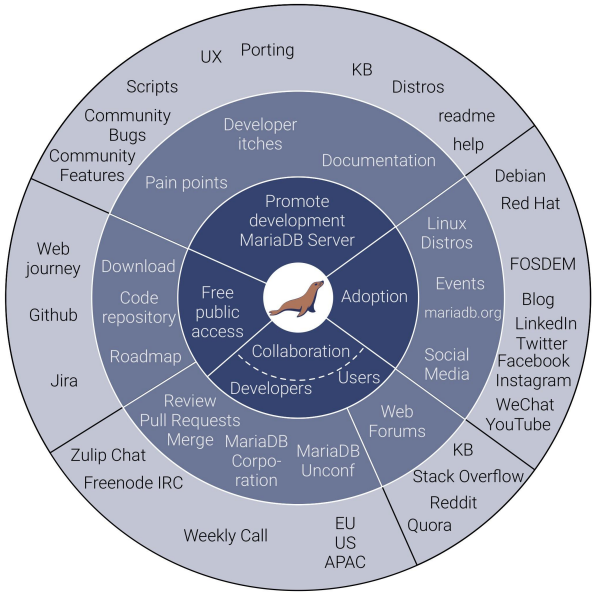


MariaDB Vector: Why your AI data should be in an RDBMS

SFSCON, Bozen, Italy
Fri 8 Nov 2024

Kaj Arnö
kaj@mariadb.org
CEO, MariaDB Foundation

Why should vectors be stored in a standard Open Source RDBMS?

Standard RDBMS:

- The **rest of your data** is there! (Source, results)
- You may need to **audit** the intermediate steps
- AI apps are largely “**standard IT**” apps
- Single queries can **combine** vector & standard data
- **Developers** know the standard RDBMSes
- Numerous **tools** work with the standard RDBMSes

Open Source RDBMS:

- Lack of vendor **lock in**
- Lower total **cost**

Agenda

1. What exactly **is** MariaDB Server?
2. What **AI functionality** does MariaDB have?
3. What are the main **use cases** for MariaDB Vector?
4. Steps in **creating a RAG** with MariaDB Vector
5. So **where is the advantage** of using MariaDB?

6. **Technical details** (another TOC; on your own time)

What exactly is MariaDB Server?

Definition: MariaDB is a ***mature extended fork of MySQL***.

- It's near **plug-in compatible** with MySQL
- It's **fully** open source
- It's more **performant**
- It adds plenty of **functionality** on top (sql_mode=Oracle)

Compare MariaDB to:

- **MySQL** (but no vector indexing in Open Source)
- **PostgreSQL** (PG Vector! but slower, plug-in)
- **Vector** databases (but specialised)
- **Oracle** Database (but no vector contender)

What AI functionality does MariaDB have?

Vector indexing and search: We store and search vectors!

- You decide how **you create** the vectors (outside MariaDB)
- We **store** and **index** your vectors (now HNSW, soon IVFFlat)
- We **search** your vectors (Euclidean, Cosine; soon pushdown)

This is exactly the task of a standard database:

- **Nearest-neighbour** search
- Index vectors of **any data** (text, images, videos, sound)
- **Combining** the above with all the existing other data
- **Pushdown** WHERE conditions: combine search criteria

What are the main use cases for MariaDB Vector?

Classic RAG: Retrieval-Augmented Generation

- Do a "specialised ChatGPT", which gives answers only based on your own data
- Implement this with high relevance ("quality") and at reasonable cost

Smarter generic search: In online stores and elsewhere

- "You may also be interested in this product"
- Give best guess responses despite vague text input
- Can be combined with exact push-down conditions

Steps in creating a RAG with MariaDB Vector 1/2

Assumption: You want to create a specialised ChatGPT

1. You have a mass of text (articles, documentation)
2. The app should answer free-form questions using that text

Step one: Embed the text mass, in a batch run (not “train”)

3. Identify the proper chunk size (unit to index)
4. Create and store the chunks in MariaDB (using eg. Python)
5. Vectorise the chunks (using your favourite LLM)
6. Store and index the vectors (using MariaDB), including “classic” database fields that point to the keys of the chunks in source data

Steps in creating a RAG with MariaDB Vector 2/2

Step two: Run-time, answer the user's free-form question

1. Vectorise the user question (using the same LLM)
2. Search the vector index for the top five-ten nearest neighbours to the vectorised user question (in MariaDB)
3. Concatenate the text chunks of the neighbours into an adequately sized text (in Python)
4. Ask the LLM using the user question as prompt and the concatenated text as context

Voilà: You have provided the user with an answer to a question, based **only** on your own data, at a **low token cost**

Sample prompt

You are tasked to answer a question using only the following information:

[chunk1], [chunk2], [...]

This is the question for you to answer:

<User query>

So where is the advantage of using MariaDB?

Audit the intermediate steps

1. Debug your chunkification; verify intermediate results
2. Test various chunkification strategies

Reuse the index

3. Once every user query
4. Make a summary, an analysis

Save cost

5. Do all the searches on your own database
6. Minimise the expensive usage of AI tokens (“words”)

Further slides: Technical Details

- 12-15 What are vectors
- 16-21 Two-dimensional nns picture (nearest neighbour)
- 23 **CREATE TABLE** .. MariaDB syntax example
- 24 **Python syntax** example
- 25-32 Schemas illustrating RAG
- 34 MariaDB **function** syntax
- 35 **How to download** MariaDB Vector
  https://mariadb.org/download
- 36 **Benchmarks**: MariaDB Vector is fast!
- 38 Deep dive: Index hierarchy
- **Docs**: https://mariadb.org/projects/mariadb-vector/

What is an embedding model vs generative model?

- ChatGPT is a **generative** model.
  - It takes a prompt.
  - Generates the most likely "correct" sequence of words as response.

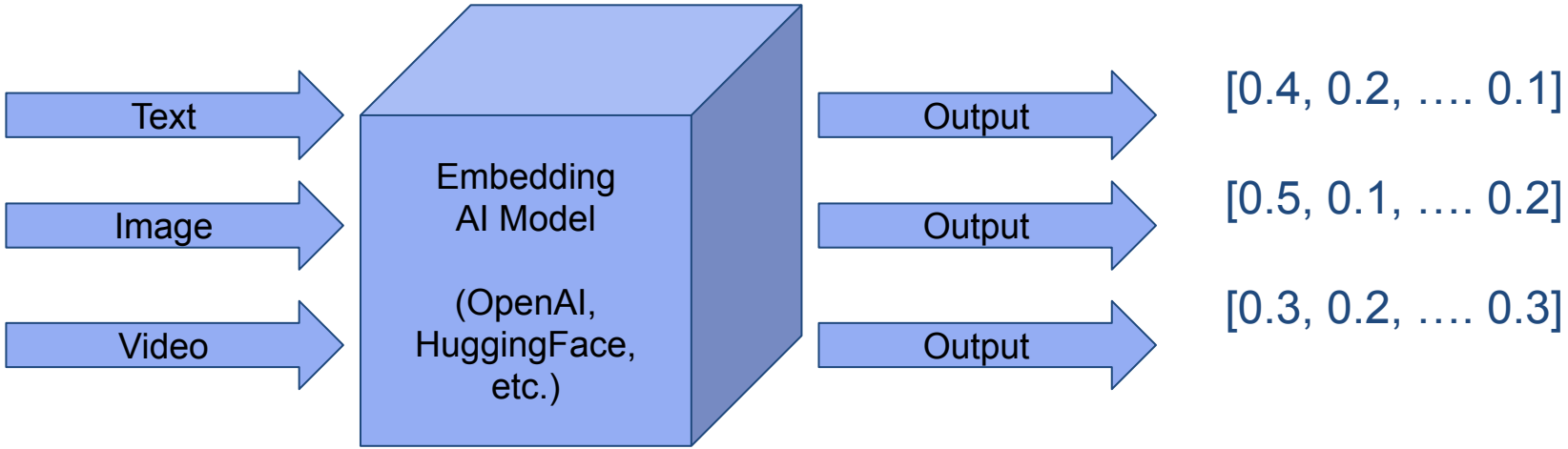
- An **embedding** model generates a vector embedding for a particular prompt.

What is a Vector Embedding?

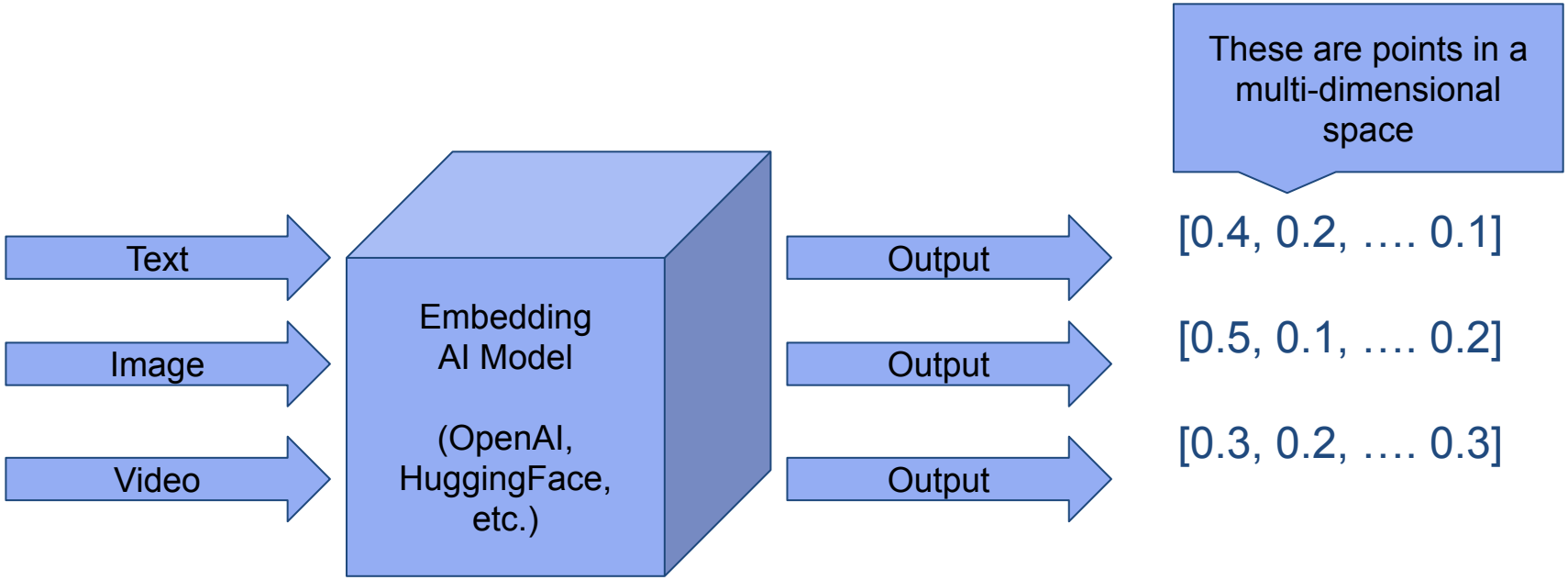
Simply a list of numbers (that describe “features” of the original)

What is a Vector Embedding?

