LangChain

Hugging Face

RAG using LangChain and Hugging Face

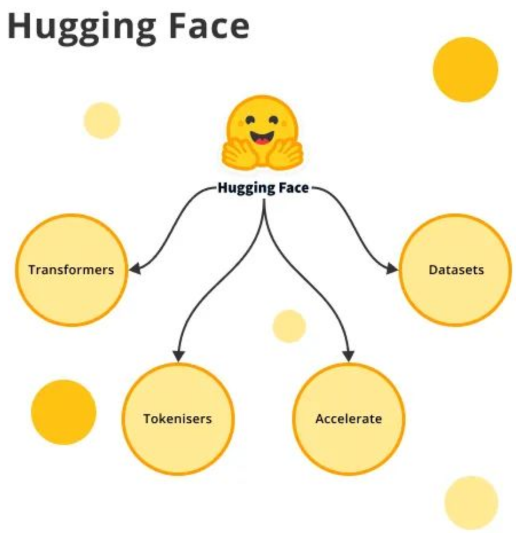
Content sourced from LangChain documentation.
Code examples and slides curated by Ahabb Sheraz.

Table of Contents

Hugging Face

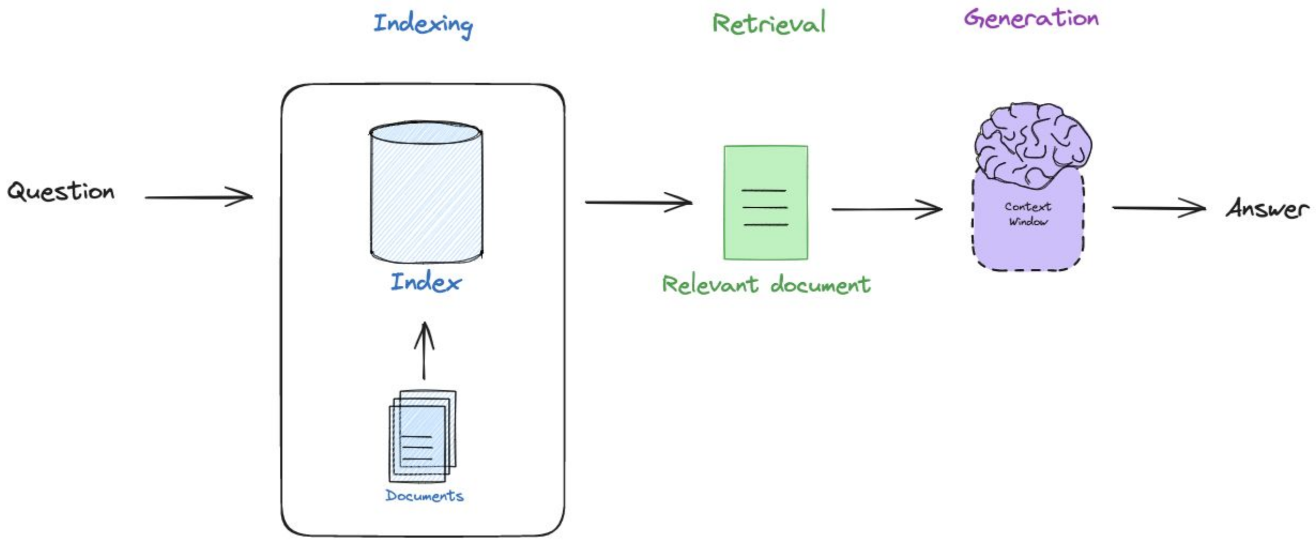
Hugging Face Transformers is an open-source Python library that provides access to thousands of pre-trained Transformers models for natural language processing (NLP), computer vision, audio tasks, and more.

Hugging Face is also popular because of the “Hugging Face hub,” which provides AI/ML researchers with access to thousands of curated datasets, machine learning models, and AI-powered demo apps.



