmest forespørgslen.

Brug begrebet i en beslutning

Brug »Hvad er vektorsøgning?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Jeres data samlet, hvor AI-agenter kan nå dem, med én udgave af hver kunde, uden at nogen ser mere, end de må, og med hvert svar sporet tilbage til kilden.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har gjort en geodatapipeline 50 gange hurtigere — 50x hurtigere geografiske datapipelines

Har gjort en geodatapipeline 50 gange hurtigere Kontakt os →

Hvorfor det betyder noget

Fuldtekstsøgning matcher ord. Med ordmatchning og stemming alene finder en kunde, der skriver »skærmen er helt sort«, ikke manualens afsnit med overskriften »displayet tænder ikke«, fordi de to ikke deler et eneste ord, heller ikke når endelserne er skåret af. Konfigurerede synonymer kan bygge bro over nogle af de huller. Vektorsøgning sammenligner betydning, sådan som embeddingmodellen har opfanget den, så klagen og afsnittet kan lande tæt på hinanden. Samme idé virker ud over tekst: et billede, der ligner dette, en supportsag, der ligner en allerede løst, et produkt, der ligner det i kurven.

Vektorsøgning er også én måde at klare hentningen på i retrieval-augmented generation, hvor en stor sprogmodel svarer ud fra tekstafsnit, der er hentet til spørgsmålet. Spørgsmålet laves om til en vektor med den samme model som afsnittene, og de nærmeste afsnit sendes med i prompten. Svaret kan aldrig blive bedre end det, søgningen fandt, så søgningen fortjener den samme omhu som modellen.

At beholde vektorerne i den database, man allerede kører, har en praktisk side, og pgvector indleder sin dokumentation med netop det argument. Vektorerne får de samme transaktioner, den samme point-in-time recovery og de samme joins som alle andre kolonner, og fordi pgvector skriver til write-ahead-loggen, følger de også med ved replikering. Et filter på kunde, dato eller rettigheder er en almindelig `WHERE` -betingelse på den samme række. Et snapshot taget efter,

at en sletning er committet, ser ikke længere dokumentet, men en forespørgsel, der bruger et ældre snapshot, kan stadig se det. Et separat vektorlager er et system mere, der skal fyldes og holdes i takt med det første.

Et team bør også vide, hvad vektorsøgning ikke gør. Den rangerer efter nærhed, ikke efter sandhed. Det nærmeste afsnit er det, der ligner spørgsmålet mest; det kan stadig være forældet, forkert eller stamme fra et dokument, læseren ikke må se. Nogle spørgsmål står og falder med et bestemt ord, for eksempel et navn, et varenummer eller en fejlkode, og pgvectors dokumentation foreslår at bruge vektorsøgning sammen med PostgreSQL's fuldtekstsøgning. Den kombination kaldes hybrid søgning.

Sådan ser det ud i praksis

pgvector, der i oktober 2026 er i version 0.8.7, understøtter PostgreSQL 13 og nyere. Udvidelsen aktiveres én gang i hver database, og derefter får tabellen en kolonne af typen `vector` med det antal dimensioner, modellen leverer:

```
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);
```

Tre dimensioner holder eksemplet til at læse; en rigtig model leverer langt flere, og kolonnen angiver præcis det tal, modellen leverer. Programmet beregner hver embedding med modellen og indsætter den ved siden af den tekst, den kom fra. Indekset hører til én bestemt afstand: `vector_cosine_ops` betjener forespørgsler, der sorteres efter cosinusafstand, og en forespørgsel, der sorteres efter L2-afstand, kræver et ekstra indeks bygget med `vector_l2_ops`. HNSW har intet træningstrin, så indekset kan oprettes på en tabel uden rækker.

Forespørgslen laver spørgsmålet om til en vektor med samme model og beder om de nærmeste afsnit:

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

`<=>` er cosinusafstand, og `$1` er forespørgslens vektor, sendt som parameter. To detaljer gør det muligt for PostgreSQL at bruge indekset: forespørgslen har både `ORDER BY` og `LIMIT`, og `ORDER BY` står på selve afstandsoperatoren i stigende rækkefølge. En lighedsscore til visning, `1 - (embedding <=> $1)`, hører hjemme i select-listen. De øvrige operatorer er `<->` for L2-afstand, `<+>` for L1 og `<#>` for indre produkt. Den sidste returnerer det negative indre produkt, fordi PostgreSQL kun scanner et indeks på en operator i stigende rækkefølge.

