

[Database](#) > [Extensions](#) > [pgvector: Embeddings and vector similarity](#) >

pgvector: Embeddings and vector similarity

[pgvector](#) is a PostgreSQL extension for vector similarity search. It can also be used for storing [embeddings](#).

 The name of pgvector's Postgres extension is vector.

Learn more about Supabase's [AI & Vector](#) offering.

Concepts

Vector similarity

Vector similarity refers to a measure of the similarity between two related items. For example, if you have a list of products, you can use vector similarity to find similar products. To do this, you need to convert each product into a "vector" of numbers, using a mathematical model. You can use a similar model for text, images, and other types of data. Once all of these vectors are stored in the database, you can use vector similarity to find similar items.

Embeddings

This is particularly useful if you're building on top of OpenAI's [GPT-3](#). You can create and store [embeddings](#) for retrieval augmented generation.

Usage

Enable the extension

Dashboard SQL

- 1 Go to the [Database](#) page in the Dashboard.
- 2 Click on **Extensions** in the sidebar.
- 3 Search for "vector" and enable the extension.

Usage

Create a table to store vectors

```
1 create table posts (  
2   id serial primary key,  
3   title text not null,  
4   body text not null,  
5   embedding vector(384)  
6 );
```

Storing a vector / embedding

In this example we'll generate a vector using Transformer.js, then store it in the database using the Supabase client.

```
1 import { pipeline } from '@xenova/transformers'  
2 const generateEmbedding = await pipeline('feature-extraction', 'Supabase/gte-small')  
3  
4 const title = 'First post!'  
5 const body = 'Hello world!'  
6  
7 // Generate a vector using Transformers.js  
8 const output = await generateEmbedding(body, {  
9   pooling: 'mean',  
10  normalize: true,
```

```
11 })
12
13 // Extract the embedding output
14 const embedding = Array.from(output.data)
15
16 // Store the vector in Postgres
17 const { data, error } = await supabase.from('posts').insert({
18   title,
19   body,
20   embedding,
21 })
```

Specific usage cases

Queries with filtering

If you use an IVFFlat or HNSW index and naively filter the results based on the value of another column, you may get fewer rows returned than requested.

For example, the following query may return fewer than 5 rows, even if 5 corresponding rows exist in the database. This is because the embedding index may not return 5 rows matching the filter.

```
1 SELECT * FROM items WHERE category_id = 123 ORDER BY embedding <-> '[3,1,2]' LIMIT 5;
```

To get the exact number of requested rows, use [iterative search](#) to continue scanning the index until enough results are found.

More pgvector and Supabase resources


[Supabase Clippy: ChatGPT for Supabase Docs](#)


[Storing OpenAI embeddings in Postgres with pgvector](#)


[A ChatGPT Plugins Template built with Supabase Edge Runtime](#)

[Template for building your own custom ChatGPT style doc search](#)

Edit this page on GitHub 

 Need some help? [Contact support](#)

 Latest product updates? [See Changelog](#)

 Something's not right? [Check system status](#)

© Supabase Inc

Contributing

Author Styleguide

Open Source

SupaSquad

Privacy Settings