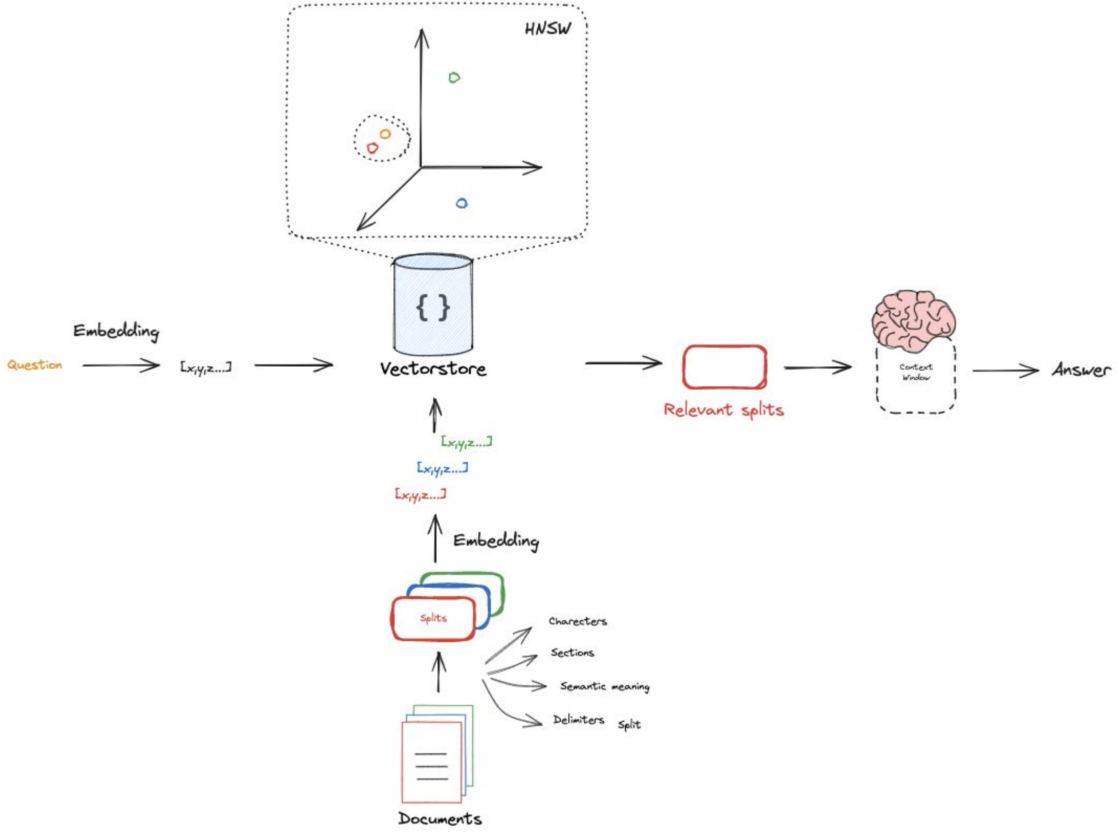
LangChain

Simple RAG Pipeline



LangChain

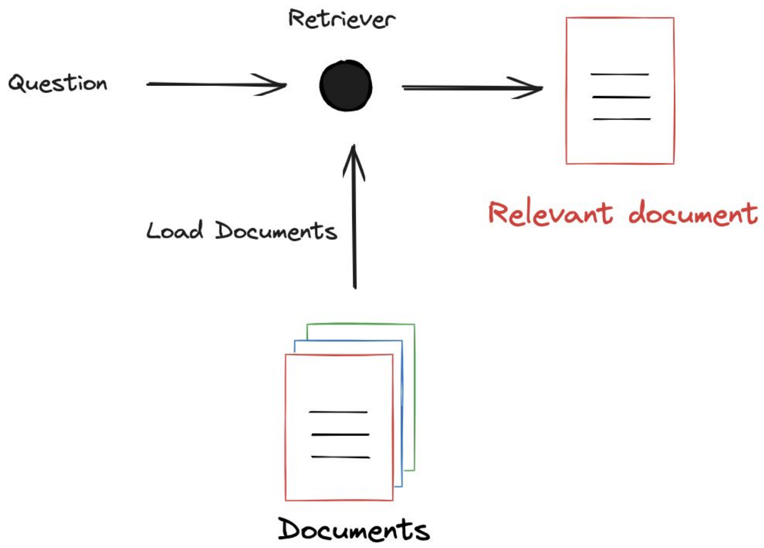
Simple RAG Pipeline (detailed diagram)



LangChain

LangChain is a python framework that contains a collection of tools for interfacing with LLMs. It contains tools that can be used for loading data, splitting text/data, storing splitted sentence embeddings to vector store (data indexing), retrieval from vector store, and prompt templates. LangChain has specialized tools for different RAG-based applications such as QnA bot, AI chatbot, etc…

Document loading



https://integrations.langchain.com/

Langchain

- Data loader
  Use document loaders to load data from a source as Document. A Document is a piece of text and associated metadata.
  - `from langchain_community.document_loaders import TextLoader, PyPDFLoader, CSVLoader`
  - Custom data loader

```
[ ] from langchain_community.document_loaders import PyPDFLoader

    loader = PyPDFLoader("/content/uber_10k.pdf")
    pages = loader.load_and_split()

    pages[0] # content of first page
```

```
Document(metadata={'source': '/content/uber_10k.pdf', 'page': 0}, page_content='2019\nAnnual  \nReport')
```

Page 0

Document loading and extraction (PyPDFLoader)

Here's we'll talk about indexing, which starts with loading documents. LangChain has over 160 different document loaders that you can use to grab data from many different sources for indexing.

```
from langchain_community.document_loaders import PyPDFLoader

file_path = "../example_data/nke-10k-2023.pdf"
loader = PyPDFLoader(file_path)

docs = loader.load()
```

```
print(docs[0].page_content[0:100])
print(docs[0].metadata)
```

```
Table of Contents
UNITED STATES
SECURITIES AND EXCHANGE COMMISSION
Washington, D.C. 20549
FORM 10-K

{'source': '../example_data/nke-10k-2023.pdf', 'page': 0}
```

https://python.langchain.com/v0.1/docs/modules/data_connection/document_loaders/pdf/

