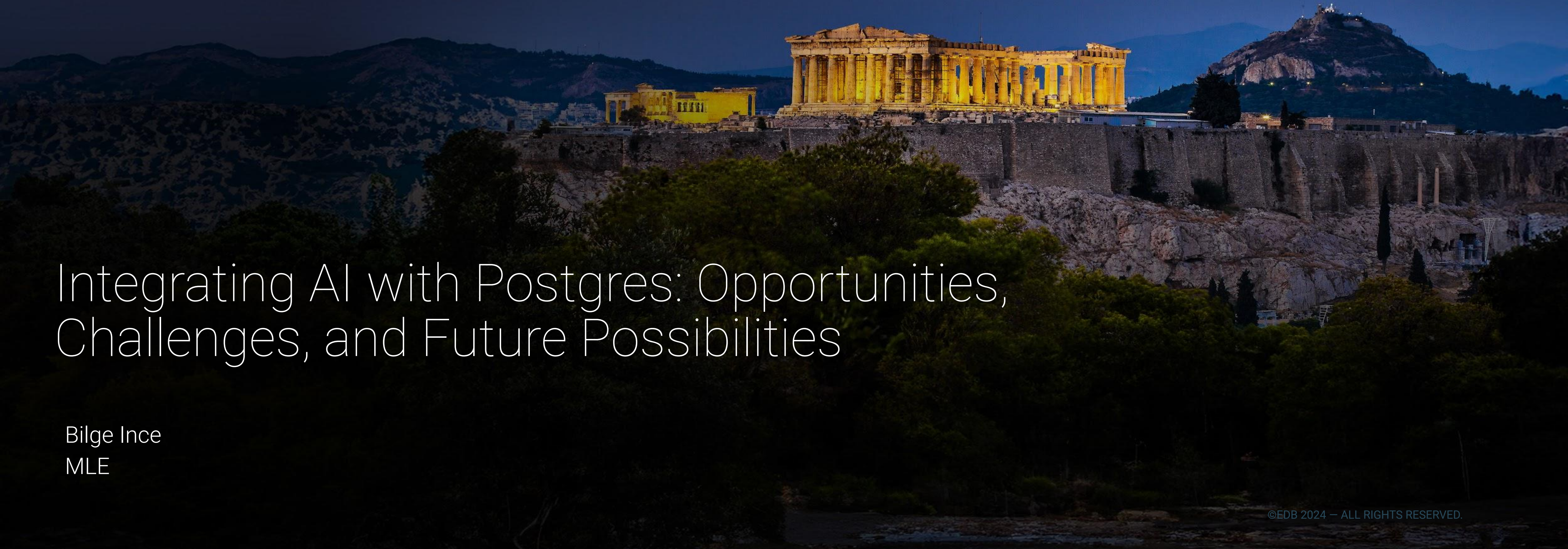
Ευρωπαϊκό Συνέδριο PostgreSQL
PostgreSQL Conference Europe
2024

EDB
Postgres® for the AI Generation



Integrating AI with Postgres: Opportunities, Challenges, and Future Possibilities

