

Ứng dụng cơ sở dữ liệu vector Milvus cho hệ thống hỏi-đáp mở

Dinh-Thang Duong và Quang-Vinh Dinh

Ngày 20 tháng 2 năm 2024

Milvus là một trong những hệ cơ sở dữ liệu vector (vector database) mã nguồn mở, chuyên dùng cho việc lưu trữ các vector embedding và tìm kiếm tương đồng (similarity search) giữa các vector với nhau. Từ đó, trở thành một công cụ cực kỳ mạnh mẽ hỗ trợ cho các ứng dụng AI trong việc truy cập vào nguồn kiến thức từ các cơ sở dữ liệu nội bộ, qua đó cải thiện độ chính xác một cách đáng kể. Các bạn có thể theo dõi và đọc thêm về thư viện này tại [trang chủ](#) hoặc trang [github](#) của thư viện.



Hình 1: Biểu tượng của Milvus

Trong bài viết này, chúng ta sẽ tìm hiểu cách cài đặt nhanh Milvus trên máy tính cá nhân và ứng dụng Milvus trong việc tìm kiếm các tài liệu (context) có liên quan nhằm hỗ trợ hệ thống hỏi-đáp mở (Open Domain Question Answering).

1. **Cài đặt Milvus:** Trong phần này, chúng ta sẽ tìm hiểu cách cài đặt Milvus thông qua Docker. Đây là cách đơn giản và nhanh chóng để cài đặt thư viện này. Phần demo được thực hiện trên hệ điều hành MacOS, vì vậy có một số bước thực hiện sẽ khác so với các hệ điều hành còn lại. Bạn đọc hãy thay đổi cho phù hợp với hệ điều hành của máy mình nhé. Đầu tiên, các bạn hãy kiểm tra cấu hình yêu cầu để cài đặt Milvus theo như ảnh dưới đây:

Component	Requirement	Recommendation	Note
CPU	<ul style="list-style-type: none"> Intel CPU Sandy Bridge or later Apple M1 CPU 	<ul style="list-style-type: none"> standalone: 8 core or more cluster: 16 core or more 	Current version of Milvus does not support AMD CPUs.
CPU instruction set	<ul style="list-style-type: none"> SSE4.2 AVX AVX2 AVX-512 	<ul style="list-style-type: none"> SSE4.2 AVX AVX2 AVX-512 	Vector similarity search and index building within Milvus require CPU's support of single instruction, multiple data (SIMD) extension sets. Ensure that the CPU supports at least one of the SIMD extensions listed. See CPUs with AVX for more information.
RAM	<ul style="list-style-type: none"> standalone: 16G cluster: 64G 	<ul style="list-style-type: none"> standalone: 32G cluster: 128G 	The size of RAM depends on the data volume.
Hard drive	SATA 3.0 SSD or higher	NVMe SSD or higher	The size of hard drive depends on the data volume.

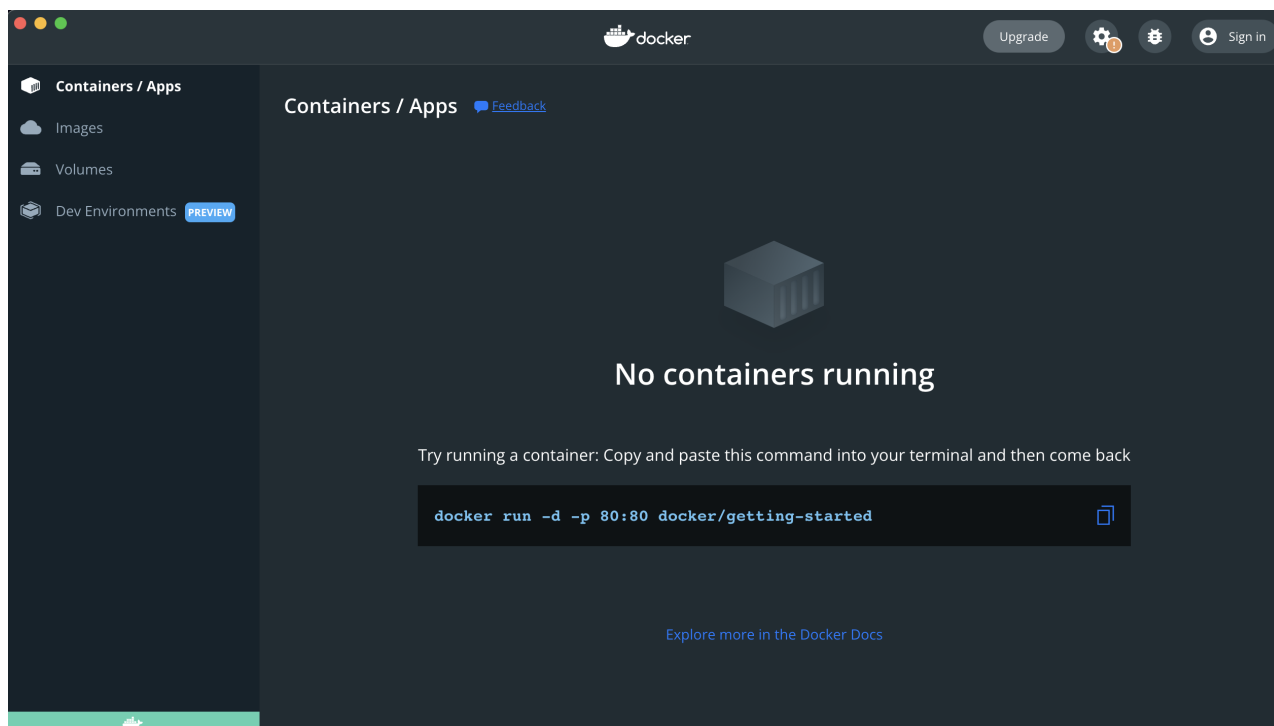
Hình 2: Yêu cầu cấu hình về phần cứng cho Milvus