Document loading and extraction (custom data loader)

BeautifulSoup library is used for web scraping.

```python
import requests
from bs4 import BeautifulSoup
from langchain_community.document_loaders import WebBaseLoader
from langchain.schema import Document

class BusinessRecorderWebScrapper(WebBaseLoader):
    def load(self):
        response = requests.get(self.web_path)
        soup = BeautifulSoup(response.content, 'html.parser')

        story_title = soup.find(class_="story__title")
        story_excerpt = soup.find(class_="story__excerpt")
        story_content = soup.find(class_="story__content")

        title = story_title.get_text(strip=True) if story_title else ""
        excerpt = story_excerpt.get_text(strip=True) if story_excerpt else ""
        content = story_content.get_text(strip=True) if story_content else ""

        full_content = f"Title: {title}\n\nExcerpt: {excerpt}\n\nContent: {content}"

        return [Document(page_content=full_content, metadata={"source": self.web_path})]

# Use the custom loader
url = "https://www.brecorder.com/news/40324594"
loader = BusinessRecorderWebScrapper(web_path=url)
docs = loader.load()
```

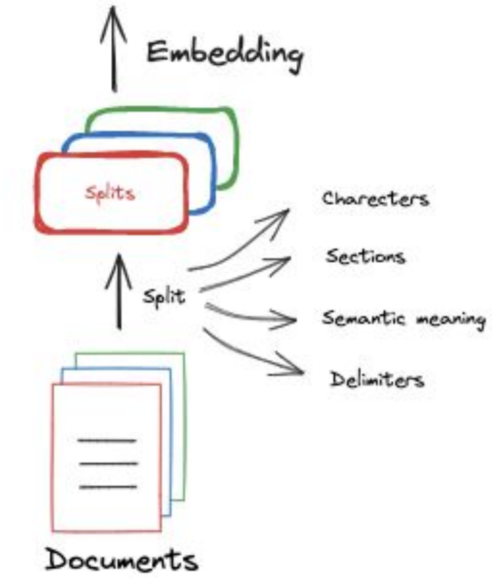
class = "story__title"

class = "story__excerpt"

class = "story__content"

Langchain

- **Document Splitting**
  Once you've loaded documents, you'll often want to transform them to better suit your application. The simplest example is you may want to split a long document into smaller chunks that can fit into your model's context window. LangChain has a number of built-in document transformers that make it easy to split, combine, filter, and otherwise manipulate documents.



Recursively split by character

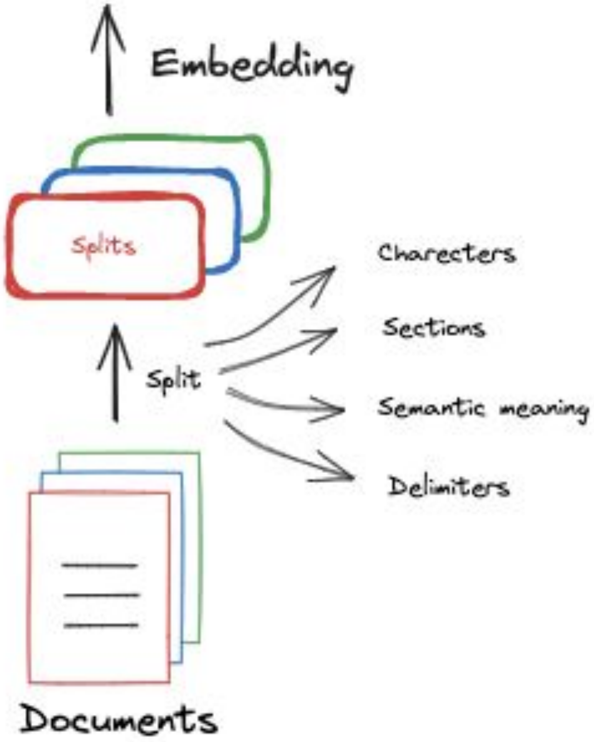
The "recursively split by character" method is a way to break up text into smaller pieces (or chunks) based on certain characters.

