Các bạn hãy đọc hết nha đừng bỏ sót dòng nào, mk cảm thấy mọi thông tin đều hữu ích !

Hybrid Search

Hybrid Search là phương pháp tìm kiếm thực hiện nhiều tìm kiếm ANN đồng thời, xếp hạng lại nhiều bộ kết quả từ các tìm kiếm ANN này và cuối cùng trả về một bộ kết quả duy nhất. Sử dụng Hybrid Search có thể tăng cường độ chính xác của tìm kiếm. Zilliz hỗ trợ thực hiện Hybrid Search trên một collection có nhiều trường vectơ.

Hybrid Search thường được sử dụng trong các tình huống bao gồm tìm kiếm sparse-dense vector và tìm kiếm đa phương thức(hình ảnh, âm thanh, ...). Tài liệu này sẽ trình bày cách thực hiện Tìm Hybrid Search trong Zilliz với một ví dụ cụ thể.

Các kịch bạn

Hybrid Search phù hợp với hai tình huống sau:

Sparse-Dense Vector Search

Các loại vectơ khác nhau có thể biểu diễn thông tin khác nhau và việc sử dụng các mô hình embedding khác nhau có thể biểu diễn toàn diện hơn các đặc trưng và khía cạnh khác nhau của dữ liệu. Ví dụ, việc sử dụng các mô hình embedding khác nhau cho cùng một câu có thể tạo ra một vectơ dày đặc để biểu diễn ngữ nghĩa của câu và một sparse vector (vectơ thưa thớt) để biểu diễn tần suất từ trong câu.

- **Sparse vectors**: Sparse vectors được đặc trưng bởi tính đa chiều của vectơ và sự hiện diện của một số ít giá trị khác không. Cấu trúc này làm cho chúng đặc biệt phù hợp với các ứng dụng truy xuất thông tin truyền thống. Trong hầu hết các trường hợp, số chiều được sử dụng trong các vectơ thưa thớt tương ứng với số lượng token khác nhau (tổng số lượng từ khác nhau trong toàn bộ văn bản mà ta đang làm việc) trên một hoặc nhiều ngôn ngữ. Mỗi chiều được gán một giá trị biểu thị tầm quan trọng tương đối của token đó trong tài liệu. Việc này có lợi cho các tác vụ liên quan đến việc khớp văn bản(text matching).

- **Dense vectors**: Dense vectors là các embedding được tạo ra từ mạng nơ-ron. Khi được sắp xếp trong một mảng có thứ tự, các vectơ này nắm bắt được thông tin về ngữ nghĩa của văn bản đầu vào. Lưu ý rằng các Dense vector không chỉ giới hạn ở xử lý văn bản; chúng cũng được sử dụng rộng rãi trong thị giác máy tính để biểu diễn ngữ nghĩa của dữ liệu trực quan. Các Dense vector thường được tạo ra bởi các mô hình embedding văn bản, được biểu diễn bởi gần như tất cả các phần tử khác không. Do đó, các Dense vector đặc biệt hiệu quả đối với trường hợp tìm kiếm ngữ nghĩa, vì chúng có thể trả về các kết quả giống nhau nhất dựa trên khoảng cách vectơ ngay cả khi không có các kết quả khớp chính xác với văn bản. Khả năng này cho phép có kết quả tìm kiếm đa dạng hơn và có sự tương đồng về ngữ cảnh, nắm bắt được các mối quan hệ giữa các khái niệm có thể bị bỏ sót bởi các phương pháp tìm kiếm dựa trên khớp từ khóa.

For more details, refer to Sparse Vector and Dense Vector.