Tiếp theo, chúng ta đến với phần cài đặt Milvus. Như đã đề cập ở trên, chúng ta sẽ cài đặt thông qua Docker. Vì vậy, chúng ta sẽ tiến hành cài đặt Docker và Docker compose tại bước này. Các bạn hãy lên trang chủ của Docker và thực hiện theo hướng dẫn cài đặt theo đúng hệ điều hành của máy mình tại [đây](#):



Hình 3: Các lựa chọn cài đặt Docker cho từng hệ điều hành riêng biệt

Khi quá trình cài đặt Docker hoàn tất, chúng ta sẽ kiểm tra Docker đã sẵn sàng để sử dụng hay chưa. Đối với MacOS/Windows, ta mở ứng dụng Docker Desktop để khởi động Docker:



Hình 4: Giao diện Docker Desktop trên hệ điều hành MacOS

Sau đó, các bạn mở Terminal/CMD lên và chạy lần lượt các dòng lệnh sau:

```
1 $ docker -v
2 $ docker-compose -v
```

```
[(base) thangduong@Duongs-MacBook-Pro ~ % docker -v
Docker version 20.10.14, build a224086
[(base) thangduong@Duongs-MacBook-Pro ~ % docker-compose -v
docker-compose version 1.29.2, build 5becea4c
(base) thangduong@Duongs-MacBook-Pro ~ %
```

Hình 5: Kết quả kiểm tra phiên bản của Docker và Docker Compose

Nếu không có lỗi gì xảy ra khi chạy 2 dòng lệnh trên, chúng ta coi như đã cài đặt thành công Docker. Từ đây, ta tiến hành cài đặt Milvus, các bạn hãy chạy các dòng lệnh sau:

(a) Tải file script chứa các lệnh sử dụng Milvus:

```
1 $ wget https://raw.githubusercontent.com/milvus-io/milvus/master/scripts/standalone_embed.sh
```

Khi chạy xong lệnh này, tại vị trí chạy lệnh, các bạn sẽ thấy file `standalone_embed.sh`. Các bạn cần lưu ý vị trí tải file này, vì chúng ta cần phải ở đúng vị trí tải file hoặc thay đổi đường dẫn hợp lý thì mới chạy lệnh gọi file này được.

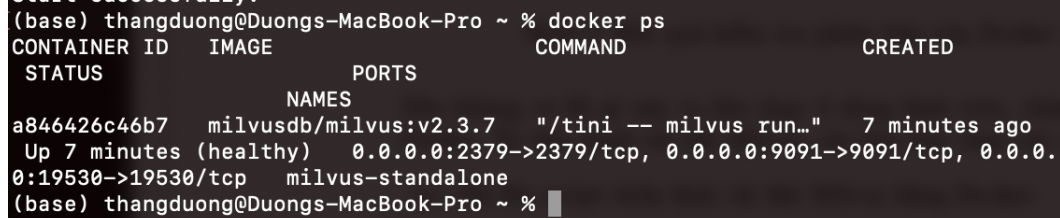
(b) Chạy file script:

```
1 $ bash standalone_embed.sh start
```

Lưu ý rằng, lệnh này sẽ mất một khoảng thời gian để hoàn tất tùy vào tốc độ mạng.