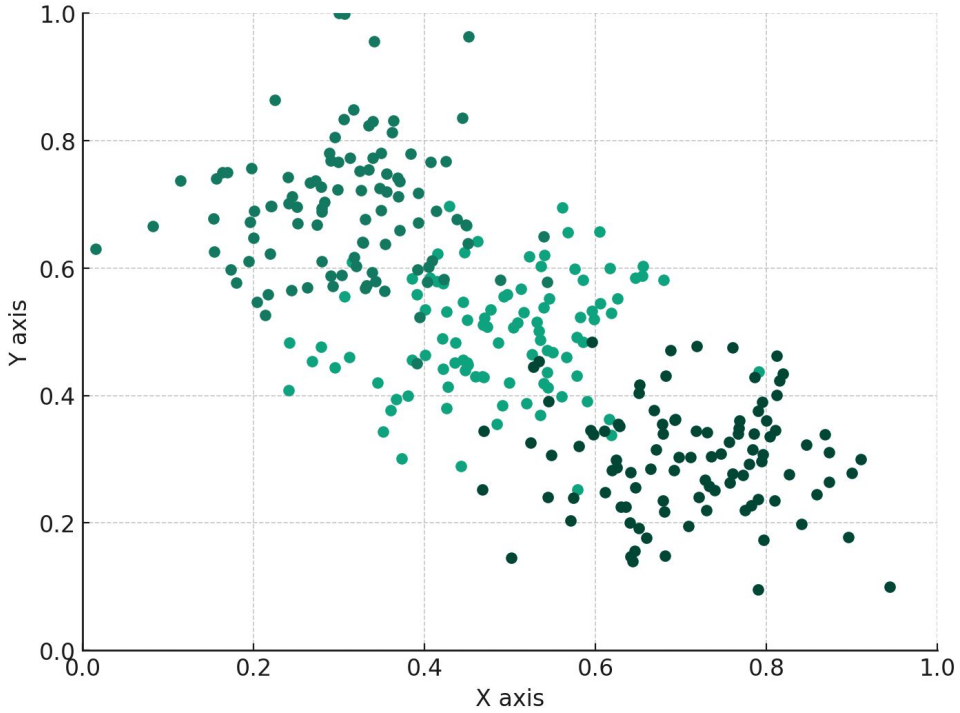
Simply a list of numbers (that describe “features” of the original)



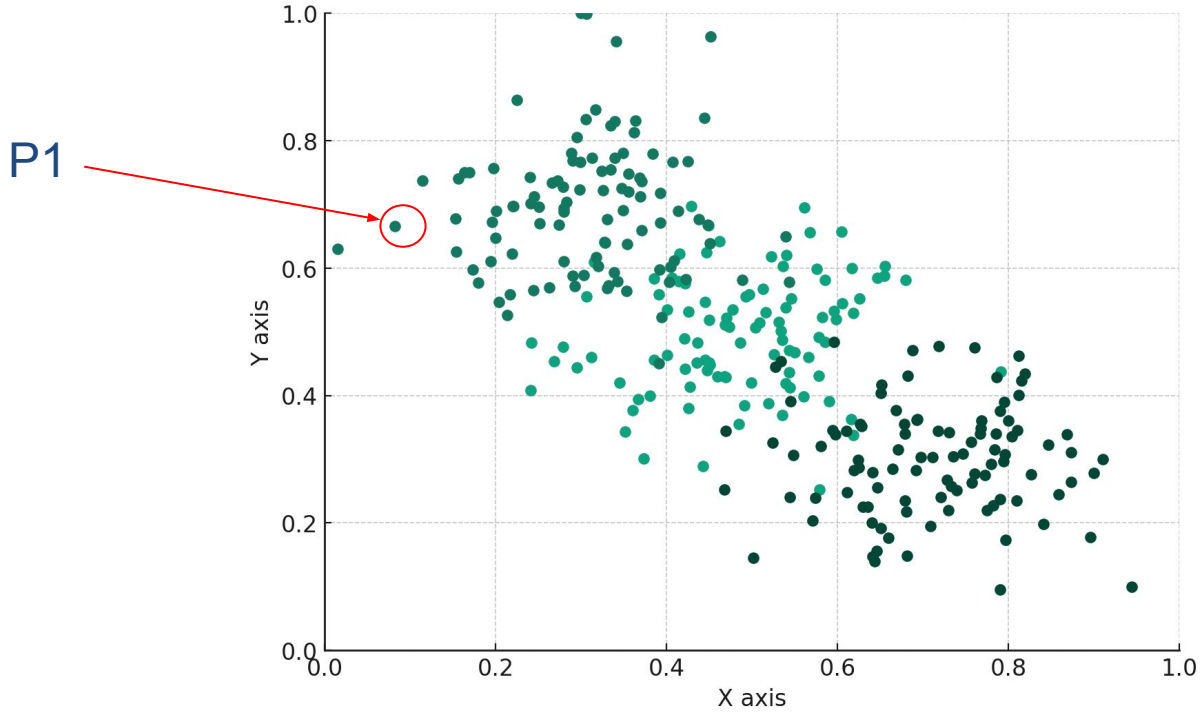
What is a Vector Embedding?



