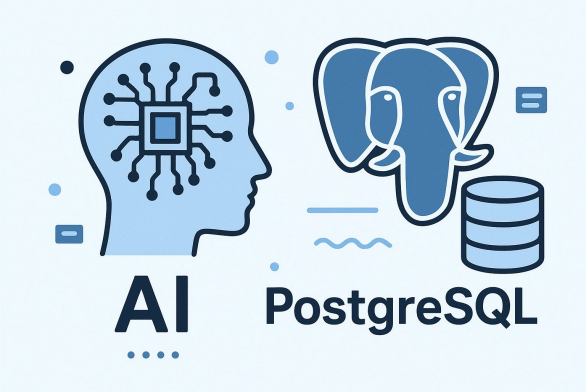
# Harnessing the Power of **AI with Postgres**




August 2025, PG Armenia x Percona University

Emma Saroyan, Generative AI for Web Development Co-author

# Generative Artificial Intelligence

**LLMs**

**OpenAI APIs**

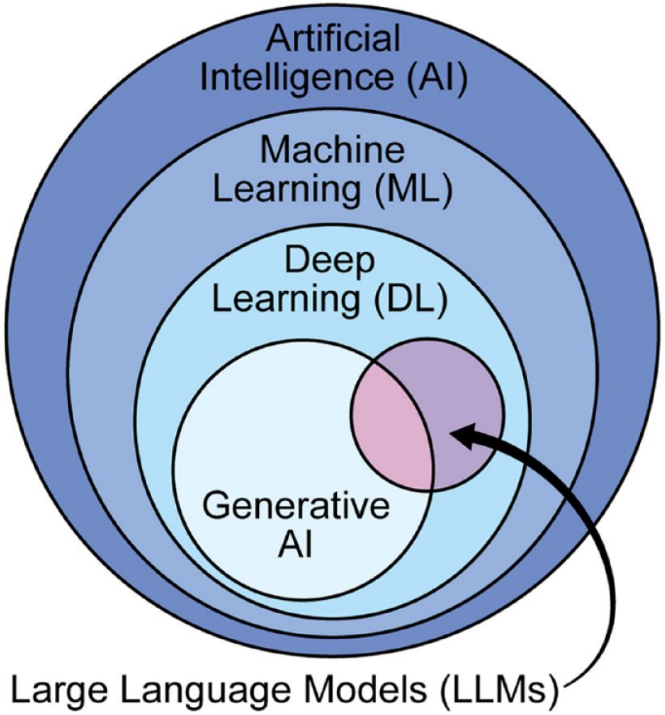
**ChatGPT**

**Vector Database**

**RAG**

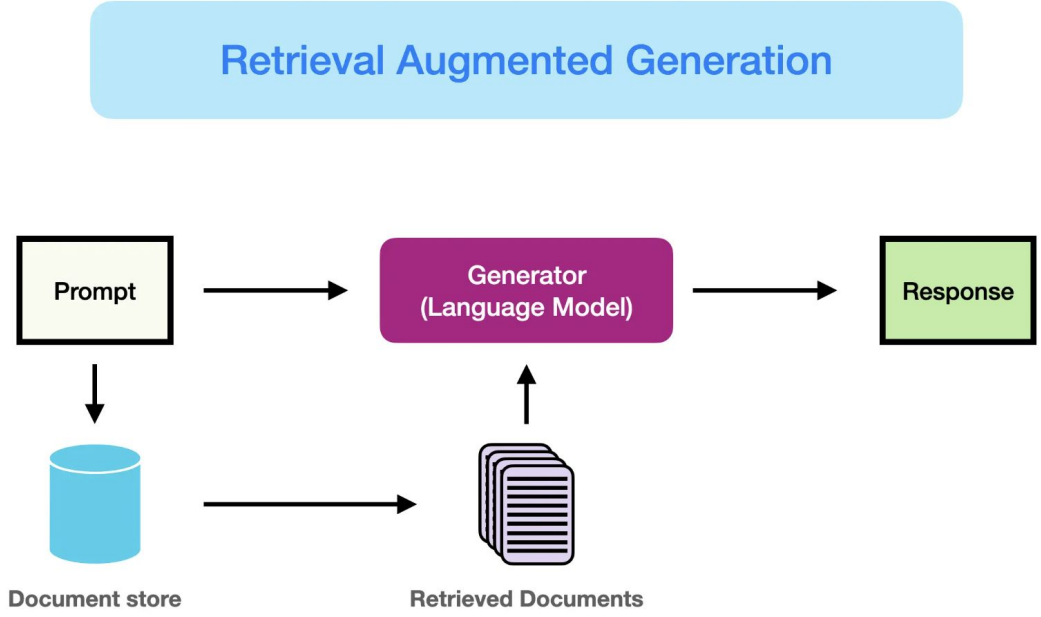
**Embeddings**

**AI Agents**



# Retrieval Augmented Generation (RAG)

- The go to database for implementing RAG is a vector database or vector store
- RAG you could pair with any database



