pdf_qa_summary

December 15, 2024

1 Building Robust RAG Systems step by step!

- In this notebook, we'll walk through creating an advanced Retrieval Augmented Generation (RAG) system to intelligently answer questions about building effective RAG solutions.

1.0.1 Things you'll learn

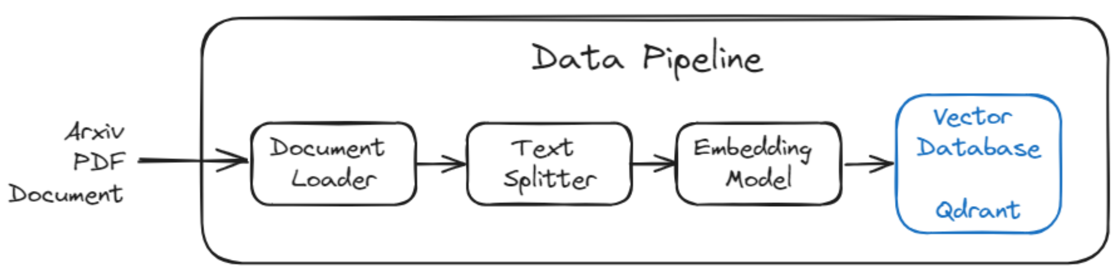
- LangSmith - Set up a crude end-to-end framework to test and evaluate the RAG solution
- Qdrant - initialize a vector store retriever that can run independently from the document loader
- LCEL - the ascii art helps bring this concept home, and confirm that the flow is set-up properly

1.0.2 For next time...

- Some of the key themes I'll do a deep dive on soon are captured in this LangSmith Trace

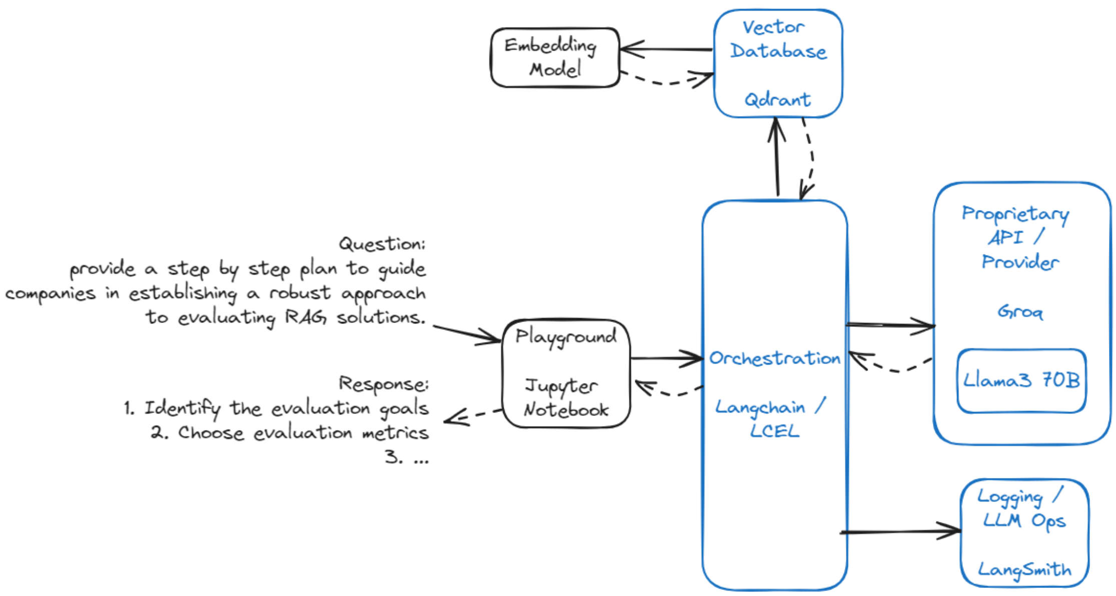
1.1 Loading the Data...