(c) Kiểm tra cài đặt: Khi đã tải và triển khai xong, các bạn có thể kiểm tra bằng lệnh sau:

```
1 $ docker ps
```



```
(base) thangduong@Duongs-MacBook-Pro ~ % docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED
STATUS        NAMES
a846426c46b7   milvusdb/milvus:v2.3.7             "/tiny -- milvus run..." 7 minutes ago
Up 7 minutes (healthy)   0.0.0.0:2379->2379/tcp, 0.0.0.0:9091->9091/tcp, 0.0.0.0:19530->19530/tcp   milvus-standalone
(base) thangduong@Duongs-MacBook-Pro ~ %
```

Hình 6: Kết quả kiểm tra cài đặt Milvus trên Terminal

Như vậy, chúng ta đã hoàn tất cài đặt và triển khai Milvus.

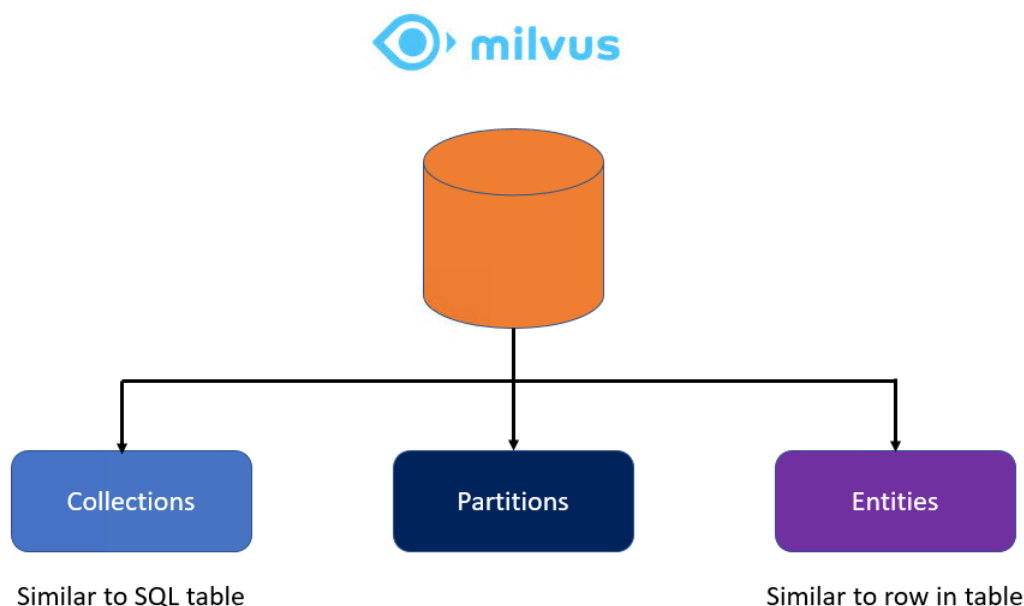
2. **Cài đặt các thư viện Python cần thiết:** Để tương tác được với Milvus trong môi trường Python, chúng ta cần tải một vài các thư viện được liệt kê ở phía dưới đây. Để thuận tiện trong việc cài đặt, các bạn hãy copy danh sách thư viện này vào trong một file tên là `requirements.txt`:

```
1 # > requirements.txt
2 pandas
3 transformers
4 torch
5 datasets
6 milvus-cli==0.4.2
7 protobuf==3.20.0
8 pymilvus==2.3.4
```

Sau đó, ta gọi lệnh pip để cài đặt, ở đây mình sẽ cài đặt trên môi trường conda:

```
1 $ conda create -n milvus_env -y
2 $ conda activate milvus_env
3 $ pip3 install --upgrade pip
4 $ pip3 install -r requirements.txt
```

3. **Kiểm tra hoạt động của Milvus:** Chúng ta sẽ thử tương tác với Milvus trong Python thông qua thư viện `pymilvus`. Milvus có sử dụng một số từ khóa mới, song các bạn có thể nắm cơ bản rằng chúng ta sẽ có các **Collection**, một dạng bảng dữ liệu của Milvus. Như vậy, để lưu trữ một vector database trong Milvus, chúng ta sẽ cần tạo một Collection, từ đó kết nối và tương tác với Collection này:



