MultiVector Retriever

It can often be beneficial to store multiple vectors per document. There are multiple use cases where this is beneficial. LangChain has a base `MultiVectorRetriever` which makes querying this type of setup easy. A lot of the complexity lies in how to create the multiple vectors per document. This notebook covers some of the common ways to create those vectors and use the `MultiVectorRetriever`.

The methods to create multiple vectors per document include:

- Smaller chunks: split a document into smaller chunks, and embed those (this is ParentDocumentRetriever).
- Summary: create a summary for each document, embed that along with (or instead of) the document.
- Hypothetical questions: create hypothetical questions that each document would be appropriate to answer, embed those along with (or instead of) the document.

Note that this also enables another method of adding embeddings - manually. This is great because you can explicitly

add questions or queries that should lead to a document being recovered, giving you more control.

```
from langchain.retrievers.multi_vector import
MultiVectorRetriever
```

```
from langchain.storage import
InMemoryByteStore
from langchain.text_splitter import
RecursiveCharacterTextSplitter
from langchain_community.document_loaders
import TextLoader
from langchain_community.vectorstores import
Chroma
from langchain_openai import OpenAIEmbeddings
```

```
loaders = [
    TextLoader("../../paul_graham_essay.txt"),
    TextLoader("../../state_of_the_union.txt"),
]
docs = []
for loader in loaders:
    docs.extend(loader.load())
text_splitter =
RecursiveCharacterTextSplitter(chunk_size=10000
docs = text_splitter.split_documents(docs)
```

Smaller chunks

Often times it can be useful to retrieve larger chunks of information, but embed smaller chunks. This allows for embeddings to capture the semantic meaning as closely as possible, but for as much context as possible to be passed downstream. Note that this is what the `ParentDocumentRetriever` does. Here we show what is going on under the hood.

```
# The vectorstore to use to index the child chunks
vectorstore = Chroma(
    collection_name="full_documents", embedding_function=OpenAIEmbeddings()
)
# The storage layer for the parent documents
store = InMemoryByteStore()
id_key = "doc_id"
# The retriever (empty to start)
retriever = MultiVectorRetriever(
    vectorstore=vectorstore,
    byte_store=store,
    id_key=id_key,
)
import uuid
```

```
doc_ids = [str(uuid.uuid4()) for _ in docs]
```

```
# The splitter to use to create smaller chunks
child_text_splitter =
RecursiveCharacterTextSplitter(chunk_size=400)
```

```
sub_docs = []
for i, doc in enumerate(docs):
    _id = doc_ids[i]
    _sub_docs =
child_text_splitter.split_documents([doc])
    for _doc in _sub_docs:
        _doc.metadata[id_key] = _id
    sub_docs.extend(_sub_docs)
```

```
retriever.vectorstore.add_documents(sub_docs)
retriever.docstore.mset(list(zip(doc_ids,
docs)))
```

```
# Vectorstore alone retrieves the small chunks
retriever.vectorstore.similarity_search("justic
breyer")[0]
```

```
Document(page_content='Tonight, I’d like to
honor someone who has dedicated his life to
serve this country: Justice Stephen Breyer—an
Army veteran, Constitutional scholar, and
retiring Justice of the United States Supreme
Court. Justice Breyer, thank you for your
service. \n\nOne of the most serious
constitutional responsibilities a President
has is nominating someone to serve on the
United States Supreme Court.', metadata=
{'doc_id': '2fd77862-9ed5-4fad-bf76-
e487b747b333', 'source':
'../../state_of_the_union.txt'})
```

```
# Retriever returns larger chunks
len(retriever.get_relevant_documents("justice
breyer")[0].page_content)
```

```
9875
```

The default search type the retriever performs on the vector database is a similarity search. LangChain Vector Stores also support searching via Max Marginal Relevance so if you want this instead you can just set the `search_type` property as follows:

```
from langchain.retrievers.multi_vector import
SearchType

retriever.search_type = SearchType.mmr

len(retriever.get_relevant_documents("justice
breyer")[0].page_content)
```

```
9875
```