- the items in blue simply show some of my early decisions
- due to the standardization and flexibility of the LangChain APIs I was able to experiment



1.2 Retrieving the Data...

- important to remember to choose the same Embedding Model for the retriever that was used to load the data
- LLM is `gemini-1.5-pro`
- Embedding Model is VertexAI's `text-embedding-004`



1.3 Assembling Our AI Toolkit

[1]:
```python
import os
print(os.path.expanduser("~"))
```

```
/home/jupyter
```

[2]:
```python
%pip install -qU pymupdf
%pip install -qU langchain langchain-core langchain-community␣
↪langchain-experimental langchain-text-splitters
%pip install -q langchain-google-vertexai
%pip install -qU langchain-qdrant
%pip install -qU grandalf
```

```
Note: you may need to restart the kernel to use updated packages.
Note: you may need to restart the kernel to use updated packages.
Note: you may need to restart the kernel to use updated packages.
Note: you may need to restart the kernel to use updated packages.
Note: you may need to restart the kernel to use updated packages.
```

[3]:
```python
import os
import getpass  # Import the getpass module for secure input
```

[4]:
```python
QDRANT_API_KEY = getpass.getpass("Enter your QDRANT_API_KEY key: ")
os.environ["QDRANT_API_KEY"] = QDRANT_API_KEY
```

```
Enter your QDRANT_API_KEY key:  ········
```

[5]:
```python
LANGCHAIN_API_KEY = getpass.getpass("Enter your LANGCHAIN_API_KEY key: ")
os.environ["LANGCHAIN_API_KEY"] = LANGCHAIN_API_KEY
```

```
Enter your LANGCHAIN_API_KEY key:  ········
```

[6]:
```
import os

import vertexai

PROJECT_ID = "[your-project-id]"  # @param {type:"string", isTemplate: true}
if PROJECT_ID == "[your-project-id]":
    PROJECT_ID = str(os.environ.get("GOOGLE_CLOUD_PROJECT"))

LOCATION = os.environ.get("GOOGLE_CLOUD_REGION", "us-central1")

vertexai.init(project=PROJECT_ID, location=LOCATION)
```

[7]:
```
import os
from langchain import hub
from langchain_google_vertexai import ChatVertexAI, VertexAI, VertexAIEmbeddings

llm = ChatVertexAI(model="gemini-1.5-pro", temperature=0)

QDRANT_API_URL = "https://dcc50e13-4537-4069-9b9f-26da8f65900c.us-east4-0.gcp.
↪cloud.qdrant.io"

# LangSmith tracing and
os.environ["LANGCHAIN_PROJECT"] = "Gemini RAG Doc Test"
os.environ["LANGCHAIN_ENDPOINT"] = "https://api.smith.langchain.com"
os.environ["LANGCHAIN_TRACING_V2"] ="true"

# Leverage a prompt from the LangChain hub
LLM_PROMPT = """You are an assistant for question-answering tasks. Use the␣
↪following pieces of retrieved context to answer the question. If you don't␣
↪know the answer, just say that you don't know. Use three sentences maximum␣
↪and keep the answer concise.
Question: {question}
Context: {context}
Answer: """
```

[8]:
```
# Parameterize some stuff

LOAD_NEW_DATA = True

FILE_PATH = "https://arxiv.org/pdf/2309.15217"
# FILE_PATH = "https://arxiv.org/pdf/2405.17813"
# FILE_PATH = "https://arxiv.org/pdf/2406.05085"
# FILE_PATH = "https://arxiv.org/pdf/2212.10496"

COLLECTION_NAME = "rag_collection"
```

```
QUESTION = "provide a step by step plan to guide companies in establishing a␣
↪robust approach to evaluating Retrieval Augmented Generation (RAG) solutions.
↪"
```

1.4 Piecing Together the Perfect RAG System

Building a high-performance RAG system is like solving a complex puzzle. Each piece - the document loader, text splitter, embeddings, and vector store - must be carefully chosen to fit together seamlessly.

In this section, we'll walk through the key implementation choices we've made for each component, and how they contribute to a powerful, efficient, and flexible RAG solution.