# Retrieval Augmented Generation/ Vector Search

- When implementing RAG with vectors, whatever LLM you choose to create the embeddings is also the LLM you use to create the embedding of your questions

- When you are asking a question about your data given to the LLM it gives you this embedding, and this embedding more or less is compared with all the data

# Similarity Search

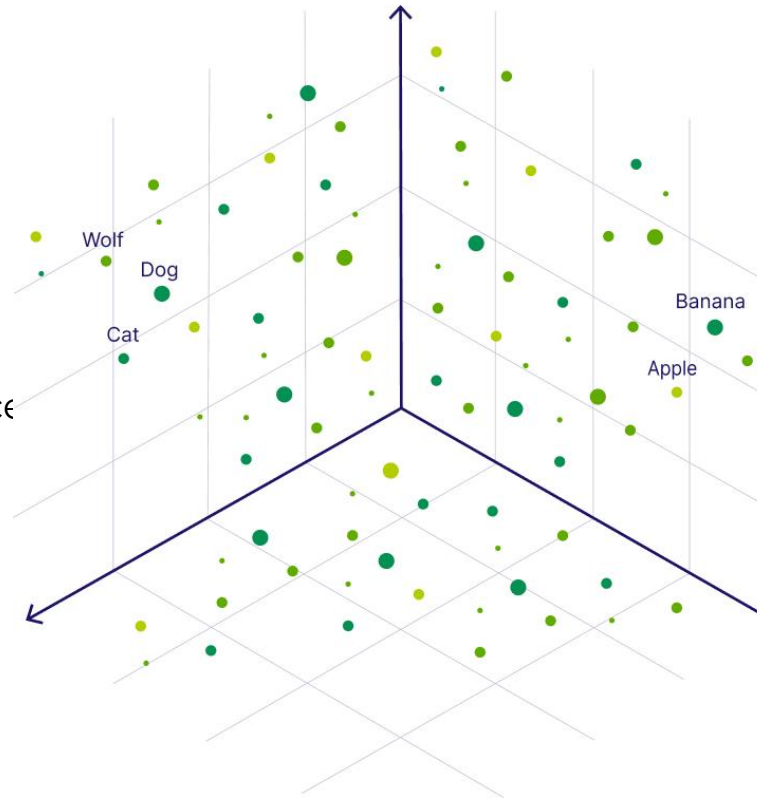
Words like **“Wolf”** and **“Dog”** should be close in meaning

With vectors, you can store them as points in space

and measure distance (cosine similarity, Euclidean distance

This allows **semantic search** (finding by meaning,

not just exact keywords)



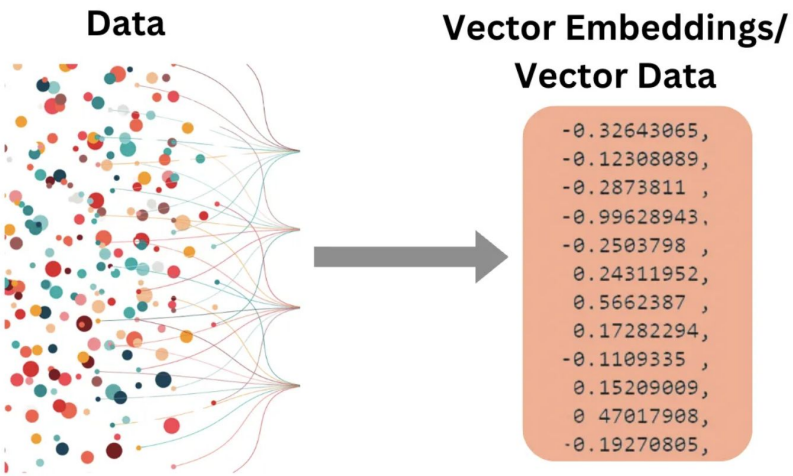
# Embeddings (They are vectors!)