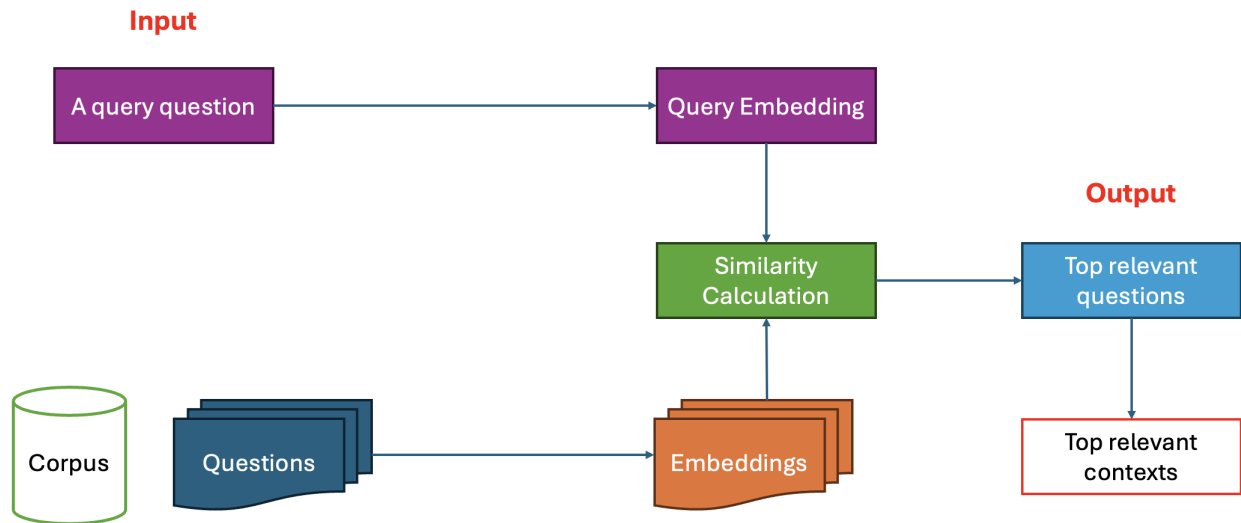
Hình 7: Một vài thành phần trong Milvus. Nguồn: [link](#)

Các bạn có thể tìm hiểu những khái niệm khác trong Milvus tại [đây](#). Bây giờ, chúng ta sẽ thử kiểm tra danh sách các Collection có trong Milvus hiện tại. Các bạn tạo một file .py bất kì, ở đây mình tạm đặt tên là `check_milvus.py`, có nội dung như sau:

```
1 from pymilvus import connections, utility
2
3 connections.connect('default', host='localhost', port='19530')
4
5 print(utility.list_collections())
6 # Output: []
```

Chương trình trên sử dụng phương thức `connections.connect()` để kết nối tới Milvus Standalone mà chúng ta đã host ở bước đầu tiên. Sau đó, sử dụng `utility.list_collections()` để kiểm tra danh sách các Collections hiện có, kết quả trả về là một list rỗng cho thấy chúng ta đang chưa có một bảng dữ liệu nào trên kho lưu trữ. Phần tiếp theo chúng ta sẽ tiến hành xây dựng một vector database cho bài QA.

4. **Xây dựng vector database cho bộ dữ liệu QA:** Chúng ta sẽ xây dựng một vector database trên bộ dữ liệu QA là SQuAD. Mục tiêu của chúng ta khi sử dụng cơ sở dữ liệu này nhằm tìm kiếm các câu hỏi có liên quan đến câu hỏi input, từ đó tìm được các context có khả năng cao chứa đáp án cho câu hỏi.