1.4.1 Intelligent Document Loading

- **PyMuPDFLoader**: For lightning-fast processing of complex PDFs

[9]:
```
# Document Loader Concepts - https://python.langchain.com/v0.2/docs/concepts/
↪#document-loaders
# PDF: https://python.langchain.com/v0.2/docs/how_to/document_loader_pdf/
# HTML:  https://python.langchain.com/v0.2/docs/how_to/document_loader_html/
# Microsoft Office files:  https://python.langchain.com/v0.2/docs/how_to/
↪document_loader_office_file/
from langchain_community.document_loaders import (
    PyMuPDFLoader,
)
```

[10]:
```
# I chose the PyMuPDFLoader for its speed, ability to handle complex PDFs, and␣
↪more extensive metadata.

DOCUMENT_LOADER = PyMuPDFLoader
```

1.4.2 Strategic Text Splitting

- **RecursiveCharacterTextSplitter**: The smart way to keep related info together

[11]:
```
# Text Splitters concepts - https://python.langchain.com/v0.2/docs/concepts/
↪#text-splitters
# Splitting by Token using HF tokenizers:  https://python.langchain.com/v0.2/
↪docs/how_to/split_by_token/#hugging-face-tokenizer
# Use of RecursiveCharacterTextSplitter to split code - https://python.
↪langchain.com/v0.2/docs/how_to/code_splitter/
from langchain_text_splitters import (
    RecursiveCharacterTextSplitter,
)
```

[12]:
```
# select the text splitter to use
```

```
# worth investigating using the RecursiveCharacterTextSplitter with the
↪length_function based on a tokenizer VS the TokenTextSplitter

TEXT_SPLITTER = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=200
    )
```

1.4.3 Powerful Embeddings

- **VertexAIEmbeddings**: Harnessing the power of cutting-edge language models

[13]:
```
# select the embedding model to use
EMBEDDING_MODEL = VertexAIEmbeddings("text-embedding-004")
```

[14]:
```
# uncomment to validate environment variables
# %whos
```

1.5 Time for New Docs? Let's Check!

The `LOAD_NEW_DATA` flag is a key part of our simple data ingestion pipeline. When set to `True`, it allows the loading of new documents.

1.5.1 Ingesting Fresh Docs: Embracing Adaptability

By using a flag like `LOAD_NEW_DATA`, we can control when new data is ingested without modifying the code itself. This supports rapid experimentation and iteration, as we can test our RAG system with different datasets by simply toggling the flag.

In this case, we're using `PyMuPDFLoader` to load a PDF file, but the beauty of this setup is that we can easily switch to other loaders like `UnstructuredHTMLLoader` for HTML files or `CSVLoader` for CSV data by changing the `DOCUMENT_LOADER` variable. This flexibility is crucial for adapting our pipeline to experiment with various data sources.

[15]:
```
# run loader if LOAD_NEW_DATA is True
if LOAD_NEW_DATA:
    loader = DOCUMENT_LOADER(FILE_PATH)
    docs = loader.load()
```