Bilge Ince
MLE

Bilge INCE

MLE @ EDB

Organizer of Diva: Dive Into AI

🎧 Kırılma Noktası

Muay Thai, Running ❤️

@abilgeince

LLMs

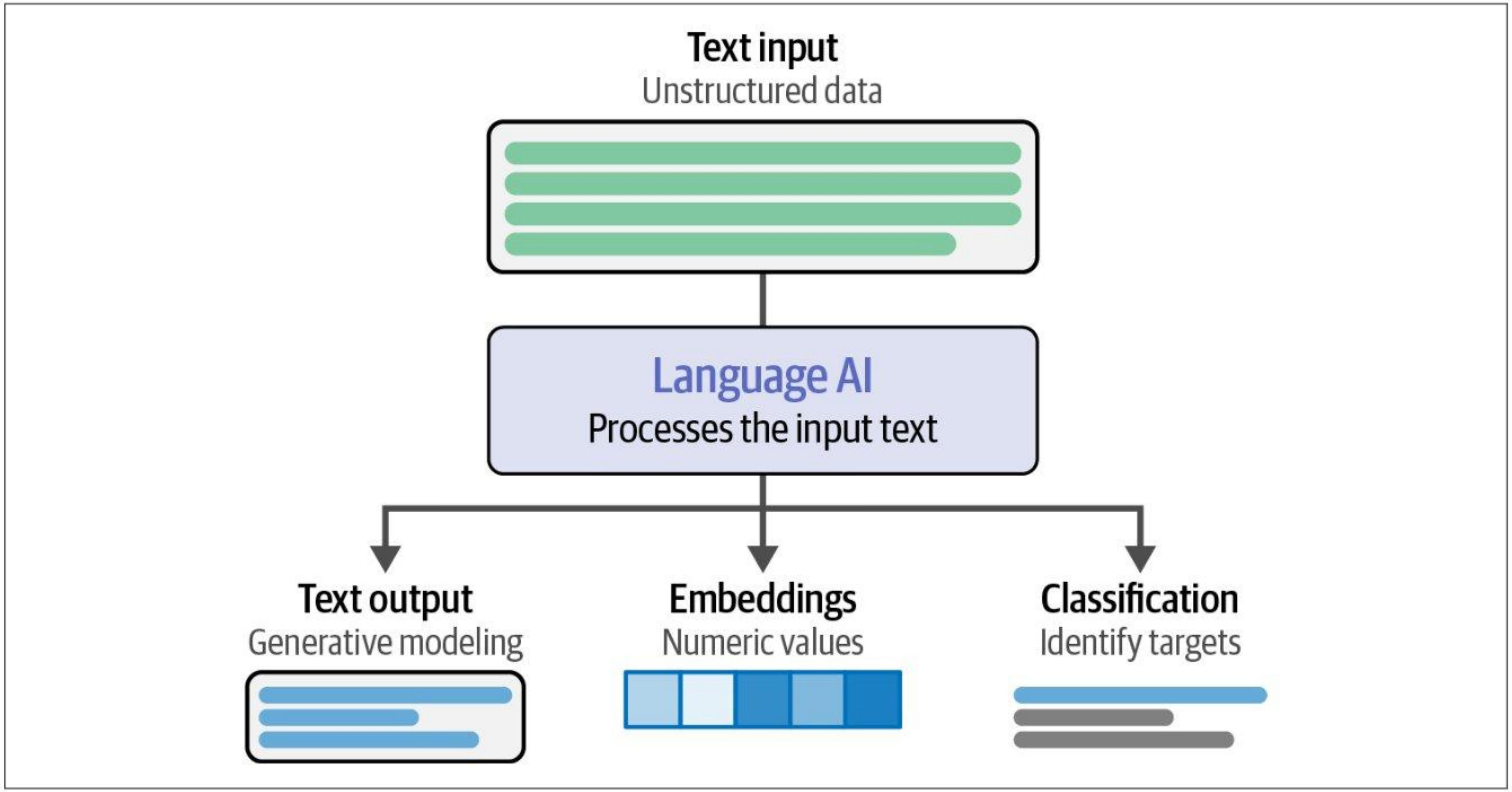


Fig: Hands on LLMs - Jay Alammar & Maarten Grootendorst

LLMs

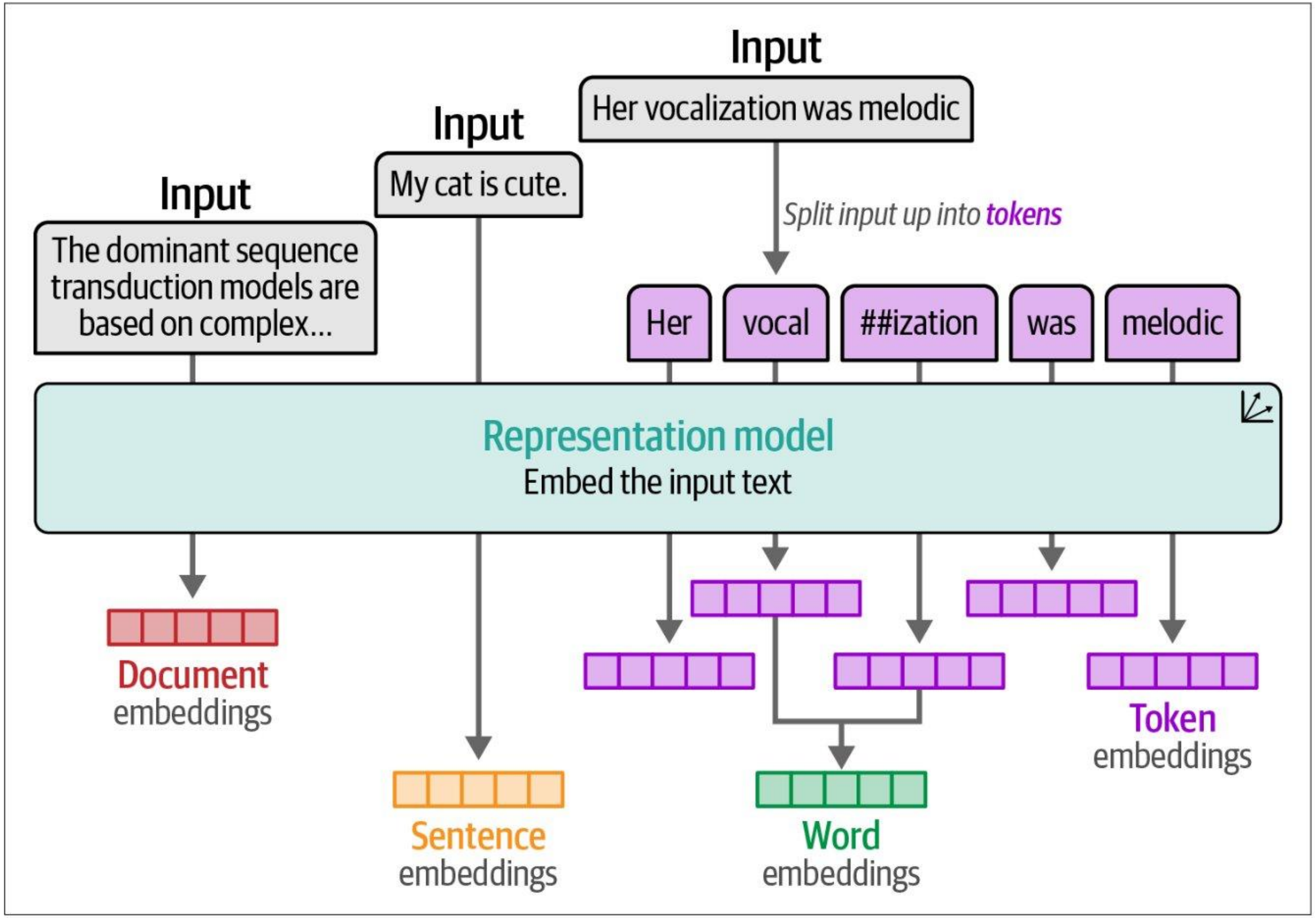


Fig: Hands on LLMs - Jay Alammar & Maarten Grootendorst

Traditional ML vs LLMs

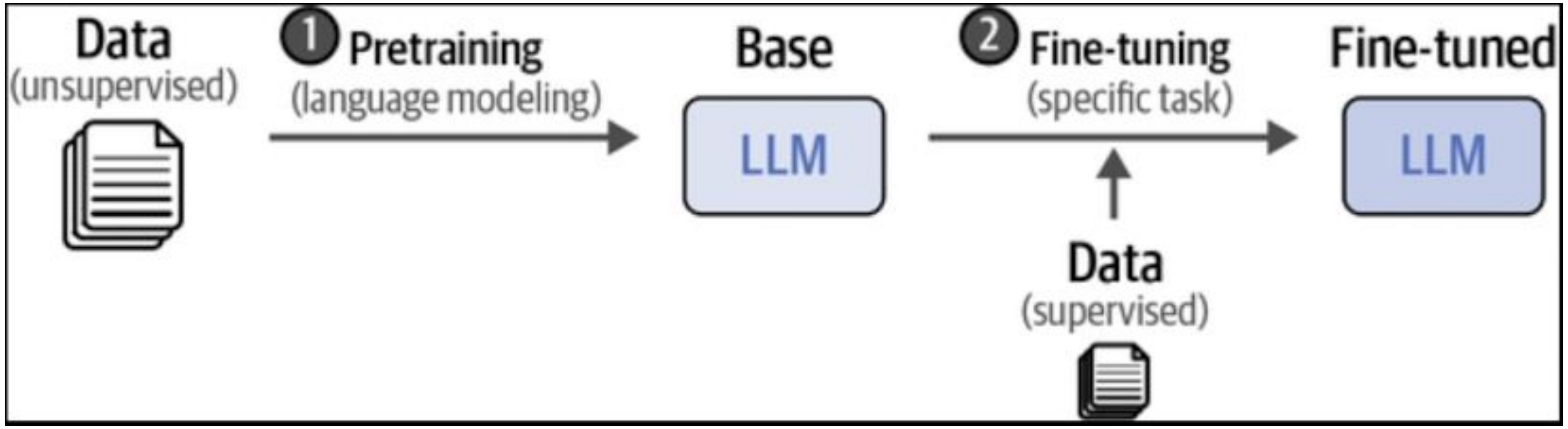
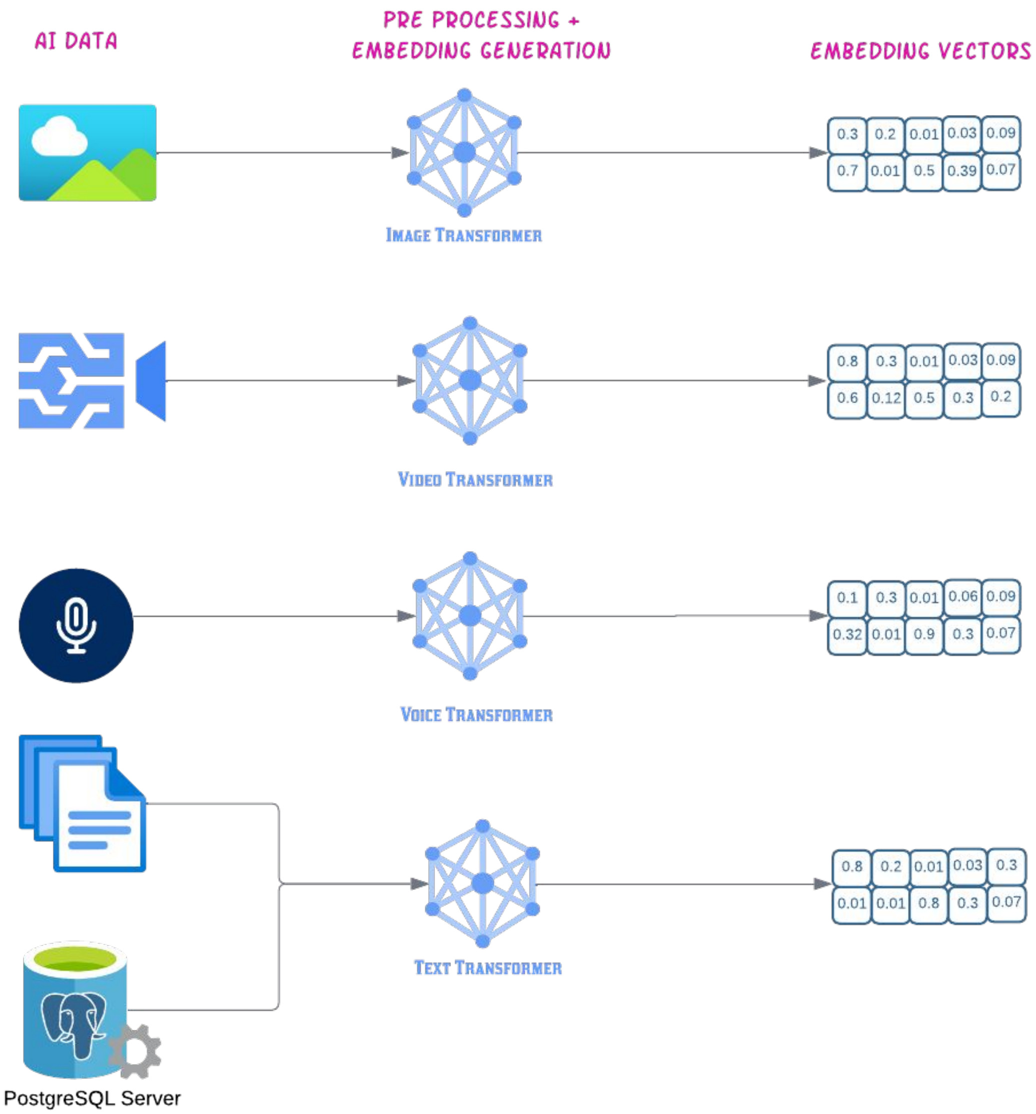


