Albert0i / mariadb-vss-chinese

Code Issues Pull requests Actions Projects Wiki Security

main mariadb-vss-chinese / README.md

Albert0i write more 7e64bef · 1 minute ago

269 lines (204 loc) · 10.7 KB

Preview Code Blame Raw

Vector Semantic Search in Chinese using MariaDB

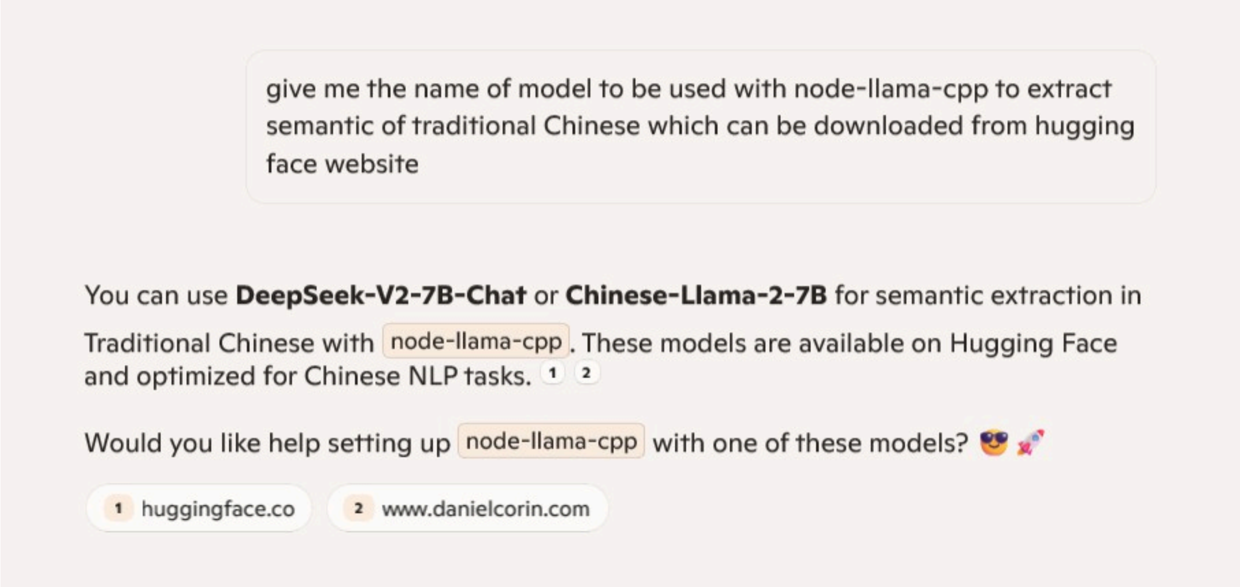
> "the ridiculous confusion which, in some circumstances, can determine the course of a man's life."
> The Castle by Franz Kafka

Prologue

- Vector Semantic Search using MariaDB (Part 1/3)
- Vector Semantic Search using MariaDB (Part 2/3)
- Vector Semantic Search using MariaDB (Part 3/3)