[16]:
```
# Document Loader validation
if LOAD_NEW_DATA:
    print(f"len(docs): {len(docs)}")
    print(f"\ndocs[0].page_content[0:100]:\n{docs[0].page_content[0:100]}")
    print(f"\ndocs[0].metadata):\n{docs[0].metadata}")

    print(f"\ndocs[1].page_content[0:100]:\n{docs[1].page_content[0:100]}")
    print(f"\ndocs[1].metadata):\n{docs[1].metadata}")

    print(f"\ndocs[-2].page_content[0:100]:\n{docs[-2].page_content[0:100]}")
```

```python
    print(f"\ndocs[-2].metadata):\n{docs[-2].metadata}")

    print(f"\ndocs[-1].page_content[0:100]:\n{docs[-1].page_content[0:100]}")
    print(f"\ndocs[-1].metadata):\n{docs[-1].metadata}")
```

```
len(docs): 8

docs[0].page_content[0:100]:
RAGAS: Automated Evaluation of Retrieval Augmented Generation
Shahul Es†, Jithin James†, Luis Espino

docs[0].metadata):
{'source': 'https://arxiv.org/pdf/2309.15217', 'file_path':
'https://arxiv.org/pdf/2309.15217', 'page': 0, 'total_pages': 8, 'format': 'PDF
1.5', 'title': '', 'author': '', 'subject': '', 'keywords': '', 'creator':
'LaTeX with hyperref', 'producer': 'pdfTeX-1.40.25', 'creationDate':
'D:20230928011700Z', 'modDate': 'D:20230928011700Z', 'trapped': ''}

docs[1].page_content[0:100]:
ment of retrieval augmented generation systems.
We focus on settings where reference answers may
not

docs[1].metadata):
{'source': 'https://arxiv.org/pdf/2309.15217', 'file_path':
'https://arxiv.org/pdf/2309.15217', 'page': 1, 'total_pages': 8, 'format': 'PDF
1.5', 'title': '', 'author': '', 'subject': '', 'keywords': '', 'creator':
'LaTeX with hyperref', 'producer': 'pdfTeX-1.40.25', 'creationDate':
'D:20230928011700Z', 'modDate': 'D:20230928011700Z', 'trapped': ''}

docs[-2].page_content[0:100]:
Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay,
Amnon Shashua, Kevin Leyton-Brown, and Yoav
Shoh

docs[-2].metadata):
{'source': 'https://arxiv.org/pdf/2309.15217', 'file_path':
'https://arxiv.org/pdf/2309.15217', 'page': 6, 'total_pages': 8, 'format': 'PDF
1.5', 'title': '', 'author': '', 'subject': '', 'keywords': '', 'creator':
'LaTeX with hyperref', 'producer': 'pdfTeX-1.40.25', 'creationDate':
'D:20230928011700Z', 'modDate': 'D:20230928011700Z', 'trapped': ''}

docs[-1].page_content[0:100]:
Question
Context
Answer
Who directed the film Op-
penheimer and who stars
```

```
as J. Robert Oppenheimer
i

docs[-1].metadata):
{'source': 'https://arxiv.org/pdf/2309.15217', 'file_path':
'https://arxiv.org/pdf/2309.15217', 'page': 7, 'total_pages': 8, 'format': 'PDF
1.5', 'title': '', 'author': '', 'subject': '', 'keywords': '', 'creator':
'LaTeX with hyperref', 'producer': 'pdfTeX-1.40.25', 'creationDate':
'D:20230928011700Z', 'modDate': 'D:20230928011700Z', 'trapped': ''}
```

1.5.2 Intelligent Text Splitting

Once our data is loaded, the next step is splitting it into manageable chunks. We're using the `RecursiveCharacterTextSplitter` for this, which intelligently splits text while keeping related pieces together.

The splitter works by recursively dividing the text on specified characters (like newlines and periods) until each chunk is within our desired `chunk_size`. The `chunk_overlap` parameter ensures some overlap between chunks to maintain context.

By adjusting these parameters, we can fine-tune the output to suit our specific use case. For example, a larger `chunk_size` results in fewer, longer chunks, while more `chunk_overlap` helps preserve context across chunks.

[17]:
```python
if LOAD_NEW_DATA:
    text_splitter = TEXT_SPLITTER
    splits = text_splitter.split_documents(docs)
```

[18]:
```python
# capture the split chunks for use in the vector store
if LOAD_NEW_DATA:
    print(f"len(splits): {len(splits)}")

    print(f"\nsplits[0]:\n{splits[0]}")
    print(f"\nsplits[1]:\n{splits[1]}")
    print(f"\nsplits[-2]:\n{splits[-2]}")
    print(f"\nsplits[-1]:\n{splits[-1]}")

    for i, split in enumerate(splits):
        print(f"\nSplit # {i}:")
        # print page number from split.metadata

        print(f"split.metadata.get('page'): {split.metadata.get('page')}")
        print(f"len(splits[{i}]): {len(split.page_content)}")
        print(f"splits[{i}][0:25]: {split.page_content[0:25]}")
```

```
len(splits): 43

splits[0]:
page_content='RAGAS: Automated Evaluation of Retrieval Augmented Generation
Shahul Es†, Jithin James†, Luis Espinosa-Anke* , Steven Schockaert*
```