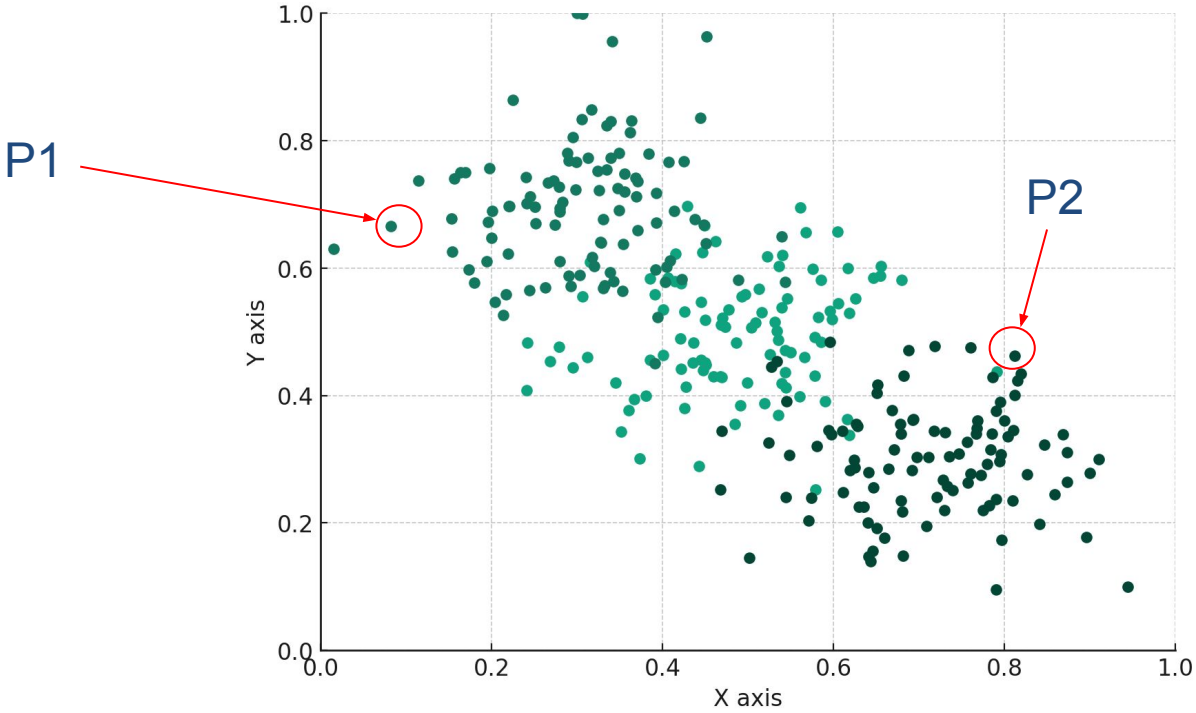
2D example



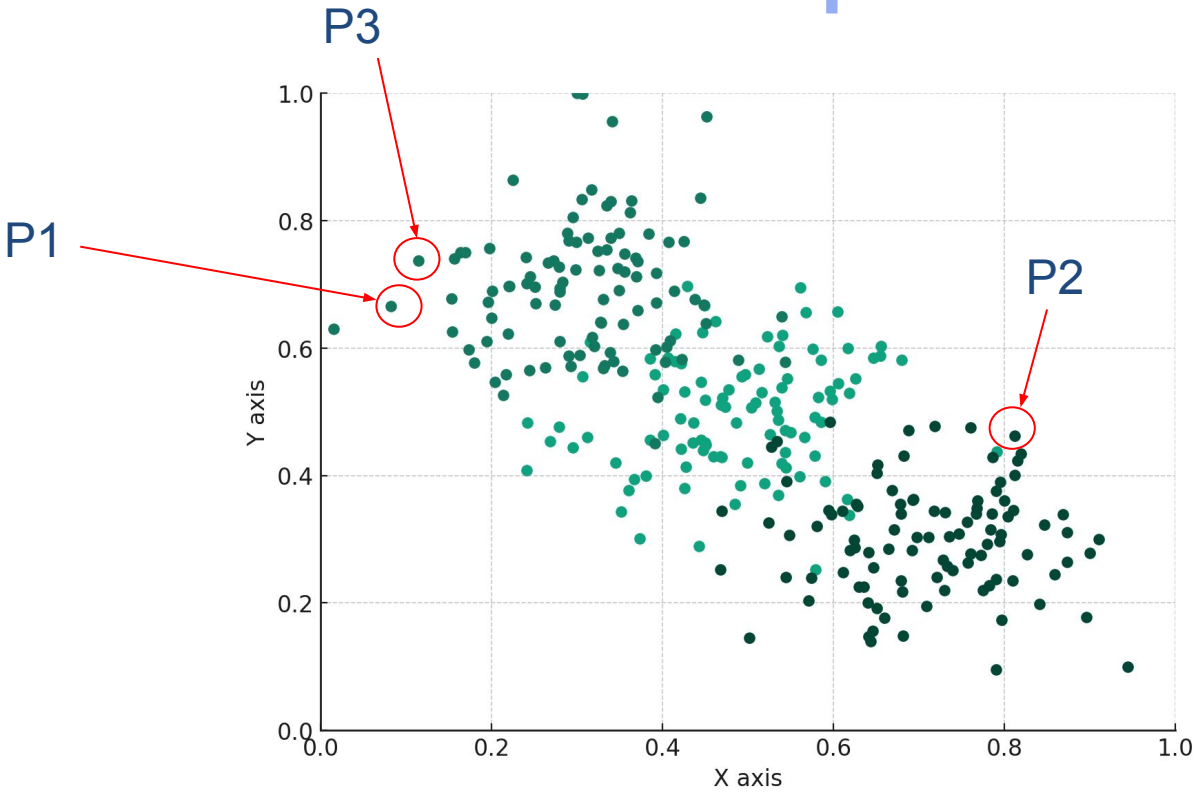
2D example



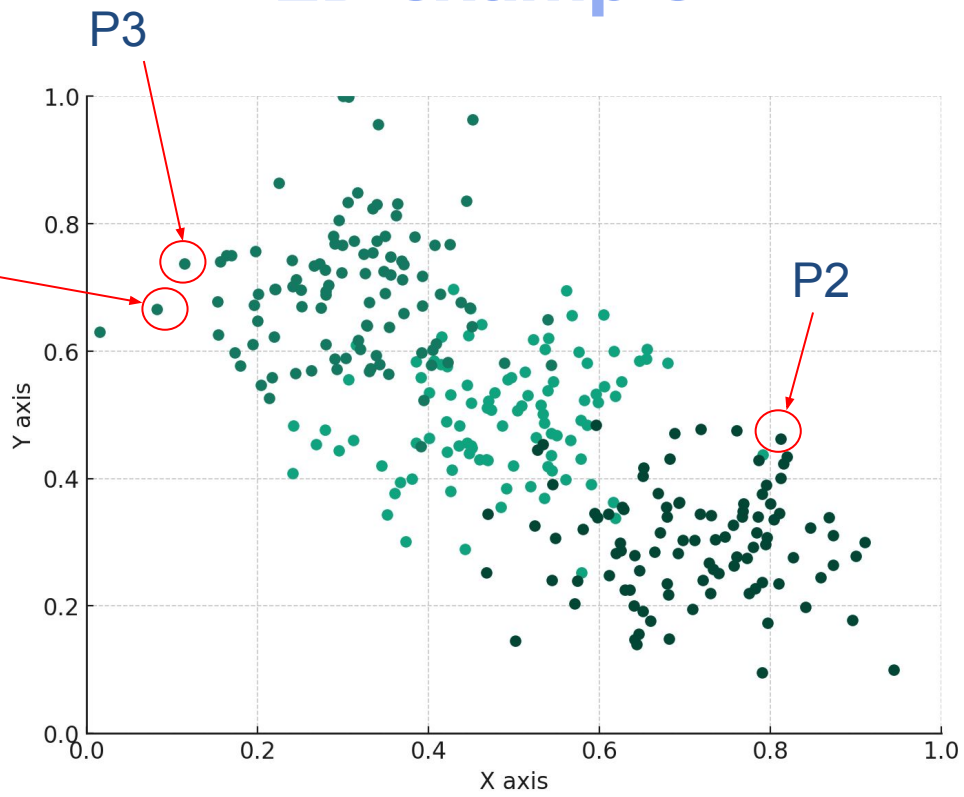
2D example



2D example



2D example



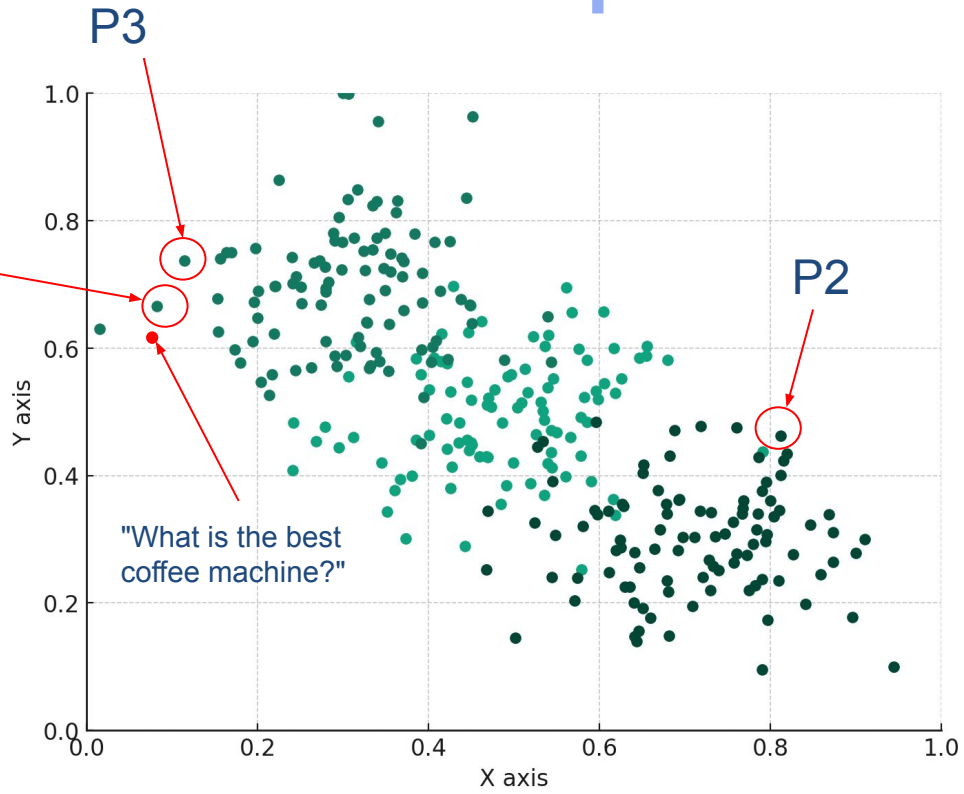
The document associated with P1 is more similar to P3 than P2,

according to the model that generated the points!

2D example

The document associated with P1 is more similar to P3 than P2,

according to the model that generated the points!



As a database user, what must you do?

1. Install a vector database (MariaDB Vector preview now available)

2. Install an Embedding Model
   or
   Setup a cloud hosted model API.

3. Change your application to query the Embedding Model for each document insert and insert the embeddings into the database.

4. Make use of VEC_DISTANCE function to get the (approximate) nearest neighbors.

Create an embedding index

PRODUCTS		
NAME	**DESCRIPTION**	**EMBEDDING**
"Coffee Maker"	"Can brew 10 different coffee types. 5 years warranty."	[0.4, 0.5, 0.3, ….., 0.2]