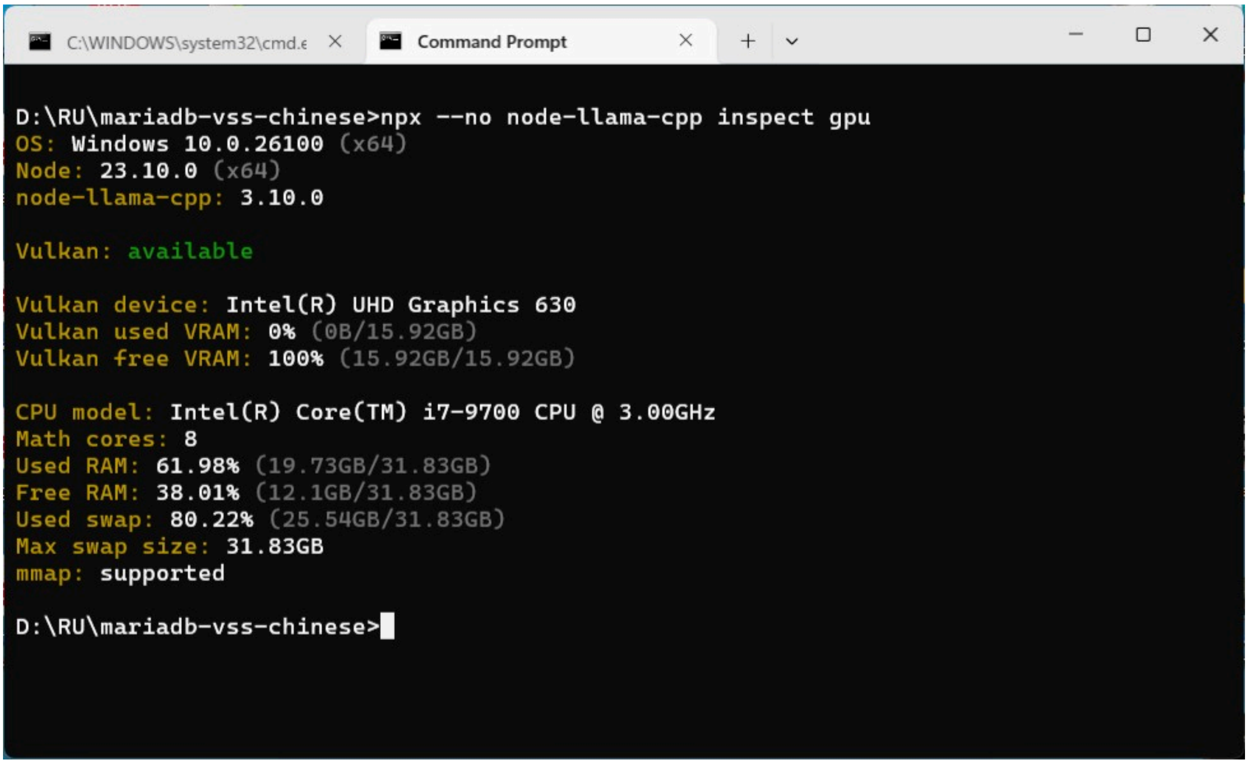
I. Choosing a Model

Previously we used VSS on English data. *Every tool would render useless if it can not work on one's native language*. We need to do the same thing on Chinese data, typically traditional Chinese. To be able to extract semantic, we need a different model as it is suggested by AI.



I head into [Hugging Face](https://huggingface.co) website and search for [Chinese-Llama-2-7B]() model and inspect my computer with:

```
npx --no node-llama-cpp inspect gpu
```