Multimodal Search

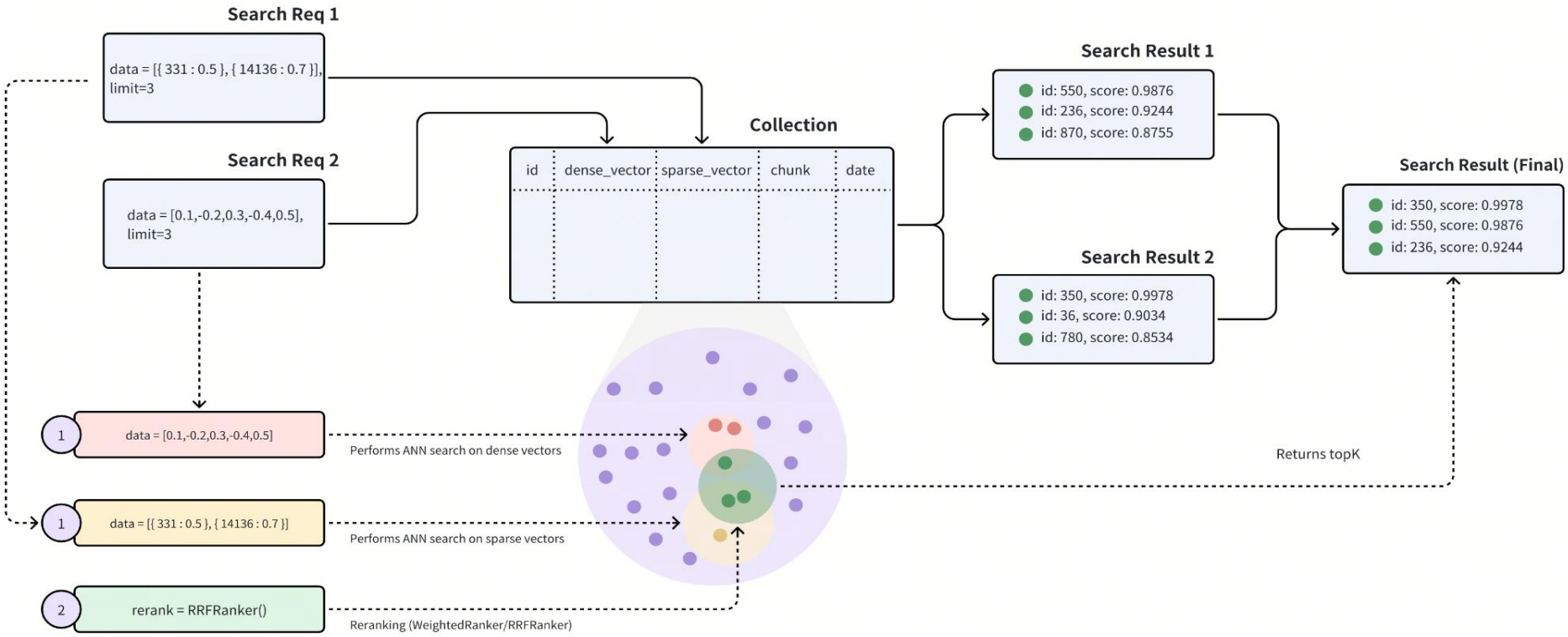
Tìm kiếm đa phương thức đề cập đến việc tìm kiếm sự tương đồng của dữ liệu phi cấu trúc trên nhiều phương thức (như hình ảnh, video, âm thanh, văn bản, v.v.). Ví dụ, một người có thể được đại diện bằng nhiều phương thức dữ liệu khác nhau như dấu vân tay, dấu giọng nói và đặc điểm khuôn mặt. Việc tìm kiếm kết hợp hỗ trợ nhiều tìm kiếm đồng thời. Ví dụ, tìm kiếm một người dựa trên dấu vân tay và đặc trưng giọng nói.

Quy trình

Quy trình để tiến hành Hybrid Search gồm các bước như sau :

1. Tạo ra các dense vector thông qua các embedding models như BERT và Transformers. Nếu dùng tiếng Việt các bạn tham khảo vietnamese-sbert (kiểu embedding về mặt ngữ nghĩa cả câu)
2. Tạo ra các sparse vectors thông qua các embedding models như BM25, BGE-M3, SPLADE, etc. Trong Milvus, bạn có thể sử dụng Function có sẵn để tạo ra các sparse vector. Xem thêm thông tin chi tiết hơn tại Full Text Search.
3. Tạo một collection và khai báo collection schema trong đó bao gồm 2 trường dense và trường sparse vector.
4. Thêm các sparse-dense vector vào collection vừa tạo ở bước trên.

