tembo

What’s our Vector, Victor?

Taking the pain out of AI with pg_vectorize

Postgres Conf
Seattle 2024

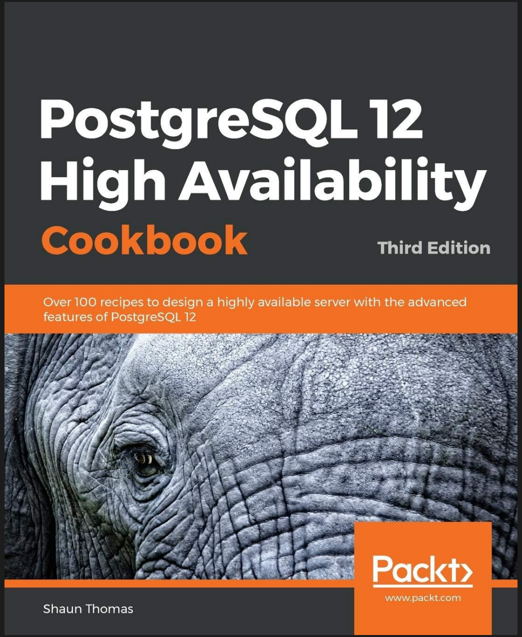
Who are Tembo?

- Cloud Services - cloud.tembo.io
- Trunk (Extensions) - pgt.dev
- **pg_vectorize**
- pgmq
- pg_tier
- pg_timeseries
- And more!

Who am I?

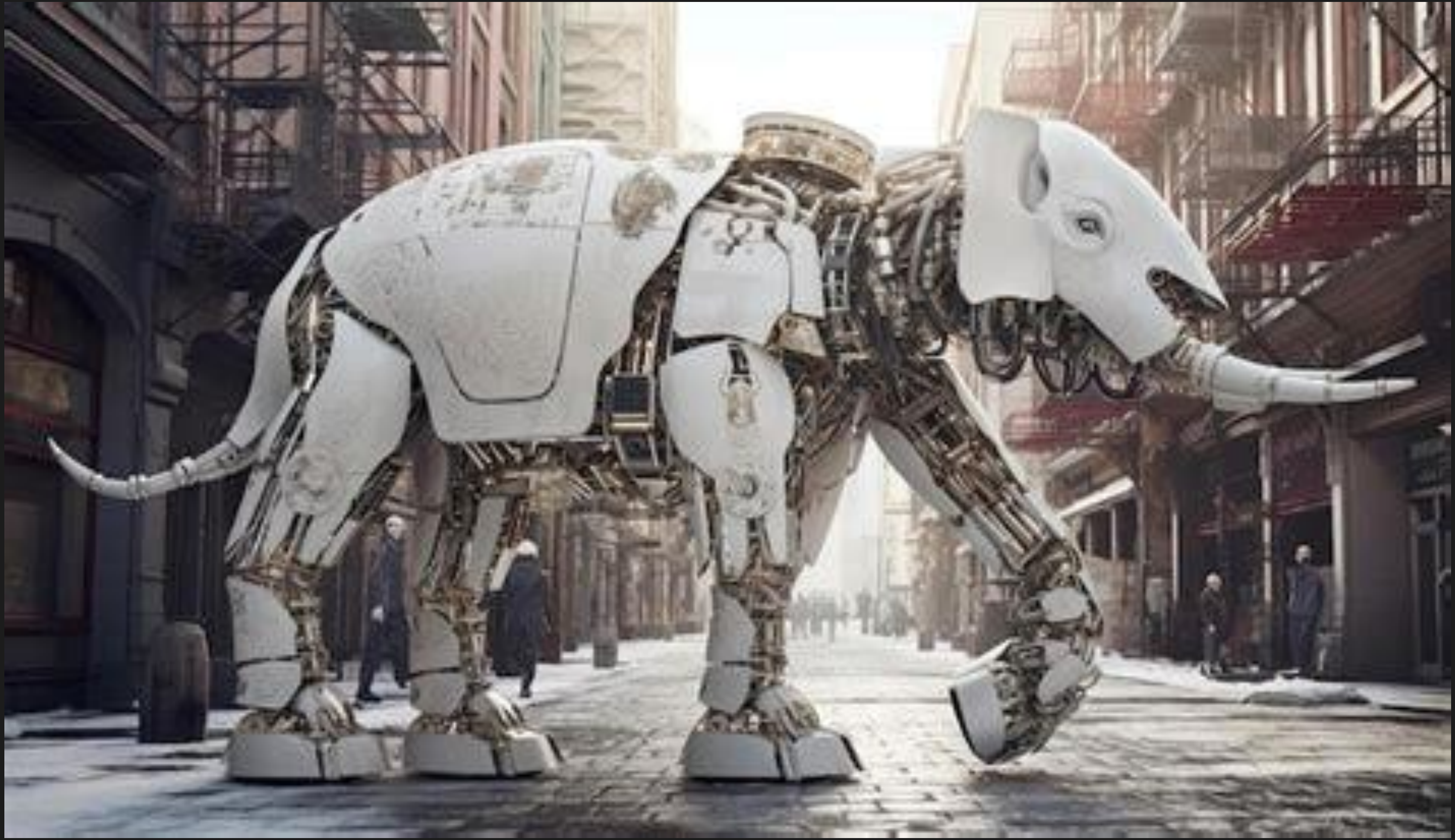
- Author
- Speaker
- Blogger
- Mentor
- Dev

shaun@tembo.io



What is AI?