†Exploding Gradients
*CardiffNLP, Cardiff University, United Kingdom
AMPLYFI, United Kingdom
shahules786@gmail.com,jamesjithin97@gmail.com
{espinosa-ankel,schockaerts1}@cardiff.ac.uk
Abstract
We introduce RAGAS (Retrieval Augmented
Generation Assessment), a framework for
reference-free evaluation of Retrieval Aug-
mented Generation (RAG) pipelines.
RAG
systems are composed of a retrieval and an
LLM based generation module, and provide
LLMs with knowledge from a reference textual
database, which enables them to act as a natu-
ral language layer between a user and textual
databases, reducing the risk of hallucinations.
Evaluating RAG architectures is, however, chal-
lenging because there are several dimensions to
consider: the ability of the retrieval system to
identify relevant and focused context passages,
the ability of the LLM to exploit such passages' metadata={'source':
'https://arxiv.org/pdf/2309.15217', 'file_path':
'https://arxiv.org/pdf/2309.15217', 'page': 0, 'total_pages': 8, 'format': 'PDF
1.5', 'title': '', 'author': '', 'subject': '', 'keywords': '', 'creator':
'LaTeX with hyperref', 'producer': 'pdfTeX-1.40.25', 'creationDate':
'D:20230928011700Z', 'modDate': 'D:20230928011700Z', 'trapped': ''}