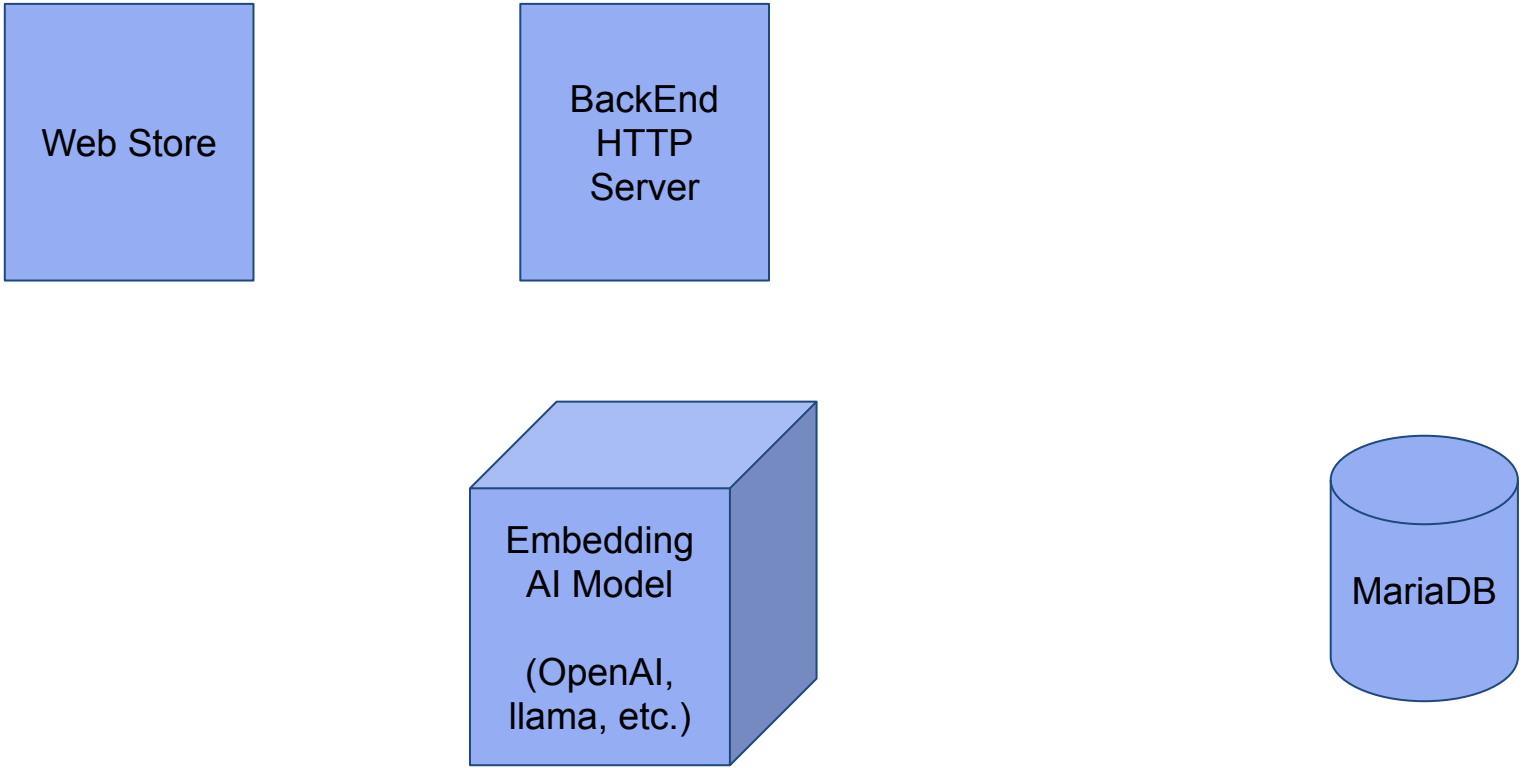
```
CREATE TABLE PRODUCTS (
    name varchar(200) primary key,
    description longtext,
    embedding blob,
    VECTOR INDEX (embedding) MHNSW_M=5
)
```

Modify insert

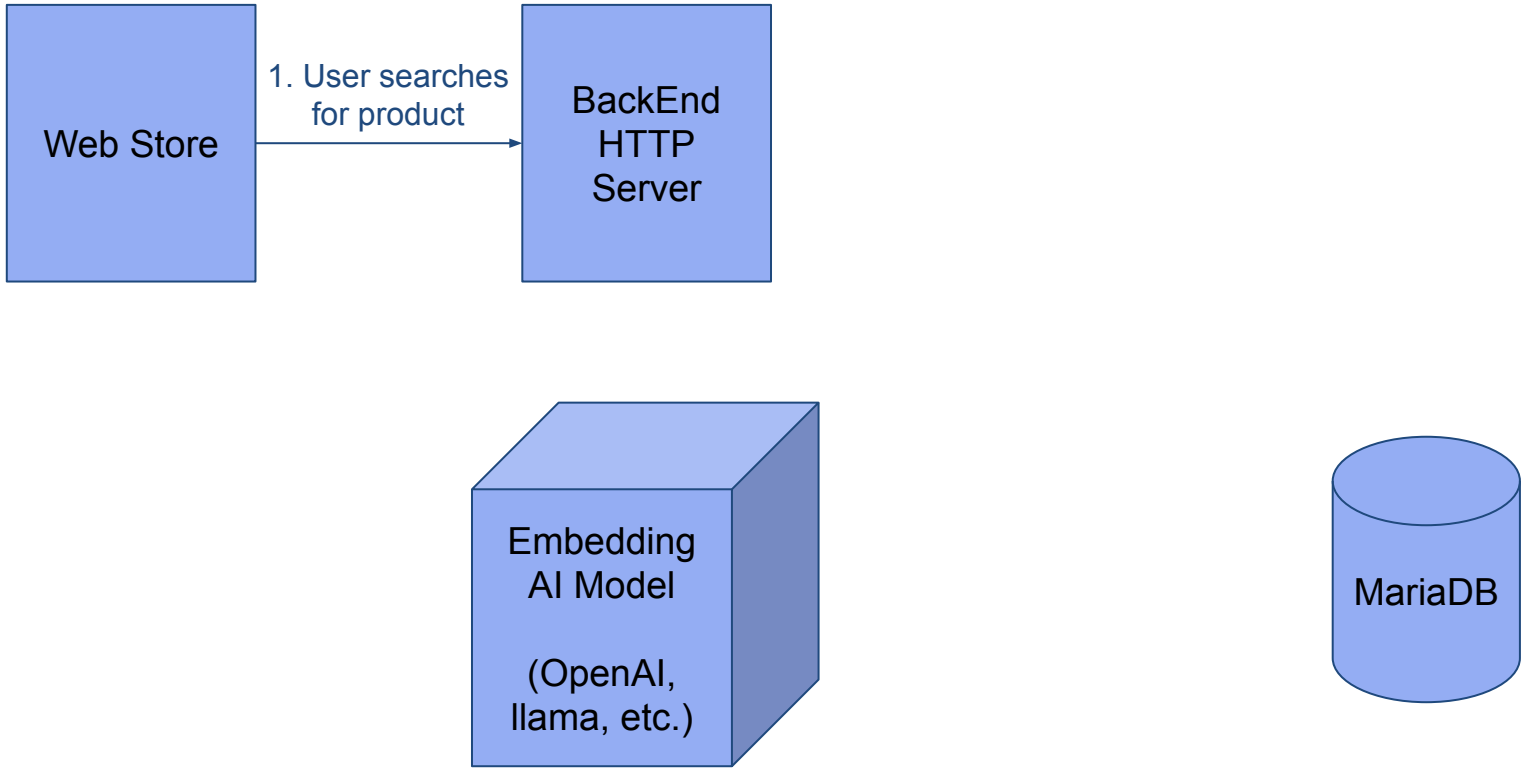
PRODUCTS		
NAME	**DESCRIPTION**	**EMBEDDING**
"Coffee Maker"	"Can brew 10 different coffee types. 5 years warranty."	[0.4, 0.5, 0.3, ….., 0.2]

```
def add_product(db_conn, product, ai_model):
  vector = ai_model.get_embedding(product.name +
                                  product.description)
  db_conn.execute(
      'INSERT INTO products (name, description, embedding)'
      'values (?, ?, ?)',
      (product.name, product.description, vector))
```

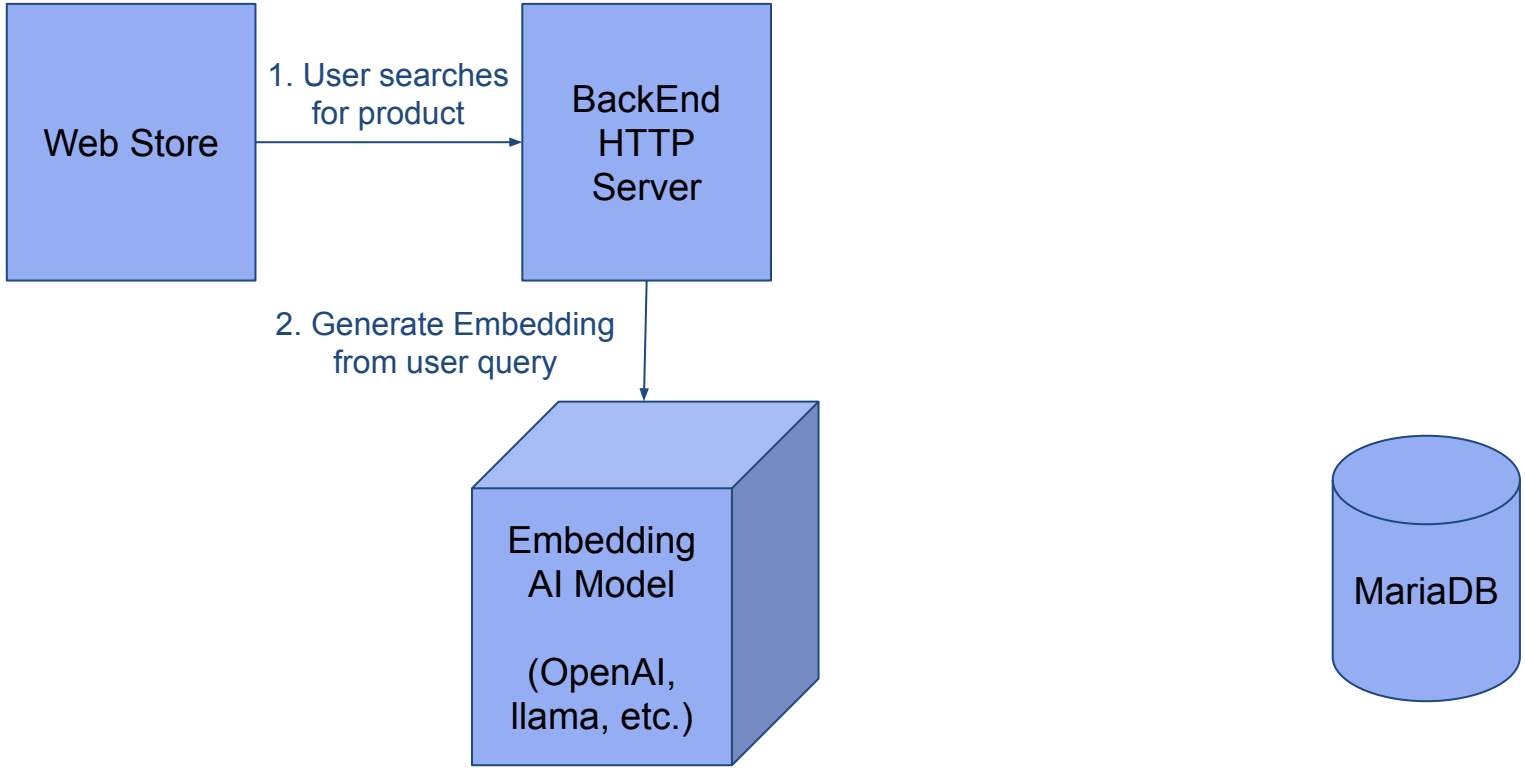
Where Vector search comes into play?