- A way to represent complex things

(like words or pictures) as numbers

that computers can easily compare

and calculate with



# Vector Databases

- An LLM like **ChatGPT** creates a vector embedding of the data which is assigning a numerical value to each one of the points of data

Why did vectors come into play?

- They became popular because **traditional data formats weren't enough** for modern AI tasks like search, recommendations, and chatbots.

# LLM Pipeline with RAG

Here’s where RAG fits in:

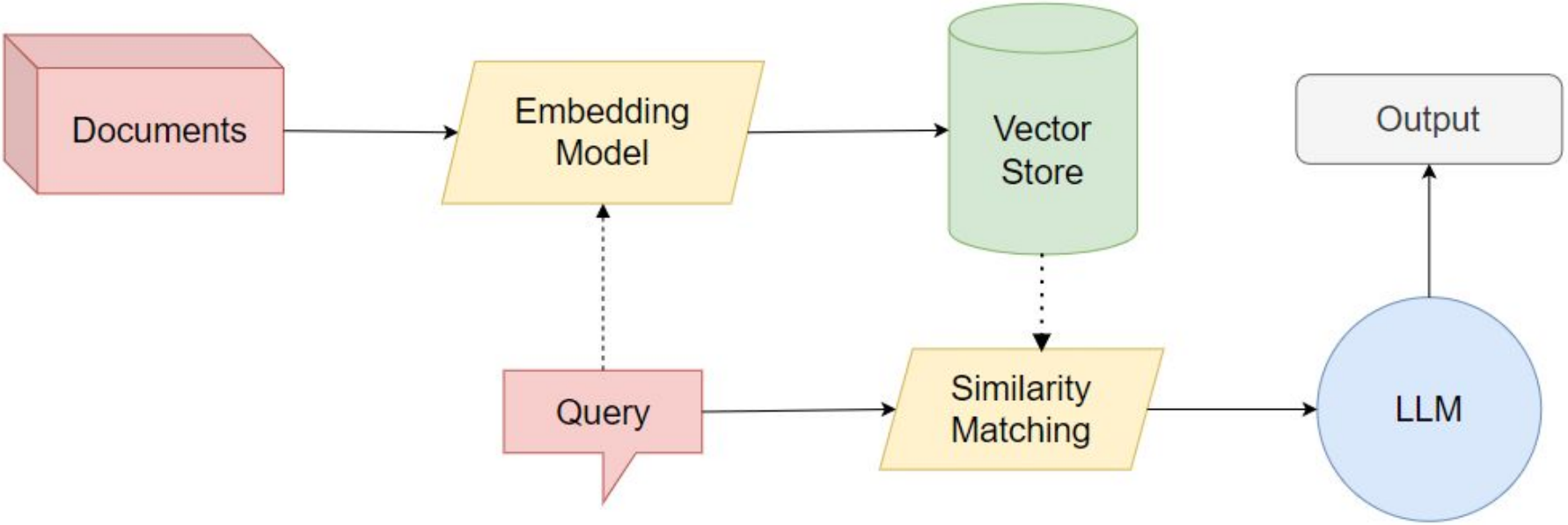
1. **User Query (Prompt)**
   "What’s the latest feature in Postgres 16?"