5. Tiến hành Hybrid Search: ANN Search trên dense vectors sẽ trả về top K kết quả có điểm cao nhất tương đồng về mặt ngữ nghĩa, và khớp từ trên sparse vectors sẽ trả về top K kết quả câu có khớp từ mà bạn nhập vào (ví dụ tìm "thủ tướng Việt Nam là ai ?" thì sẽ khớp từ "thủ tướng VN" trong tập văn bản mà bạn đang tìm kiếm)
6. Chuẩn hóa: Chuẩn hóa điểm của 2 bộ kết quả trả về bên trên, đưa scores về trong khoảng [0,1].
7. Chọn một chiến lược reranking phù hợp để gộp và reranking 2 tập kết quả top-K ở bước 5 và cuối cùng trả về một tập hợp kết quả top-K.



Ví dụ

Phần này sẽ lấy một ví dụ cụ thể để minh họa cách thực hiện Hybrid Search trên các sparse-dense vector nhằm tăng độ chính xác của việc tìm kiếm văn bản(cả ngữ nghĩa cả khớp từ).

Tạo một collection với nhiều trường vector

Quy trình tạo một collection bao gồm 3 phần: Khai báo các thuộc tính của schema, cấu hình các tham số chỉ mục(index parameters), và tạo collection.

Định nghĩa schema

Trong ví dụ này, nhiều trường vector cần được định nghĩa trong collection schema. Hiện tại, mỗi collection có thể bao gồm tối đa 4 trường vector theo mặc định. Nhưng bạn cũng có thể sửa đổi giá trị của proxy.maxVectorFieldNum để tăng lên tối đa 10 trường vector trong một collection khi cần.

Ví dụ sau đây định nghĩa một collection schema, nơi mà `dense` và `sparse` là 2 trường vector :

- `id`: Trường này đóng vai trò là khóa chính để lưu trữ ID. Kiểu dữ liệu của trường này là INT64.
- `text`: Trường này được sử dụng để lưu trữ nội dung văn bản. Kiểu dữ liệu của trường này là VARCHAR, với độ dài tối đa là 1000 ký tự.

- dense: Trường này được sử dụng để lưu trữ các vectơ dày đặc của văn bản. Kiểu dữ liệu của trường này là FLOAT_VECTOR, với số chiều là 768.
- sparse: Trường này được sử dụng để lưu trữ các sparse vector của văn bản. Kiểu dữ liệu của trường này là SPARSE_FLOAT_VECTOR. Trong ví dụ này, chúng tôi sử dụng Function có sẵn của milvus để tạo các sparse vectors.

```
# Create a collection in customized setup mode
from pymilvus import (
    MilvusClient, DataType
)

client = MilvusClient(
    uri="http://localhost:19530",
    token="root:Milvus"
)

# Create schema
schema = MilvusClient.create_schema(
    auto_id=False,
    enable_dynamic_field=True,
)
# Add fields to schema
schema.add_field(field_name="id", datatype=DataType.INT64, is_primary=True)
schema.add_field(field_name="text", datatype=DataType.VARCHAR, max_length=1000, enable_analyzer=True)
# Define a sparse vector field to generate spare vectors with BM25
schema.add_field(field_name="sparse", datatype=DataType.SPARSE_FLOAT_VECTOR)
schema.add_field(field_name="dense", datatype=DataType.FLOAT_VECTOR, dim=5)
```

Trong quá trình tìm kiếm sparse vector, bạn có thể đơn giản hóa quá trình tạo vectơ nhúng sparse bằng cách tận dụng khả năng của Full Text Search. Để biết thêm chi tiết, hãy xem Full Text Search.

