Deep Lake self-querying

> DeepLake is a multimodal database for building AI applications.

In the notebook we'll demo the `SelfQueryRetriever` wrapped around a DeepLake vector store.

Creating a Deep Lake vectorstore

First we'll want to create a DeepLake VectorStore and seed it with some data. We've created a small demo set of documents that contain summaries of movies.

NOTE: The self-query retriever requires you to have `lark` installed (`pip install lark`). We also need the `deeplake` package.

```
# !pip install lark
```

```
# in case if some queries fail consider installing libdeeplake manually
# !pip install libdeeplake
```

We want to use `OpenAIEmbeddings` so we have to get the OpenAI API Key.

```
import os
import getpass

os.environ["OPENAI_API_KEY"] = getpass.getpass("OpenAI API Key:")
os.environ["ACTIVELOOP_TOKEN"] = getpass.getpass("Activeloop token:")
```

```
from langchain.schema import Document
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import DeepLake

embeddings = OpenAIEmbeddings()
```

API Reference:

API Reference:

- Document from `langchain.schema`
- OpenAIEmbeddings from `langchain.embeddings.openai`
- DeepLake from `langchain.vectorstores`

```
docs = [
    Document(
        page_content="A bunch of scientists bring back dinosaurs and
mayhem breaks loose",
        metadata={"year": 1993, "rating": 7.7, "genre": "science
fiction"},
    ),
    Document(
        page_content="Leo DiCaprio gets lost in a dream within a dream
within a dream within a ...",
        metadata={"year": 2010, "director": "Christopher Nolan",
"rating": 8.2},
    ),
    Document(
        page_content="A psychologist / detective gets lost in a series
of dreams within dreams within dreams and Inception reused the idea",
        metadata={"year": 2006, "director": "Satoshi Kon", "rating":
8.6},
    ),
    Document(
        page_content="A bunch of normal-sized women are supremely
wholesome and some men pine after them",
        metadata={"year": 2019, "director": "Greta Gerwig", "rating":
8.3},
    ),
    Document(
        page_content="Toys come alive and have a blast doing so",
        metadata={"year": 1995, "genre": "animated"},
    ),
    Document(
        page_content="Three men walk into the Zone, three men walk out
of the Zone",
        metadata={
            "year": 1979,
            "rating": 9.9,
            "director": "Andrei Tarkovsky",
            "genre": "science fiction",
            "rating": 9.9,
        },
    ),
```

```
]
username_or_org = "<USERNAME_OR_ORG>"
vectorstore = DeepLake.from_documents(
    docs, embeddings,
dataset_path=f"hub://{username_or_org}/self_queery", overwrite=True,
)
```

```
    Your Deep Lake dataset has been successfully created!


    /

    Dataset(path='hub://adilkhan/self_queery', tensors=['embedding',
'id', 'metadata', 'text'])

      tensor      htype       shape     dtype  compression
      -------    -------    -------   -------  -------
     embedding  embedding  (6, 1536)  float32   None
        id         text      (6, 1)      str     None
     metadata      json      (6, 1)      str     None
       text        text      (6, 1)      str     None
```

Creating our self-querying retriever

Now we can instantiate our retriever. To do this we'll need to provide some information upfront about the metadata fields that our documents support and a short description of the document contents.

```
from langchain.llms import OpenAI
from langchain.retrievers.self_query.base import SelfQueryRetriever
from langchain.chains.query_constructor.base import AttributeInfo

metadata_field_info = [
    AttributeInfo(
        name="genre",
        description="The genre of the movie",
        type="string or list[string]",
    ),
    AttributeInfo(
```

```
        name="year",
        description="The year the movie was released",
        type="integer",
    ),
    AttributeInfo(
        name="director",
        description="The name of the movie director",
        type="string",
    ),
    AttributeInfo(
        name="rating", description="A 1-10 rating for the movie",
type="float"
    ),
]
document_content_description = "Brief summary of a movie"
llm = OpenAI(temperature=0)
retriever = SelfQueryRetriever.from_llm(
    llm, vectorstore, document_content_description,
metadata_field_info, verbose=True
)
```

API Reference:

- OpenAI from `langchain.llms`
- SelfQueryRetriever from `langchain.retrievers.self_query.base`
- AttributeInfo from `langchain.chains.query_constructor.base`

