Building AI Applications with PostgreSQL: A busy dev's guide

Avthar Sewrathan (@avthars on X)
PM for AI and Vector @ Timescale

Timescale

What will you learn today?

→ **For beginners:**

- Understand what kind of AI apps you can build with PostgreSQL.
- Learn enough to start building.

→ **For intermediates:**

- Learn ideas and tactics for how to improve your AI application.

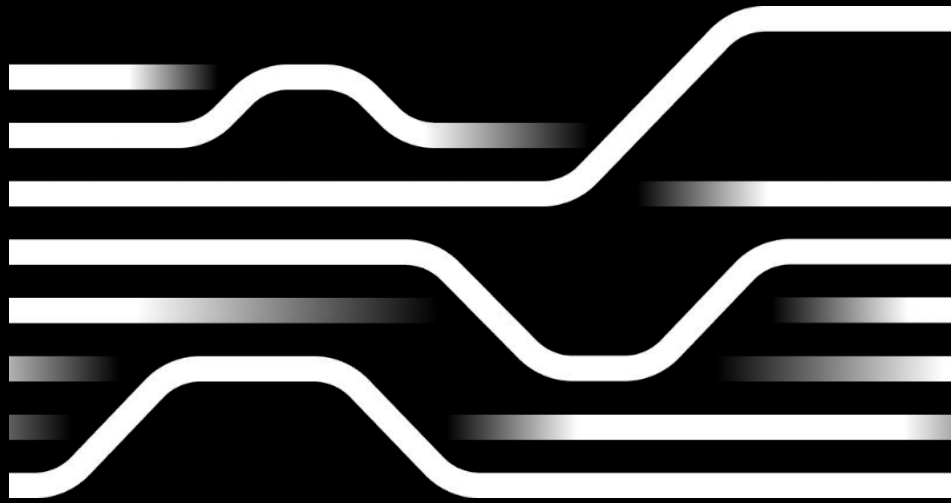
Webinar Roadmap

1. Foundations: PostgreSQL for AI Systems
2. Vector search indexes in PostgreSQL
3. Improving your AI system: Advanced Topics
4. Q+A

01

Foundations

Building AI Systems with PostgreSQL



You can build AI applications with PostgreSQL

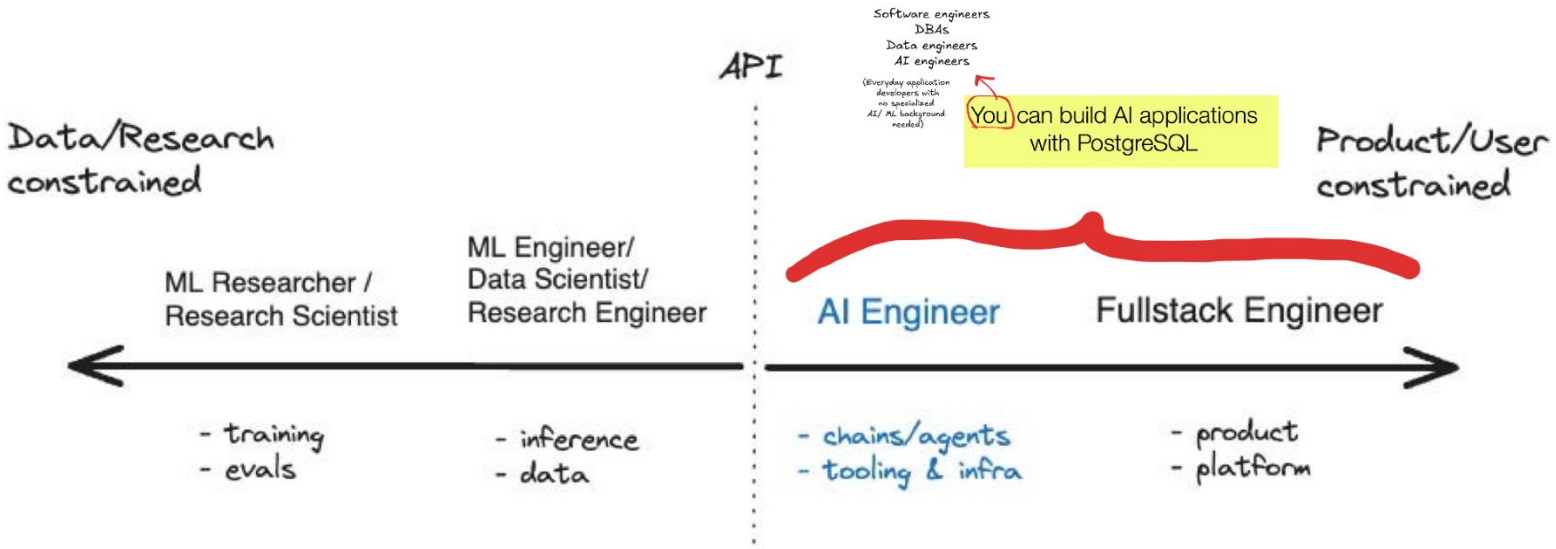
Software engineers
DBAs
Data engineers
AI engineers