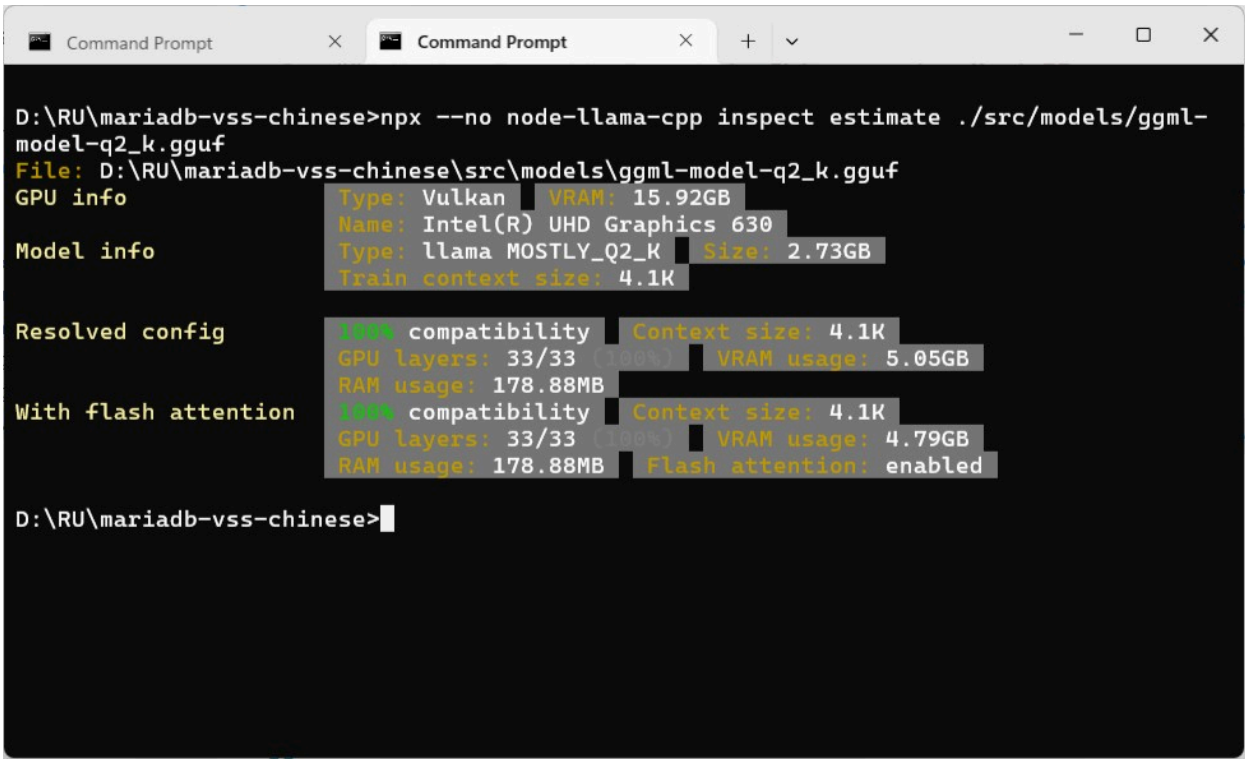


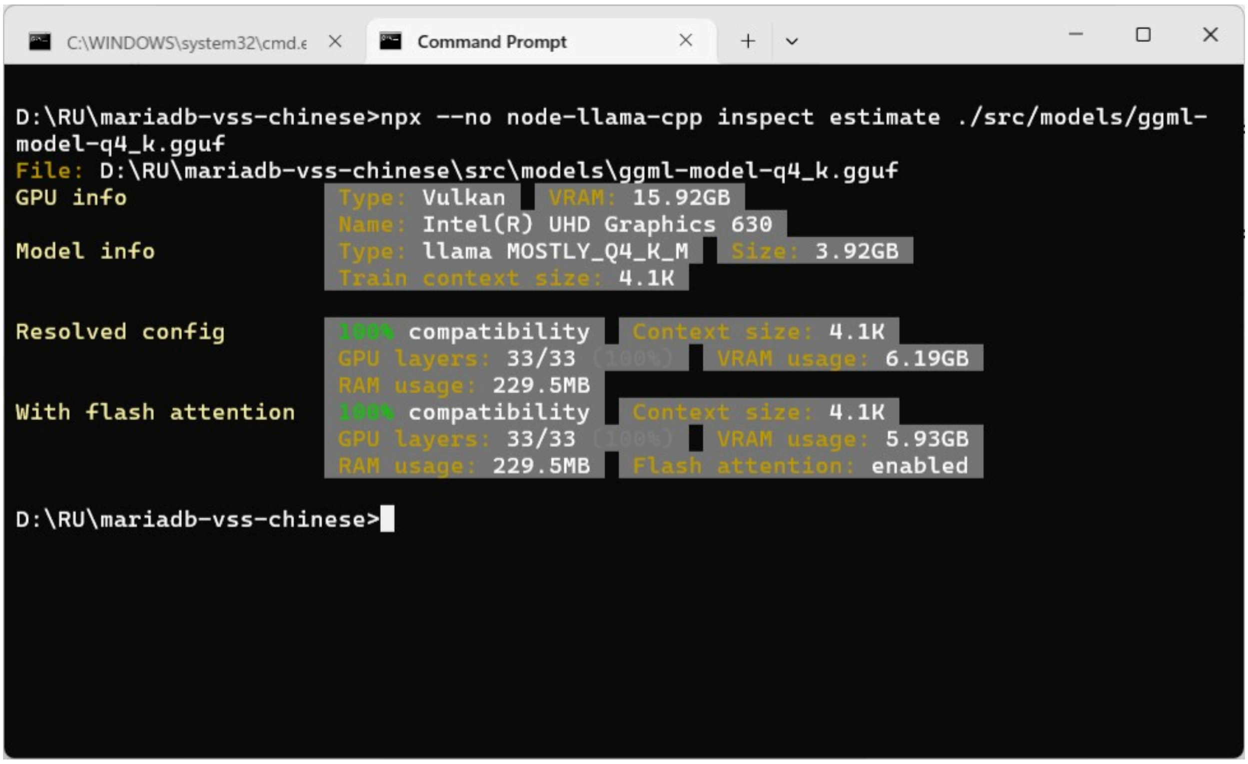
> If the machine you plan to run this model on doesn't have a GPU, you'd probably want to use a small model that can run on a CPU with decent performance.

