Split Documents

As a starting point on generic text, we recommend the use of LangChain's RecursiveCharacterTextSplitter.

Splitting documents into smaller segments called chunks is an essential step when embedding your data into a vector store. RAG pipelines retrieve relevant chunks to serve as context for the LLM to pull from when generating responses, which makes it important that the retrieved chunks provide the right amount of contextual information to answer the question, and no more than that.

The `RecursiveCharacterTextSplitter` tries to split on a list of characters, in order, until the chunks are small enough. The default list is `["\n\n", "\n", " ", ""]`. This has the effect of trying to keep all paragraphs (and then sentences, and then words) together as long as possible, as those would generically seem to be the strongest semantically related pieces of text.

The LangChain TextSplitters have several parameters that can be tuned:

- `chunk_size`: Determining the best chunk size for your setup can be difficult. Smaller chunk sizes may result in lower retrieval and generation cost by providing fewer tokens to the context window, but the embedding may miss out on broader contextual information. Larger chunk sizes include that contextual information but may produce diluted responses due to unnecessary information being included in the context.
- `chunk_overlap`: Chunk overlap is used to specify the number of overlapping tokens between consecutive chunks. This is useful when splitting text to maintain context continuity between chunks. By including some overlapping tokens, you can ensure that a small portion of context is shared between adjacent chunks, which can help with preserving the meaning and coherence when processing the text with language models.

The `chunk_size` represents the total size of the chunk. (The total size is not `chunk_size + chunk_overlap`.)

For example, if `chunk_size` is 1000 and `chunk_overlap` is 100, then the first 100 tokens will be copied from the end of the previous chunk, and the next 900 tokens will be new.

We recommend starting with a `chunk_size` of 1024 tokens and a `chunk_overlap` of 128. Chunk sizes over 1500 tokens may not work with Astra DB at this time.

Additionally you should make sure to use a length function that splits on tokens and not individual characters. The token splitter should match the embedding model you are using. LangChain includes shortcuts for OpenAI and HuggingFace embedding models:

- OpenAI:

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
    model_name="text-embedding-ada-002",
    chunk_size = 1024,
    chunk_overlap = 128,
)
```

- HuggingFace:

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
splitter = RecursiveCharacterTextSplitter.from_huggingface_tokenizer(
    tokenizer=tokenizer,
    chunk_size = 1024,
    chunk_overlap = 128,
)
```

Split Documents with RecursiveCharacterTextSplitter

In this example we will split Edgar Allan Poe's "The Cask of Amontillado" using the `RecursiveCharacterTextSplitter`.

- We assume that the text will be embedded using OpenAI's `text-embedding-ada-002` model.
- For illustrative purposes, the `chunk_size` and `chunk_overlap` are set to smaller values than recommended.

1. Download a copy of the text from our repository.

```
curl https://raw.githubusercontent.com/CassioML/cassio-website/main/docs/frameworks/l
```

2. Use the `RecursiveCharacterTextSplitter` to split the text into smaller chunks. Output the count of chunks and the contents of the first two chunks. Observe the chunk overlap.

Python | Result

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_core.documents import Document

with open("amontillado.txt") as textfile:
    amontillado = textfile.read()

doc = Document(page_content=amontillado)

splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
    model_name="text-embedding-ada-002",
    chunk_size = 100,
    chunk_overlap = 20,
)

chunks = splitter.split_documents([doc])
```

```python
print(f"Chunk count: {len(chunks)}\n")
print(f"Chunk 1 contents:\n\n{chunks[0].page_content}\n")
print(f"Chunk 2 contents:\n\n{chunks[1].page_content}\n")
```