splits[1]:
page_content='lenging because there are several dimensions to
consider: the ability of the retrieval system to
identify relevant and focused context passages,
the ability of the LLM to exploit such passages
in a faithful way, or the quality of the gener-
ation itself. With RAGAS, we put forward a
suite of metrics which can be used to evaluate
these different dimensions without having to
rely on ground truth human annotations. We
posit that such a framework can crucially con-
tribute to faster evaluation cycles of RAG archi-
tectures, which is especially important given
the fast adoption of LLMs.
1
Introduction
Language Models (LMs) capture a vast amount
of knowledge about the world, which allows them
to answer questions without accessing any exter-
nal sources. This idea of LMs as repositories of

```
knowledge emerged shortly after the introduction
of BERT (Devlin et al., 2019) and became more
firmly established with the introduction of ever
larger LMs (Roberts et al., 2020). While the most' metadata={'source':
'https://arxiv.org/pdf/2309.15217', 'file_path':
'https://arxiv.org/pdf/2309.15217', 'page': 0, 'total_pages': 8, 'format': 'PDF
1.5', 'title': '', 'author': '', 'subject': '', 'keywords': '', 'creator':
'LaTeX with hyperref', 'producer': 'pdfTeX-1.40.25', 'creationDate':
'D:20230928011700Z', 'modDate': 'D:20230928011700Z', 'trapped': ''}

splits[-2]:
page_content='Question
Context
When was the Chimnabai
Clock Tower completed,
and who was it named af-
ter?
High context relevance: The Chimnabai Clock Tower, also known as the Raopura
Tower, is
a clock tower situated in the Raopura area of Vadodara, Gujarat, India. It was
completed
in 1896 and named in memory of Chimnabai I (1864-1885), a queen and the first
wife of
Sayajirao Gaekwad III of Baroda State.
Low context relevance: The Chimnabai Clock Tower, also known as the Raopura
Tower, is
a clock tower situated in the Raopura area of Vadodara, Gujarat, India. It was
completed
in 1896 and named in memory of Chimnabai I (1864-1885), a queen and the first
wife of
Sayajirao Gaekwad III of Baroda State. It was built in Indo-Saracenic
architecture style.
History. Chimnabai Clock Tower was built in 1896. The tower was named after
Chimnabai
I (1864-1885), a queen and the first wife of Sayajirao Gaekwad III of Baroda
State. It was' metadata={'source': 'https://arxiv.org/pdf/2309.15217',
'file_path': 'https://arxiv.org/pdf/2309.15217', 'page': 7, 'total_pages': 8,
'format': 'PDF 1.5', 'title': '', 'author': '', 'subject': '', 'keywords': '',
'creator': 'LaTeX with hyperref', 'producer': 'pdfTeX-1.40.25', 'creationDate':
'D:20230928011700Z', 'modDate': 'D:20230928011700Z', 'trapped': ''}

splits[-1]:
page_content='History. Chimnabai Clock Tower was built in 1896. The tower was
named after Chimnabai
I (1864-1885), a queen and the first wife of Sayajirao Gaekwad III of Baroda
State. It was
inaugurated by Mir Kamaluddin Hussainkhan, the last Nawab of Baroda. During the
rule of
```

```
Gaekwad, it was a stoppage for horse drawn trams. The clock tower was erected at
the cost
of 25,000 (equivalent to 9.2 million or USD 120,000 in 2023).
Table 4: Example from WikiEval, showing answers with high and low context
relevance.' metadata={'source': 'https://arxiv.org/pdf/2309.15217', 'file_path':
'https://arxiv.org/pdf/2309.15217', 'page': 7, 'total_pages': 8, 'format': 'PDF
1.5', 'title': '', 'author': '', 'subject': '', 'keywords': '', 'creator':
'LaTeX with hyperref', 'producer': 'pdfTeX-1.40.25', 'creationDate':
'D:20230928011700Z', 'modDate': 'D:20230928011700Z', 'trapped': ''}

Split # 0:
split.metadata.get('page'): 0
len(splits[0]): 996
splits[0][0:25]: RAGAS: Automated Evaluati

Split # 1:
split.metadata.get('page'): 0
len(splits[1]): 987
splits[1][0:25]: lenging because there are

Split # 2:
split.metadata.get('page'): 0
len(splits[2]): 974
splits[2][0:25]: knowledge emerged shortly

Split # 3:
split.metadata.get('page'): 0
len(splits[3]): 982
splits[3][0:25]: Generation (RAG) (Lee et

Split # 4:
split.metadata.get('page'): 0
len(splits[4]): 974
splits[4][0:25]: a significant amount of t

Split # 5:
split.metadata.get('page'): 0
len(splits[5]): 244
splits[5][0:25]: tative of how the system

Split # 6:
split.metadata.get('page'): 1
len(splits[6]): 968
splits[6][0:25]: ment of retrieval augment

Split # 7:
split.metadata.get('page'): 1
len(splits[7]): 977
```

```
splits[7][0:25]: with detecting hallucinat

Split # 8:
split.metadata.get('page'): 1
len(splits[8]): 980
splits[8][0:25]: they essentially convert

Split # 9:
split.metadata.get('page'): 1
len(splits[9]): 995
splits[9][0:25]: factual, we can expect th

Split # 10:
split.metadata.get('page'): 1
len(splits[10]): 983
splits[10][0:25]: a particular aspect of th

Split # 11:
split.metadata.get('page'): 1
len(splits[11]): 797
splits[11][0:25]: the availability of one o

Split # 12:
split.metadata.get('page'): 2
len(splits[12]): 970
splits[12][0:25]: we usually do not have ac

Split # 13:
split.metadata.get('page'): 2
len(splits[13]): 971
splits[13][0:25]: tion that was provided. F

Split # 14:
split.metadata.get('page'): 2
len(splits[14]): 960
splits[14][0:25]: made in the answer can be

Split # 15:
split.metadata.get('page'): 2
len(splits[15]): 962
splits[15][0:25]: c(q) using a verification

Split # 16:
split.metadata.get('page'): 2
len(splits[16]): 969
splits[16][0:25]: an appropriate way. In pa

Split # 17:
```

```
split.metadata.get('page'): 2
len(splits[17]): 263
splits[17][0:25]: answer aligns with the in

Split # 18:
split.metadata.get('page'): 3
len(splits[18]): 991
splits[18][0:25]: inclusion of redundant in

Split # 19:
split.metadata.get('page'): 3
len(splits[19]): 974
splits[19][0:25]: need examples of question

Split # 20:
split.metadata.get('page'): 3
len(splits[20]): 961
splits[20][0:25]: given context satisfying

Split # 21:
split.metadata.get('page'): 3
len(splits[21]): 997
splits[21][0:25]: tion from the given conte

Split # 22:
split.metadata.get('page'): 3
len(splits[22]): 625
splits[22][0:25]: the question and correspo

Split # 23:
split.metadata.get('page'): 4
len(splits[23]): 969
splits[23][0:25]: Faith.
Ans. Rel.
Cont. Re

Split # 24:
split.metadata.get('page'): 4
len(splits[24]): 959
splits[24][0:25]: with the answer/context p

Split # 25:
split.metadata.get('page'): 4
len(splits[25]): 998
splits[25][0:25]: in the answer that cannot

Split # 26:
split.metadata.get('page'): 4
```

```
len(splits[26]): 953
splits[26][0:25]: vancy.
question: [questio

Split # 27:
split.metadata.get('page'): 4
len(splits[27]): 832
splits[27][0:25]: ticular, we have argued t

Split # 28:
split.metadata.get('page'): 5
len(splits[28]): 973
splits[28][0:25]: References
Amos Azaria an

Split # 29:
split.metadata.get('page'): 5
len(splits[29]): 967
splits[29][0:25]: of Machine Learning Resea

Split # 30:
split.metadata.get('page'): 5
len(splits[30]): 972
splits[30][0:25]: Liu. 2023. Gptscore: Eval

Split # 31:
split.metadata.get('page'): 5
len(splits[31]): 958
splits[31][0:25]: Ganguli, Danny Hernandez,

Split # 32:
split.metadata.get('page'): 5
len(splits[32]): 964
splits[32][0:25]: Omar Khattab, Keshav Sant

Split # 33:
split.metadata.get('page'): 5
len(splits[33]): 970
splits[33][0:25]: ral Information Processin

Split # 34:
split.metadata.get('page'): 5
len(splits[34]): 538
splits[34][0:25]: ume 1: Long Papers), page

Split # 35:
split.metadata.get('page'): 6
len(splits[35]): 965
```

```
splits[35][0:25]: Ori Ram, Yoav Levine, Ita

Split # 36:
split.metadata.get('page'): 6
len(splits[36]): 956
splits[36][0:25]: Zhou. 2023a. Is chatgpt a

Split # 37:
split.metadata.get('page'): 6
len(splits[37]): 954
splits[37][0:25]: Fang, Luc Gaitskell, Thom

Split # 38:
split.metadata.get('page'): 6
len(splits[38]): 350
splits[38][0:25]: cessing (EMNLP-IJCNLP), p

Split # 39:
split.metadata.get('page'): 7
len(splits[39]): 998
splits[39][0:25]: Question
Context
Answer
W

Split # 40:
split.metadata.get('page'): 7
len(splits[40]): 946
splits[40][0:25]: Robert Oppenheimer in the

Split # 41:
split.metadata.get('page'): 7
len(splits[41]): 921
splits[41][0:25]: Question
Context
When was

Split # 42:
split.metadata.get('page'): 7
len(splits[42]): 501
splits[42][0:25]: History. Chimnabai Clock
```