Fig: Hands on LLMs - Jay Alammar & Maarten Grootendorst

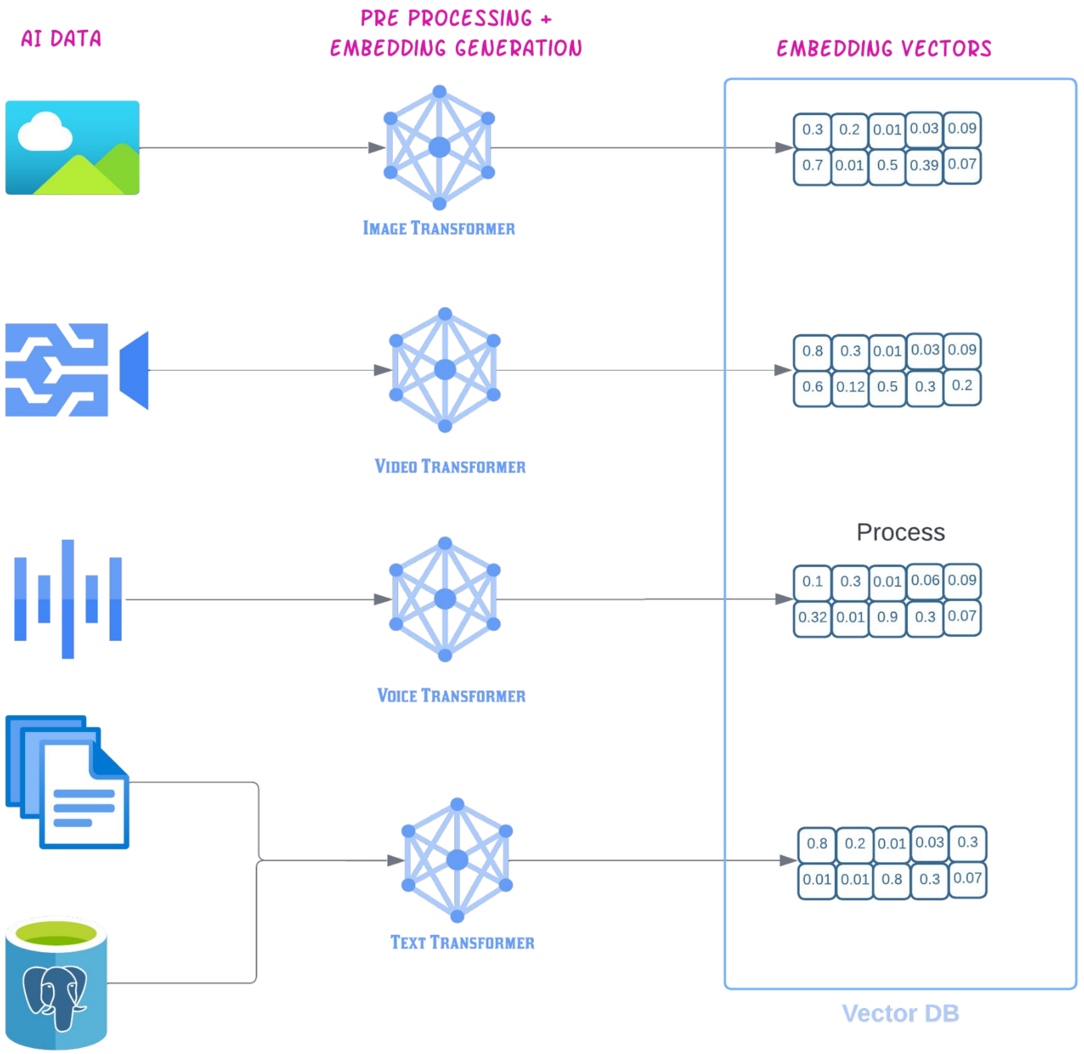
What is PostgreSQL for an AI Engineer?

What is PostgreSQL after pg vector extension for an AI Scientist?

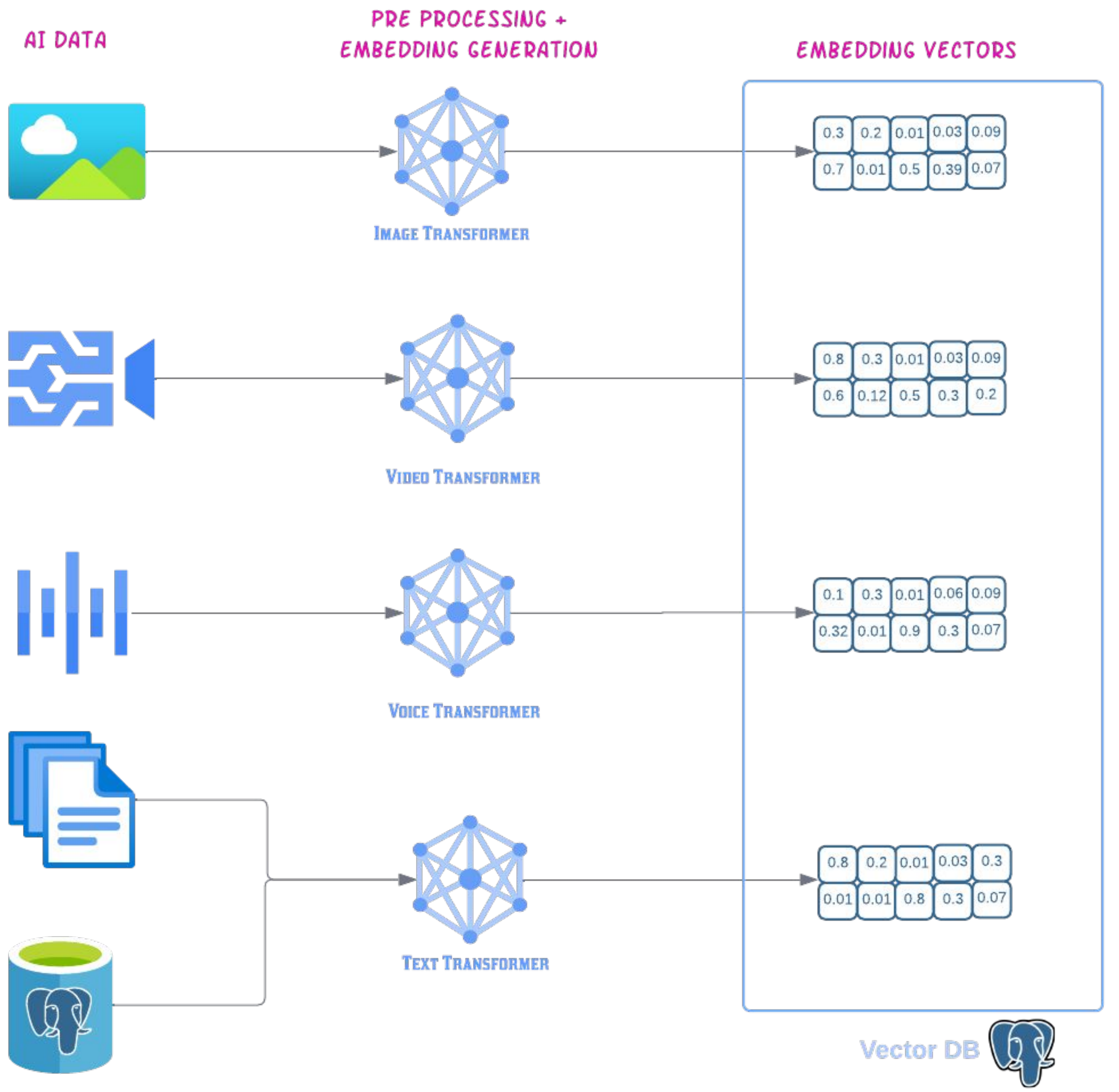
How data & vectors are connected to each other?



How data & vectors are connected to each other?



How data & vectors are connected to each other?



