MariaDB Vector

Sergei Golubchik
Server Architect @ MariaDB plc

Vicențiu Ciorbaru
Chief Development Officer
@ MariaDB Foundation

FOSDEM 2024
MariaDB Fringe Event
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)

Text  →  Embedding AI Model  →  Output

Image  →  Embedding AI Model  →  Output

Video  →  Embedding AI Model  →  Output

[0.4, 0.2, …. 0.1]
[0.5, 0.1, …. 0.2]
[0.3, 0.2, …. 0.3]
What is a Vector Embedding?

These are points in a multi-dimensional space

[0.4, 0.2, …, 0.1]

[0.5, 0.1, …, 0.2]

[0.3, 0.2, …, 0.3]
2D example
2D example

P1
2D example
2D example
The document associated with P1 is more similar to P3 than P2, according to the model that generated the points!
Where Vector search comes into play?

Web Store

BackEnd
HTTP
Server

Embedding
AI Model
(OpenAI,
llama, etc.)

MariaDB

MariaDB Foundation
https://mariadb.org
Where Vector search comes into play?

1. User searches for product

Web Store → BackEnd HTTP Server

Embedding AI Model (OpenAI, llama, etc.)

MariaDB

[Diagram showing the flow of data from the web store to the back-end HTTP server, through an embedding AI model, and finally to MariaDB.]
Where Vector search comes into play?

1. User searches for product
   - Web Store
   - BackEnd HTTP Server

2. Generate Embedding from user query
   - Embedding AI Model
     - (OpenAI, llama, etc.)

   - MariaDB
Where Vector search comes into play?

1. User searches for product
2. Generate Embedding from user query
3. Return Embedding [0.5, 0.7, … 0.3]
Where Vector search comes into play?

1. User searches for product
2. Generate Embedding from user query
3. Return Embedding [0.5, 0.7, … 0.3]
4. Run SQL Query
Where Vector search comes into play?

1. User searches for product
2. Generate Embedding from user query
3. Return Embedding [0.5, 0.7, … 0.3]
4. Run SQL Query

SELECT p.name, p.description
FROM products p
ORDER BY VEC_DISTANCE([0.5, 0.7, ..., 0.3], p.embedding)
LIMIT 10
Where Vector search comes into play?

1. User searches for product
2. Generate Embedding from user query
3. Return Embedding [0.5, 0.7, …, 0.3]
4. Run SQL Query
5. Database returns results

SELECT p.name, p.description
FROM products p
ORDER BY
  VEC_DISTANCE([0.5, 0.7, ..., 0.3], p.embedding)
LIMIT 10
Where Vector search comes into play?

1. User searches for product
2. Generate Embedding from user query
3. Return Embedding
   \[ [0.5, 0.7, \ldots, 0.3] \]
4. Run SQL Query
   \[
   \text{SELECT p.name, p.description}
   \text{FROM products p}
   \text{ORDER BY}
   \text{VEC\_DISTANCE([0.5, 0.7, \ldots, 0.3], p.embedding)}
   \text{LIMIT 10}
   \]
5. Database returns results
6. User recommendation
Other applications?

- Q&A systems based on documentation (Sergei demo)
- **Augment GPT prompts**
  - GPTs can handle a lot of context in a prompt
  - Use vector search to find the most relevant documents for a prompt.
  - Augment the prompt with the content of the documents.

For example, one could program prompts to be sent like this:

```plaintext
Using only the following information: [document_from_db1], [document_from_db2], ...
Answer the following question: <User query>
```

MariaDB Foundation
https://mariadb.org
As a database user, what must you do?

1. Install a vector database (MariaDB Vector will come with MariaDB Server soon)

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.
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" neighbor.
3. Depending on dataset, some indexing strategies perform better than others.
4. Indexing generally requires a lot of memory.
   a. IVFFlat — Low resource usage, poor search quality, present in pgvector
   b. HNSW — Hierarchical Navigable Small Worlds
      i. de-facto industry standard.
         Will be implemented in MariaDB
      ii. Large memory usage.
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)


4. Performance optimizations - Index Condition pushdown
Demo
Thank you!

Contact details:

vicentiu@mariadb.org
serg@mariadb.com

About:

https://mariadb.org/serg
https://mariadb.org/vicentiu