(Everyday application developers with no specialized AI/ ML background needed)

You can build AI applications with PostgreSQL

Rise of the AI Engineer:

Application Dev who uses AI to build products (that's you!)



Source: https://www.latent.space/p/ai-engineer

PostgreSQL as a **vector database**

Vector data + Vector databases crash course

Vector = compressed representation of data

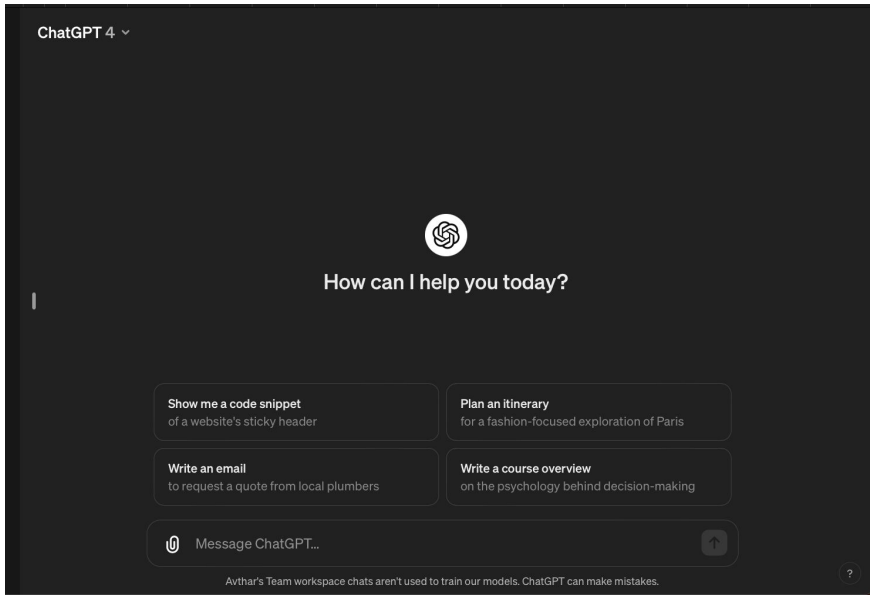
[1, 2, 4, 5 ,6 …]

[50, 345, 2231, 200…]

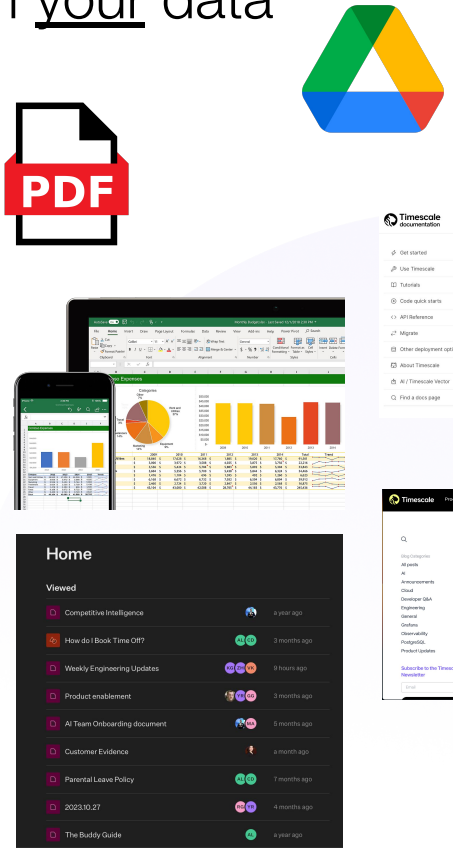
[40, 34, 09, 73, 11, 111 …]