Vector Arithmetic

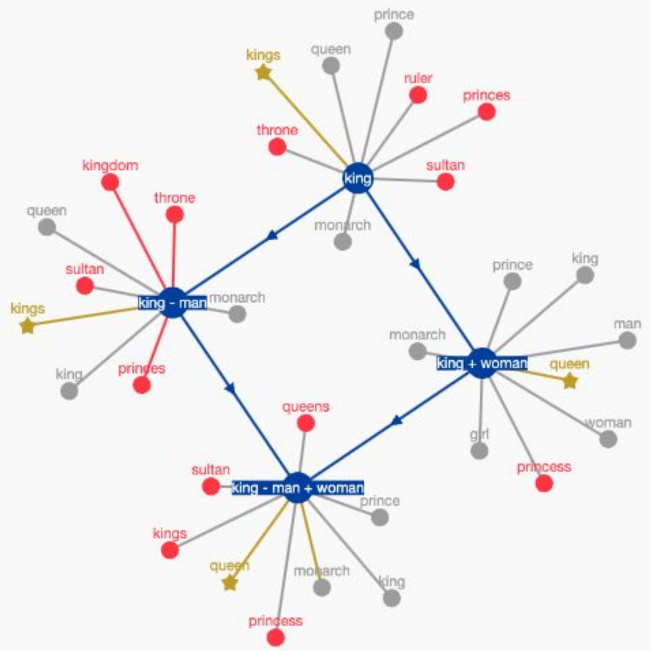
Preset: king - man + woman

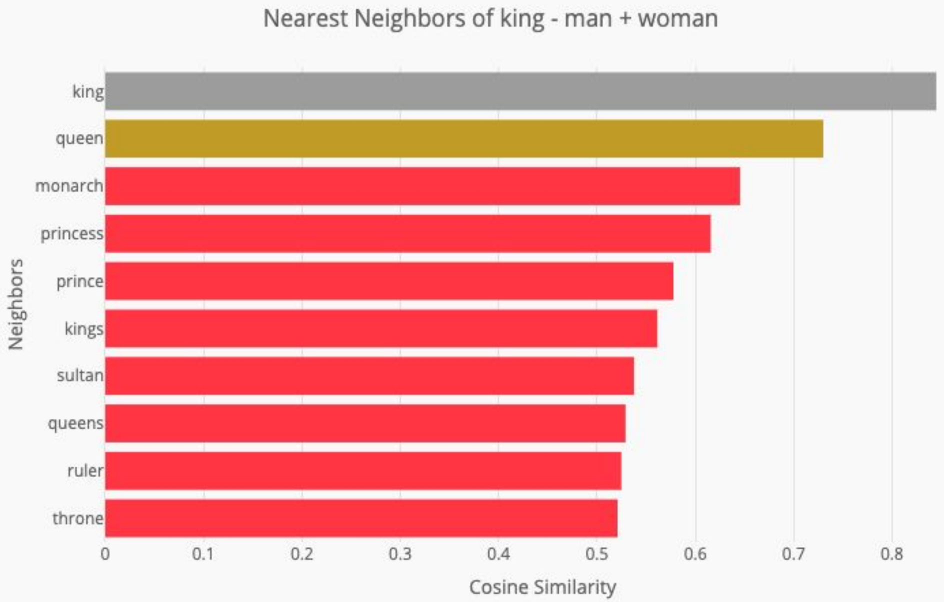
Start Word: king

Subtract this word: man

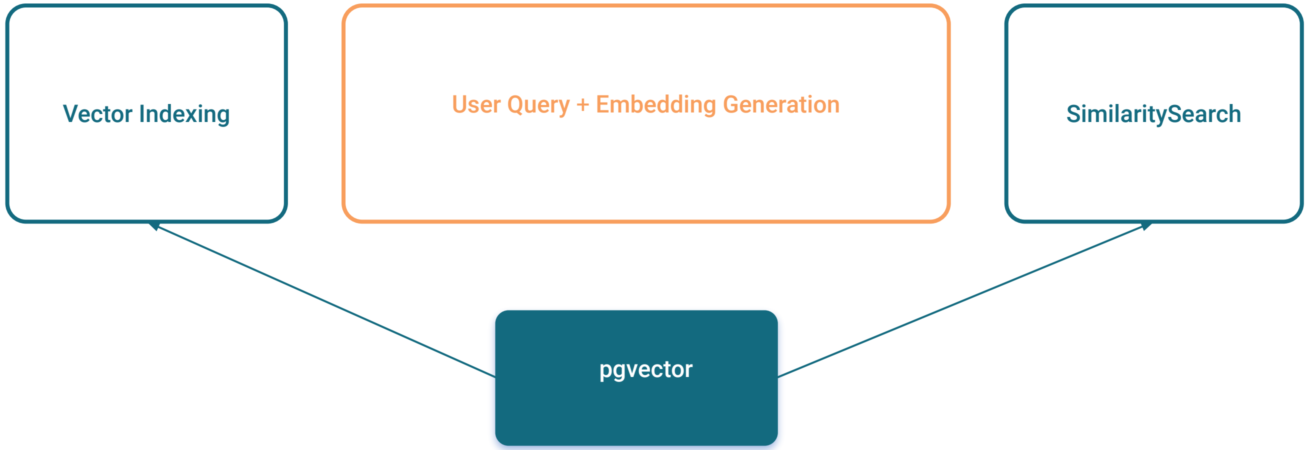
Add this word: woman

RUN





PgVector


Vector Indexing
User Query + Embedding Generation
SimilaritySearch
pgvector

Similarity Search

distance - Euclidean sq

$$(0.3 - 0.3)^2 + (0.5 - 0.2)^2 + (0.01 - 0.01)^2 + (0.08 - 0.03)^2 + (0.09 - 0.09)^2$$

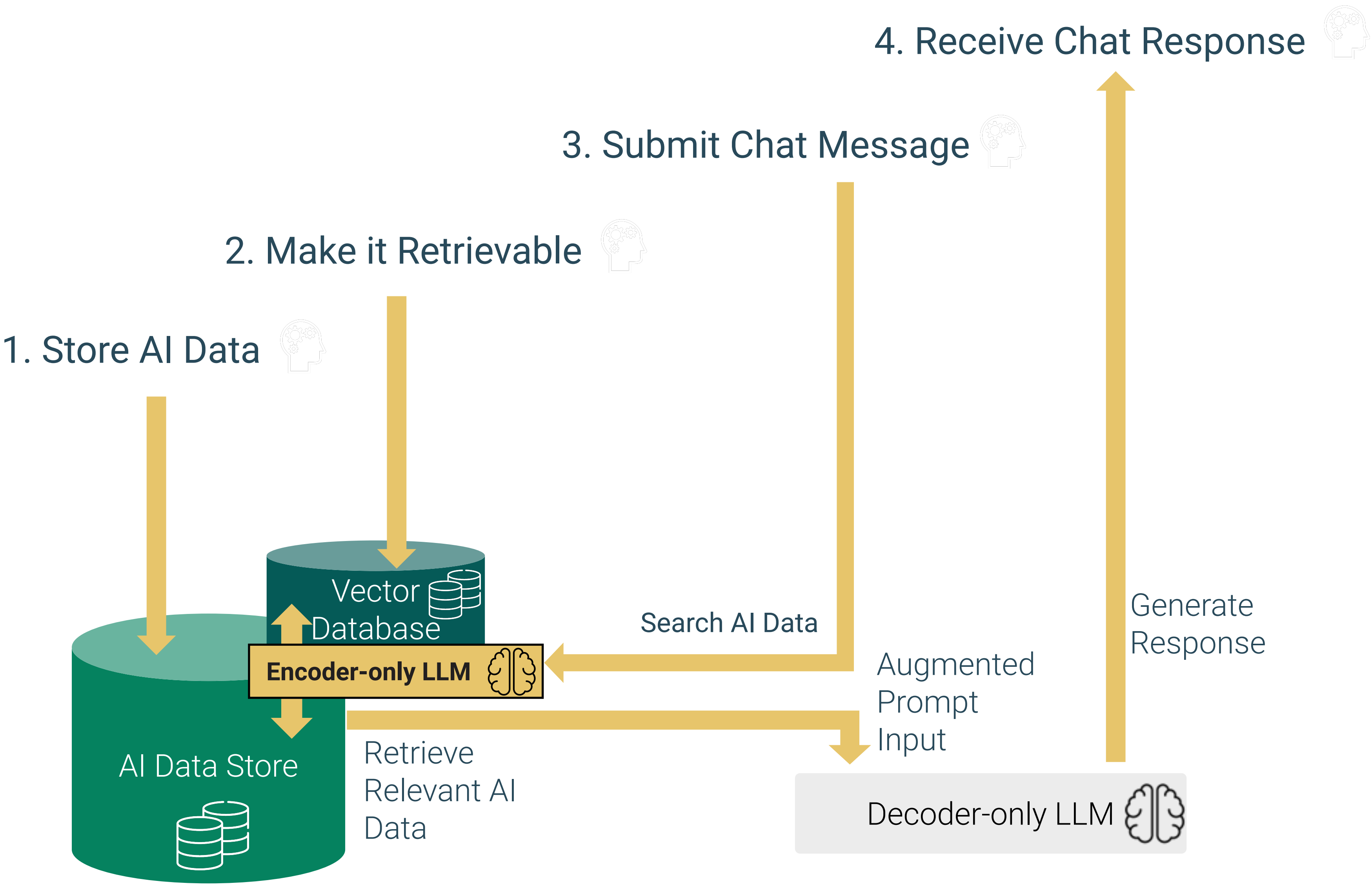
Postgres is perfectly positioned as THE AI database

- Absolute **battle proof** Enterprise QoS
  - In **community** distro but also **very vital** commercial Enterprise option ecosystem
- Perfect **extensibility** & customization
  - With **AI relevant** languages & ecosystems: **Python, Rust**
  - **Custom Data** Types
  - Index & Table **Access Methods**
- Already houses the most valuable enterprise **business data**
  - in **fully relational** manner

Always On ✓	Resilient ✓	Secure ✓
Responsive ✓	Scalable ✓	Enterprise Support ✓
Ecosystem ✓	Business Data ✓	Business Events ✓

What about aidb?