Not This!



What it Really is

Large Language Model

Retrieval Augmented Generation

Word Chunk

Embedding

Big-Ass Array

Embedding

~~Big Ass Array~~

Vector to Token Coordinates

Translates Text into Embeddings

Coordinates Where?

What Does pgvector Do?

Transform Individual Phrases

```sql
SELECT vectorize.encode(
    input       => 'Is Postgres the best database engine?',
    model_name  => 'sentence-transformers/all-MiniLM-L12-v2'
);
```

- Easily transform a prompt or search terms into compatible vectors
- Output is compatible with pgvector vector search

Create and Maintain Embeddings

```sql
SELECT vectorize.table(
    job_name      => 'rt_article_embed',
    "table"       => 'blog_article',
    primary_key   => 'article_id',
    update_col    => 'last_updated',
    columns       => ARRAY['author', 'title', 'content'],
    transformer   => 'sentence-transformers/all-MiniLM-L12-v2',
    schedule      => 'realtime'
);
```

Embeddings are maintained by pg_cron job, or **pgmq live updates**

Writes can spawn embeddings via queue

Natural Language Search

```sql
SELECT * FROM vectorize.search(
    job_name       => 'rt_article_embed',
    query          => 'Is Postgres the best database engine?',
    return_columns => ARRAY['author', 'title', 'content'],
    num_results    => 5
);
```

Consider this like Full Text Search, but better

Bootstrap a RAG Stack

```sql
SELECT vectorize.init_rag(
    agent_name          => 'blog_chat',
    table_name          => 'blog_article',
    "column"            => 'article',
    unique_record_id    => 'article_id',
    transformer         => 'sentence-transformers/all-MiniLM-L12-v2',
    schedule            => 'realtime'
);
```

Realtime embeddings are queued to avoid write latency

Perform a RAG Request

```sql
SELECT vectorize.rag(
    agent_name  => 'blog_chat',
    query       => 'Is Postgres the best database?',
    chat_model  => 'ollama/llama3.1'
) -> 'chat_response';
```

The result is a JSON object that includes context if we need it

tembo

Works with OpenAI

Just supply your OpenAI token:

```
ALTER SYSTEM SET vectorize.openai_key TO '<your api key>';
```

Or Roll Your Own

Search using Ollama or vLLM instead:

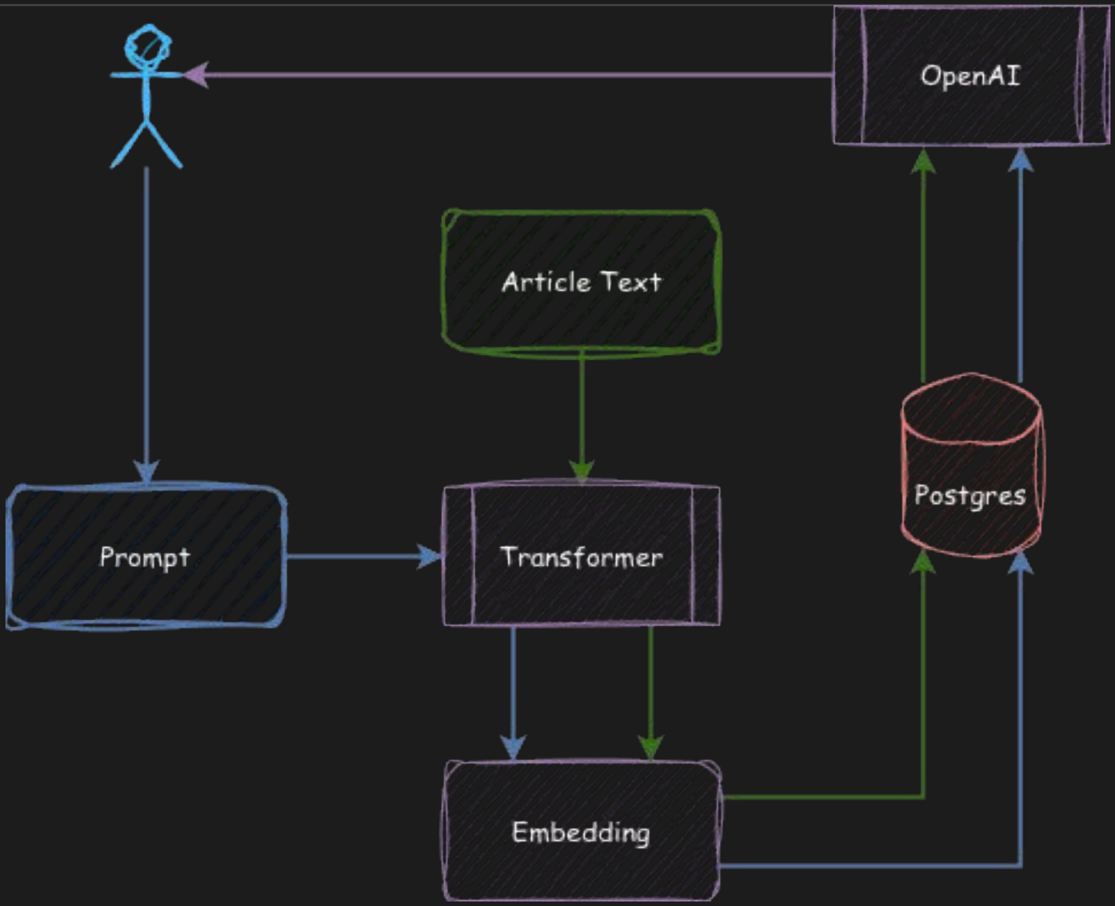
```
ALTER SYSTEM SET vectorize.openai_service_url
   TO 'https://api.myserver.com/v1';
```