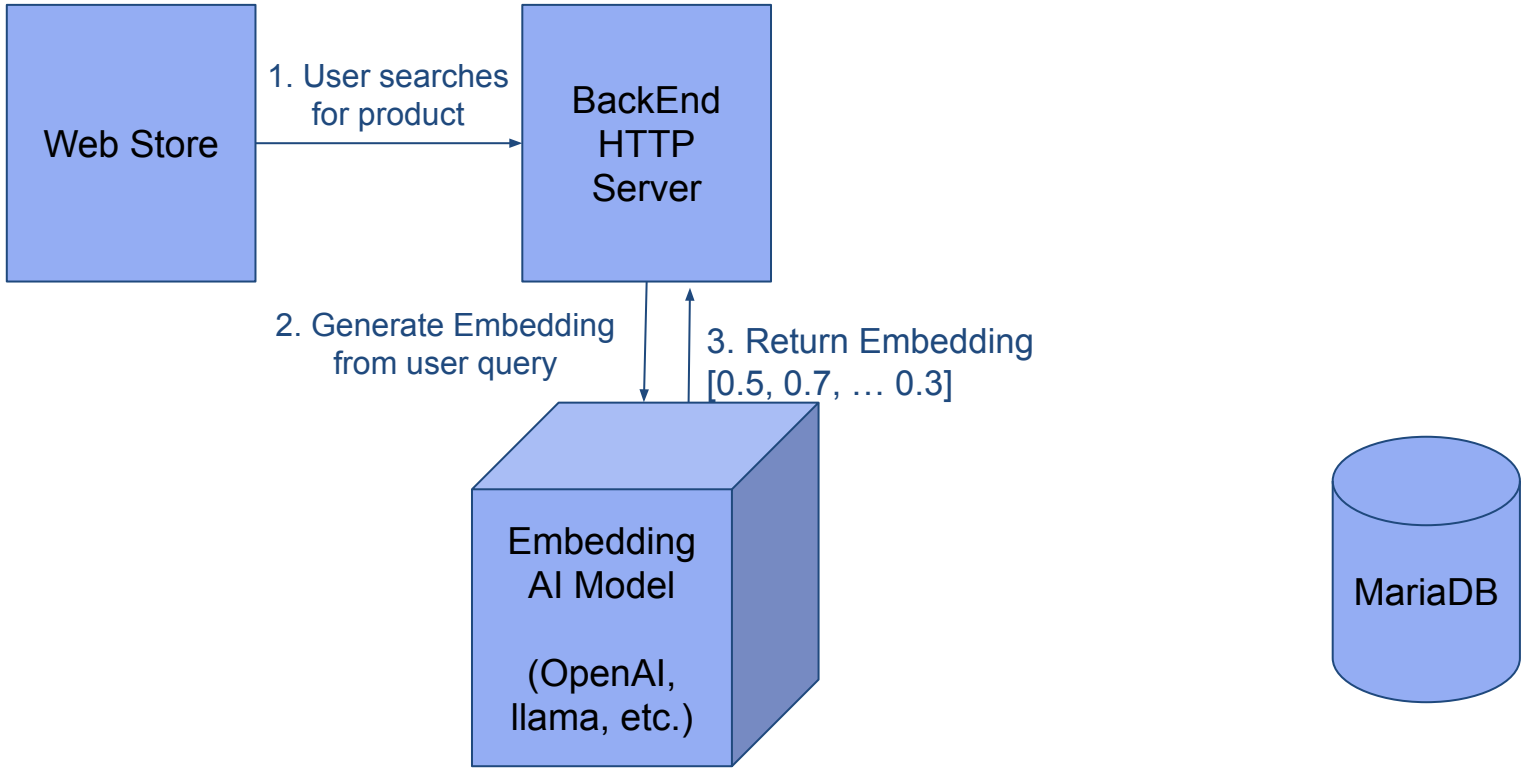
Where Vector search comes into play?



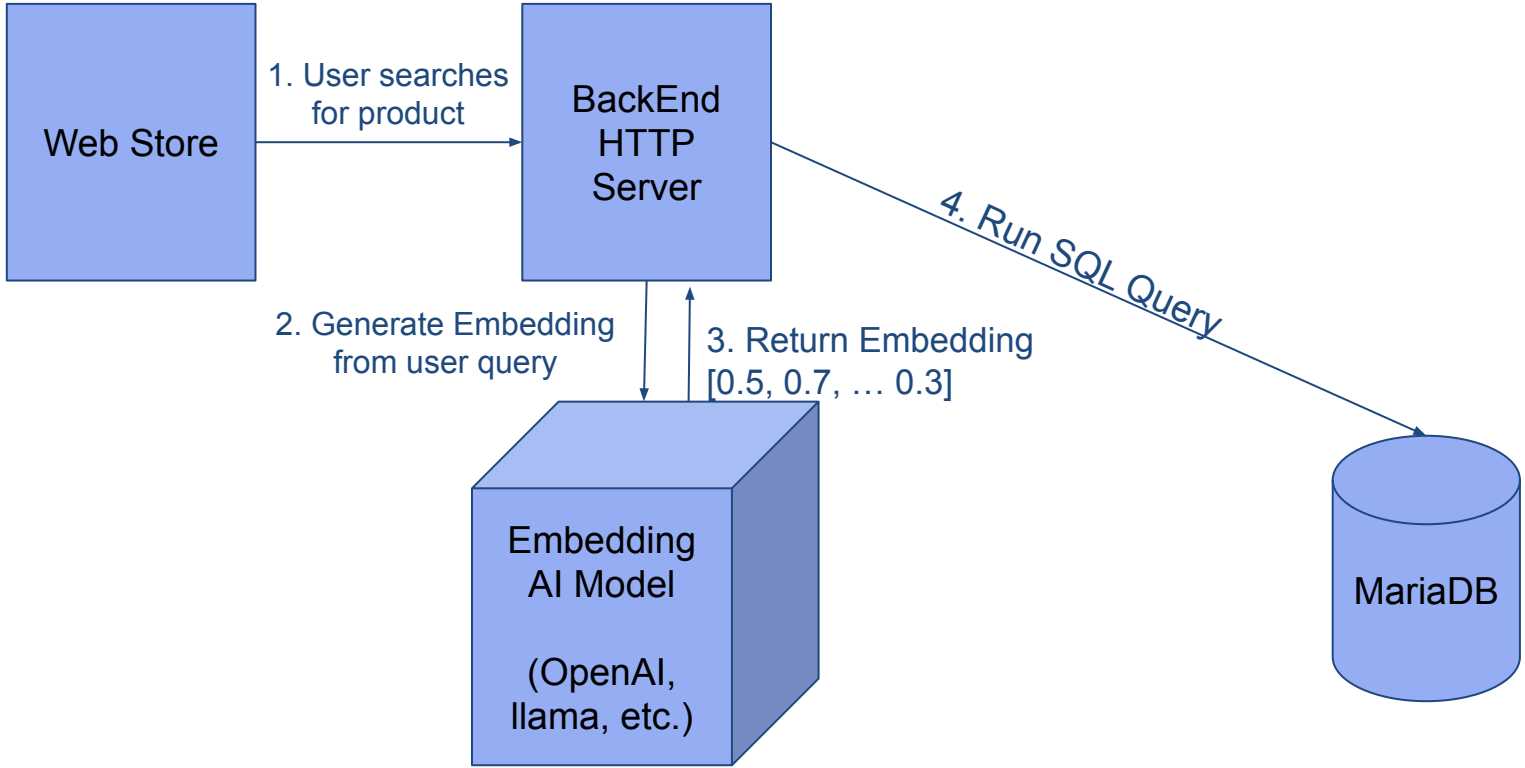
Where Vector search comes into play?



Where Vector search comes into play?