Chat Bots – The John Doe of Gen AI Applications



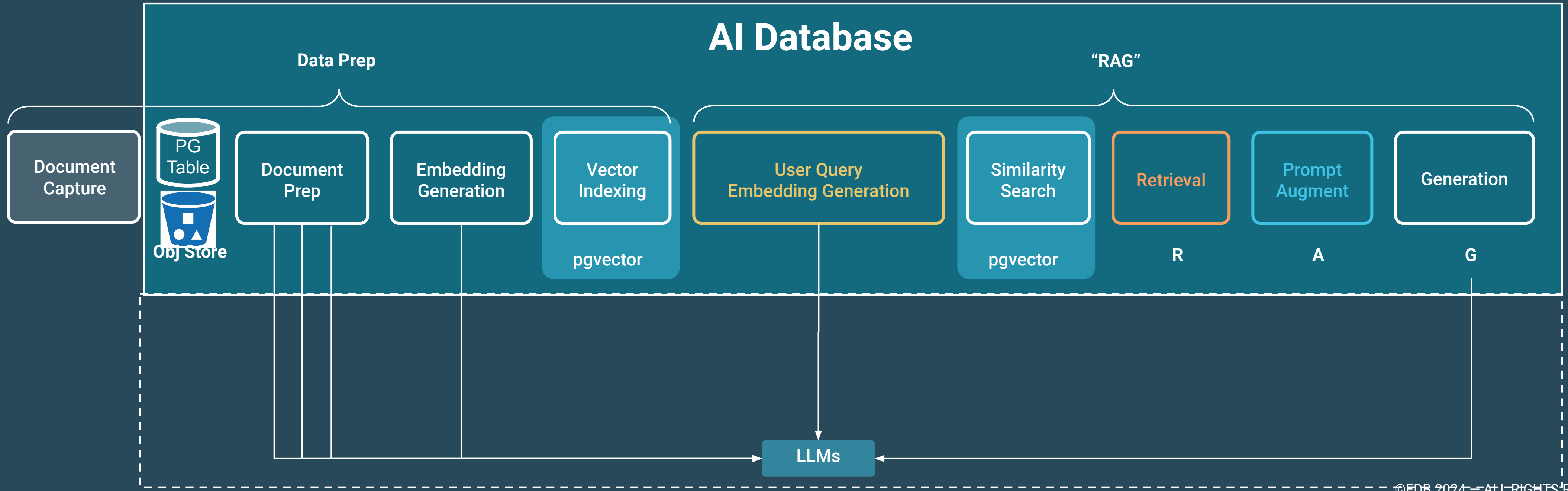
A.k.a.:
Retrieval
Augmented
Generation
= **RAG**

Building GenAI applications with EDB Postgres AI

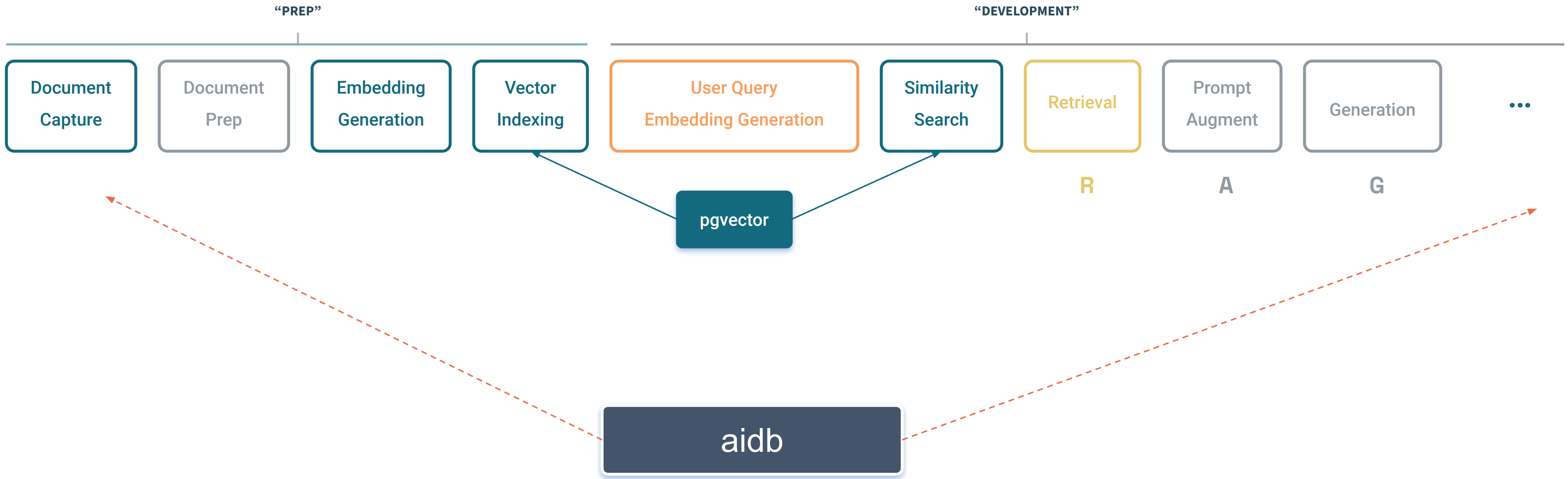
BEYOND VECTOR SUPPORT

1 **Postgres as GenAI Retriever & Generator:** Automating document (and other modalities) prep, embedding generation & vector indexing, providing a simple semantic retriever interface, and even chat completion in database

2 **Enabling Sovereign AI for enterprises:** Runs with either, embedded LLMs (in PG memory), external model provider of your choice, or EDB Postgres AI platform hosted models.



aidb



"PREP"
"DEVELOPMENT"
Document Capture
Document Prep
Embedding Generation
Vector Indexing
User Query Embedding Generation
Similarity Search
Retrieval
Prompt Augment
Generation
...
R
A
G
pgvector
aidb

aidb - embeddings

```
postgres=# select * from aidb.encoders;
 id |                  name                   |  provider   | max_tokens | default_distance_metric | dimensions
----+-----------------------------------------+-------------+------------+-------------------------+------------
  1 | text-embedding-ada-002                  | openai      |       8191 | cosine                  |       1536
  2 | text-embedding-3-small                  | openai      |       8191 | cosine                  |       1536
  3 | text-embedding-3-large                  | openai      |       8191 | cosine                  |       2000
  4 | clip-vit-base-patch32                   | openai      |        512 | cosine                  |        512
  5 | gtr-t5-xxl                              | huggingface |        512 | dot                     |        768
  6 | gtr-t5-xl                               | huggingface |        512 | dot                     |        768
  7 | sentence-t5-xxl                         | huggingface |        256 | dot                     |        768
  8 | gtr-t5-large                            | huggingface |        512 | dot                     |        768
  9 | all-mpnet-base-v1                       | huggingface |        512 | dot                     |        768
 10 | multi-qa-mpnet-base-cos-v1              | huggingface |        512 | dot                     |        768
 11 | all-roberta-large-v1                    | huggingface |        256 | dot                     |       1024
 12 | sentence-t5-xl                          | huggingface |        256 | dot                     |        768
 13 | all-MiniLM-L12-v1                       | huggingface |        256 | dot                     |        384
 14 | gtr-t5-base                             | huggingface |        512 | dot                     |        768
 15 | sentence-t5-large                       | huggingface |        256 | dot                     |        768
 16 | all-MiniLM-L6-v1                        | huggingface |        256 | dot                     |        384
 17 | msmarco-bert-base-dot-v5                | huggingface |        512 | dot                     |        768
 18 | multi-qa-MiniLM-L6-dot-v1               | huggingface |        512 | dot                     |        384
 19 | sentence-t5-base                        | huggingface |        256 | dot                     |        768
 20 | msmarco-distilbert-base-tas-b           | huggingface |        512 | dot                     |        768
 21 | msmarco-distilbert-dot-v5               | huggingface |        512 | dot                     |        768
 22 | multi-qa-mpnet-base-dot-v1              | huggingface |        512 | dot                     |        384
 23 | multi-qa-distilbert-dot-v1              | huggingface |        512 | dot                     |        768
 24 | paraphrase-MiniLM-L6-v2                 | huggingface |        128 | cosine                  |        384
 25 | paraphrase-TinyBERT-L6-v2               | huggingface |        128 | cosine                  |        768
 26 | paraphrase-MiniLM-L12-v2                | huggingface |        256 | cosine                  |        384
 27 | paraphrase-distilroberta-base-v2        | huggingface |        256 | cosine                  |        768
 28 | paraphrase-mpnet-base-v2                | huggingface |        512 | cosine                  |        768
 29 | all-mpnet-base-v2                       | huggingface |        384 | cosine                  |        768
 30 | all-distilroberta-v1                    | huggingface |        512 | cosine                  |        768
 31 | all-MiniLM-L12-v2                       | huggingface |        256 | cosine                  |        384
 32 | multi-qa-distilbert-cos-v1              | huggingface |        512 | cosine                  |        768
 33 | all-MiniLM-L6-v2                        | huggingface |        256 | cosine                  |        384
 34 | multi-qa-MiniLM-L6-cos-v1               | huggingface |        512 | cosine                  |        384
 35 | paraphrase-multilingual-mpnet-base-v2   | huggingface |        128 | cosine                  |        768
 36 | paraphrase-albert-small-v2              | huggingface |        256 | cosine                  |        768
 37 | paraphrase-multilingual-MiniLM-L12-v2   | huggingface |        128 | cosine                  |        384
 38 | paraphrase-MiniLM-L3-v2                 | huggingface |        128 | cosine                  |        384
 39 | distiluse-base-multilingual-cased-v1    | huggingface |        128 | cosine                  |        512
 40 | distiluse-base-multilingual-cased-v2    | huggingface |        128 | cosine                  |        512
(40 rows)
```

```sql
SELECT provider, count(*) encoder_model_count FROM aidb.encoders gro
```