Hình 8: Pipeline của hệ thống End-to-end QA trong bài

Các bạn tạo một file code .py mới (ở đây mình sẽ tạo file `build_database.py`) và thực hiện các bước sau đây:

(a) Import các thư viện cần thiết:

```

1 from pymilvus import (
2     connections,
3     utility,
4     FieldSchema,
5     CollectionSchema,
6     DataType,
7     Collection,
8 )
9
10 from datasets import load_dataset, Dataset
11 from transformers import AutoTokenizer, AutoModel
12 from torch import clamp, sum

```

(b) Khai báo các hyperparameters sẽ dùng trong code:

```

1 DATASET_NAME = 'squad_v2' # Huggingface Dataset to use
2 MODEL_NAME = 'distilbert-base-uncased' # Transformer to use for embeddings
3 TOKENIZATION_BATCH_SIZE = 1000 # Batch size for tokenizing operation
4 INFERENCE_BATCH_SIZE = 64 # batch size for transformer
5 INSERT_RATIO = 0.001 # How many samples to embed and insert
6 COLLECTION_NAME = 'huggingface_squad_db' # Collection name
7 DIMENSION = 768 # Embeddings size
8 LIMIT = 3 # How many results to search for
9 MILVUS_HOST = "localhost"
10 MILVUS_PORT = "19530"
11 REPLICAS_NUMBER = 1

```

Một vài tham số các bạn cần quan tâm:

- **INFERENCE_BATCH_SIZE:** Số lượng mẫu dữ liệu đưa vào mô hình BERT để lấy vector embedding, các bạn hãy điều chỉnh nhỏ hơn nếu không đủ GPU hoặc cao hơn trong trường hợp ngược lại.

- **INSERT_RATIO:** Kích thước bộ dữ liệu để đưa vào database. Ở đây mình chỉnh tỉ lệ rất thấp để việc demo trở nên nhanh hơn. Các bạn muốn test nhiều hơn có thể tăng tỉ lệ này lên.
- **LIMIT:** Số lượng kết quả truy vấn trả về từ Milvus. Các bạn muốn tăng số lượng tài liệu trả về có thể tăng tham số này lên.

(c) **Xây dựng hàm tạo Collection:** Ta dùng hàm này để tạo một bảng dữ liệu (Collection), lưu ý rằng kết quả của hàm sẽ là một Collection có đầy đủ các trường thông tin (các cột) nhưng chưa có dữ liệu (records):

```

1 def create_collection(collection_name, dim):
2     if utility.has_collection(collection_name):
3         utility.drop_collection(collection_name)
4
5     fields = [
6         FieldSchema(name='id', dtype=DataType.INT64, is_primary=True, auto_id=True),
7         FieldSchema(name='title', dtype=DataType.VARCHAR, max_length=1000),
8         FieldSchema(name='question', dtype=DataType.VARCHAR, max_length=1000)
9     ],
10    FieldSchema(name='context', dtype=DataType.VARCHAR, max_length=10000)
11    ],
12    FieldSchema(name='answer', dtype=DataType.VARCHAR, max_length=1000),
13    FieldSchema(name='question_embedding', dtype=DataType.FLOAT_VECTOR,
14                dim=dim)
15    ]
16    schema = CollectionSchema(fields=fields, description='question search')
17    collection = Collection(name=collection_name, schema=schema)
18
19    # create IVF_FLAT index for collection.
20    index_params = {
21        'metric_type': 'L2',
22        'index_type': "IVF_FLAT",
23        'params': {"nlist": 2048}
24    }
25    collection.create_index(field_name="question_embedding", index_params=index_params)
26
27    return collection

```

(d) **Xây dựng hàm tokenization:**

```

1 tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
2 def tokenize_question(batch):
3     results = tokenizer(
4         batch['question'],
5         add_special_tokens=True,
6         truncation=True,
7         padding="max_length",
8         return_attention_mask=True,
9         return_tensors="pt"
10    )
11
12    batch['input_ids'] = results['input_ids']
13    batch['attention_mask'] = results['attention_mask']
14
15    return batch