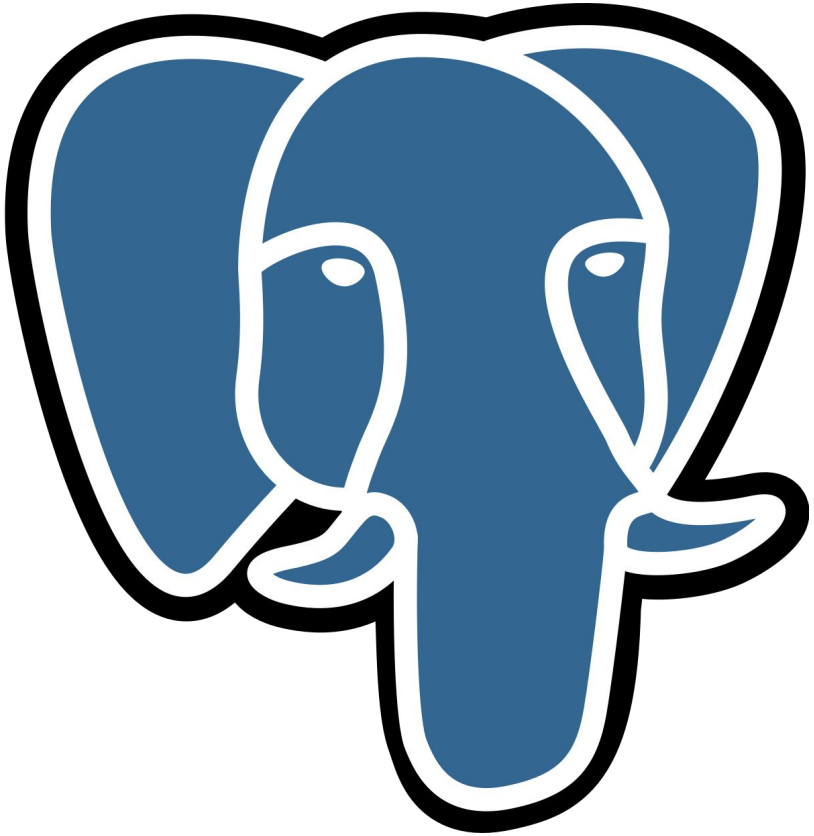
2. **Embed the Query**
   Convert it into a vector representation.

3. **Retrieve Relevant Data**
   Use the embedding to **search a vector database** (like pgvector, Pinecone, Weaviate, etc.) for semantically relevant documents/snippets.

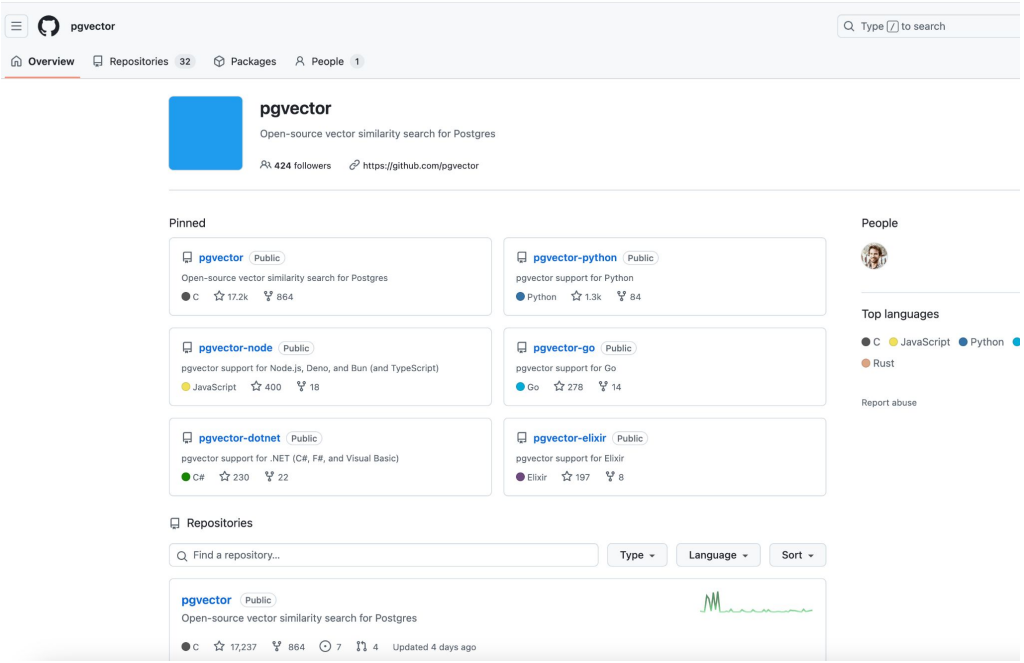
4. **Augment the Prompt**
   Combine the user query **plus the retrieved documents** into a new enriched prompt.