Vector databases store and search vectors efficiently

Vector databases make LLMs work with your data



+

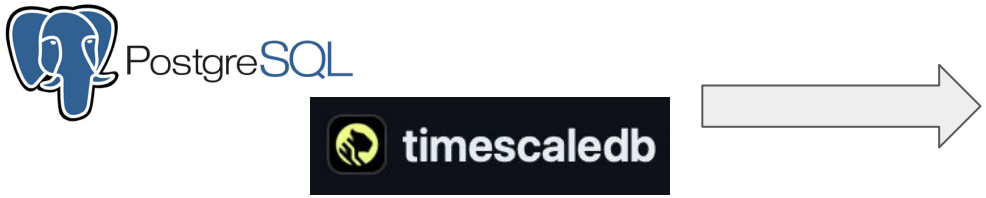


What AI systems is PostgreSQL good for?

Use case	**RAG**	**Search**	**Agents**	**Text to SQL**	**And more**
Description	"ChatGPT" but with your company/ customer data *(Retrieval Augmented Generation)*	Search and find relevant information by meaning (not keyword/ tag)	ChatGPT that can use tools, plan, and act autonomously Tools = web search, database query, code, APIs etc.	ChatGPT but for numerical, structured, tabular data Asking questions in English vs writing formulas / code	Recommendation Systems (RecSys) Anomaly Detection
Example apps	- Customer support chatbot - Research Copilot - Docs chatbot	- Semantic Search - Image/ video search	- AI Software engineer (e.g Devin, Replit Agents)	- "Chat with your data" - Data / Financial Analysis Agent	- Time-series anomaly detection - E-commerce purchase recommendations

PostgreSQL Extensions for AI and Vector

The power of PostgreSQL extensions



Postgres is a time-series database!

Postgres is a vector database!

PostgreSQL extensions for AI applications

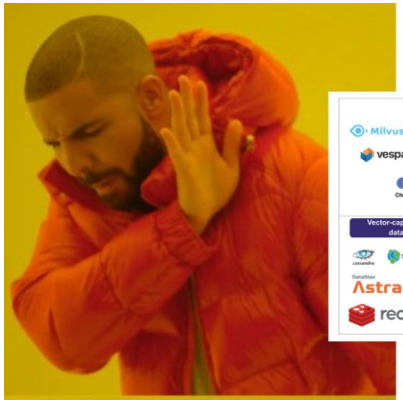
AI extension	**Why it's useful**	**License**
pgvector	• Gives PostgreSQL vector database super powers! • Vector data type, distance functions (cosine, L1, L2, inner product) • Vector search indexes (HNSW, IVFFLAT)	Open-source (PostgreSQL)
pgvectorscale	• Speeds up pgvector for large scale workloads • Complement to pgvector (you use them together) • High accuracy filtered search • Vector search index (StreamingDiskANN)	Open-source (PostgreSQL)
pgai	• Brings AI workflows to PostgreSQL • Embedding creation • In-database LLM reasoning (summarization, moderation, categorization)	Open-source (PostgreSQL)

PostgreSQL extensions for AI applications

vector_and_ai_extensions.sql

```sql
-- install pgvector, pgai and pgvectorscale extensions
create extension if not exists pgvector;
create extension if not exists ai;
create extension if not exists vectorscale;

-- installing pgai or pgvectorscale will also install pgvector
create extension if not exists ai cascade;
create extension if not exists vectorscale cascade;
```