```

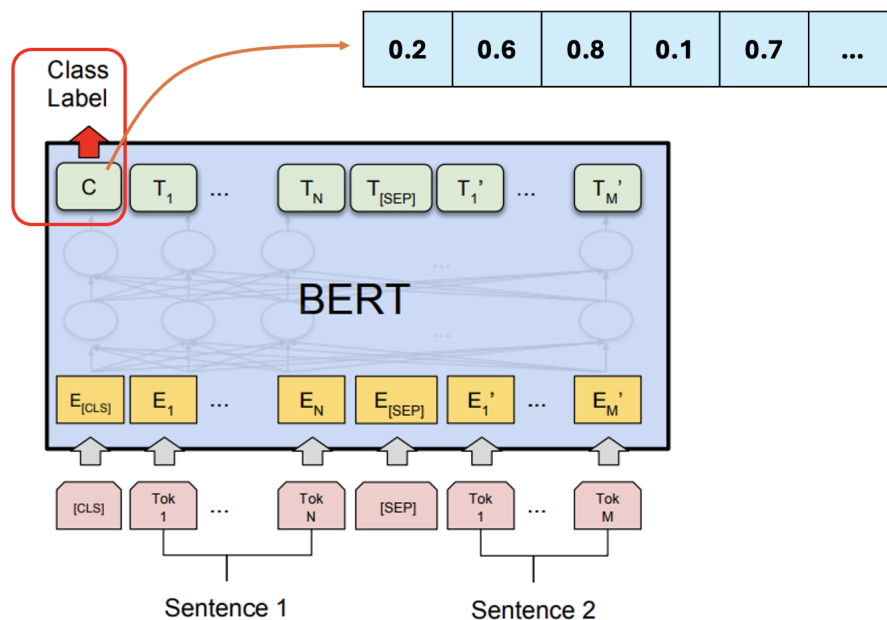
(e) **Xây dựng hàm get embedding:** Ta cần xây dựng hàm đổi từ text sang dạng vector embedding của nó. Tương tự như trong bài học chính, ở đây ta cũng sử dụng model BERT

và lấy final hidden state của token [CLS] để làm vector embedding:

```

1 model = AutoModel.from_pretrained(MODEL_NAME)
2 def quest_embedding(batch):
3     sentence_embs = model(
4         input_ids=batch['input_ids'],
5         attention_mask=batch['attention_mask']
6     )
7     batch['question_embedding'] = sentence_embs.last_hidden_state[:, 0]
8
9     return batch

```



Hình 9: Final hidden state của token [CLS] trong BERT

- (f) **Xây dựng hàm cập nhật dữ liệu SQuADv2 vào Collection:** Hàm này sẽ tải về bộ dữ liệu SQuADv2 gốc và thực hiện đưa từng sample vào Collection:

```

1 def create_squad_database(qa_collection):
2     squad_v2_dataset = load_dataset(DATASET_NAME, split='all')
3     squad_v2_dataset = squad_v2_dataset.train_test_split(test_size=
4         INSERT_RATIO, seed=0)['test']
5     squad_v2_dataset = squad_v2_dataset.map(lambda val: {'answer': val['
6         answers'] ['text'] [0]} if val ['answers'] ['text'] else {'answer': ''},
7         remove_columns=['answers'])
8
9     # Generate the tokens for each entry.
10    squad_v2_dataset = squad_v2_dataset.map(tokenize_question, batch_size=
11        TOKENIZATION_BATCH_SIZE, batched=True)
12    squad_v2_dataset.set_format('torch', columns=['input_ids', '
13        attention_mask'], output_all_columns=True)
14
15    squad_v2_dataset = squad_v2_dataset.map(
16        quest_embedding,
17        remove_columns=['input_ids', 'attention_mask'],
18        batched=True,
19        batch_size=INFERENCE_BATCH_SIZE

```



```

15     )
16
17     # Due to the varchar constraint we are going to limit the question size
    when inserting
18     def insert_function(batch):
19         insertable = [
20             batch['title'],
21             batch['question'],
22             [x[:9995] + '...' if len(x) > 9999 else x for x in batch['context
    ']],
23             [x[:995] + '...' if len(x) > 999 else x for x in batch['answer'
    ]],
24             batch['question_embedding'].tolist()
25         ]
26         qa_collection.insert(insertable)
27
28     squad_v2_dataset.map(insert_function, batched=True, batch_size=64)
29     qa_collection.flush()

```

Lưu ý rằng trong đoạn code này ở dòng số 3 các sử dụng hàm `train_test_split()` để tách nhỏ bộ dữ liệu ra nhằm mục đích có thể test trên một lượng sample nhỏ. Các bạn có thể điều chỉnh số lượng này thông qua tham số `INSERT_RATIO` đã khai báo ở đầu code.