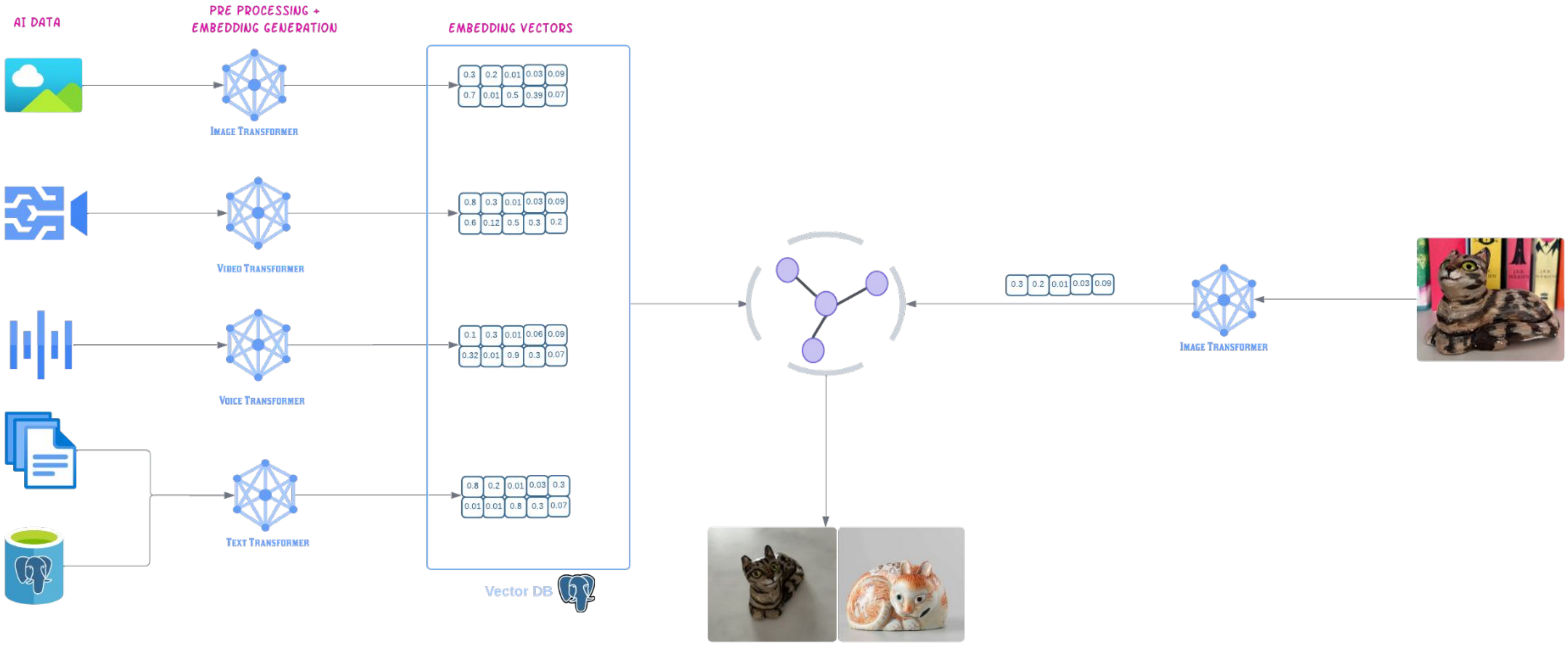
OUTPUT

```
  provider   | encoder_model_count
-------------+---------------------
 huggingface |                  36
 openai      |                   4
(2 rows)
```

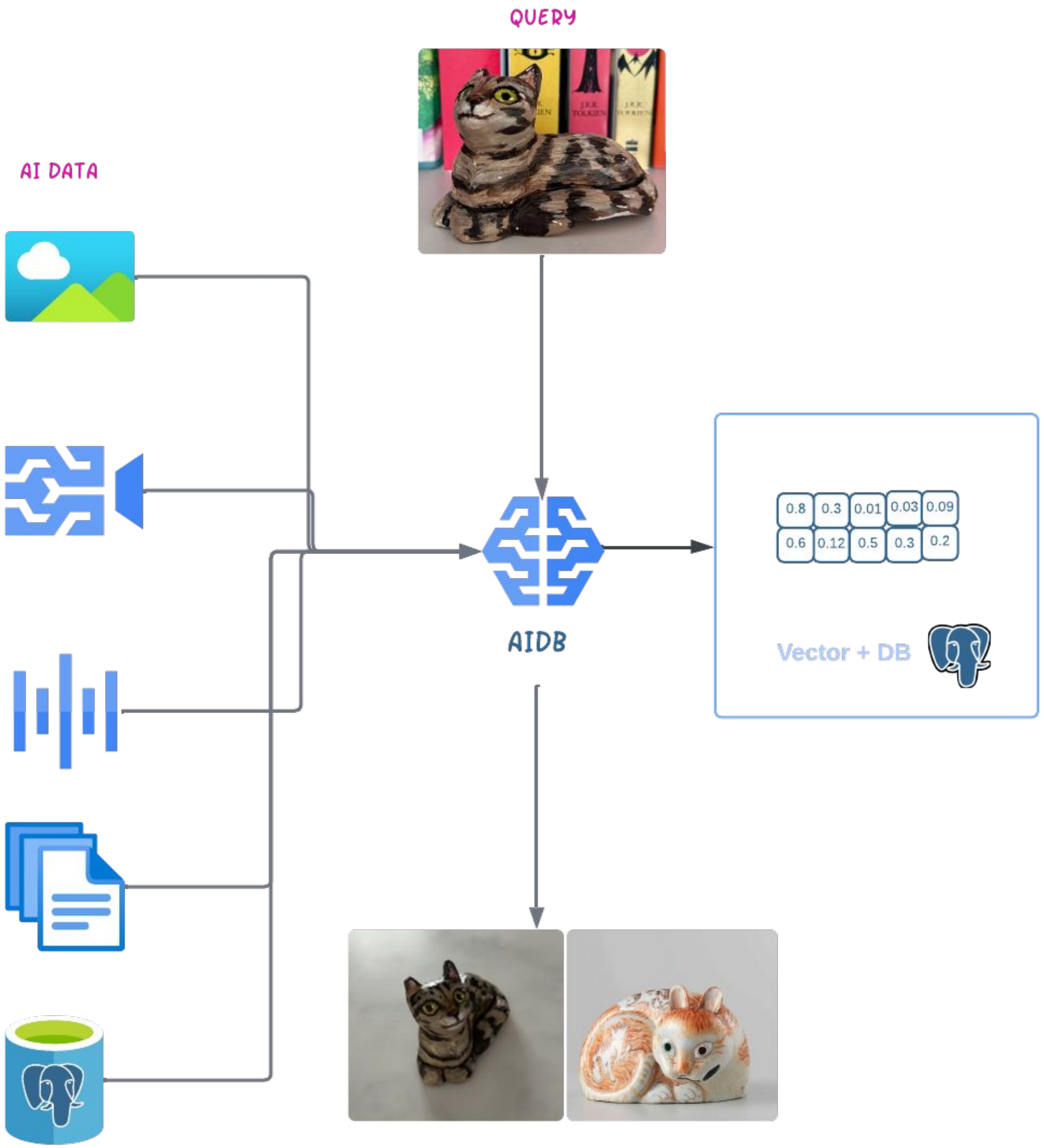
aidb-functions

```
postgres=# select routine_name from information_schema.routines where routine_schema='aidb';
          routine_name
----------------------------------
 init
 create_pg_retriever
 create_s3_retriever
 _embed_table_update
 refresh_retriever
 retrieve
 retrieve_via_s3
 register_prompt_template
 render_prompt
 generate
 ag
 rag
 generate_text_embedding
 generate_single_image_embedding
(14 rows)
```