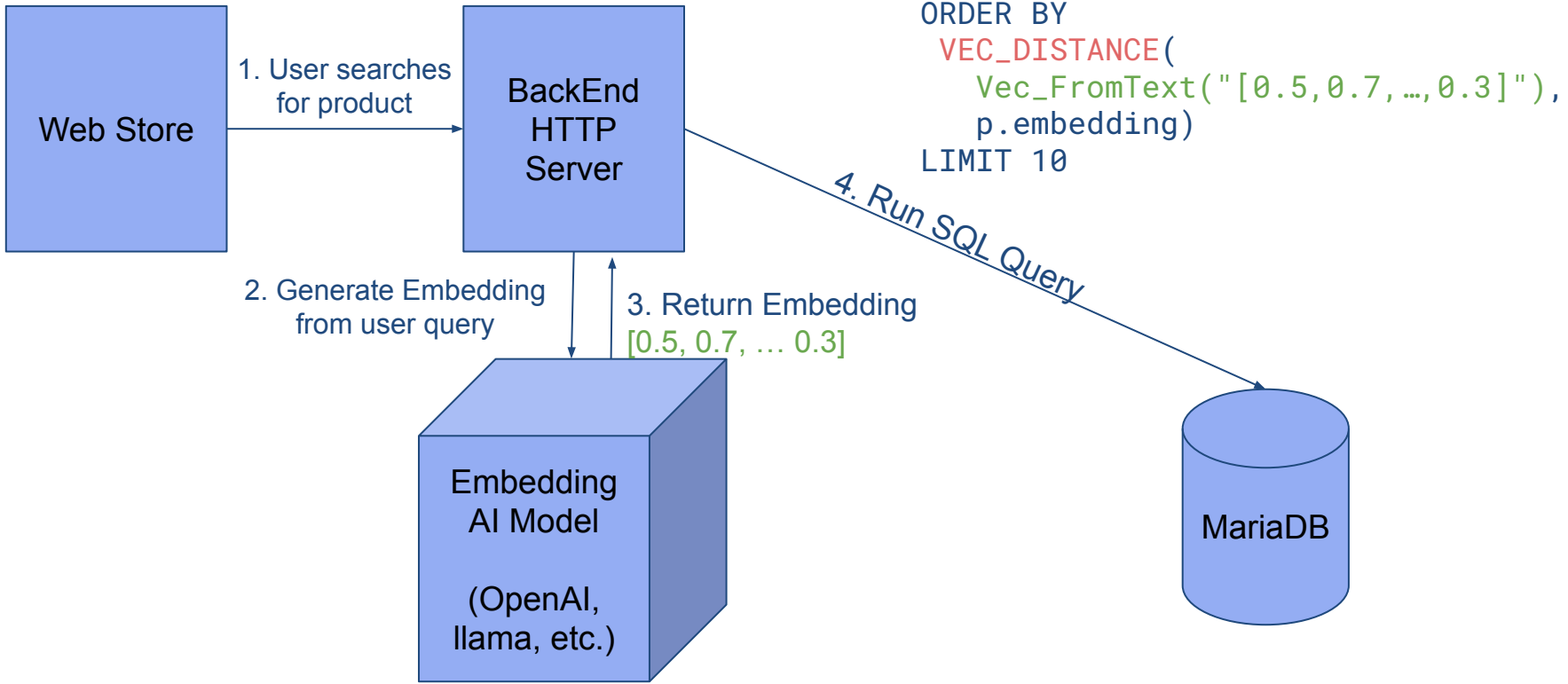


Where Vector search comes into play?



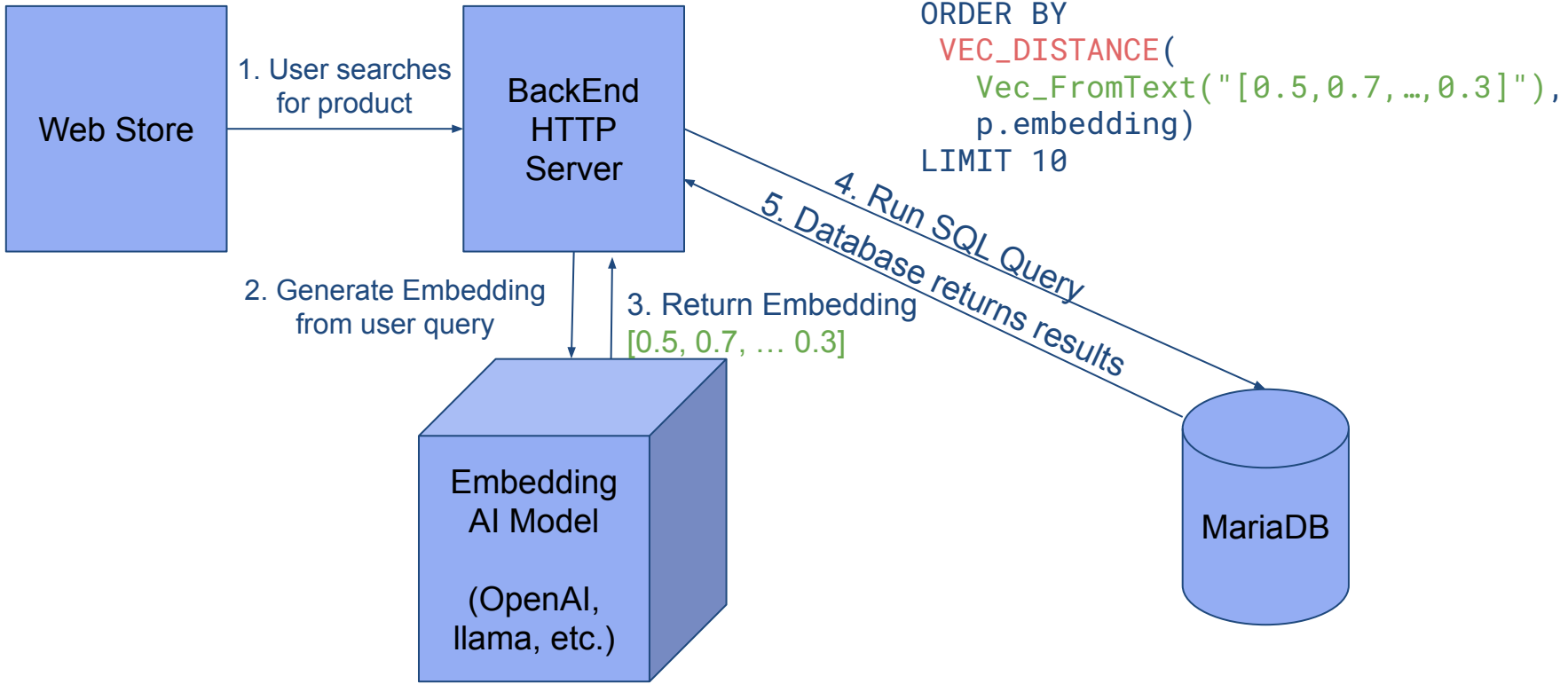
Where Vector search comes into play?

```
SELECT p.name, p.description
FROM products p
ORDER BY
 VEC_DISTANCE(
   Vec_FromText("[0.5,0.7,…,0.3]"),
   p.embedding)
LIMIT 10
```



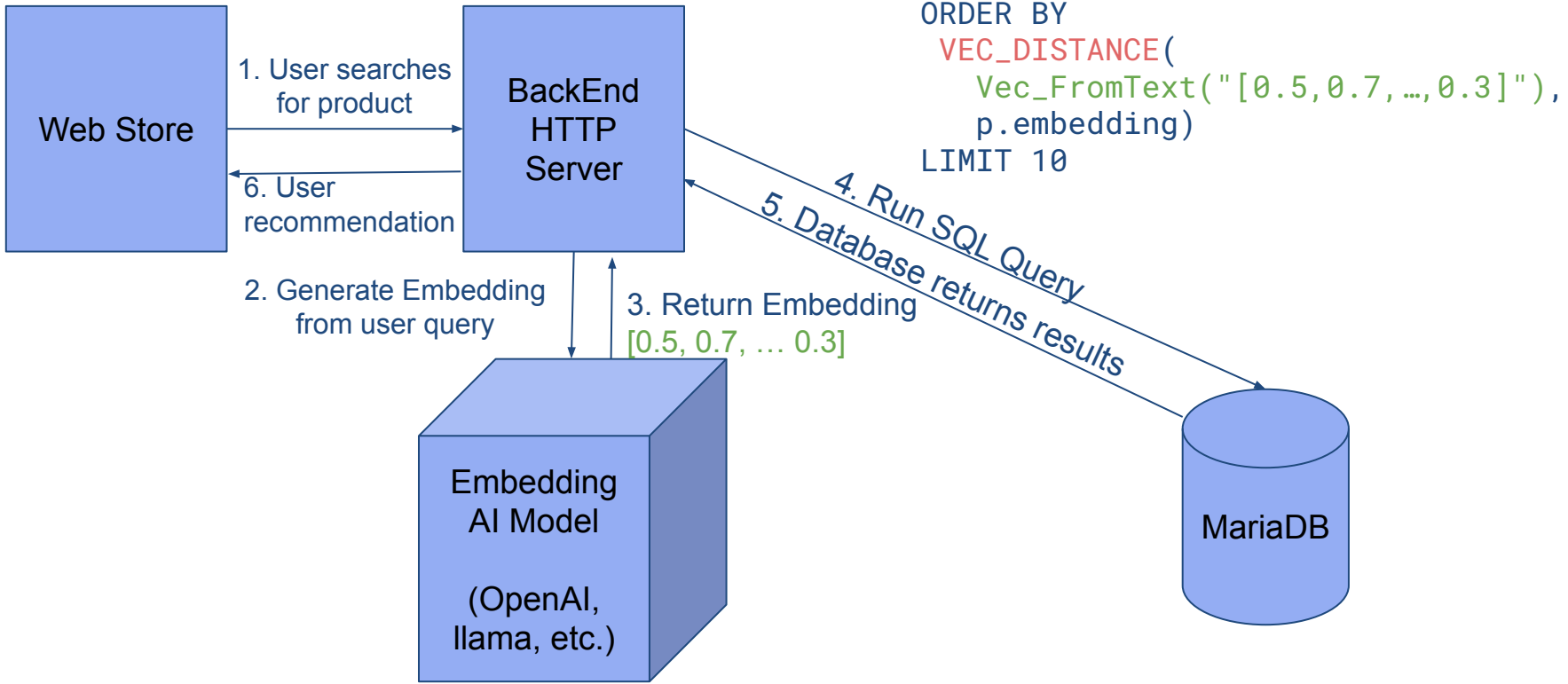
Where Vector search comes into play?

```
SELECT p.name, p.description
FROM products p
ORDER BY
 VEC_DISTANCE(
   Vec_FromText("[0.5,0.7,…,0.3]"),
   p.embedding)
LIMIT 10
```



Where Vector search comes into play?

```
SELECT p.name, p.description
FROM products p
ORDER BY
 VEC_DISTANCE(
   Vec_FromText("[0.5,0.7,…,0.3]"),
   p.embedding)
LIMIT 10
```



What's the catch?