- (g) **Khởi tạo vector database:** Với các hàm trên, ta tiến hành thực hiện lời gọi hàm để khởi tạo vector database cho bộ dữ liệu QA:

```

1 connections.connect(host=MILVUS_HOST, port=MILVUS_PORT)
2 if not utility.has_collection(COLLECTION_NAME):
3     qa_collection = create_collection(COLLECTION_NAME, DIMENSION)
4     qa_collection.load(replica_number=REPLICA_NUMBER)
5 else:
6     qa_collection = Collection(COLLECTION_NAME)
7     qa_collection.load(replica_number=REPLICA_NUMBER)
8
9 if qa_collection.is_empty:
10     create_squad_database(qa_collection)

```

- (h) **Xây dựng hàm search:** Khi đã tạo xong vector database, chúng ta sẽ xây dựng một hàm cho phép nhận vào một batch các vector embedding của câu truy vấn (trong trường hợp này là các câu hỏi). Sau đó, tìm kiếm tương đồng và trả về các mẫu dữ liệu có liên quan nhất:

```

1 def search(question_batch):
2     res = qa_collection.search(
3         question_batch['question_embedding'].tolist(),
4         anns_field='question_embedding',
5         param={
6             "metric_type": "L2",
7             "params": {"nprobe": 10},
8         },
9         output_fields=['question', 'context'],
10        limit=LIMIT
11    )
12    overall_id = []
13    overall_distance = []
14    overall_question = []
15    overall_context = []
16
17    for hits in res:
18        ids = []
19        distances = []
20        questions = []

```

```

21     contexts = []
22
23     for hit in hits:
24         ids.append(hit.id)
25         distances.append(hit.distance)
26         questions.append(hit.entity.get('question'))
27         contexts.append(hit.entity.get('context'))
28
29     overall_id.append(ids)
30     overall_distance.append(distances)
31     overall_question.append(questions)
32     overall_context.append(contexts)
33
34     return {
35         'id': overall_id,
36         'distance': overall_distance,
37         'context': overall_context,
38         'similar_question': overall_question
39     }

```

5. **Kết hợp công cụ tìm kiếm và mô hình hỏi-đáp:** Khi chạy xong file `build_dataset.py`, chúng ta đã có một vector database mong muốn. Bây giờ, để kết hợp với mô hình QA để trở thành End-to-end QA, chúng ta sẽ viết một file code để triển khai vấn đề này. Tại đây, mình sẽ tạo một file mới mang tên `qa.py` và có nội dung như sau:

(a) **Import các thư viện, hàm và tham số cần thiết:**

```

1  import argparse
2  from datasets import Dataset
3  from transformers import pipeline
4  from pymilvus import connections, utility, Collection
5  from build_database import tokenize_question, quest_embedding, search
6  from build_database import (
7      MILVUS_HOST,
8      MILVUS_PORT,
9      COLLECTION_NAME,
10     REPLICATION_NUMBER,
11     TOKENIZATION_BATCH_SIZE,
12     INFERENCE_BATCH_SIZE
13 )

```

Các bạn lưu ý có một số hàm và biến sẽ được import từ file code `build_dataset.py`.

(b) **Kết nối tới vector database:**

```

1  connections.connect(host=MILVUS_HOST, port=MILVUS_PORT)
2  if utility.has_collection(COLLECTION_NAME):
3      qa_collection = Collection(COLLECTION_NAME)
4      qa_collection.load(replica_number=REPLICATION_NUMBER)
5  else:
6      raise RuntimeError

```

- (c) **Khai báo mô hình QA:** Chúng ta sẽ dùng mô hình đã huấn luyện ở buổi học về QA để sử dụng trong chương trình code này. Ở đây, mình sẽ sử dụng mô hình đã huấn luyện và được lưu trên HuggingFace:

```

1  PIPELINE_NAME = 'question-answering'
2  MODEL_NAME = 'thangduong0509/distilbert-finetuned-squadv2'
3  qa_pipeline = pipeline(PIPELINE_NAME, model=MODEL_NAME)

```

Các bạn nên sử dụng mô hình mình đã huấn luyện và thay tên ở biến `MODEL_NAME` nhé.