# LLM Pipeline with RAG





# Pgvector




Pgvector makes PostgreSQL function as a vector database

# RAG with PostgreSQL

**RAG** (Retrieval-Augmented Generation) = **LLM + external knowledge base**.

- The **LLM generates text**
- but before answering, it retrieves **relevant context** from your data (docs, FAQs, code, etc.)
- Retrieval is usually based on **vector similarity search** (finding text chunks that are semantically closest to the query)

## Why **Pgvector?**

- It enables RAG to work well at scale

# Where does Pgvector come in?

Without something like pgvector, your RAG system is not efficient to ***store and search embeddings***

**RAG alone** is an idea / method

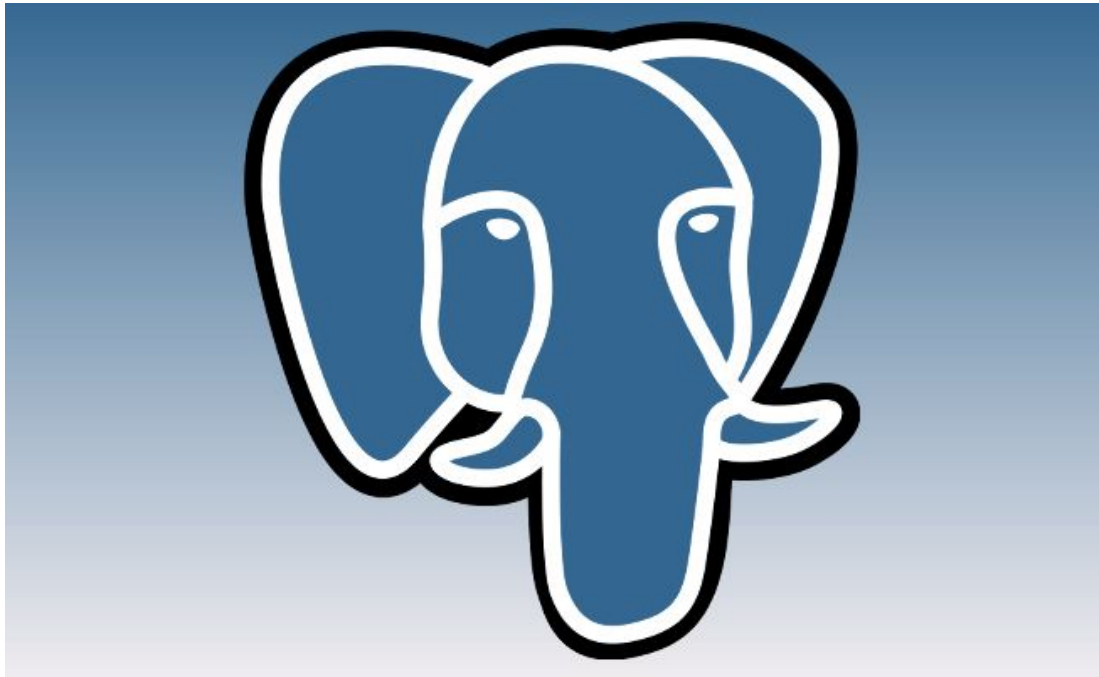
**RAG + pgvector** is an actual working system, because you can:

- Store embeddings (vector representations of your documents)
- Perform similarity search directly in Postgres
- Keep structured data + unstructured embeddings in one place