The default list for splitting is ["\n\n", "\n", " ", ""]. The size of each chunk is based on the number of characters in it, not the number of words.

https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/
https://python.langchain.com/docs/how_to/recursive_text_splitter/

Langchain

- Document Splitting



Name	Classes	Splits On	Adds Metadata	Description
Recursive	RecursiveCharacterTextSplitter, RecursiveJsonSplitter	A list of user defined characters		Recursively splits text. This splitting is trying to keep related pieces of text next to each other. This is the `recommended way` to start splitting text.
HTML	HTMLHeaderTextSplitter, HTMLSectionSplitter	HTML specific characters	✅	Splits text based on HTML-specific characters. Notably, this adds in relevant information about where that chunk came from (based on the HTML)
Markdown	MarkdownHeaderTextSplitter	Markdown specific characters	✅	Splits text based on Markdown-specific characters. Notably, this adds in relevant information about where that chunk came from (based on the Markdown)
Code	many languages	Code (Python, JS) specific characters		Splits text based on characters specific to coding languages. 15 different languages are available to choose from.
Token	many classes	Tokens		Splits text on tokens. There exist a few different ways to measure tokens.
Character	CharacterTextSplitter	A user defined character		Splits text based on a user defined character. One of the simpler methods.
[Experimental] Semantic Chunker	SemanticChunker	Sentences		First splits on sentences. Then combines ones next to each other if they are semantically similar enough. Taken from Greg Kamradt
AI21 Semantic Text Splitter	AI21SemanticTextSplitter	✅	Identifies distinct topics that form coherent pieces of text and splits along those.	

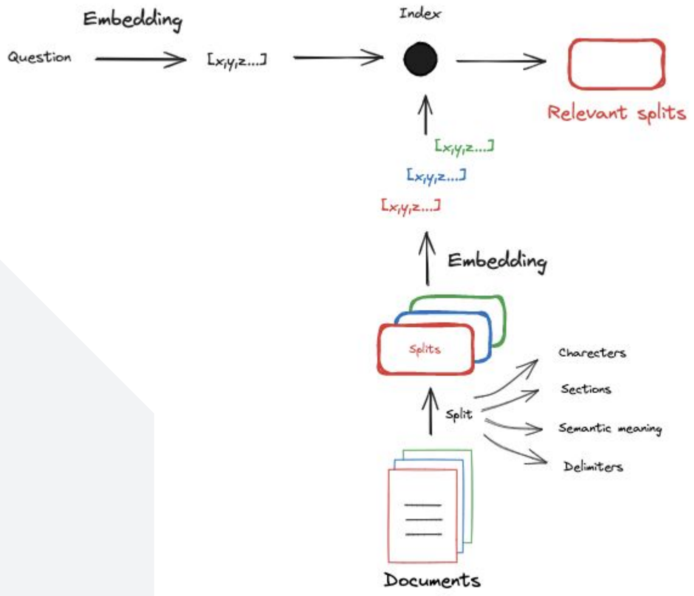
https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/

LangChain

Loading, splitting, and embedding

```python
from langchain import hub
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import Chroma
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough
from langchain_huggingface import HuggingFacePipeline
from sentence_transformers import SentenceTransformer
from langchain.embeddings import HuggingFaceEmbeddings
```

```python
# Load Documents
url = "https://www.brecorder.com/news/40324594"
loader = BusinessRecorderWebScrapper(web_path=url)
docs = loader.load()

# Split
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
splits = text_splitter.split_documents(docs)
print(splits[:5])

# Use model.encode to embed the documents
# Initialize an instance of HuggingFaceEmbeddings with the specified parameters
embeddings = HuggingFaceEmbeddings(
    model_name='sentence-transformers/stsb-xlm-r-multilingual'
)

# Embed, index data, and store it
vectorstore = Chroma.from_documents(documents=splits, embedding=embeddings, persist_directory='db')

retriever = vectorstore.as_retriever()
```



LangChain

Loading, splitting, and embedding

chroma

```
# Post-processing
def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)

# Function to print the formatted context
def print_formatted_context(question):
    context = retriever.get_relevant_documents(question)
    formatted_context = format_docs(context)
    print("\nFormatted context:")
    print(formatted_context[:1000])  # Print first 1000 characters of the context

# Bring context that relate to the word FBR
print_formatted_context('FBR')
```

```
Formatted context:
Content: Finance Minister Muhammad Aurangzeb stated Sunday that Pakistan economy's "DNA needed to be fundamentally ch
anged" to ensure that the latest International Monetary Fund (IMF) agreement is the country's final one. The minister
also stated that Pakistan must declare a "nuclear war" against the cash-based economy as part of the reforms.The mini
ster made the remarks during a news conference in Islamabad, where he, sitting alongside chairman of the Federal Boar
d of Revenue (FBR), outlined many fiscal issues and the government's efforts to address them.At the start of the pres
s conference, he stated that the newly negotiated IMF deal is good news for Pakistan.China, UAE, Saudi Arabia: Pakist
an wins additional financing assurances: IMFThe IMF Executive Board on September 25 approved the 37-month, $7-billion
Extended Fund Facility for Pakistan. The Pakistani authorities and the IMF team reached staff-level agreement on the
EFF in the amount equivalent to SDR 5,320 million (or about
```