- (d) **Xây dựng hàm main cho chương trình:** Cuối cùng, ta viết code nhận đầu vào là câu hỏi từ command line, thực hiện embedding câu hỏi và chạy hàm search. Từ đó, với các tài liệu có liên quan, ta chạy mô hình QA để trả lời câu hỏi từ input:

```

1 def main():
2     parser = argparse.ArgumentParser()
3     parser.add_argument('--question', type=str, required=True)
4     args = parser.parse_args()
5
6     questions = {'question': [f'{args.question}']}
7     question_dataset = Dataset.from_dict(questions)
8
9     question_dataset = question_dataset.map(
10         tokenize_question,
11         batched=True,
12         batch_size=TOKENIZATION_BATCH_SIZE
13     )
14     question_dataset.set_format(
15         'torch',
16         columns=['input_ids', 'attention_mask'],
17         output_all_columns=True
18     )
19     question_dataset = question_dataset.map(
20         quest_embedding,
21         remove_columns=['input_ids', 'attention_mask'],
22         batched=True,
23         batch_size=INFERENCE_BATCH_SIZE
24     )
25
26     retrieval_results = question_dataset.map(search, batched=True, batch_size=1)
27     for result in retrieval_results:
28         print()
29         print('Input Question:')
30         print(result['question'])
31         print()
32         for rank_idx, candidate in enumerate(zip(result['similar_question'],
33 result['context'], result['distance'])):
34             context = candidate[1]
35             distance = candidate[2].tolist()
36             predicted_answer = qa_pipeline(
37                 context=context,
38                 question=args.question
39             )
40
41             print(f'Relevant Context Rank {rank_idx+1}:')
42             print(f'Context: {context}')
43             print(f'Score: {distance}')
44             print(f'Predicted Answer: {predicted_answer}')
45             print()
46 if __name__ == '__main__':
47     main()
48     qa_collection.release()

```

Cuối cùng, chúng ta sẽ chạy file này để xem thử thành quả. Ở đây, mình sẽ chạy với câu hỏi sau (câu hỏi này thuộc bộ dữ liệu SQuADv2):

```
1 $ python3 qa.py --question 'In what year did Wesley Clark retire?'
```

```

Input Question:
In what year did Wesley Clark retire?

Relevant Context Rank 1:
Context: In 1970, President Nasser died and was succeeded by Anwar Sadat. Sadat switched Egypt's Cold War allegiance from the Soviet Union to the United States, expelling Soviet advisors i
n 1972. He launched the Infitah economic reform policy, while clamping down on religious and secular opposition. In 1973, Egypt, along with Syria, launched the October War, a surprise atta
ck to regain part of the Sinai territory Israel had captured 6 years earlier. It presented Sadat with a victory that allowed him to regain the Sinai later in return for peace with Israel.
Score: 122.26721954345703
Predicted Answer: {'score': 0.027695229277014732, 'start': 3, 'end': 7, 'answer': '1970'}

Relevant Context Rank 2:
Context: In September 2003, retired four-star general Wesley Clark announced his intention to run in the presidential primary election for the Democratic Party nomination. His campaign foc
used on themes of leadership and patriotism; early campaign ads relied heavily on biography. His late start left him with relatively few detailed policy proposals. This weakness was appare
nt in his first few debates, although he soon presented a range of position papers, including a major tax-relief plan. Nevertheless, the Democrats did not flock to support his campaign.
Score: 124.70985412597656
Predicted Answer: {'score': 0.028020793572068214, 'start': 13, 'end': 17, 'answer': '2003'}

```

Hình 10: Kết quả End-to-end QA sử dụng hàm search trên Milvus vector database

Như vậy, thông qua việc cài đặt theo các bước trên, các bạn đã thành công ứng dụng Milvus vector database để xây dựng một chương trình về End-to-end Question Answering. Các bạn muốn hiểu thêm về Milvus có thể tìm đọc code đính kèm có file `hello_milvus.py` để hiểu thêm về các hàm cơ bản trong Milvus nhé.

6. **Trường hợp muốn ngắt kết nối với Milvus và xóa dữ liệu:** Để nhanh chóng ngắt kết nối với Milvus, các bạn hãy sử dụng lệnh sau trong Terminal:

```
1 $ bash standalone_embed.sh stop
```

Để xóa hẳn dữ liệu được lưu trong Milvus, đầu tiên các bạn hãy chạy lệnh dưới đây trong Terminal:

```
1 $ docker ps -a
```

Tại đây, các bạn sẽ thấy một danh sách các CONTAINER ID, các bạn hãy tìm hàng có tên tại dòng IMAGE là milvusdb:

```

(milvus_env) thangduong@Duongs-MacBook-Pro milvus_test % docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS          NAMES
a846426c46b7   milvusdb/milvus:v2.3.7             "/tiny -- milvus run..." 28 hours ago   Exited (137) 16 seconds ago           milvus-standalone

```

Hình 11: Hàng Container ID của Milvus

Các bạn hãy copy CONTAINER ID của milvusdb, trong trường hợp ở ảnh trên sẽ là a846426c46b7. Sau đó, các bạn chạy lệnh sau:

```
1 $ docker rm a846426c46b7
```

Như vậy, các bạn đã ngắt kết nối khỏi Milvus cũng như xóa toàn bộ dữ liệu đã đưa vào.

- Hết -