Use a custom transformer service:

```
ALTER SYSTEM SET vectorize.embedding_service_url
   TO 'https://api.myserver.com/v1';
```

Let’s Make a RAG App

Anatomy of a RAG App



tembo

How it Works

Data Side

1. Gather content
2. Pass through transformer
3. Store vector in database

User Side

1. Asks a question
2. Pass through transformer
3. Match against stored vectors
4. Question + Results sent to AI
5. Send answer to user

The Full Monty

To build a RAG app, we need to:

1. Parse and load the content and metadata into Postgres
2. Generate the embeddings
3. Transform user input into an embedding
4. Match results from user search vector
5. Build new prompt from results and user search
6. Send full instructions to OpenAI
7. Return results to user

From the Perspective of pg_vectorize

Or if we're using pg_vectorize:

1. Parse and load the content and metadata into Postgres
2. Call `vectorize.init_rag( … )`
3. Call `vectorize.rag( … )`

Which would you rather do?

A Place for Blogs

```sql
CREATE TABLE blog_articles (
     article_id    BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
     author        TEXT,
     title         TEXT,
     content       TEXT,
     publish_date  DATE,
     last_updated  TIMESTAMPTZ NOT NULL DEFAULT now()
);
```

Chunky Style

```sql
CREATE TABLE blog_article_chunks (
    chunk_id       BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    article_id     BIGINT NOT NULL REFERENCES blog_articles,
    chunk          TEXT,
    last_updated   TIMESTAMPTZ NOT NULL DEFAULT now()
);
```

- Embeddings are usually “fuzzy” (only 384 coordinates)
- We need chunks for sharper context

More than Meets the Eye

```sql
SELECT vectorize.init_rag(
    agent_name         => 'blog_chat',
    table_name         => 'blog_article_chunks',
    "column"           => 'chunk',
    unique_record_id   => 'chunk_id',
    transformer        => 'sentence-transformers/all-MiniLM-L12-v2',
    schedule           => 'realtime'
);
```

Look familiar? Now we're indexing chunks rather than full articles.

Slice and Dice

Here’s a closer look at a chunk splitter in Python:

```python
from langchain_text_splitters import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(
    separators = ["\n\n", "\n", ' ', '.', '```'],
    chunk_size = 500,
    chunk_overlap = 20,
    length_function = len,
    is_separator_regex = False
)

def chunk_content(content):
    return splitter.split_text(content)
```

A Pleasing Result

Now we can finally figure out which database is best:

```
SELECT vectorize.rag(
    agent_name => 'blog_chat',
    query      => 'Is Postgres the best database engine?',
    chat_model => 'ollama/llama3.1'
) -> 'chat_response';
```

```
"Four times since 2017, it has won the DB-Engines \"DBMS of the Year\"
award."
```


tembo

Conclusion!

If you can write queries
You can build AI apps with Postgres

Thanks!

shaun@tembo.io
@BonesMoses
/in/bonesmoses

tembo.io

Want to experiment?
Use the Tembo free trial!

- Two weeks to test
- $300 USD credit
- Reverts to Hobby tier instance after trial ends

learn more at

tembo.io

Easily deploy one of our Postgres stacks

- **AI / RAG**
- Geospatial
- Analytics
- Timeseries

tembo

Other ways to extend / focus slides

-