PostgreSQL is all you need

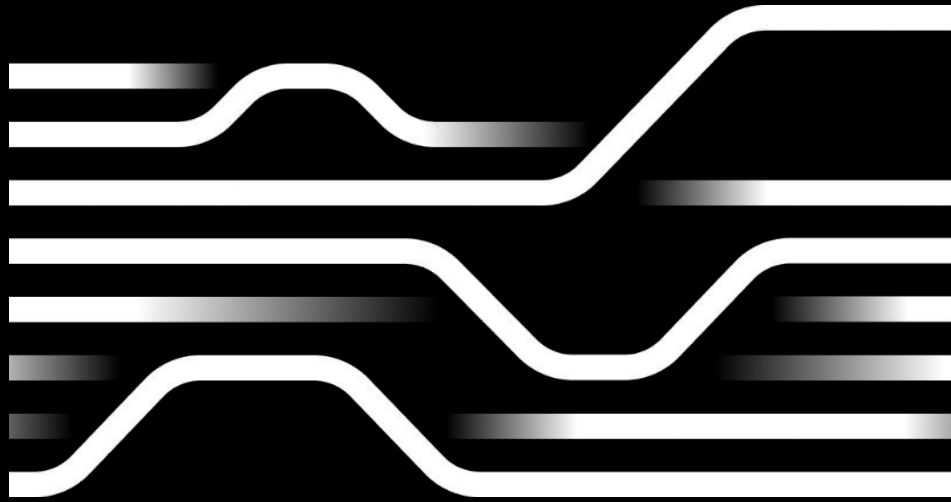


[Demo time]
Intro to pgvector and pgai

02

Vector search indexes

Improve performance of vector search in PostgreSQL



When / why do you need a vector index?

- TL;DR: More than ~100k vectors in a single index
- Approximate Nearest Neighbour search:
  - Trades off some accuracy for faster search speed.
- 3 vector search indexes in PostgreSQL
  - IVFFLAT - Inverted File Flat
  - HNSW - Hierarchical Navigable Small Worlds
  - StreamingDiskANN

Vector Search Indexes for pgvector / PostgreSQL

Vector index	**Best for**	**Pros**	**Cons**
IVFFLAT	Medium workloads (100k-1M vectors)	• Low memory usage • Faster than linear scan	• Updates require rebuilding the index every time • Lower accuracy vs HNSW, StreamingDiskANN
HNSW	Real-time search on medium workloads (100k-1M vectors)	• Good balance between speed and accuracy • Handles updates without rebuild • Quantization (halfvec, PQ, BQ) • Fast (parallel) index build	• Problems with filtered search accuracy • High memory usage • Cost scales with RAM • Scale limitations
StreamingDiskANN (pgvectorscale)	• Real-time search • Filtered search use cases • Large scale workloads (10M+ vectors)	• High accuracy filtered search • Scales to 1B+ vectors • Statistical Binary Quantization on by default • Cost scales with disk and RAM • Handles updates without rebuild	• Longer build times time than HNSW

Vector search indexes in PostgreSQL

pgvector_indexes.sql

```sql
-- create an index on the embedding column using ivfflat
CREATE INDEX quotes_embedding_idx ON quotes
USING ivfflat (embedding vector_cosine_ops);

-- create an index on the embedding column using HNSW
CREATE INDEX quotes_embedding_idx ON quotes
USING hnsw (embedding vector_cosine_ops);

-- Create an index on the embedding column using StreamingDiskANN
CREATE INDEX quotes_embedding_idx ON quotes
USING diskann (embedding);
```

Vector search indexes in PostgreSQL

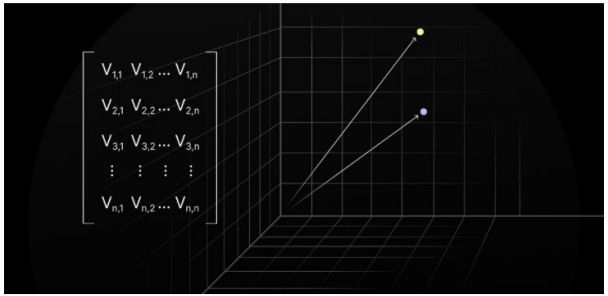
pgvector_indexes.sql

```sql
-- Create an IVFFlat index on the embedding column
-- Custom options: lists = 100
CREATE INDEX ON quotes
USING ivfflat (embedding vector_cosine_ops)
WITH (lists = 100);

-- Create an HNSW index on the embedding column
-- Custom options: m = 16, ef_construction = 64
CREATE INDEX ON quotes
USING hnsw (embedding vector_cosine_ops)
WITH (m = 16, ef_construction = 64);

-- Create a StreamingDiskANN index on the embedding column
-- Custom options: storage_layout = 'memory_optimized', num_neighbors = 50, max_alpha = 1.2
CREATE INDEX quotes_embedding_idx ON quotes
USING diskann (embedding) WITH (
    storage_layout = 'memory_optimized',
    num_neighbors = 50,
    max_alpha = 1.2
);
```