Summary

Oftentimes a summary may be able to distill more accurately what a chunk is about, leading to better retrieval. Here we show how to create summaries, and then embed those.

```
import uuid

from langchain_core.documents import Document
from langchain_core.output_parsers import
StrOutputParser
from langchain_core.prompts import
ChatPromptTemplate
from langchain_openai import ChatOpenAI
```

```
chain = (
    {"doc": lambda x: x.page_content}
    |
ChatPromptTemplate.from_template("Summarize
the following document:\n\n{doc}")
    | ChatOpenAI(max_retries=0)
    | StrOutputParser()
)
```

```
summaries = chain.batch(docs,
{"max_concurrency": 5})
```

```
# The vectorstore to use to index the child
chunks
vectorstore =
Chroma(collection_name="summaries",
embedding_function=OpenAIEmbeddings())
# The storage layer for the parent documents
store = InMemoryByteStore()
id_key = "doc_id"
# The retriever (empty to start)
retriever = MultiVectorRetriever(
    vectorstore=vectorstore,
    byte_store=store,
    id_key=id_key,
)
doc_ids = [str(uuid.uuid4()) for _ in docs]
```

```
summary_docs = [
    Document(page_content=s, metadata=
{id_key: doc_ids[i]})
    for i, s in enumerate(summaries)
]
```

```
retriever.vectorstore.add_documents(summary_doc
retriever.docstore.mset(list(zip(doc_ids,
docs)))
```

```
# # We can also add the original chunks to
the vectorstore if we so want
# for i, doc in enumerate(docs):
#     doc.metadata[id_key] = doc_ids[i]
# retriever.vectorstore.add_documents(docs)
```

```
sub_docs =
vectorstore.similarity_search("justice
breyer")
```

```
sub_docs[0]
```

```
Document(page_content="The document is a speech given by President Biden addressing various issues and outlining his agenda for the nation. He highlights the importance of nominating a Supreme Court justice and introduces his nominee, Judge Ketanji Brown Jackson. He emphasizes the need to secure the border and reform the immigration system, including providing a pathway to citizenship for Dreamers and essential workers. The President also discusses the protection of women's rights, including access to healthcare and the right to choose. He calls for the passage of the Equality Act to protect LGBTQ+ rights. Additionally, President Biden discusses the need to address the opioid epidemic, improve mental health services, support veterans, and fight against cancer. He expresses optimism for the future of America and the strength of the American people.", metadata={'doc_id': '56345bff-3ead-418c-a4ff-dff203f77474'})
```

```
retrieved_docs = retriever.get_relevant_documents("justice breyer")
```

```
len(retrieved_docs[0].page_content)
```

```
9194
```

Hypothetical Queries

An LLM can also be used to generate a list of hypothetical questions that could be asked of a particular document. These questions can then be embedded

```
functions = [
    {
        "name": "hypothetical_questions",
        "description": "Generate hypothetical
questions",
        "parameters": {
            "type": "object",
            "properties": {
                "questions": {
                    "type": "array",
                    "items": {"type":
"string"},
                },
            },
            "required": ["questions"],
        },
```

```
    }
]
```

```
from langchain.output_parsers.openai_functions
import JsonKeyOutputFunctionsParser

chain = (
    {"doc": lambda x: x.page_content}
    # Only asking for 3 hypothetical questions,
but this could be adjusted
    | ChatPromptTemplate.from_template(
        "Generate a list of exactly 3 hypotheti
questions that the below document could be used
answer:\n\n{doc}"
    )
    | ChatOpenAI(max_retries=0, model="gpt-
4").bind(
        functions=functions, function_call=
{"name": "hypothetical_questions"}
    )
    |
JsonKeyOutputFunctionsParser(key_name="question
)
```

```
chain.invoke(docs[0])
```

```
["What was the author's first experience with
programming like?",
 'Why did the author switch their focus from
AI to Lisp during their graduate studies?',
 'What led the author to contemplate a career
in art instead of computer science?']
```

```
hypothetical_questions = chain.batch(docs,
{"max_concurrency": 5})
```

```
# The vectorstore to use to index the child
chunks
vectorstore = Chroma(
    collection_name="hypo-questions",
embedding_function=OpenAIEmbeddings()
)
# The storage layer for the parent documents
store = InMemoryByteStore()
id_key = "doc_id"
# The retriever (empty to start)
retriever = MultiVectorRetriever(
    vectorstore=vectorstore,
    byte_store=store,
    id_key=id_key,
)
doc_ids = [str(uuid.uuid4()) for _ in docs]
```

```
question_docs = []
for i, question_list in
enumerate(hypothetical_questions):
    question_docs.extend(
        [Document(page_content=s, metadata=
{id_key: doc_ids[i]}) for s in question_list]
    )
```

```
retriever.vectorstore.add_documents(question_do
retriever.docstore.mset(list(zip(doc_ids, docs)
```

```
sub_docs =
vectorstore.similarity_search("justice
breyer")
```

```
sub_docs
```

```
[Document(page_content='Who has been
nominated to serve on the United States
Supreme Court?', metadata={'doc_id':
'0b3a349e-c936-4e77-9c40-0a39fc3e07f0'}),
 Document(page_content="What was the context
and content of Robert Morris' advice to the
document's author in 2010?", metadata=
{'doc_id': 'b2b2cdca-988a-4af1-ba47-
```

```
46170770bc8c'}),
 Document(page_content='How did personal
circumstances influence the decision to pass
on the leadership of Y Combinator?',
metadata={'doc_id': 'b2b2cdca-988a-4af1-ba47-
46170770bc8c'}),
 Document(page_content='What were the reasons
for the author leaving Yahoo in the summer of
1999?', metadata={'doc_id': 'ce4f4981-ca60-
4f56-86f0-89466de62325'})]
```

```
retrieved_docs =
retriever.get_relevant_documents("justice
breyer")
```

```
len(retrieved_docs[0].page_content)
```

```
9194
```