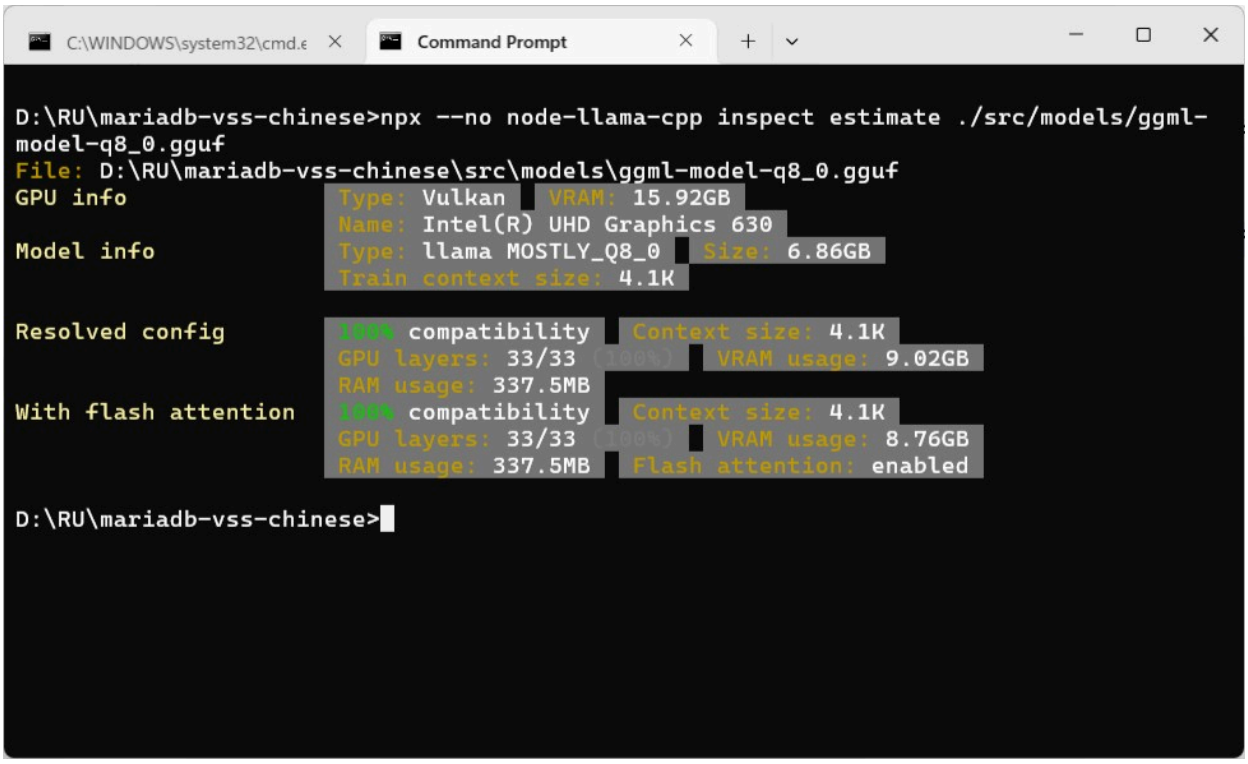
> If you have a GPU, the amount of VRAM you have will determine the size of the model you can run. Ideally, you'd want to fit the entire model in the VRAM to use only the GPU and achieve maximum performance. If the model requires more memory than the available VRAM, parts of it will be offloaded to the RAM and be evaluated using the CPU, significantly reducing the efficiency and speed of inference.