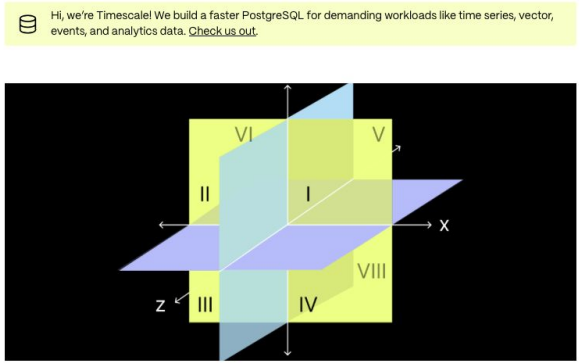
Vector Search Indexes for pgvector / PostgreSQL

Nearest Neighbor Indexes: What Are IVFFlat Indexes in Pgvector and How Do They Work

Vector Database Basics: HNSW



How We Made PostgreSQL as Fast as Pinecone for Vector Data



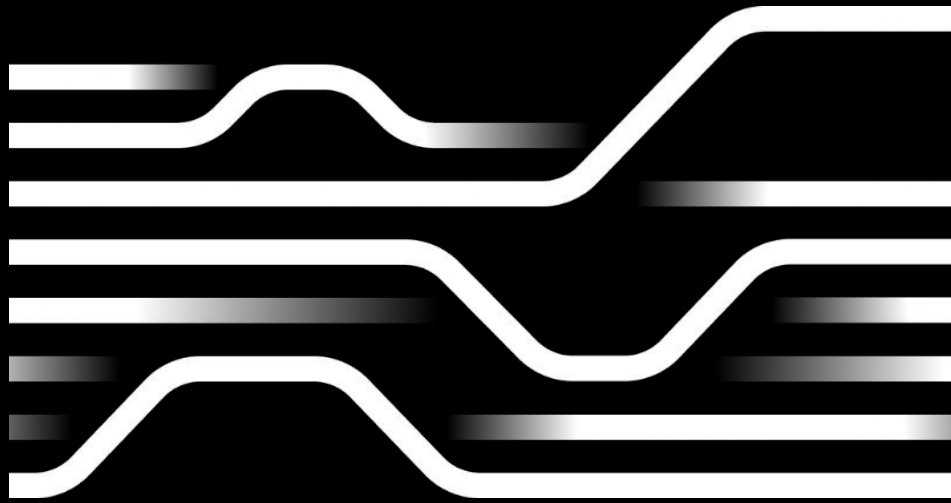
Further reading

- https://www.timescale.com/blog/nearest-neighbor-indexes-what-are-ivfflat-indexes-in-pgvector-and-how-do-they-work/
- https://www.timescale.com/blog/vector-database-basics-hnsw/
- https://www.timescale.com/blog/how-we-made-postgresql-as-fast-as-pinecone-for-vector-data/

03

Improving your AI system

Subtitle



Advanced Topics for building AI apps with PostgreSQL

1. **Evaluation-driven development**
2. **Filtered search:** Combining vector search with filters / WHERE clauses
3. **Hybrid search:** Combining vector and keyword search
4. **Multi-tenancy for RAG apps**
5. **Text to SQL (and RAG/ Agents with tool use)**

Evaluation driven development

Evals driven development

1. Start with set of questions you want the system to answer → this is your evaluation set.
2. As you make changes to the system, you can now measure improvement/ decline!
    - New embedding model, new filter, new LLM etc.
    - Add to the eval set as you make your app more complex
3. Decompose the problem, don't just focus on the LLM answer
    - RAG: Are we fetching the right chunks/ documents?
    - Agents: Is the LLM selecting the right tools?
    - Text to SQL: Are we querying the right tables?