HNSW-indekset har to byggeparametre: `m`, det største antal forbindelser pr. lag, 16 som standard, og `ef_construction`, størrelsen på kandidatlisten under opbygningen, 64 som standard. Når der søges, bestemmer `hnsw.ef_search` størrelsen på kandidatlisten, 40 som standard. En højere værdi giver bedre recall på bekostning af hastigheden, og `SET LOCAL` hæver den indtil slutningen af den aktuelle transaktion. Skal indstillingen kun gælde én forespørgsel, skal transaktionen kun indeholde den forespørgsel. Vil man vide, hvad indekset koster i recall, kører man den samme forespørgsel i en transaktion med `SET LOCAL enable_indexscan = off`, som giver det eksakte svar, og sammenligner de to lister.

Virksomhedens arbejde med skræddersyede AI-agenter lægger agentens hukommelse om tidligere sager i kundens egen database frem for i en leverandørs chathistorik. En tabel som `passages` er én form, sådan en hukommelse kan have: når en ny sag kommer ind, er de tidligere sager, der ligner den mest, kun én forespørgsel væk, i den samme database og under de samme rettigheder som resten af dataene.

Hybrid søgning kører begge søgninger og fletter de to lister. pgvectors dokumentation nævner Reciprocal Rank Fusion fra en artikel fra 2009 af Cormack, Clarke og Büttcher: hvert dokument får 1 / (k + rang) for hver liste, det optræder i, og pointene lægges sammen. Forfatterne fastsatte k til 60, og deres indledende forsøg viste, at 60 var »near-optimal, but that the choice was not critical«. Med en `tsvector`-kolonne `search_vec` i samme tabel, bygget som til fuldtekstsøgning, ser den flettede forespørgsel sådan ud:

```
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('danish', $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;
```

Hver søgning beholder sine egne 20 bedste og nummererer dem fra 1. Et afsnit, som begge søgninger placerer højt, får to andele og stiger til tops; et afsnit, som kun den ene søgning finder, beholder én andel. Fletningen bruger rang frem for score, fordi en cosinusafstand og en tekstrang ligger på skalaer, der ikke kan sammenlignes. Sproget i `plainto_tsquery` skal være det samme som i kolonnen, her `danish`. pgvector nævner en cross-encoder som den anden måde at kombinere listerne på.

Hvor det går galt

Færre rækker, end `LIMIT` bad om. Med et approksimativt indeks anvendes `WHERE`-betingelsen først efter indeksscanningen. pgvector viser regnestykket: hvis en betingelse matcher 10 % af rækkerne, giver et HNSW-indeks med standardværdien 40 for `hnsw.ef_search` i gennemsnit kun omkring 4 rækker. Siden version 0.8.0 scanner iterative indeksscanninger mere af indekset, indtil nok rækker er kommet igennem filteret, eller scanningsgrænserne er nået. Slå dem til med `SET hnsw.iterative_scan = strict_order`, eller med `relaxed_order` for bedre recall med resultater, der står lidt ude af rækkefølge. For HNSW er `hnsw.max_scan_tuples`, 20.000 som standard, en omtrentlig grænse for yderligere besøgte tupler, som ikke påvirker den første scanning. En særskilt hukommelsesgrænse kan også stoppe yderligere scanning. Har filteret kun få forskellige værdier, virker et partielt indeks pr. værdi også.

En forespørgsel, der ikke rammer nærmeste-nabo-indekset. `ORDER BY 1 - (embedding <=> $1) DESC` sorterer efter et udtryk, og en forespørgsel uden `LIMIT` beder ikke om et afgrænset nærmeste-nabo-resultat. Ingen af delene opfylder pgvectors krav til at bruge det indeks. Hvor mange rækker den valgte plan læser, afhænger også af filtre og andre valg i planlæggeren. Et indeks bygget med `vector_l2_ops` gør intet for en forespørgsel, der sorteres efter `<=>`. Sortér efter selve operatoren, stigende og med `LIMIT`, byg ét indeks for hver afstand, der søges med, og tjek planen med `EXPLAIN (ANALYZE, BUFFERS)`.