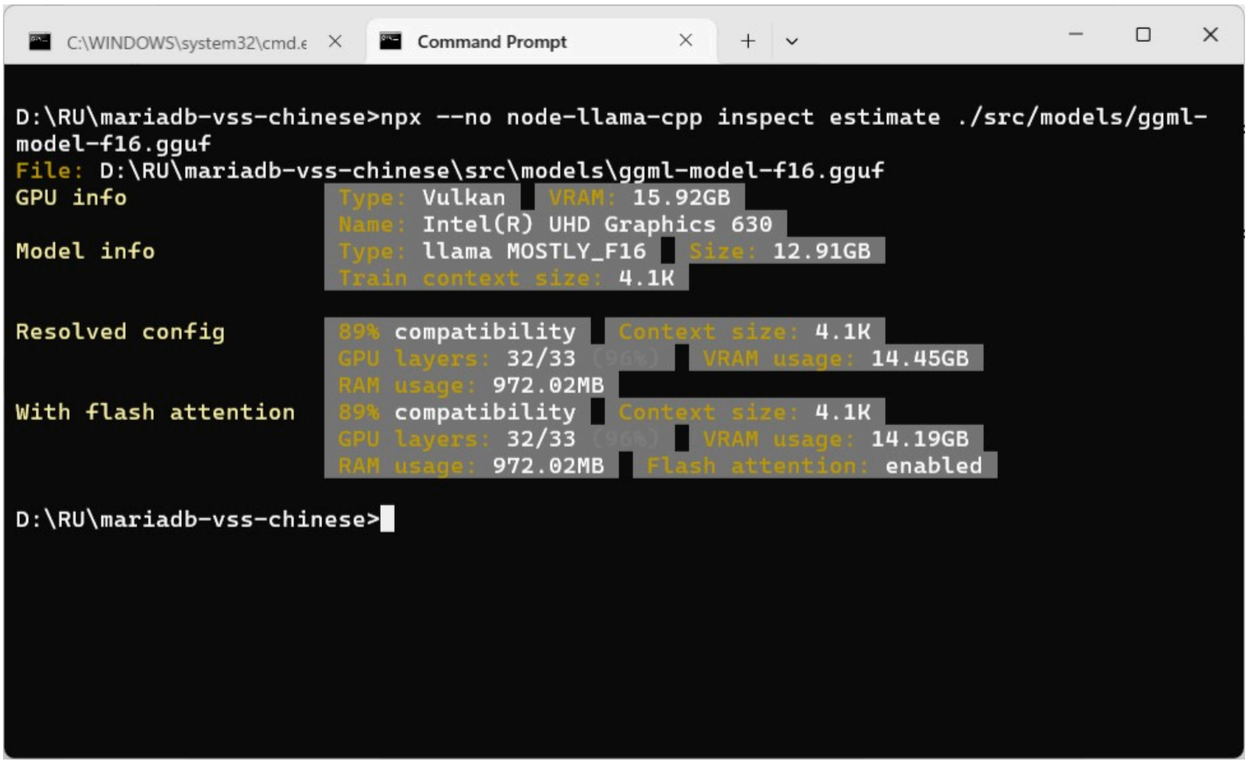
To get a more accurate estimation of how well a model will run:

```
npx --no node-llama-cpp inspect estimate ./src/models/ggml-model-q2_k.gguf
```




C:\WINDOWS\system32\cmd.e
Command Prompt
D:\RU\mariadb-vss-chinese>npx --no node-llama-cpp inspect estimate ./src/models/ggml-model-q4_k.gguf
File: D:\RU\mariadb-vss-chinese\src\models\ggml-model-q4_k.gguf
GPU info Type: Vulkan VRAM: 15.92GB
Name: Intel(R) UHD Graphics 630
Model info Type: llama MOSTLY_Q4_K_M Size: 3.92GB
Train context size: 4.1K
Resolved config 100% compatibility Context size: 4.1K
GPU layers: 33/33 (100%) VRAM usage: 6.19GB
RAM usage: 229.5MB
With flash attention 100% compatibility Context size: 4.1K
GPU layers: 33/33 (100%) VRAM usage: 5.93GB
RAM usage: 229.5MB Flash attention: enabled
D:\RU\mariadb-vss-chinese>


C:\WINDOWS\system32\cmd.e
Command Prompt
D:\RU\mariadb-vss-chinese>npx --no node-llama-cpp inspect estimate ./src/models/ggml-model-q8_0.gguf
File: D:\RU\mariadb-vss-chinese\src\models\ggml-model-q8_0.gguf
GPU info Type: Vulkan VRAM: 15.92GB
Name: Intel(R) UHD Graphics 630
Model info Type: llama MOSTLY_Q8_0 Size: 6.86GB
Train context size: 4.1K
Resolved config 100% compatibility Context size: 4.1K
GPU layers: 33/33 (100%) VRAM usage: 9.02GB
RAM usage: 337.5MB
With flash attention 100% compatibility Context size: 4.1K
GPU layers: 33/33 (100%) VRAM usage: 8.76GB
RAM usage: 337.5MB Flash attention: enabled
D:\RU\mariadb-vss-chinese>



Suggestion from AI

> Your GPU **can handle models up to around 15GB VRAM usage**, but for **efficient performance**, models should **stay below 80% of total VRAM** (~12GB). If you want **bigger models**, you may need a GPU with more memory or offload layers to **CPU RAM (slower but possible).**

1. ggml-model-q2_k.gguf (2.73G)
2. ggml-model-q4_k.gguf (3.92G)
3. ggml-model-q8_0.gguf (6.85G)
4. ggml-model-f16.gguf (12.9G)

To strike a balance between speed and accuracy, I opt `ggml-model-q4_k.gguf` (3.92G) instead of `ggml-model-q8_0.gguf` (6.85G). A short comparison is in here. Both of them generate vector of 4096 dimensions.