1. Searching for vectors is expensive
2. Indexing strategies for vectors are only "approximate", they don't guarantee the exact "nearest" neighbour.
3. Depending on dataset, some indexing strategies perform better than others.
4. Indexing generally requires a lot of memory.
    a. **HNSW – Hierarchical Navigable Small Worlds**
        i. **de-facto industry standard. Implemented in MariaDB**
        ii. **Large memory usage.**
    b. IVFFlat – Low resource usage, poor search quality, present in pgvector

Project status

- Targeting 11.7 as first stable release (MDEV-33408)
- Performance faster than pgVector on SELECTS (better scaling)
  - More optimizations planned (ARM, PowerPC operations).

- Preview of MariaDB Vector syntax supports:
  - VEC_DISTANCE
  - VEC_DISTANCE_COSINE (euclidean / cosine distance)
  - VEC_FromText() VEC_ToText()

- Work collaboratively with MariaDB plc and other vendors (large contributions from Amazon)

MariaDB Server Version

MariaDB Server 11.7.0 Preview

Display older releases: ☑

Operating System

Linux

Architecture

x86_64

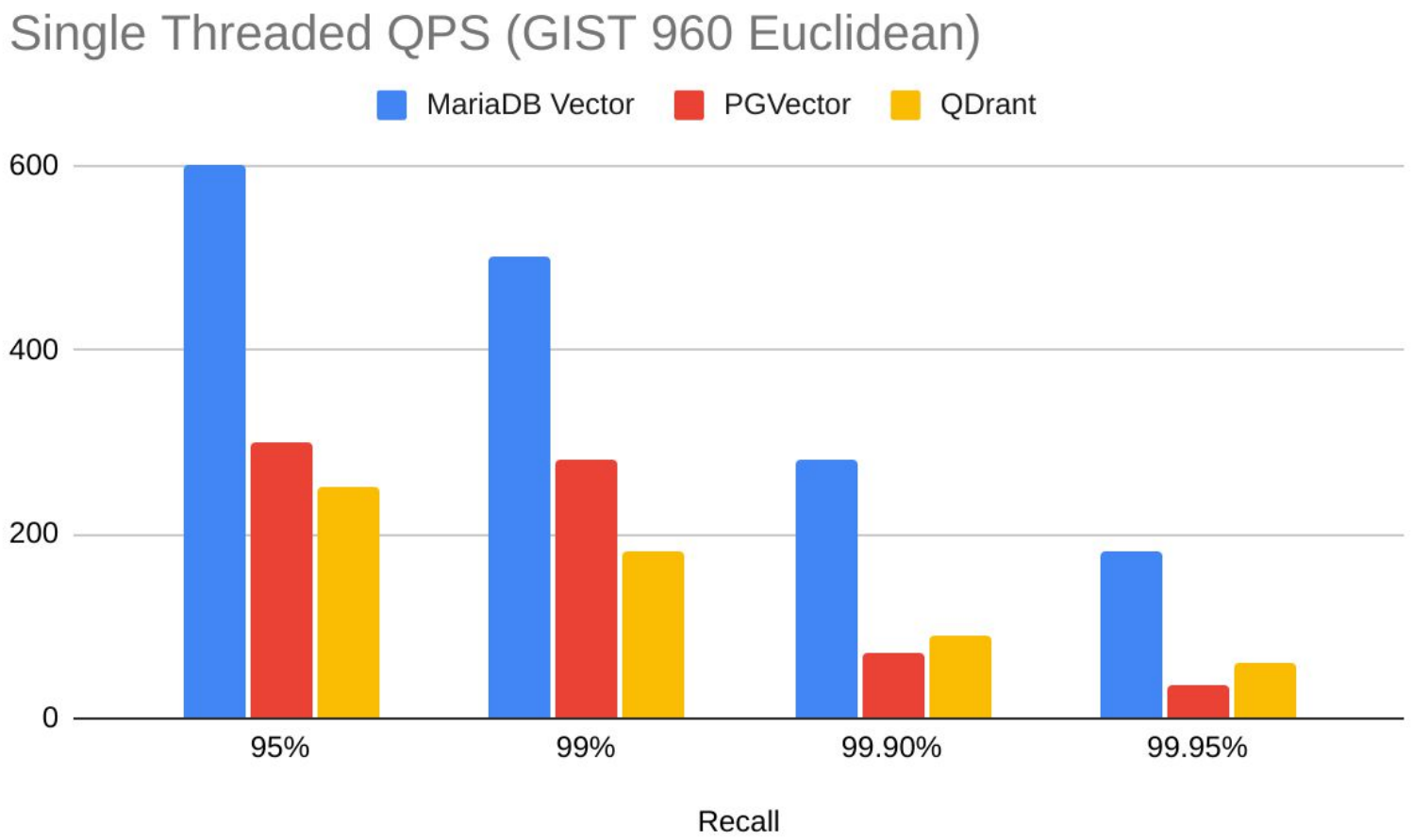
Init System

Systemd

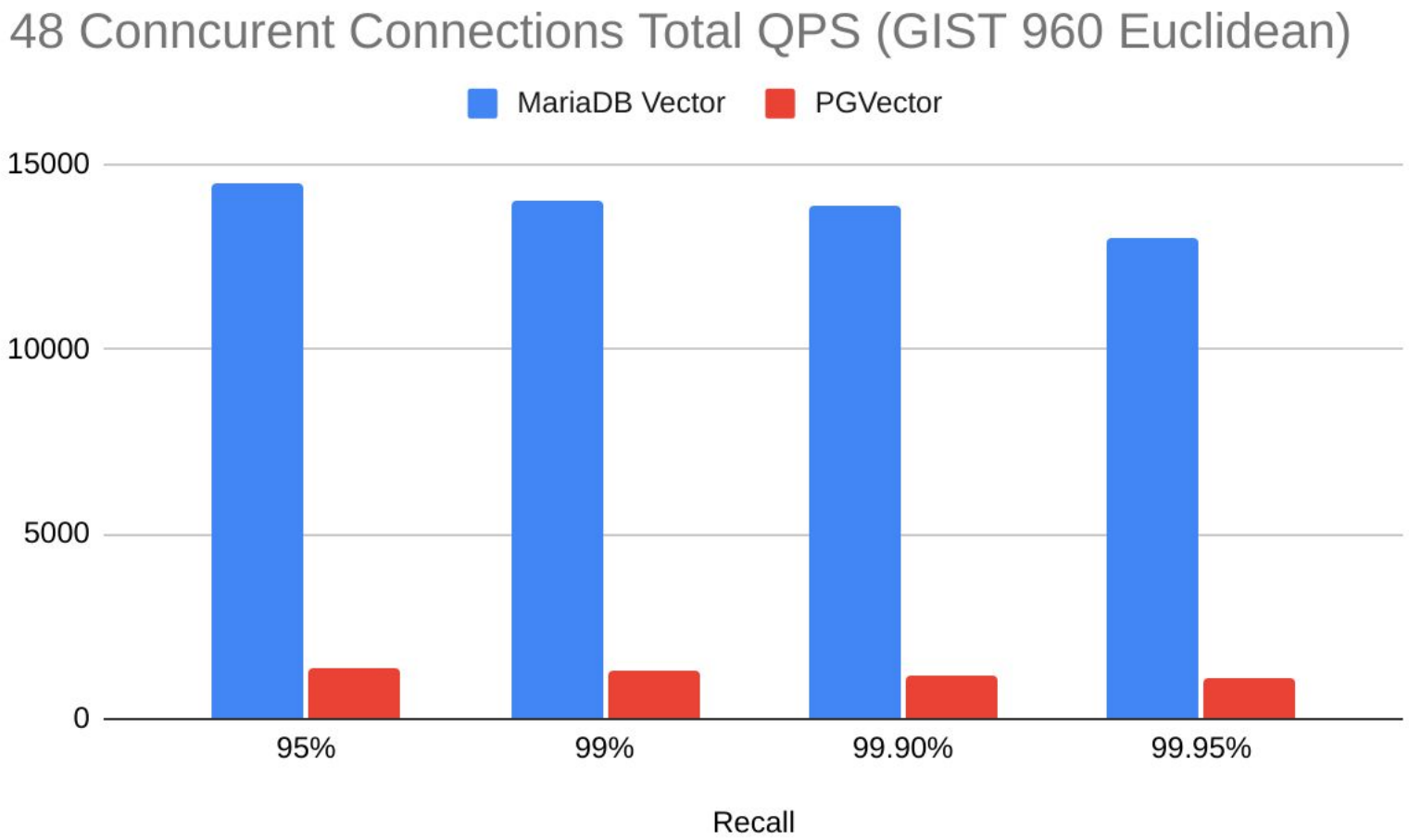
Download

Mirror

Bharat Datacenter - New Delhi



https://mariadb.com/es/resources/blog/how-fast-is-mariadb-vector/



https://mariadb.com/es/resources/blog/how-fast-is-mariadb-vector/

Index Construction

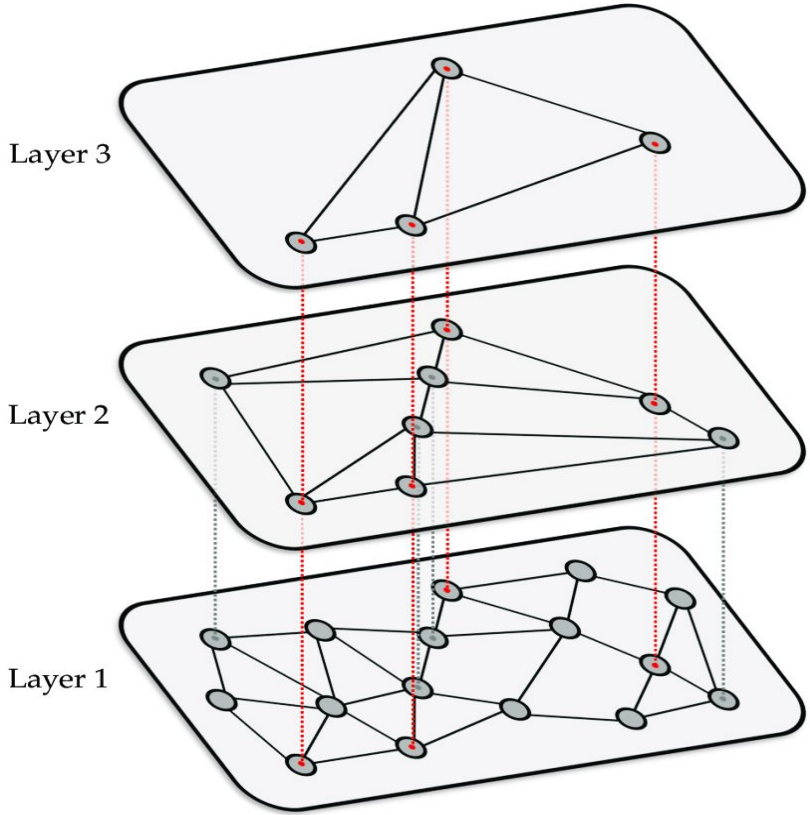
1. HNSW index is stored as a separate auxiliary table