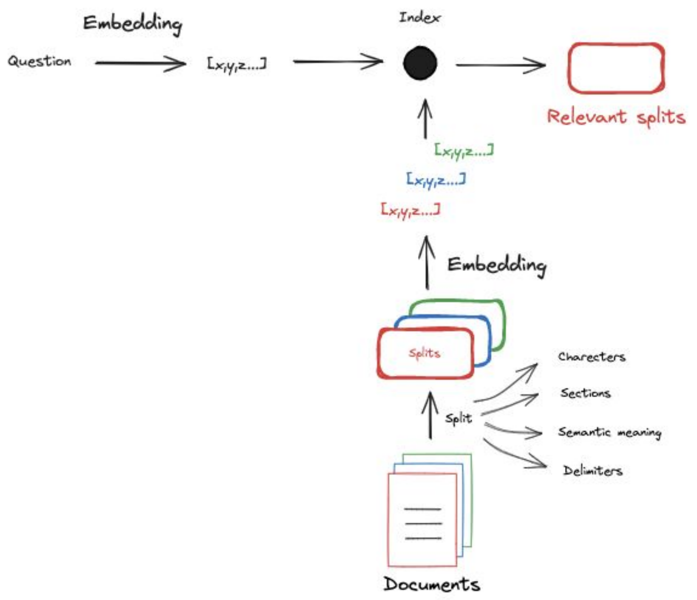
🦜🔗 LangChain

Loading, splitting, and embedding

Pinecone

```python
import getpass
import os
import time
from google.colab import userdata
from pinecone import Pinecone, ServerlessSpec

os.environ["PINECONE_API_KEY"] = userdata.get('Pinecone_API')

pinecone_api_key = os.environ.get("PINECONE_API_KEY")

pc = Pinecone(api_key=pinecone_api_key)
```

```python
# Create the serverless spec (you can adjust replicas, pod_type, etc. as neede
spec = ServerlessSpec(
            cloud='aws',
            region='us-east-1'
        )

# Create the index using the spec
pc.create_index(name='langchain', metric='cosine', dimension=768, spec=spec)

index = pc.Index('langchain')
```



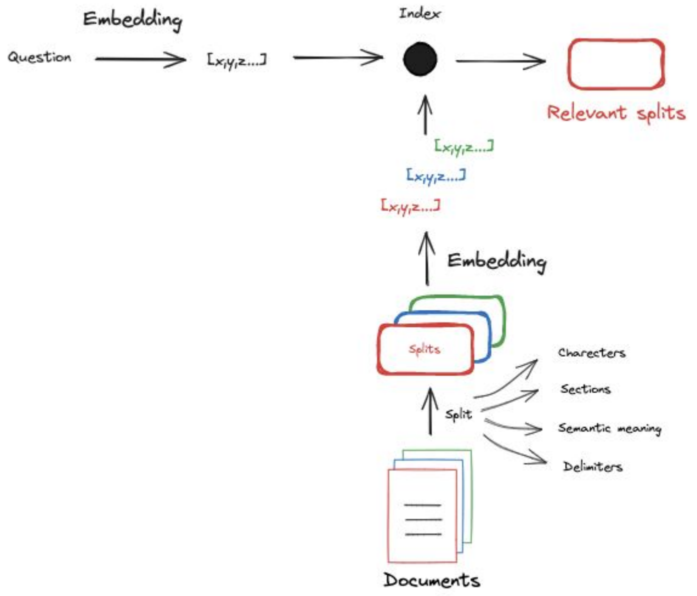
🦜🔗 LangChain

Loading, splitting, and embedding

Pinecone

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import PyPDFLoader

loader = PyPDFLoader("/content/uber_10k.pdf")
doc = loader.load()

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=200
)

texts = text_splitter.split_documents(doc)
```

```
from langchain_pinecone import PineconeVectorStore
from langchain.embeddings import HuggingFaceEmbeddings
from uuid import uuid4

# Initialize an instance of HuggingFaceEmbeddings with the specified parameters
embeddings = HuggingFaceEmbeddings(
    model_name='sentence-transformers/stsb-xlm-r-multilingual'
)

vector_store = PineconeVectorStore(index=index, embedding=embeddings)

uuids = [str(uuid4()) for _ in range(len(texts))] # generate unique ids for each split

vector_store.add_documents(documents=texts, ids=uuids)