I have the AI generated 1,200 sentences in traditional Chinese and put it in an array for sake of simplicity. A more realistic scenario would load the data from Oracle or by parsing a CSV file, for example.

It is *crucial* to choose the right model in the first place, change of model involves re-creating all vector embeddings may take hours or even days.

II. Using Embedding

First of all, create a table in target database:

```
USE vss;

CREATE TABLE documents (
  id          INT AUTO_INCREMENT PRIMARY KEY,

  textChi     VARCHAR(512) NOT NULL,
  visited     INT DEFAULT 0,
  embedding   VECTOR(4096) NOT NULL,

  createdAt   TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt   TIMESTAMP,
  updateIdent INT DEFAULT 0,

  UNIQUE (textChi),
  VECTOR INDEX (embedding) M=16 DISTANCE=cosine
);
```

Note:

1. Number of dimensions is determined by model output;
2. Only one vector index per table and can not be NULL;
3. `M` — Larger values mean slower SELECTs and INSERTs, larger index size and higher memory consumption but more accurate results. The valid range is from `3` to `200`.
4. `DISTANCE` — Distance function to build the vector index for. Searches using a different distance function will not be able to use a vector index. Valid values are `cosine` and `euclidean` (the default).

Next, install Prisma, pull in model and generate client code.

```
model documents {
  id          Int                           @id @default(autoincrement())
  textChi     String                        @unique(map: "textChi")
@db.VarChar(512)
  visited     Int?                          @default(0)
  embedding   Unsupported("vector(4096)")
  createdAt   DateTime?                     @default(now())
@db.Timestamp(0)
  updatedAt   DateTime?                     @db.Timestamp(0)
  updateIdent Int?                          @default(0)

  @@index([embedding], map: "embedding")
}
```

The template to generate vector embedding is from here:

```
import {fileURLToPath} from "url";
import path from "path";
import {getLlama, LlamaEmbedding} from "node-llama-cpp";

const __dirname = path.dirname(
    fileURLToPath(import.meta.url)
);

const llama = await getLlama();
const model = await llama.loadModel({
    modelPath: path.join(__dirname, "bge-small-en-v1.5-q8_0.gguf")
});
const context = await model.createEmbeddingContext();

async function embedDocuments(documents: readonly string[]) {
    const embeddings = new Map<string, LlamaEmbedding>();

    await Promise.all(
        documents.map(async (document) => {
            const embedding = await context.getEmbeddingFor(document);
            embeddings.set(document, embedding);

            console.debug(
                `${embeddings.size}/${documents.length} documents
embedded`
            );
        })
    );

    return embeddings;
}

function findSimilarDocuments(
    embedding: LlamaEmbedding,
    documentEmbeddings: Map<string, LlamaEmbedding>
) {
    const similarities = new Map<string, number>();
    for (const [otherDocument, otherDocumentEmbedding] of
documentEmbeddings)
        similarities.set(
            otherDocument,
            embedding.calculateCosineSimilarity(otherDocumentEmbedding)
        );

    return Array.from(similarities.keys())
        .sort((a, b) => similarities.get(b)! - similarities.get(a)!);
}

const documentEmbeddings = await embedDocuments([
    "The sky is clear and blue today",
    "I love eating pizza with extra cheese",
    "Dogs love to play fetch with their owners",
```

```
    "The capital of France is Paris",
    "Drinking water is important for staying hydrated",
    "Mount Everest is the tallest mountain in the world",
    "A warm cup of tea is perfect for a cold winter day",
    "Painting is a form of creative expression",
    "Not all the things that shine are made of gold",
    "Cleaning the house is a good way to keep it tidy"
]);


const query = "What is the tallest mountain on Earth?";
const queryEmbedding = await context.getEmbeddingFor(query);

const similarDocuments = findSimilarDocuments(
    queryEmbedding,
    documentEmbeddings
);
const topSimilarDocument = similarDocuments[0];

console.log("query:", query);
console.log("Document:", topSimilarDocument);
```

Which is a good start!