Khai báo hàm để tạo sparse vectors

Để tạo sparse vectors, bạn có thể sử dụng Hàm chức năng trong Milvus. Ví dụ sau đây định nghĩa một Function để tạo sparse vectors sử dụng thuật toán BM25. Để biết thêm chi tiết, hãy xem Full Text Search.

```
# Define function to generate sparse vectors

bm25_function = Function(
    name="text_bm25_emb", # Function name
    input_field_names=["text"], # Name of the VARCHAR field containing raw text data
    output_field_names=["sparse"], # Name of the SPARSE_FLOAT_VECTOR field reserved to store generated embeddings
    function_type=FunctionType.BM25,
)

schema.add_function(bm25_function)
```

Tạo index

Sau khi định nghĩa collection schema, cần thiết lập các vector indexes và phương pháp đo sự tương đồng (similarity metrics). Trong ví dụ này, IVF_FLAT index được tạo cho trường dense vector, và SPARSE_INVERTED_INDEX được tạo cho trường sparse. Để tìm hiểu về các loại index được hỗ trợ, xem tại Index Explained.

```python
from pymilvus import MilvusClient

# Prepare index parameters
index_params = client.prepare_index_params()

# Add indexes
index_params.add_index(
    field_name="dense",
    index_name="dense_index",
    index_type="IVF_FLAT",
    metric_type="IP",
    params={"nlist": 128},
)

index_params.add_index(
    field_name="sparse",
    index_name="sparse_index",
    index_type="SPARSE_INVERTED_INDEX",  # Index type for sparse vectors
    metric_type="BM25",  # Set to `BM25` when using function to generate sparse vectors
    params={"inverted_index_algo": "DAAT_MAXSCORE"},  # The ratio of small vector values to be dropped during indexing
```

Tạo collection

Tạo một collection có tên là hybrid_search_collection với collection schema và indexes được thiết lập ở các bước trước.

```
from pymilvus import MilvusClient

client.create_collection(
    collection_name="hybrid_search_collection",
    schema=schema,
    index_params=index_params
)
```

Insert data

Đưa dữ liệu sparse-dense vectors vào collection có tên **hybrid_search_collection**.

```
from pymilvus import MilvusClient

docs = [
    "Artificial intelligence was founded as an academic discipline in 1956.",
    "Alan Turing was the first person to conduct substantial research in AI.",
    "Born in Maida Vale, London, Turing was raised in southern England.",
]
```

```
data = [
    {"id": 1, "text": docs[0], "dense": [2.7242085933685303, 6.021071434020996, 0.4754035174846649, 9.358858108520508,
5.173221111297607]},
    {"id": 2, "text": docs[1], "dense": [8.584294319152832, 2.7640628814697266, 9.558855056762695, 2.584272861480713,
4.705013275146484]},
    {"id": 3, "text": docs[2], "dense": [2.5525057315826416, 3.8815805912017822, 9.343480110168457, 7.888997554779053,
4.500918388366699]},
]

res = client.insert(
    collection_name="hybrid_search_collection",
    data=data
)
```

Tạo nhiều đối tượng AnnSearchRequest

Để thực hiện tìm kiếm kết hợp (Hybrid Search), hàm hybrid_search() sẽ tạo ra nhiều đối tượng AnnSearchRequest. Mỗi đối tượng AnnSearchRequest này đại diện cho một yêu cầu tìm kiếm ANN cơ bản trên một trường vector cụ thể. Do đó, trước khi tiến hành tìm kiếm kết hợp, bạn cần tạo một đối tượng AnnSearchRequest cho mỗi trường vector mà bạn muốn sử dụng trong quá trình tìm kiếm.

> Trong Hybrid Search, mỗi AnnSearchRequest chỉ hỗ trợ một vector truy vấn

Giả sử đoạn văn bản truy vấn là “Who started AI research?” đã được chuyển đổi thành sparse vectors và dense vectors. Dựa vào đó, hai yêu cầu tìm kiếm **AnnSearchRequest** được tạo cho các trường **sparse** và **dense** vector tương ứng, như trong ví dụ sau.

```python
from pymilvus import AnnSearchRequest

search_param_1 = {
    "data": [[0.7425515055656433, 7.774101734161377, 0.7397570610046387, 2.429982900619507, 3.8253049850463867]],
    "anns_field": "dense",
    "param": {
        "metric_type": "IP",
        "params": {"nprobe": 10}
    },
    "limit": 2
}
request_1 = AnnSearchRequest(**search_param_1)

search_param_2 = {
    "data": ['Who started AI research'],
    "anns_field": "sparse",
    "param": {
        "metric_type": "BM25",
    },
    "limit": 2
}
request_2 = AnnSearchRequest(**search_param_2)

reqs = [request_1, request_2]
```

vì tham số **limit** được đặt là 2, mỗi **AnnSearchRequest** trả về 2 kết quả. Trong ví dụ này, Có 2 **AnnSearchRequest** được tạo ra, do đó tổng cộng sẽ có 4 kết quả tìm kiếm được trả về.