   [**layer, tref,** vec, neighbors]

2. ACID benefits of the underlying storage engine.

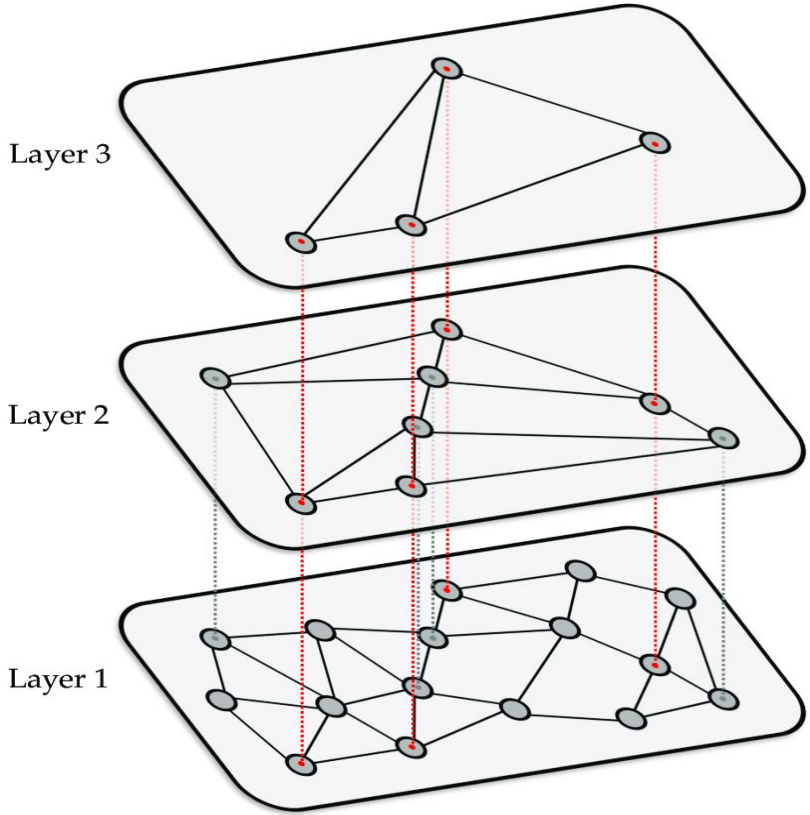
3. A bit more overhead than having it natively within the SE.

4. More flexible.



Index Construction

1. HNSW allows online construction.
2. HNSW does not have a native DELETE method.
3. Parameters that influence index quality / speed:
   a. **mhnsw_max_edges_per_node** (**5-8** is ok)



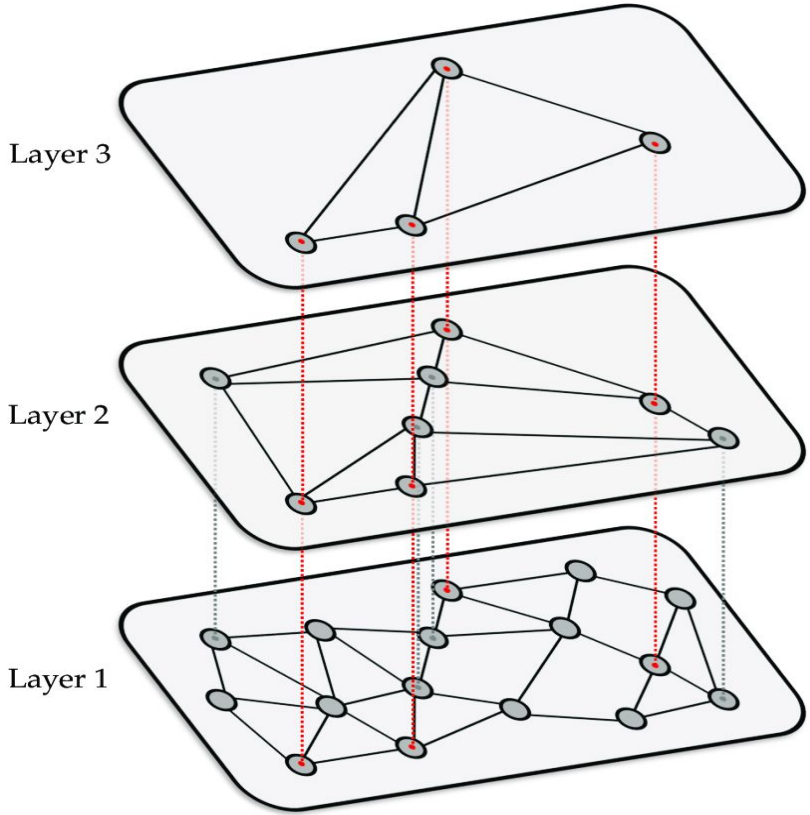
Index Lookup

1. Traverse the graph from upper layer to lower layer.

2. Parameters that influence results:

3. **mhnsw_ef_search** from HNSW paper

4. Higher values produce better recall.
   (percentage of results which are true minimums)

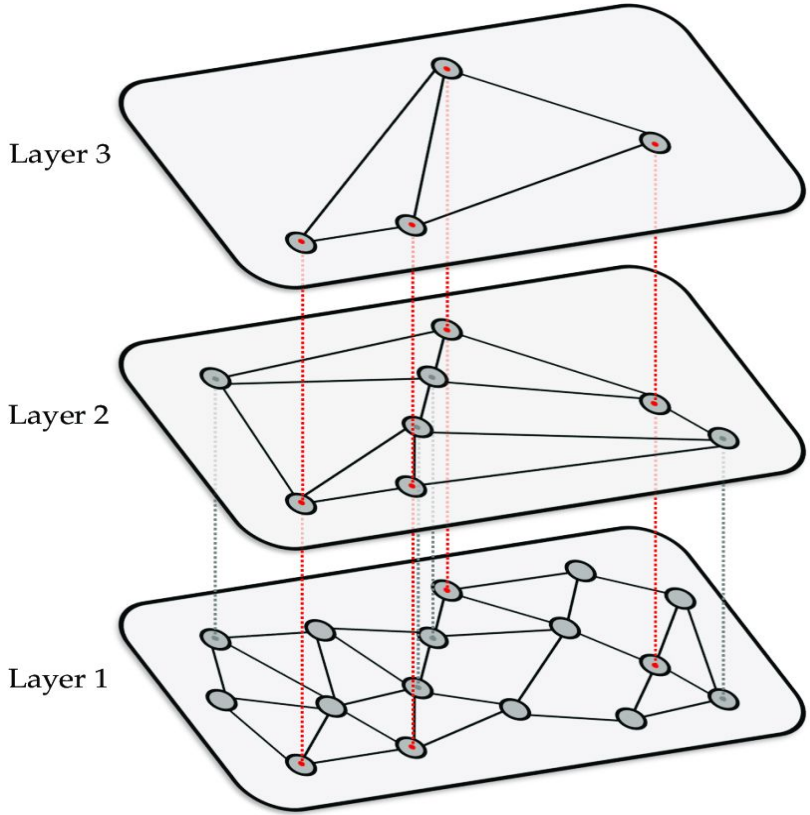


Index Lookup

1. MariaDB has a dedicated shared-statement cache to store the graph in memory.

   mhnsw_cache_size

   Ideally this should fit all your vector data for best performance.



Possible future directions?

1. Plugins to generate embedding on insert.
2. Storage Engine for Vector Embeddings generation (CONNECT SE can fulfill this to some degree already)
3. More vector indexing algorithms.
    a. IVFFlat is a Google Summer of Code project this year.
4. Performance optimizations - Index Condition pushdown

Thank you!

Contact details:

kaj@mariadb.org

vicentiu@mariadb.org

About:

https://mariadb.org/projects/mariadb-vector/

https://mariadb.org/kaj

https://mariadb.org/vicentiu