# Why Pgvector?

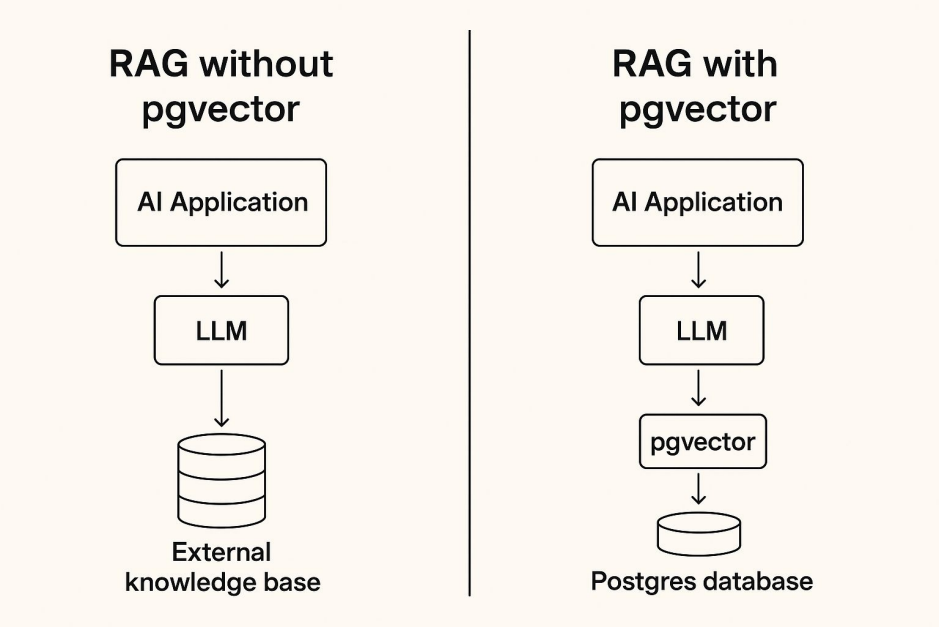
- Pgvector lets you **implement RAG directly inside Postgres**,

no need for extra infrastructure



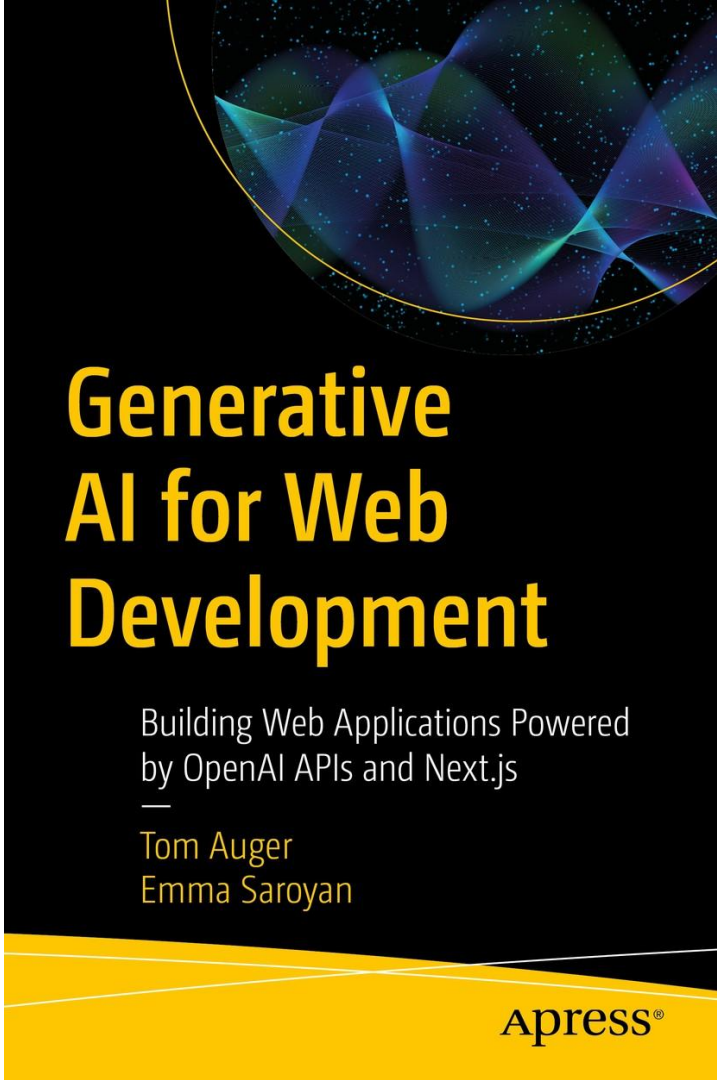
# When to use RAG with Pgvector?

- If your app needs to **search across a large collection** of documents
- If you want **fast and relevant retrieval** (semantic, not just keyword)
- If you already use Postgres and prefer not to introduce a new database



# Check out my latest book

- Published in December 2024



# Let’s connect!