Retrieval Augmented Generation Internals

Compiled by D. Gueorguiev, 9/20/2025

# Introductory Notes

A typical RAG application has two main components:

*Indexing pipeline*: a pipeline for ingesting data from a source and indexing it. The indexing pipeline works *offline*.

*Retrieval and generation*: the actual RAG chain takes user query at run time and retrieves the relevant data from the index, then passes that to the model.

# The Semantic Search Engine

We will discuss how to build a semantic search engine using LangChain document loader, embedding model, and vector store.

## The Document class

LangChain implements the Document abstraction which represents a unit of text and associated metadata. It has three attributes:

page\_content: a string representing the content

metadata: a dict containing arbitrary metadata

id: (optional) a string identifier for the document

The metadata attribute can capture information about the source of the document, its relationship to other documents, and other information. An individual Document object often represents a chunk of a larger document.

Example: Generating sample documents

from langchain\_core.documents import Document

documents = [

Document(

page\_content="Dogs are great companions, known for their loyalty and friendliness.",

metadata={"source": "mammal-pets-doc"},

),

Document(

page\_content="Cats are independent pets that often enjoy their own space.",

metadata={"source": "mammal-pets-doc"},

),

]

LangChain implements document loaders that integrate with a large set of common sources.

Example: loading a PDF into a sequences of Document objects

from langchain\_community.document\_loaders import PyPDFLoader

file\_path = "../example\_data/nke-10k-2023.pdf"

loader = PyPDFLoader(file\_path)

docs = loader.load()

print(len(docs))

107

PyPDFLoader loads one Document object per PDF page. For each object access is provided for the string content of the page and the metadata containing the file name and page number.

Example:

print(f"{docs[0].page\_content[:200]}\n")

print(docs[0].metadata)

Table of Contents

UNITED STATES

SECURITIES AND EXCHANGE COMMISSION

Washington, D.C. 20549

FORM 10-K

(Mark One)

☑ ANNUAL REPORT PURSUANT TO SECTION 13 OR 15(D) OF THE SECURITIES EXCHANGE ACT OF 1934

FO

{'source': '../example\_data/nke-10k-2023.pdf', 'page': 0}

## Splitting

# References

[1] [Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, Patrick Lewis et al, 2021](https://github.com/dimitarpg13/rag_architectures_and_concepts/blob/main/articles/Retrieval-Augmented_Generation_for_Knowledge-Intensive_NLP_Tasks_Lewis_2021.pdf)

[2] [Foundations of Vector Retrieval, S. Bruch, 2024](https://github.com/dimitarpg13/vector_db_intro/blob/main/articles/Foundations_of_Vector_Retrieval_Bruch_2024.pdf)

[3] [Retrieval-Augmented Generation for Large Language Models: A Survey, Y. Gao et al, 2024](https://github.com/dimitarpg13/rag_architectures_and_concepts/blob/main/articles/Retrieval-Augmented_Generation_for_Large_Language_Models-A_Survey_Gao_2024.pdf)

[4] [Build a Retrieval Augmented Generation (RAG) App: Part 1, LangChain](https://python.langchain.com/docs/tutorials/rag/),

<https://github.com/langchain-ai/langchain/blob/master/docs/docs/tutorials/rag.ipynb>

[5] Tutorial: Build a Semantic Search Engine with LangChain,

<https://python.langchain.com/docs/tutorials/retrievers/>

[6] Document loaders in LangChain: <https://python.langchain.com/docs/concepts/document_loaders/>

[7] Embedding models in LangChain: <https://python.langchain.com/docs/concepts/embedding_models/>

[8] Vector stores in LangChain: <https://python.langchain.com/docs/concepts/vectorstores/>

[9] [Build a Retrieval Augmented Generation (RAG) App: Part 2, LangChain](https://python.langchain.com/docs/tutorials/qa_chat_history/),

<https://github.com/langchain-ai/langchain/blob/master/docs/docs/tutorials/qa_chat_history.ipynb>

[] [Retrieval-Augmented Generation (RAG) from basics to advanced, Tejpal Kumawat, Medium, 2024](https://medium.com/@tejpal.abhyuday/retrieval-augmented-generation-rag-from-basics-to-advanced-a2b068fd576c)

[] [Retrieval-Augmented Generation (RAG) using LangChain, LlamaIndex, and OpenAI, Medium, Prasad Mahamulkar, 2024](https://pub.towardsai.net/introduction-to-retrieval-augmented-generation-rag-using-langchain-and-lamaindex-bd0047628e2a)

# Appendix

## Document loaders in LangChain

Interface: [BaseLoader](https://python.langchain.com/api_reference/core/document_loaders/langchain_core.document_loaders.base.BaseLoader.html)

class langchain\_core.document\_loaders.base.BaseLoader[source]

Interface for Document Loader.

Implementations should implement the lazy-loading method using generators to avoid loading all Documents into memory at once.

load is provided just for user convenience and should not be overridden.

Methods

async alazy\_load() -> AsyncIterator[Document] # A lazy loader for Document

async aload() -> list[Document] # Load data into Document objects.

lazy\_load(): A lazy loader for Documents.

load()

Load data into Document objects.

load\_and\_split([text\_splitter])

Load Documents and split into chunks.