Further reading
RAG Is More Than Just Vector Search

Filtered vector search

Metadata Filter

pgvector_metadata_filter.sql

```sql
-- 1. Metadata filter
-- Use case: A technical documentation search system for a software company
-- with multiple products.

SELECT
    content,
    (embedding <=> query_embedding) AS cosine_distance
FROM documents
WHERE product = 'CRM Software' AND document_type = 'API Reference'
ORDER BY cosine_distance
LIMIT 5;
```

Composite Filter

pgvector_composite_filter.sql

```sql
-- 2. Composite filter
-- Use case: An e-commerce product recommendation system that considers both
user preferences and product attributes.

SELECT
    product_name,
    (product_embedding <=> user_preference_embedding) AS cosine_distance
FROM products
WHERE category IN ('Electronics', 'Computers')
  AND price BETWEEN 500 AND 2000
  AND stock_status = 'In Stock'
  AND rating >= 4.0
ORDER BY cosine_distance
LIMIT 10;
```

Time-based Filter

Pro-tip: Combine with TimescaleDB hypertables for most efficient search

pgvector_time_filter.sql

```sql
-- 3. Time-based filter
-- Use case: A news article recommendation system that prioritizes recent
content.

SELECT
    title,
    content,
    (article_embedding <=> query_embedding) AS cosine_distance
FROM news_articles
WHERE published_date >= CURRENT_DATE - INTERVAL '7 days'
ORDER BY cosine_distance, published_date DESC
LIMIT 5;
```

Permissions -based Filter

pgvector_permissions_filter.sql

```sql
-- 4. Permissions-based filter
-- Use case: A company-wide knowledge base where access to certain documents
is restricted based on user roles.

SELECT
    document_title,
    content,
    (document_embedding <=> query_embedding) AS cosine_distance
FROM knowledge_base
WHERE (
    document_access_level <= :user_access_level
    OR document_id IN (
        SELECT document_id
        FROM user_document_permissions
        WHERE user_id = :current_user_id
    )
)
ORDER BY cosine_distance
LIMIT 10;
```