retriever = vector_store.as_retriever()
```



🦜🔗 LangChain

Loading, splitting, and embedding

Pinecone

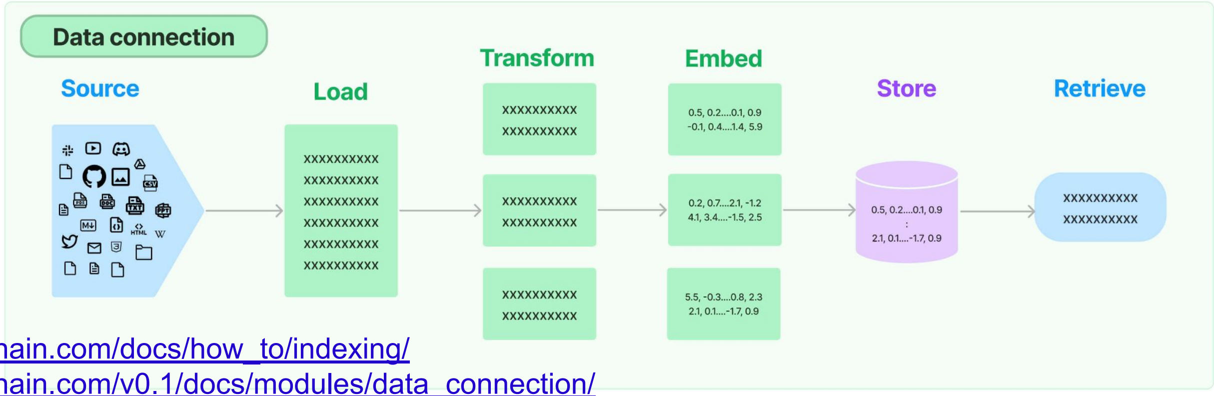
```python
retriever = vector_store.as_retriever()
retriever.get_relevant_documents("What does EBITDA mean?")
```

```
[Document(id='0e5b6db3-2863-46a7-9220-f6e3e270eb7b', metadata={'page': 117.0, 'source': '/content/uber_10k.pdf'}, page_content='Recently Issued Accounting Pronouncements Not Yet Adopted \nIn June 2016, the FASB issued ASU 2016-13, “Financial  Instruments - Credit Losses (Topic 326): Measurement of \nCredit Losses on Financial Instruments” to require the measurement of expected credit losses for financial assets held at the reporting date based on historical experience, current conditions, and reas onable and supportable forecasts. The \nguidance also amends the impairment model for available fo r sale debt securities and requires entities to determine \nwhether all or a portion of the unrealized loss on such debt s ecurity is a credit loss. The standard is effective for public \ncompanies for fiscal years, an d interim periods within those fiscal years, beginning after December 15, 2019. Early'),
 Document(id='fb345cd4-14d6-4928-812b-626c1875fedb', metadata={'page': 116.0, 'source': '/content/uber_10k.pdf'}, page_content='Equity Method Investments for further information. The $9 million difference between the total derecognized assets and \ntotal derecognized liabilities was recorded in the opening balance of  accumulated deficit, net of tax, as of January 1, 2019. \nIn July 2017, the FASB issued ASU 2017-11, “Earnings Per Share (T opic 260); Distinguishing Liabilities from \nEquity (Topic 480); Derivatives and Hedging (Topic 815): (Part I) Accounting for Certain Financial Instruments with Down Round Features, (Part II) Replacement of the Indefinite Deferral for Mandatorily Redeemable Financial \nInstruments of Certain Nonpublic Entities and Certain Manda torily Redeemable Noncontrolling Interests with a Scope \nException” to simplify the accounting fo r certain instruments with down roun d features. The amendments require \ncompanies to disregard the down round feature when assessing whether the instrument is indexed to its own stock, for'),
 Document(id='c61d27d5-ac8e-42d3-bb58-17eb11daf95d', metadata={'page': 155.0, 'source': '/content/uber_10k.pdf'}, page_content='that are not in-substance common stock. As a result, the Grab investment is classified as an available-for-sale debt \nsecurity initially recorded at fair value, with changes in the fair value of the investment recorded in other comprehensive \nincome (loss), net of tax. Refer to Note 3 - Investments and Fair Value Measurement for further information regarding the amortized cost, unrealized holding gains, and fair va lue of the Company’s available-for-sale debt securities. \nThere is significant uncertainty over the collectability of th e contractual interest payable on the Grab investment on \nor after the redemption date due to, among other factors, the reasonable possibility of a Grab  IPO. For these reasons, the \nCompany has not recognized a ny interest income as of December 31, 2 018 and 2019. If the Co mpany had recorded \naccrued interest on the Series G preference shares, approxi mately $102 million and $142 m illion of additional interest'),
 Document(id='f695b8a2-08d6-4959-8cf6-955530326875', metadata={'page': 91.0, 'source': '/content/uber_10k.pdf'}, page_content='88 ITEM 8. FINANCIAL STATEMENTS AND SU PPLEMENTARY DATA \nINDEX TO CONSOLIDATED FINANCIAL STATEMENTS AND SCHEDULE  \n Pages \nReport of Independent Registered Public Accounting Firm 89\nConsolidated Financial Statements \nConsolidated Balance Sheets 90\nConsolidated Statements of Operations 91Consolidated Statements of Comprehensive Income (Loss) 92Consolidated Statements of Mezzanin e Equity and Equity (Deficit) 93\nConsolidated Statements of Cash Flows 96Notes to the Consolidated Financial Statements 98\nFinancial Statement Schedule \nSchedule II - Valuation and Qualif\nying Accounts for the Years Ended Decem ber 31, 2017, 2018 and 2019 154\n \nThe supplementary financial information required by this It em 8 is included in Item 7 under the caption “Selected \nQuarterly Financial Data.”')]
```

Langchain

- Indexing
  The indexing API lets you load and keep in sync documents from any source into a vector store. Specifically, it helps:
  - Avoid writing duplicated content into the vector store
  - Avoid re-writing unchanged content
  - Avoid re-computing embeddings over unchanged content

- Retrieval
  A retriever is an interface that returns documents given an unstructured query. It is more general than a vector store. A retriever does not need to be able to store documents, only to return (or retrieve) them.



https://python.langchain.com/docs/how_to/indexing/
https://python.langchain.com/v0.1/docs/modules/data_connection/

Name	Index Type	Uses an LLM	When to Use	Description
Vectorstore	Vectorstore	No	If you are just getting started and looking for something quick and easy.	This is the simplest method and the one that is easiest to get started with. It creates embeddings for each piece of text.
ParentDocument	Vectorstore + Document Store	No	If your pages have lots of smaller pieces of distinct information that are best indexed by themselves, but best retrieved all together.	This indexes multiple chunks for each document. Then you find the chunks that are most similar in embedding space, but you retrieve the whole parent document and return that (rather than individual chunks).
Multi Vector	Vectorstore + Document Store	Sometimes during indexing	If you are able to extract information from documents that you think is more relevant to index than the text itself.	This creates multiple vectors for each document. Each vector could be created in a myriad of ways - examples include summaries of the text and hypothetical questions.

https://python.langchain.com/v0.1/docs/modules/data_connection/retrievers/

Langchain

- Generation
    - PromptTemplate

`prompts.base.BasePromptTemplate`	Base class for all prompt templates, returning a prompt.
`prompts.base.BasePromptTemplate[ImageURL]`	Base class for all prompt templates, returning a prompt.
`prompts.chat.AIMessagePromptTemplate`	AI message prompt template.
`prompts.chat.BaseChatPromptTemplate`	Base class for chat prompt templates.
`prompts.chat.BaseMessagePromptTemplate`	Base class for message prompt templates.
`prompts.chat.BaseStringMessagePromptTemplate`	Base class for message prompt templates that use a string prompt template.
`prompts.chat.ChatMessagePromptTemplate`	Chat message prompt template.
`prompts.chat.ChatPromptTemplate`	Prompt template for chat models.
`prompts.chat.HumanMessagePromptTemplate`	Human message prompt template.
`prompts.chat.MessagesPlaceholder`	Prompt template that assumes variable is already list of messages.
`prompts.chat.SystemMessagePromptTemplate`	System message prompt template.
`prompts.few_shot.FewShotChatMessagePromptTemplate`	Chat prompt template that supports few-shot examples.
`prompts.few_shot.FewShotPromptTemplate`	Prompt template that contains few shot examples.

https://python.langchain.com/api_reference/core/prompts.html

🦜🔗 LangChain

Prompt Template