1.5.3 Blazing-Fast Vector Stores

- **Qdrant**: The high-performance, scalable choice for demanding workloads

With our text split into manageable chunks, it's time to vectorize and store them for fast retrieval. That's where Qdrant comes in - a vector database that offers performance, scalability, and flexibility.

Qdrant utilizes the HNSW algorithm for blazing-fast similarity search, delivering up to 4x higher requests per second compared to alternatives. Its advanced compression features reduce memory usage by up to 97%, while its flexible storage options allow us to fine-tune for our specific needs.

But Qdrant isn't just fast - it's also incredibly versatile. With support for hybrid search (combining vector similarity and filtering), sparse vectors, and rich JSON payloads, Qdrant enables powerful querying patterns that go beyond simple similarity search.

[19]:
```python
from langchain_qdrant import QdrantVectorStore

# Store the chunks in Qdrant
if LOAD_NEW_DATA:
    from_splits = QdrantVectorStore.from_documents(
        documents=splits,
        embedding=EMBEDDING_MODEL,
        url=QDRANT_API_URL,
        prefer_grpc=True,
        api_key=QDRANT_API_KEY,
        collection_name=COLLECTION_NAME,
        force_recreate=True
    )
```

1.6 Implementing a Robust Vector Store Retriever

- depends on the "Initialize the Vector Store client" section above

[20]:
```python
from langchain_qdrant import RetrievalMode

qdrant = QdrantVectorStore.from_documents(
    documents=splits,
    embedding=EMBEDDING_MODEL,
    url=QDRANT_API_URL,
    prefer_grpc=True,
    api_key=QDRANT_API_KEY,
    collection_name=COLLECTION_NAME,
    retrieval_mode=RetrievalMode.DENSE,
)
```