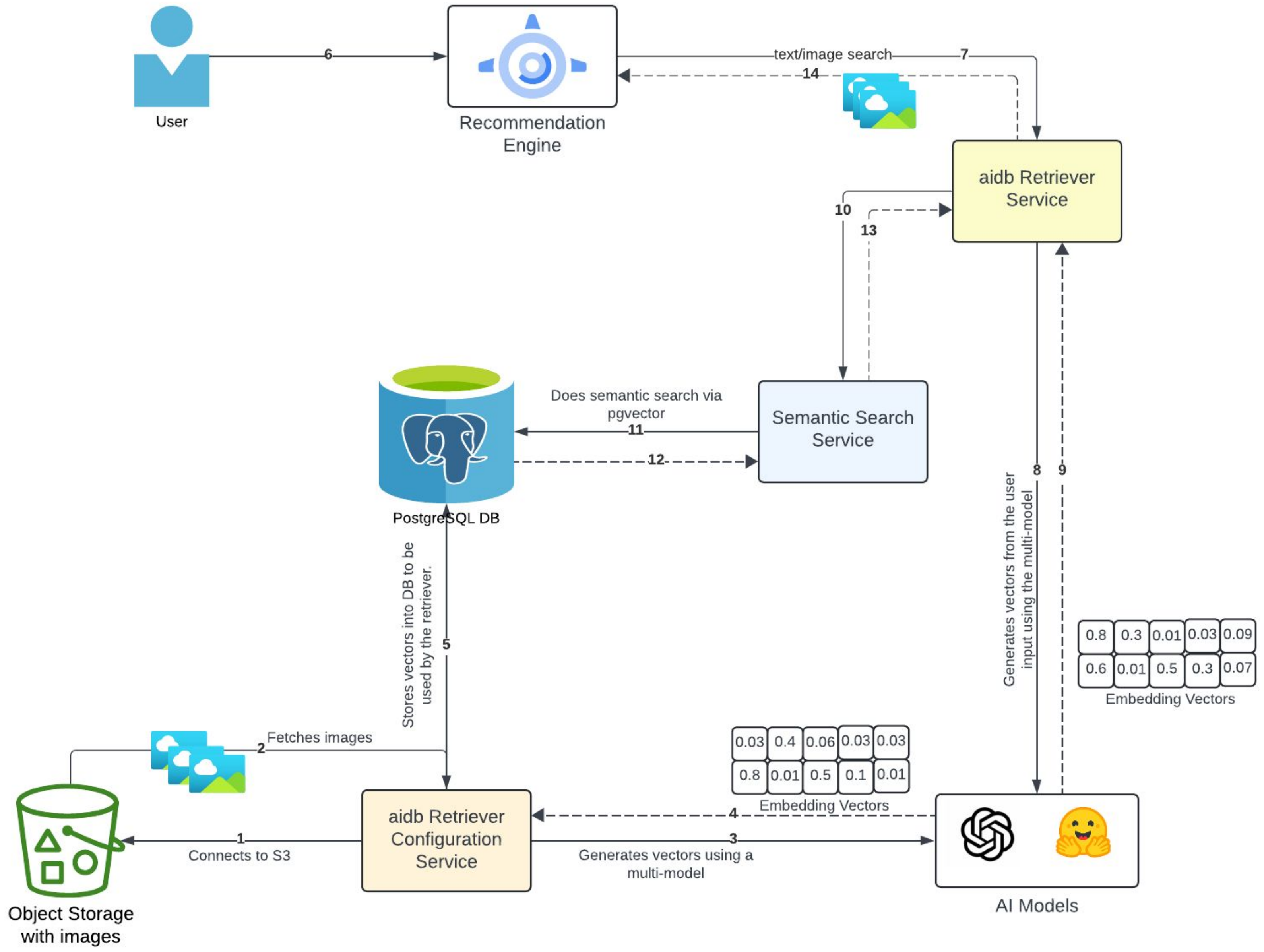
A recommendation engine with pgvector



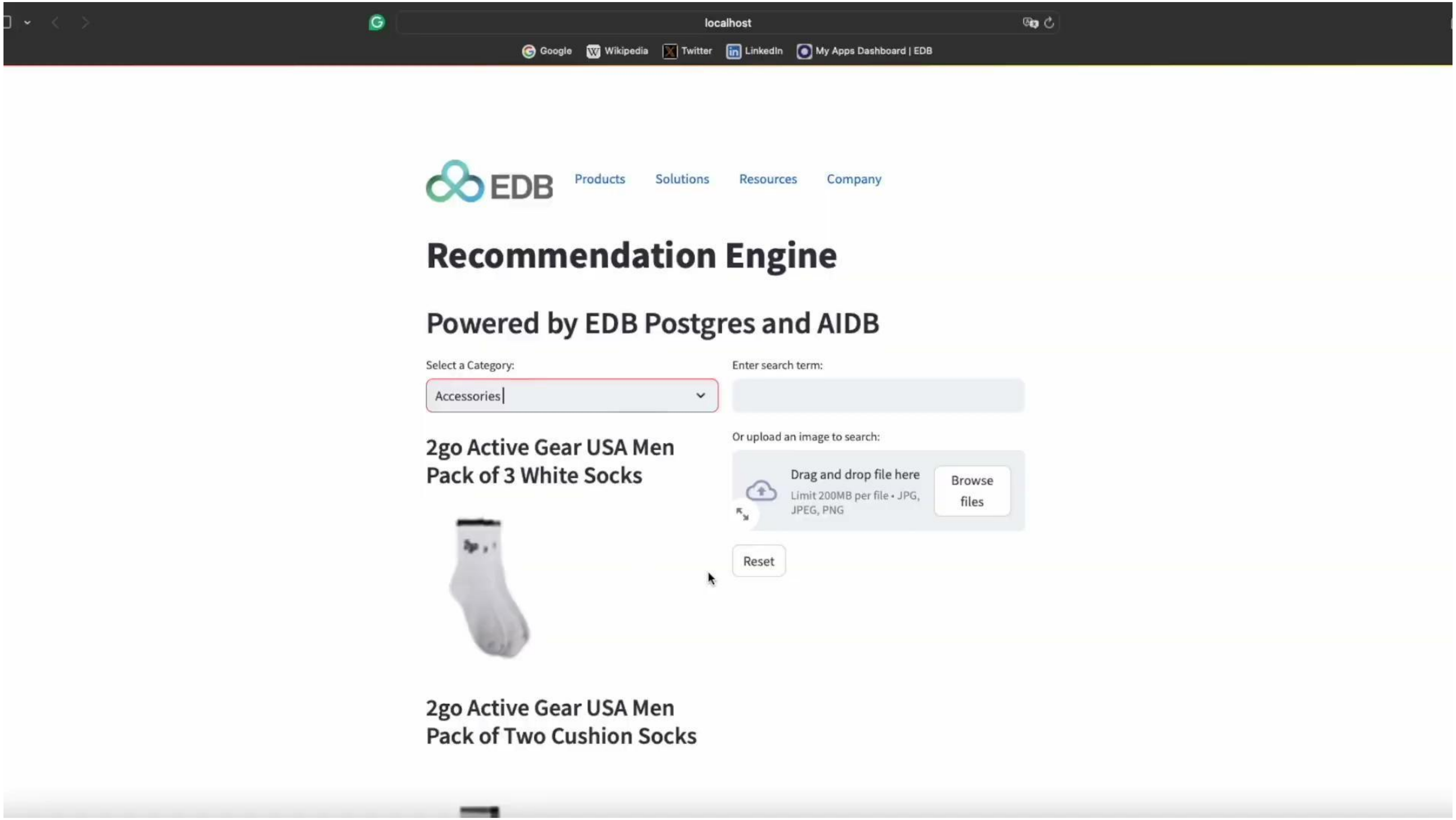
A recommendation engine with aidb



A recommendation engine as an app



Demo


localhost
Google
Wikipedia
Twitter
LinkedIn
My Apps Dashboard | EDB
EDB
Products
Solutions
Resources
Company
Recommendation Engine
Powered by EDB Postgres and AIDB
Select a Category:
Accessories
Enter search term:
Or upload an image to search:
Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files
Reset
2go Active Gear USA Men Pack of 3 White Socks
2go Active Gear USA Men Pack of Two Cushion Socks

Implementation with pgvector

132 lines of code without, **vs** 5 line of function with aidb

```
function_start_time = time.time()

# Load the model and processor with timing
model_loading_start = time.time()
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
model_loading_end = time.time()

fetch_start = time.time()
cursor = conn.cursor()
cursor.execute("SELECT id, gender, mastercategory, subcategory, articletype, basecolour, season, year, usage, productdisplayname FROM products;")
result = cursor.fetchall()
fetch_end = time.time()

batch_size = batch
total_rows_inserted = 0
total_image_processing_time = 0

for i in range(0, 50, batch_size):
    batch_ids = [row[0] for row in result[i:i+batch_size]]
    inputs, valid_paths = load_images_batch(batch_ids, base_path, processor, tag)
    if inputs is not None:
        image_processing_start_time = time.time()
        outputs = model(**inputs)
        image_processing_end_time = time.time()
        embeddings = outputs.image_embeds
        image_processing_time = image_processing_end_time - image_processing_start_time
        total_image_processing_time += image_processing_time

        embeddings_list = embeddings.detach().cpu().numpy().tolist()

        with conn.cursor() as cursor:
            for idx, embedding in enumerate(embeddings_list):
                row = result[i + idx]
                image_path = valid_paths[idx]
                cursor.execute(
                    """INSERT INTO products_emb
                    (id, gender, mastercategory, subcategory, articletype, basecolour, season, year, usage, productdisplayname, image_path, embedding) """
                    "VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)",
                    (row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], image_path, embedding)
                )
                total_rows_inserted += 1

function_end_time = time.time()
total_time = function_end_time - function_start_time
print(f"Total Rows: {total_rows_inserted}")
print(f"Total function execution time: {total_time} seconds")
print(f"Model loading time: {model_loading_end - model_loading_start} seconds")
print(f"Fetching time: {fetch_end - fetch_start} seconds")
```