III. Seeding

A cheap trick is used to implement `UPSERT` in MariaDB, which is described in [INSERT ON DUPLICATE KEY UPDATE](). An `UPSERT` prevents from inserting duplicated entry and let's keep track of the duplication.

```
export async function addDocument(document) {
    const { vector } = await
context.getEmbeddingFor(removeWords(document));

    // Add new document
    return await prisma.$executeRaw`
                    INSERT INTO documents (textChi, embedding)
                    VALUES( ${document},
VEC_FromText(${JSON.stringify(vector)}) )
                    ON DUPLICATE KEY
                    UPDATE updateIdent = updateIdent + 1;
                `;
}
```

Run command to seed database:

```
npx prisma db seed
```

```
C:\WINDOWS\system32\cmd.

D:\RU\mariadb-vss-chinese>npx prisma db seed
Environment variables loaded from .env
Running seed command `node prisma/seed.js` ...
[node-llama-cpp] load: special_eos_id is not in special_eog_ids - the tokenizer confi
g may be incorrect
Document 1: 今天的天空晴朗且蔚藍
Document 2: 我喜歡吃加了額外起司的披薩
Document 3: 狗狗喜歡和主人玩接球遊戲
Document 4: 法國的首都是巴黎
Document 5: 喝水對保持身體水分很重要
Document 6: 聖母峰是世界上最高的山
Document 7: 寒冷的冬天來一杯溫暖的茶最合適
Document 8: 繪畫是一種創意表達的方式
Document 9: 並非所有閃亮的東西都是黃金製成
Document 10: 打掃房子是保持整潔的好方法
Document 11: 早晨的陽光透過窗戶照進房間
Document 12: 雨後的空氣格外清新
Document 13: 夜晚的星空閃爍著微光
Document 14: 我喜歡在春天欣賞盛開的櫻花
Document 15: 閱讀一本好書能讓人心靈沉靜
Document 16: 貓咪喜歡窩在陽光下打盹
Document 17: 音樂能夠療癒人的心靈
Document 18: 旅行是探索世界的美好方式
Document 19: 咖啡的香氣讓人感覺放鬆
```

Depending on the machine, it may take hours or even days... AFter that verify if there is duplicated entry with:

```
SELECT * FROM documents WHERE updateIdent <> 0;
```

IV. Finding the documetns

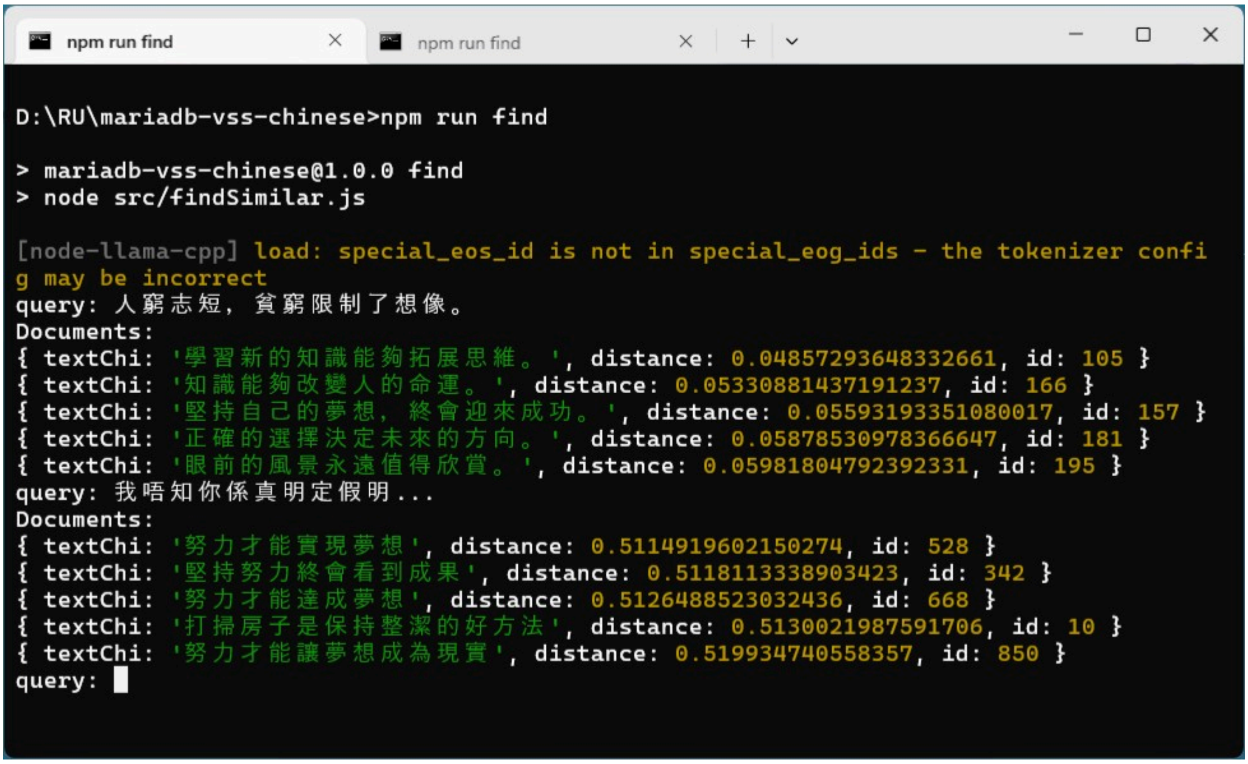
Most of the code remains the same, we only add update to `visited` field so that we can check to see search results.

```
export async function findSimilarDocuments(document, limit = 3) {
    const { vector } = await
context.getEmbeddingFor(removeWords(document));

    // Find similar documents
    const docs = await prisma.$queryRaw`
                          SELECT textChi,
                                 VEC_DISTANCE_COSINE(
                                    embedding,

VEC_FromText(${JSON.stringify(vector)})
                                    ) AS distance,
                                 id
                          FROM documents
                          ORDER BY 2 ASC
                          LIMIT ${limit} OFFSET 0;
                        `;

    // Update `visited` field
```

```
    const promises = [];    // Collect promises
    docs.forEach(doc => {
            promises.push(prisma.$executeRaw`
                            UPDATE documents
                            SET visited = visited + 1,
                                updatedAt = Now(),
                                updateIdent = updateIdent + 1
                            WHERE id=${doc.id}
                            `
            )
        })
    await Promise.all(promises); // Resolve all at once

    return docs
}
```