```python
from langchain.prompts import ChatPromptTemplate

# Prompt template
template = """You are an assistant for question-answering tasks.
Use the following pieces of retrieved context to answer the question. Keep the answers concise
and avoid filler words. If you don't know the answer, just say that you don't know:

{context}

Question: {question}
"""

prompt = ChatPromptTemplate.from_template(template)
```

LangChain

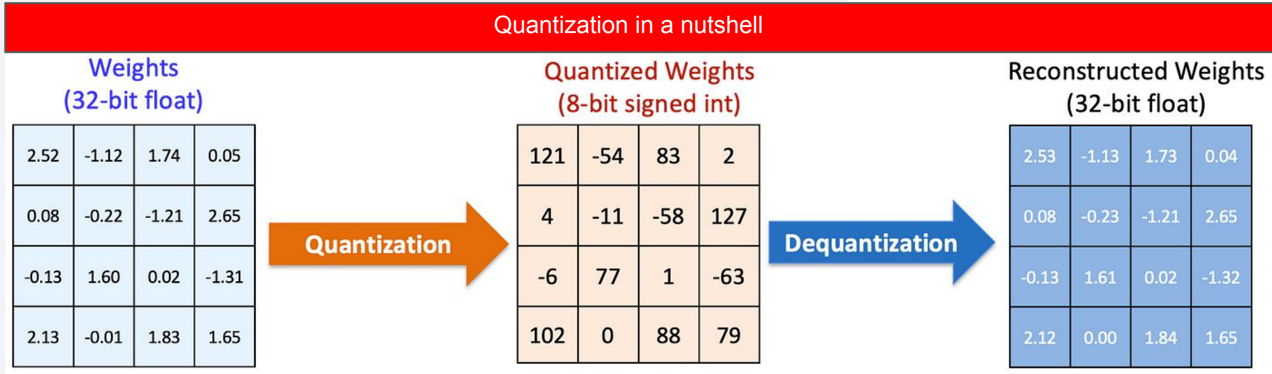
Connecting retrieval with LLMs via prompt (part a)
Integrating LangChain with Hugging Face

Transformer library is being used for loading LLM and its tokenizer from huggingface. Bits and Bytes library is being used to quantize the LLM for easier computing.