Implementation with aidb

132 lines of code without, **vs** 2 line of function with aidb

```
cur.execute(f"""
    SELECT aidb.create_s3_retriever(
        '{retriever_name}',
        'public',
        'clip-vit-base-patch32',
        'img',
        '{s3_bucket_name}',
        '',
        '{s3_endpoint}'
    );
""")
cur.execute(f"SELECT aidb.refresh_retriever('{retriever_name}');")
```

Implementation with pgvector

70 lines of code without, **vs** 1 line of function with aidb

```
CREATE OR REPLACE FUNCTION generate_embeddings_clip_bytea(
    RETURNS SETOF vector
    COST 100
    VOLATILE PARALLEL UNSAFE
    ROWS 1000
AS $BODY$

import sys
import os
path = '{}/lib/python{}.{}/site-packages'.format(
        os.environ['VIRTUAL_ENV'],
        sys.version_info.major,
        sys.version_info.minor
    )
sys.path.append(path)
from PIL import Image
from transformers import CLIPModel, CLIPProcessor
import numpy as np
from io import BytesIO  # Import BytesIO to handle bytea input

# Define the model and processor outside the loop to avoid reloading them for each image
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

# Convert the bytea data to a bytes-like object and load the image
img_bytes = BytesIO(img_bytea)
img = Image.open(img_bytes)

# Process the image and calculate embeddings
inputs = processor(text=[tag], images=img, return_tensors="pt")
outputs = model(**inputs)
embedding = outputs.image_embeds

# Convert embeddings to a list to store in the database
embeddings_list = embedding.tolist()

return embeddings_list
$BODY$;
```

```
query = text(
    "SELECT public.generate_embeddings_clip_bytea(:bytes_data, 'person'::text);"
)
with engine.connect() as connection:
    vector_result = connection.execute(query, {"bytes_data": bytes_data})
    data = [
        {"generate_embeddings_clip_bytea": row["generate_embeddings_clip_bytea"]}
        for row in vector_result.mappings().all()
    ]

    # If you expect a single embedding or a single row, extract it
    if data:
        # If there's only one row, return the first row's data
        return data[0]
```

```
query = text(
    """SELECT id, productDisplayname, image_path FROM products_emb
    ORDER BY (embedding <=> :vector_result) LIMIT 2;"""
)
if isinstance(
    vector_result, list
):  # If it's a list, format it as a string that PostgreSQL understands
    vector_result = "[" + ",".join(map(str, vector_result)) + "]"

with engine.connect() as connection:
    result = connection.execute(query, {"vector_result": vector_result})
    data = [
        {
            "id": row["id"],
            "name": row["productdisplayname"],
            "image_path": row["image_path"],
        }
        for row in result.mappings().all()
    ]
```

Implementation with aidb

70 lines of code without, **vs** 1 line of function with aidb

```
cur.execute(
    f"""SELECT data from
    aidb.retrieve_via_s3('{st.session_state.retriever_name}', 5, '{st.session_state.s3_bucket_name}', '{image_name}', '{st.session_state.s3_endpoint}');"""
)
```

Implementation with pgvector

55 lines of code without, **vs** 1 line of function with aidb

```
CREATE OR REPLACE FUNCTION generate_embeddings_clip_text(text_query text)
RETURNS float[] AS
$$
    import sys
    import os
    path = '{}/lib/python{}.{}/site-packages'.format(
            os.environ['VIRTUAL_ENV'],
            sys.version_info.major,
            sys.version_info.minor
        )
    sys.path.append(path)
    import torch
    from transformers import CLIPProcessor, CLIPModel

    model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
    processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

    inputs = processor(text=[text_query], return_tensors="pt")
    inputs = {k: v for k, v in inputs.items()}

    with torch.no_grad():
        text_embeddings = model.get_text_features(**inputs).cpu().numpy().tolist()

    return text_embeddings[0]
```

```
query = text("SELECT public.generate_embeddings_clip_text(:text_query);")
with engine.connect() as connection:
    vector_result = connection.execute(query, {"text_query": text_query})
    data = [
        {"embedding": row["generate_embeddings_clip_text"]}
        for row in vector_result.mappings().all()
    ]

    # If you expect a single embedding or a single row, extract it
    if data:
        # If there's only one row, return the first row's data
        return data[0]
```

```
query = text(
    """SELECT id, productDisplayname, image_path FROM products_emb
    ORDER BY (embedding <=> :vector_result) LIMIT 2;"""
)
if isinstance(
    vector_result, list
):  # If it's a list, format it as a string that PostgreSQL understands
    vector_result = "[" + ",".join(map(str, vector_result)) + "]"

with engine.connect() as connection:
    result = connection.execute(query, {"vector_result": vector_result})
    data = [
        {
            "id": row["id"],
            "name": row["productdisplayname"],
            "image_path": row["image_path"],
        }
        for row in result.mappings().all()
    ]
```

Implementation with aidb

55 lines of code without, **vs** 1 line of function with aidb

```
cur.execute(
    f"""SELECT data from aidb.retrieve('{text_query}', 5, '{st.session_state.retriever_name}');"""
)
```

Step by step guide

```
-   postgres=# CREATE EXTENSION IF NOT EXISTS aidb CASCADE;
-   postgres=# CREATE TABLE IF NOT EXISTS products (
                 id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
                 img_id TEXT UNIQUE,
                 gender VARCHAR(50),
                 masterCategory VARCHAR(100),
                 subCategory VARCHAR(100),
                 articleType VARCHAR(100),
                 baseColour VARCHAR(50),
                 season TEXT,
                 year INTEGER,
                 usage TEXT NULL,
                 productDisplayName TEXT NULL
             );
```

Step by step guide

- ```
  postgres=# SELECT aidb.create_s3_retriever(
              'recommendation_engine',
              'public',
              'clip-vit-base-patch32',
              'img',
              'public-ai-team',
              '',
              'http://s3.eu-central-1.amazonaws.com'
          );
  ```

- ```
  postgres=# SELECT aidb.refresh_retriever('recommendation_engine');
  ```

- ```
  Fashion git:(main): streamlit run code/app_search_aidb.py recommendation_engine
  public-ai-team http://s3.eu-central-1.amazonaws.com
  ```
  ```

Step by step guide

- ```
  postgres=# SELECT aidb.create_pg_retriever(
  'product_embeddings_pg', -- Name of the similarity retrieval setup
   'public', -- Schema of the source table
   'id', -- Primary key
   'all-MiniLM-L6-v2', -- Embeddings encoder model for similarity data
   'text',
   'products', -- Source table name default is products
   ARRAY['mastercategory', 'productdisplayname'], -- Columns in source table with the data for
  similarity retrieval
   TRUE
  );
  ```

- ```
  postgres=# INSERT INTO products (img_id, gender, masterCategory, subCategory, articleType,
  baseColour, season, year, usage, productDisplayName)
  VALUES (70, 'Men', 'Apparel', 'Topwear', 'Shirts', 'Navy Blue', 'Fall', 2011,
  'Casual', 'Turtle Check Men Navy Blue Shirt');
  ```

EDB Postgres AI – Maximum Return of Investment in Postgres

HW Acceleration

Sustainable AI

Don’t Move My Data

EDB Postgres AI Platform

100% built on Postgres

100% Leverage of Postgres Incumbency:

- Incumbent Postgres Data
- Incumbent Postgres Systems
- Incumbent Postgres Skills
- Incumbent Postgres Apps
- Incumbent Postgres Quality of Service

Customer 360

Big Data Scaling

Sovereign AI

Future of EDB Postgres AI: AI & Analytics Capabilities

EDB Postgres AI Platform

1. Data Integration

- External Storage Support
- Tiered Tables
- Lakehouse SQL INSERT
- Migration & Analytic Synch
- Metastore & ETL Vendors

2. Analytic Acceleration

- Columnar Query Engine
- Auto Compaction
- Real-time analytics
- Conversational SQL
- GPU-Accelerated Analytics

3. Search

- PGVECTOR
- Hybrid Search Index
- Text Search
- GPU-Accelerated Search

4. Orchestration

- AI Pipelines
- Embedded model hosting in PG Process
- Text/Image Embeddings
- Auto Embeddings & Retrievers
- In-DB RAG
- AI Feature Engineering
- EDB connectors in AI solution frameworks
- AI Platform Vendors

5. Serving

- Enterprise LLM hosting
- Chat models
- GPU-Accelerated Models
- Model Serving Vendors

GPU

Acceleration

Wide funnel for existing data

Scaling analytics w/ columnar engine

Scaling AI search

Broad coverage of application patterns

Scaling hosting of Enterprise AI model

Summary

- PostgreSQL was only a relational DB before pgvector for an AI Engineer.
- Data and vector are better together.
- pgVector brought vector capabilities like semantic search in PostgreSQL and fulfilled the Vector DB needs as well as relational DB.
- However it's hard to install and it's complicated for someone who don't know AI and PostgreSQL.
- EDB Postgres AI brings simplicity and hides complexity without compromising from capabilities.
- The platform also offers Lakehouse capabilities.

Thank You!