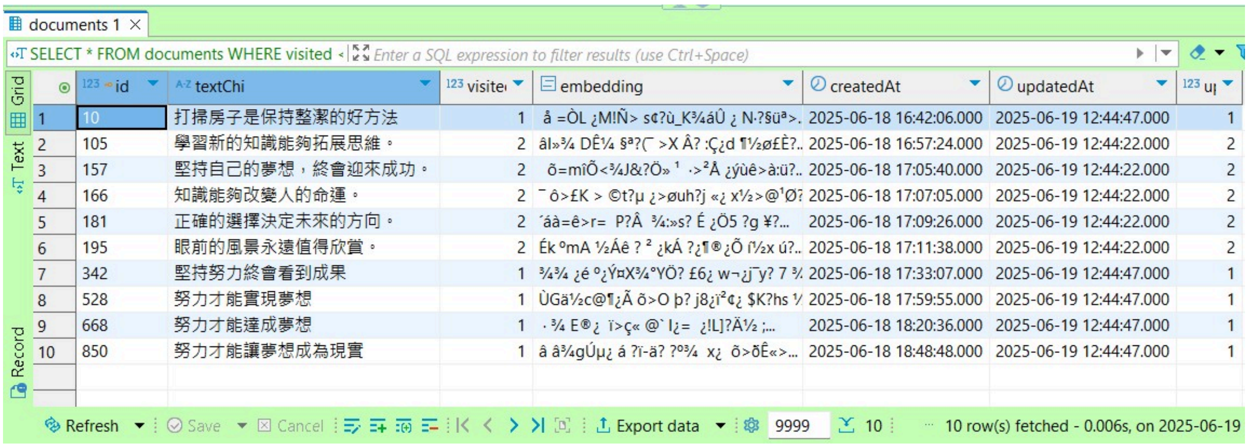
Run command to find the documents:

```
npm run find
```



Verify search results with:

```
SELECT * FROM documents WHERE visited <> 0;
```



V. Bibliography

1. Choosing a Model
2. Using Embedding
3. Raw queries
4. CRUD
5. The Castle by Franz Kafka

Epilogue

> "I am certainly ignorant, but facts are facts, which is very sad for me but also advantageous, since an ignorant man will dare to do more, so I will happily go about in my ignorance with what I am sure are its unfortunate consequences for a little longer, as long as my strength allows. "
> The Castle by Franz Kafka

EOF (2025/06/20)