```
from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline
import bitsandbytes as bnb
```

Enables quantization in 4 bits.

```
# Choose LLM from huggingface. We are using Llama 3 8b.
model_id = "glaiveai/Llama-3-8B-RAG-v1"
tokenizer = AutoTokenizer.from_pretrained(model_id)
# Quantizing model to save memory resources
model = AutoModelForCausalLM.from_pretrained(model_id, load_in_4bit=True, trust_remote_code=True)

# Create a text-generation pipeline
text_generation = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    max_new_tokens=1000,
    temperature=0.2,
    top_p=0.95,
    repetition_penalty=1.15
)

# Wrap the pipeline in HuggingFacePipeline for compatibility with LangChain
llm = HuggingFacePipeline(pipeline=text_generation)
```



Connecting retrieval with LLMs via prompt (part b)

```
# Post-processing
def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)

# Chain
rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)

# Question
result = rag_chain.invoke('''What is being done to improve the tax system in Pakistan?
                          What is being done to increase tax net?
                          Will the salaried class be less burdened? ''')
print(result)
```

- RunnablePassthrough is a utility in LangChain that allows you to pass input directly through a chain without modifying it.
- Here, RunnablePassthrough() is used for the "question" key, meaning the user query will be passed through the chain unchanged.
- Meanwhile, the "context" will have the top-k most similar chunks processed by the retriever for the user query and then formatted.

LangChain

Connecting retrieval with LLMs via prompt (part b)

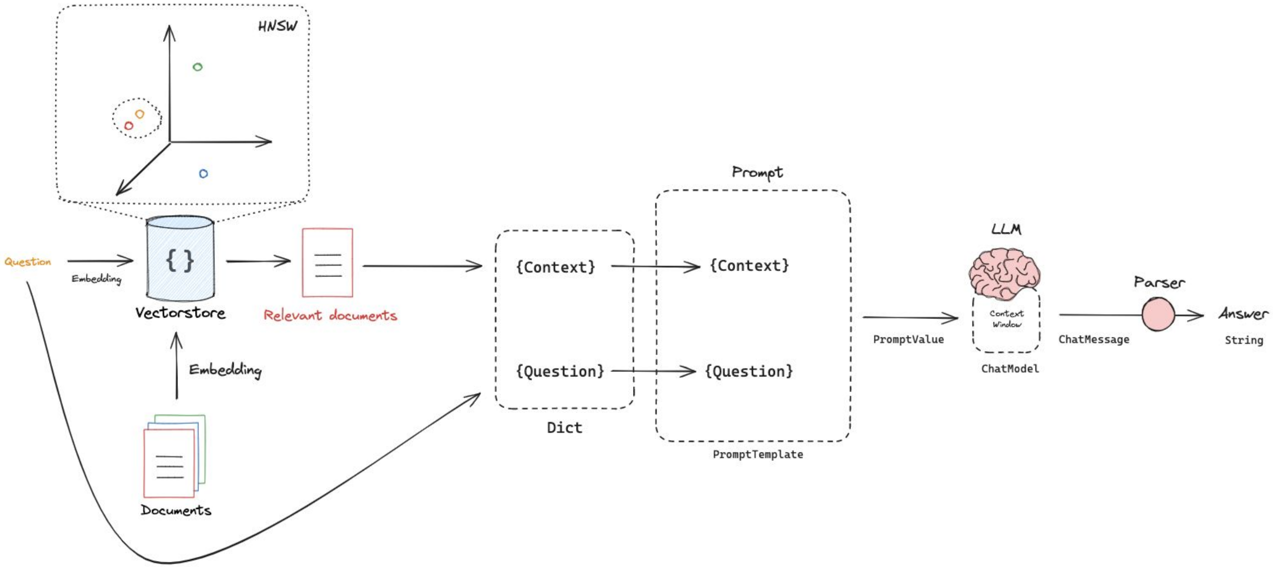
```
# Chain
rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)

# Question
result = rag_chain.invoke('''What is being done to improve the tax system in Pakistan?
                            What is being done to increase tax net?
                            Will the salaried class be less burdened? ''')
print(result)
```

Output

```
Question: What is being done to improve the tax system in Pakistan? What is being done to increase tax net? Will the salaried class be less burdened?
Answer: To improve the tax system, the government plans to employ 2,000 chartered accountants to enhance tax audit capabilities. A new interface for monitoring activit
ies will be developed to prevent harassment by auditors. Independent auditors will investigate and consult with taxpayers. Additionally, the government aims to tap int
o the salaried class and manufacturing sector for taxes, but only if the system is reformed. This suggests that the salaried class may not see significant relief witho
ut systemic changes.
```

LangChain

Connecting retrieval with LLMs via prompt



https://python.langchain.com/docs/expression_language/get_started
https://smith.langchain.com/hub/rlm/rag-prompt?organizationId=1fa8b1f4-fcb9-4072-9aa9-983e35ad61b8