Thiết lập chiến lược reranking (sắp xếp lại)

Để gộp và **reranking** hai tập kết quả tìm kiếm ANN, cần thiết phải chọn một chiến lược **reranking** phù hợp. Zilliz hỗ trợ hai loại chiến lược **reranking**: WeightedRanker và RRFRanker. Khi chọn một chiến lược **reranking**, một điều cần cân nhắc là liệu có sự nhấn mạnh nào cho một hoặc nhiều tìm kiếm ANN cơ bản trên các trường vectơ hay không.

WeightedRanker: Chiến lược này được khuyến nghị nếu bạn yêu cầu kết quả nhấn mạnh một trường vectơ cụ thể. WeightedRanker cho phép bạn gán trọng số cao hơn cho một số trường vectơ nhất định, do đó nhấn mạnh chúng hơn. Ví dụ, trong các tìm kiếm đa phương thức, mô tả văn bản của một hình ảnh có thể được coi là quan trọng hơn màu sắc trong hình ảnh đó.

RRFRanker (Reciprocal Rank Fusion Ranker): Chiến lược này được khuyến nghị khi không có sự nhấn mạnh cụ thể nào. RRF có thể cân bằng hiệu quả tầm quan trọng của từng trường vectơ.

Để biết thêm chi tiết về cơ chế của hai chiến lược sắp xếp lại này, hãy tham khảo Reranking.

Hai ví dụ sau đây minh họa cách sử dụng các chiến lược sắp xếp lại WeightedRanker và RRFRanker:

Ví dụ 1: Sử dụng WeightedRanker

Khi sử dụng chiến lược WeightedRanker, bạn cần nhập các giá trị trọng số vào hàm WeightedRanker. Số lượng tìm kiếm ANN cơ bản trong một Tìm kiếm Hỗn hợp tương ứng với số lượng giá trị cần được nhập vào. Các giá trị đầu vào phải nằm trong khoảng [0, 1], với các giá trị gần 1 hơn cho thấy tầm quan trọng lớn hơn.

```
from pymilvus import WeightedRanker

ranker = WeightedRanker(0.8, 0.3)
```

Ví dụ 2: Sử dụng RRFRanker

Khi sử dụng chiến lược RRFRanker, bạn cần nhập giá trị tham số k vào RRFRanker. Giá trị mặc định của k là 60. Tham số này giúp xác định cách các thứ hạng được kết hợp từ các tìm kiếm ANN khác nhau, nhằm mục đích cân bằng và hòa trộn tầm quan trọng giữa tất cả các tìm kiếm.

```
from pymilvus import RRFRanker

ranker = RRFRanker(100)
```

Thực hiện Hybrid Search

Trước khi tiến hành Hybrid Search, trước hết phải tải collection vào bộ nhớ. Nếu bất kỳ trường vectơ nào trong bộ sưu tập không có đánh index hoặc chưa được load, lỗi sẽ xảy ra khi gọi phương thức Hybrid Search.

```
from pymilvus import MilvusClient

res = client.hybrid_search(
    collection_name="hybrid_search_collection",
    reqs=reqs,
    ranker=ranker,
    limit=2
)
```

```python
for hits in res:
    print("TopK results:")
    for hit in hits:
        print(hit)
```

Đây là kết quả đầu ra:

```
TopK results:
{'id': 1, 'distance': 0.0099009899422252636, 'entity': {}}
{'id': 2, 'distance': 0.0099009899422252636, 'entity': {}}
```

Vì `limit=2` được thiết lập trong Hybrid Search, Zilliz sẽ sắp xếp lại bốn kết quả tìm kiếm từ bước 3 và cuối cùng chỉ trả về 2 kết quả tìm kiếm tương tự nhất.