[21]:
```python
# Concepts:  https://python.langchain.com/v0.2/docs/concepts/#retrievers
# Vector Store as Retriever:  https://python.langchain.com/v0.2/docs/how_to/
 ↪vectorstore_retriever/
# Including Similarity Search Scores:  https://python.langchain.com/v0.2/docs/
 ↪how_to/add_scores_retriever/

retriever = qdrant.as_retriever(
    search_type="similarity_score_threshold",
    search_kwargs={"score_threshold": 0.5}
)
```

1.7 Constructing the RAG Chain for Question Answering

[22]:
```python
from langchain.prompts import PromptTemplate

prompt_template = PromptTemplate(
    input_variables=["question", "context"],
    template=LLM_PROMPT,
)
```

[23]:
```python
from operator import itemgetter
from langchain.schema.runnable import RunnablePassthrough

retrieval_augmented_qa_chain = (
    {"context": itemgetter("question") | retriever, "question":␣
 ↪itemgetter("question")}
    | RunnablePassthrough.assign(context=itemgetter("context"))
    | {"response": prompt_template | llm, "context": itemgetter("context")}
)
```

[24]:
```
!pip install grandalf
```

```
Requirement already satisfied: grandalf in /opt/conda/lib/python3.10/site-
packages (0.8)
Requirement already satisfied: pyparsing in /opt/conda/lib/python3.10/site-
packages (from grandalf) (3.2.0)
```

[25]:
```python
print(retrieval_augmented_qa_chain.get_graph().draw_ascii())
```

```
        +---------------------------------+
        | Parallel<context,question>Input |
        +---------------------------------+
                  **            **
                **                **
              **                    **
       +--------+                     **
       | Lambda |                      *
       +--------+                      *
            *                          *
            *                          *
            *                          *
+------------------------+          +--------+
| VectorStoreRetriever   |          | Lambda |
+------------------------+          +--------+
                  **            **
                    **        **
                      **    **
        +----------------------------------+
        | Parallel<context,question>Output |
        +----------------------------------+
```

```
                         *
                         *
                         *
           +------------------------+
           | Parallel<context>Input |
           +------------------------+
                  ***        ***
                 *              *
               **                **
        +--------+          +-------------+
        | Lambda |          | Passthrough |
        +--------+          +-------------+
                  ***        ***
                     *      *
                      **  **
           +-------------------------+
           | Parallel<context>Output |
           +-------------------------+
                         *
                         *
                         *
       +---------------------------------+
       | Parallel<response,context>Input |
       +---------------------------------+
                ***            ***
              **                  ***
            **                       **
+----------------+                     **
| PromptTemplate |                      *
+----------------+                      *
         *                              *
         *                              *
         *                              *
 +--------------+                  +--------+
 | ChatVertexAI |                  | Lambda |
 +--------------+                 *+--------+
                ***            ***
                   **        **
                     **    **
       +----------------------------------+
       | Parallel<response,context>Output |
       +----------------------------------+
```

1.8 Moment of Truth: Testing Our RAG System!

[26]:
```
response = retrieval_augmented_qa_chain.invoke({"question" : QUESTION})
```

[27]:
```python
# return the response.  filter on the response key AIMessage content element
response["response"].content
```

[27]:
```
'This document focuses on introducing RAGAS, a framework for evaluating RAG
solutions, but it does not provide guidance for companies to establish a robust
approach for this purpose. Therefore, I cannot answer your question using the
provided context. \n'
```

[28]:
```python
print(response["response"].content)
```

```
This document focuses on introducing RAGAS, a framework for evaluating RAG
solutions, but it does not provide guidance for companies to establish a robust
approach for this purpose. Therefore, I cannot answer your question using the
provided context.
```

1.8.1 Thanks to LangSmith, this custom code is no longer required

```python
for i, context_instance in enumerate(response["context"]):
  print(f"\nvector store CONTEXT # {i}:")
  print(f"Page # : {context_instance.metadata.get('page')}")
  print(f"context.page_content:\n{context_instance.page_content}")
  print(f"context.metadata:\n{context_instance.metadata}")
```