Testing it out

And now we can try actually using our retriever!

```
# This example only specifies a relevant query
retriever.get_relevant_documents("What are some movies about
dinosaurs")
```

```
/home/ubuntu/langchain_activeloop/langchain/libs/langchain/langchain/chain
UserWarning: The predict_and_parse method is deprecated, instead pass an o
directly to LLMChain.
      warnings.warn(
```

```
    query='dinosaur' filter=None limit=None
```

```
    [Document(page_content='A bunch of scientists bring back dinosaurs and
breaks loose', metadata={'year': 1993, 'rating': 7.7, 'genre': 'science fi
     Document(page_content='Toys come alive and have a blast doing so', me
{'year': 1995, 'genre': 'animated'}),
     Document(page_content='Three men walk into the Zone, three men walk o
Zone', metadata={'year': 1979, 'rating': 9.9, 'director': 'Andrei Tarkovsk
'science fiction'}),
     Document(page_content='A psychologist / detective gets lost in a seri
within dreams within dreams and Inception reused the idea', metadata={'yea
'director': 'Satoshi Kon', 'rating': 8.6})]
```

```
# This example only specifies a filter
retriever.get_relevant_documents("I want to watch a movie rated higher
than 8.5")

# in case if this example errored out, consider installing libdeeplake
manually: `pip install libdeeplake`, and then restart notebook.
```

```
    query=' ' filter=Comparison(comparator=<Comparator.GT: 'gt'>,
attribute='rating', value=8.5) limit=None
```

```
    [Document(page_content='A psychologist / detective gets lost in a
series of dreams within dreams within dreams and Inception reused the
idea', metadata={'year': 2006, 'director': 'Satoshi Kon', 'rating':
8.6}),
     Document(page_content='Three men walk into the Zone, three men
walk out of the Zone', metadata={'year': 1979, 'rating': 9.9,
'director': 'Andrei Tarkovsky', 'genre': 'science fiction'})]
```

```
# This example specifies a query and a filter
retriever.get_relevant_documents("Has Greta Gerwig directed any movies
about women")
```

```
    query='women' filter=Comparison(comparator=<Comparator.EQ: 'eq'>,
attribute='director', value='Greta Gerwig') limit=None
```

```
    [Document(page_content='A bunch of normal-sized women are supremely
wholesome and some men pine after them', metadata={'year': 2019,
'director': 'Greta Gerwig', 'rating': 8.3})]
```

```
# This example specifies a composite filter
retriever.get_relevant_documents(
    "What's a highly rated (above 8.5) science fiction film?"
)
```

```
    query=' ' filter=Operation(operator=<Operator.AND: 'and'>,
arguments=[Comparison(comparator=<Comparator.GTE: 'gte'>,
attribute='rating', value=8.5), Comparison(comparator=<Comparator.EQ:
'eq'>, attribute='genre', value='science fiction')]) limit=None
```

```
    [Document(page_content='Three men walk into the Zone, three men
walk out of the Zone', metadata={'year': 1979, 'rating': 9.9,
'director': 'Andrei Tarkovsky', 'genre': 'science fiction'})]
```

```
# This example specifies a query and composite filter
retriever.get_relevant_documents(
    "What's a movie after 1990 but before 2005 that's all about toys,
and preferably is animated"
)
```

```
    query='toys' filter=Operation(operator=<Operator.AND: 'and'>,
arguments=[Comparison(comparator=<Comparator.GT: 'gt'>,
attribute='year', value=1990), Comparison(comparator=<Comparator.LT:
'lt'>, attribute='year', value=2005), Comparison(comparator=
<Comparator.EQ: 'eq'>, attribute='genre', value='animated')])
limit=None
```

```
    [Document(page_content='Toys come alive and have a blast doing so',
metadata={'year': 1995, 'genre': 'animated'})]
```

Filter k

We can also use the self query retriever to specify `k`: the number of documents to fetch.

We can do this by passing `enable_limit=True` to the constructor.

```
retriever = SelfQueryRetriever.from_llm(
    llm,
    vectorstore,
    document_content_description,
    metadata_field_info,
    enable_limit=True,
    verbose=True,
)
```

```
# This example only specifies a relevant query
retriever.get_relevant_documents("what are two movies about dinosaurs")
```

```
    query='dinosaur' filter=None limit=2
```

```
    [Document(page_content='A bunch of scientists bring back dinosaurs
and mayhem breaks loose', metadata={'year': 1993, 'rating': 7.7,
'genre': 'science fiction'}),
     Document(page_content='Toys come alive and have a blast doing so',
metadata={'year': 1995, 'genre': 'animated'})]
```