Geo-spatial Filter

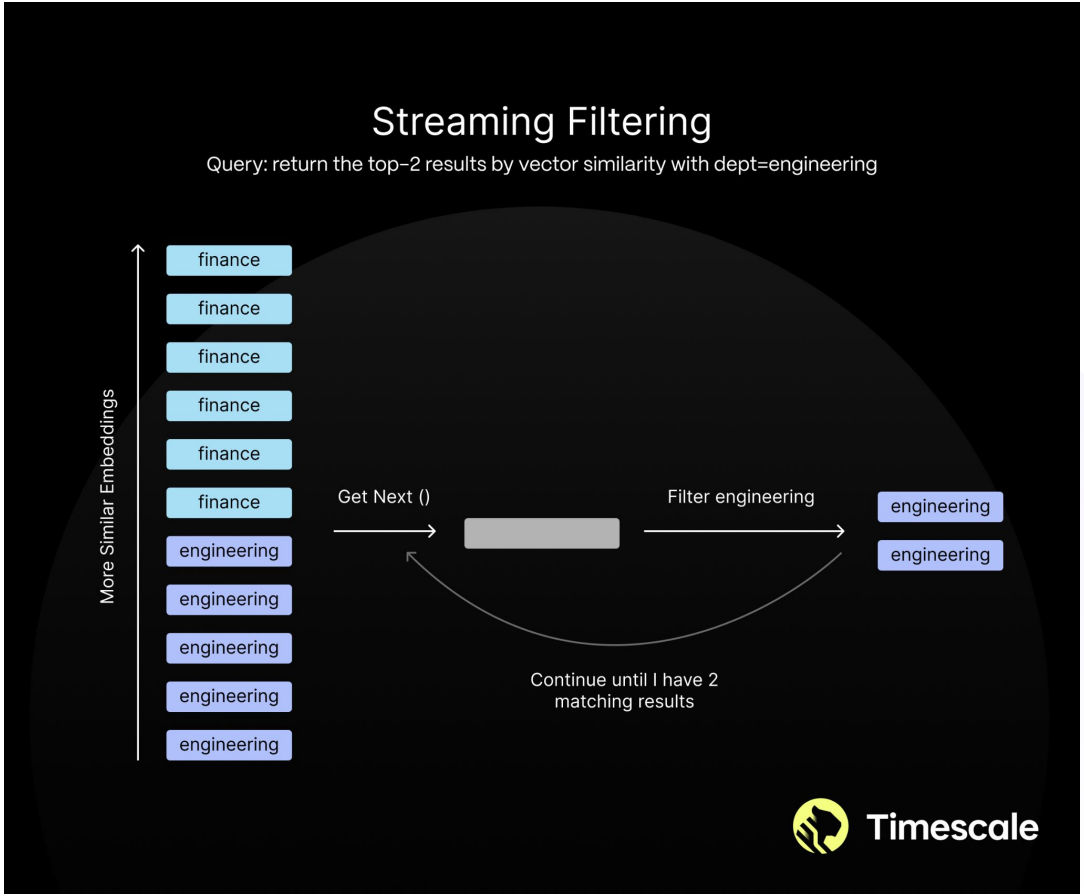
Combine pgvector with PostGIS

pgvector_geospatial_filter.sql

```sql
-- 5. Geospatial filter
-- Use case: A location-based service recommendation system for tourists.

SELECT
    place_name,
    description,
    (place_embedding <=> query_embedding) AS cosine_distance,
    ST_Distance(
        ST_MakePoint(longitude, latitude),
        ST_MakePoint(:user_longitude, :user_latitude)
    ) AS geographic_distance
FROM tourist_attractions
WHERE ST_DWithin(
    ST_MakePoint(longitude, latitude),
    ST_MakePoint(:user_longitude, :user_latitude),
    5000  -- 5000 meters radius
)
ORDER BY cosine_distance, geographic_distance
LIMIT 10;
```

Pro-tip

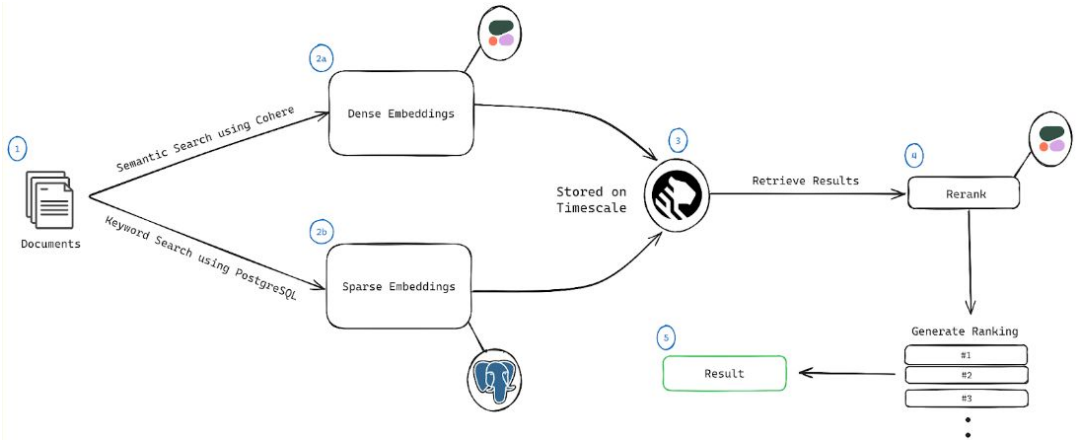
Use pgvectorscale and the StreamingDiskANN index to get higher search accuracy for filtered search



Hybrid Search

Hybrid Search: Combine vector and keyword search

1. **Goal:** Get more accurate/ useful results
2. **High level:** Combine vector / semantic search with full-text / keyword based search.
3. **PostgreSQL implementation:** Combine pgvector / pgvectorscale vector search with native PostgreSQL full text search (tsearch2/ tsvector)



Further reading

[Hybrid search with PostgreSQL and pgvector | Jonathan Katz]

[PostgreSQL Hybrid Search Using pgvector and Cohere]

Hybrid Search

- Code example using pgai extension and Cohere embedding and re-ranker models

Further reading
[Build Search and RAG Systems on PostgreSQL Using Cohere and Pgai]