Et IVFFlat-indeks bygget for tidligt. IVFFlat kører k-means over de rækker, der findes, når indekset bygges, og deler vektorerne op i lister ud fra resultatet. Et indeks, der er oprettet med for lidt data til sit antal lister, returnerer færre rækker, end forespørgslen bad om, og pgvectors råd er at droppe det, indtil tabellen rummer flere data. Indlæs dataene først, og byg indekset bagefter. pgvector foreslår at begynde med rækker / 1000 lister op til en million rækker og kvadratroden af antallet af rækker derover, og at sætte probes, der som standard er 1, til kvadratroden af antallet af lister. HNSW har intet tilsvarende træningstrin.

Rækker, der aldrig bliver fundet. NULL-vektorer kommer ikke med i indekset, og det gør nulvektorer heller ikke under cosinusafstand, for de har ingen retning at sammenligne. En række, der står uden embedding, er usynlig for en indekseret søgning. Tilføj `NOT NULL`, når indlæsningen er pålidelig, og lad et fast job tælle de rækker, der stadig mangler en embedding.

Vektorer fra to modeller i én kolonne. To embeddingmodeller, eller to versioner af den samme, placerer den samme tekst forskellige steder, og deres vektorer kan ikke sammenlignes, selv når antallet af dimensioner er det samme. Lav spørgsmålets vektor med den model, der lavede afsnittenes, gem modellens navn i en kolonne, og beregn alle vektorer igen, når modellen skifter. Til en kolonne med vektorer i flere størrelser viser pgvector et udtryksindeks med en betingelse for hver model, `WHERE (model_id = 123)`.

For mange dimensioner til indekset. Typen `vector` gemmer op til 16.000 dimensioner, men HNSW og IVFFlat indekserer højst 2.000. Indeksér en cast til `halfvec` for op til 4.000, brug binær kvantisering for op til 64.000 og rangér den korte liste igen efter de fulde vektorer, eller indeksér en delvektor, når modellen understøtter det.

Én kundes vektorer gør en andens søgning langsommere. pgvector advarer om, at når flere kunder deler et approksimativt indeks, kan én kundes vektorer påvirke recall og hastighed for de andre. Partitionér tabellen efter kunde med listepartitionering, eller giv hver kunde sin egen tabel.

En indeksopbygning, der blokerer programmet. HNSW bygges langt hurtigere, når grafen kan ligge i `maintenance_work_mem`, og pgvector udskriver en meddelelse, når den ikke længere kan. At sætte den hukommelse så højt, at serveren løber tør, giver sin egen nedetid. På et system i drift bygger man med `CREATE INDEX CONCURRENTLY`, som ikke blokerer skrivninger.

Afsnit, læseren ikke må se. En søgning over alle afsnit kan give en sprogmodel tekst fra et dokument, brugeren ikke har ret til at læse, og modellen kan citere den. Læg rettighedsbetingelsen i den samme `WHERE`-betingelse som søgningen, så sådan en række aldrig forlader databasen, og slå iterative scanninger til for at søge efter flere tilladte afsnit, når filtreringen fjerner kandidater, inden for scanningsgrænserne.

En afstand opfattet som vished. En cosinusafstand på 0,2 siger, at et afsnit ligger nærmere end et på 0,4, ikke at det besvarer spørgsmålet. En tærskel, der er indstillet til én model, siger intet om den næste. Sortér efter afstand, begræns med `LIMIT`, og bedøm søgningen på et sæt rigtige spørgsmål, hvor man kender de rigtige afsnit på forhånd.

Læs mere

- pgvector — udvidelsens egen dokumentation: typer, afstandsoperatorer, indstillinger for HNSW og IVFFlat, filtrering, iterative indeksscanninger, hybrid søgning og fejlfinding.
- Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs — Malkov og Yashunins artikel, der introducerede HNSW.
- Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods — SIGIR-artiklen fra 2009 med fletteformlen og de indledende forsøg bag dens konstant.
- Introduction to full text search — PostgreSQL's ordbaserede søgning, den anden halvdel af en hybrid forespørgsel.
- Controlling Text Search — `plainto_tsquery` og `ts_rank_cd`, som de bruges i nøgleordsdelen ovenfor.

- Transaction Isolation — hvornår snapshots i PostgreSQL ser committede ændringer, herunder hvorfor et ældre snapshot stadig kan se et dokument, efter at sletningen er committet.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring et repræsentativt datasæt, de spørgsmål det skal besvare, og den aktuelle flaskehals. Vi kan aftale målinger af kvalitet, gennemløb og gendannelse.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er vektorsøgning?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af **D.**

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/vector-search/