hybrid_search_pgai_cohere.sql

```sql
-- Hybrid search query combining full-text search, vector search, and reranking

-- Full-text search using PostgreSQL's built-in text search capabilities
with full_text_search as
(
   select article
   from cnn_daily_mail
   where article @@ to_tsquery('english', '(death | kill) & police & car & dog')
   limit 15  -- Limit to top 15 results
)
-- Generate embedding for the search query
, vector_query as
(
   select cohere_embed
   ('embed-english-v3.0'
   , 'Show me stories about police reports of deadly happenings involving cars and dogs.'
   , _input_type=>'search_query'
   ) as query_embedding
)
-- Vector similarity search using the generated embedding
, vector_search as
(
   select article
   from cnn_daily_mail
   order by embedding <=> (select query_embedding from vector_query limit 1)
   limit 15  -- Limit to top 15 results
)
-- Rerank the combined results from full-text and vector searches
, rerank as
(
   select cohere_rerank
   ( 'rerank-english-v3.0'
   , 'Show me stories about police reports of deadly happenings involving cars and dogs.'
   , (
      select jsonb_agg(x.article)
      from
      (
          select *
          from full_text_search
          union
          select * from vector_search
      ) x
    )
   , _top_n => 5  -- Return top 5 results after reranking
   , _return_documents => true
   ) as response
)
-- Final selection of reranked results
select
 x.index
, x.document->>'text' as article
, x.relevance_score
from rerank
cross join lateral jsonb_to_recordset(rerank.response->'results') x(document jsonb, index int, relevance_score float8)
order by relevance_score desc
;
```

Multi-tenancy

Strategies for multi-tenant RAG in Postgres / pgvector

Each tenant gets its own…	Best for	Pros	Cons
Table	Simple apps with shared data	• Simple to implement • Efficient resource usage • Easy to manage and update	• Limited data isolation • Potential for data leaks • Complex queries with many tenant IDs
Schema	**Most use cases,** balancing isolation and efficiency	Better data isolation • Simplified queries • Minimal operational and cost overhead	• Slightly more complex to set up • May hit schema limits in large systems
Logical database	Clients wanting stricter separation	• Strong data isolation • Independent management	• More complex to manage • Higher overhead for connections • High backup and maintenance complexity
Database service	High-security scenarios or clients with unique needs	• Complete isolation • Highest level of security • Full customization per tenant	• Most expensive • Complex to manage and scale • Highest resource overhead

Text to SQL

Building Text to SQL systems with PostgreSQL

- Use case 1: Enable users to query structured data (SQL tables) in natural language
- Use case 2: AI Agent with tools to do semantic search and SQL querying, decides when to use which depending on user query.
- Contact us if you have this use case, Timescale can help!

(Read this blog) –>

RAG Is More Than Just Vector Search

Implementing Text-to-SQL

The final step of our application also involves building a text-to-SQL (Text2SQL) tool as a catch-all for more complex queries.

Developing an effective Text2SQL agent is crucial for translating natural language queries into precise database operations.

In this section, we'll review some tips for developing these agents by looking at a prompt we developed for TimescaleDB-specific query generation.

Use verbose prompts

While concise prompts seem appealing, we've found that verbose prompts are key to unlocking accurate SQL generation. Here's why:

- **Rich context**: verbose prompts provide your AI model with comprehensive context, ensuring it grasps the nuances of your specific database schema and requirements.
- **Domain-specific knowledge**: by including detailed information about your database structure, table relationships, and TimescaleDB-specific functions like time_bucket, you equip the model with essential domain knowledge.
- **Clear boundaries**: explicit instructions and constraints create a framework for the model, preventing

Resources and Next Steps

Building

pgvectorscale Github
https://github.com/timescale/pgvectorscale/

Pgai Github
https://github.com/timescale/pgai

Want to spend more time building your app and less time managing your database?

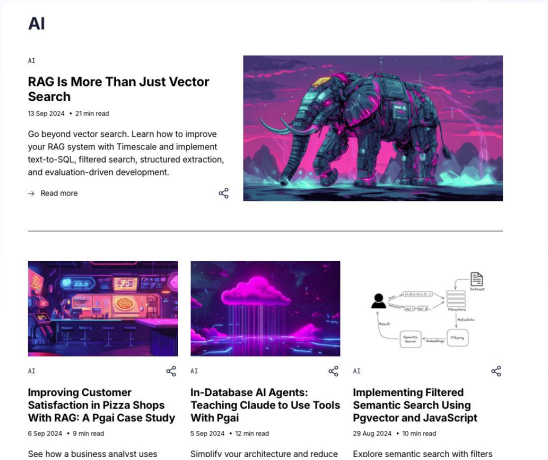
Try Timescale Cloud
https://tsdb.co/ai-webinar

Reading

RAG is more than vector search
https://www.timescale.com/blog/rag-is-more-than-just-vector-search/

Timescale AI blog
https://www.timescale.com/blog/tag/ai/

- Beginner and intermediate tutorials, guides, explainers



Question time!