Building a Local RAG Architecture

By
Pooja Dayanand Kabadi – 50575012

Contents

RAG Architecture

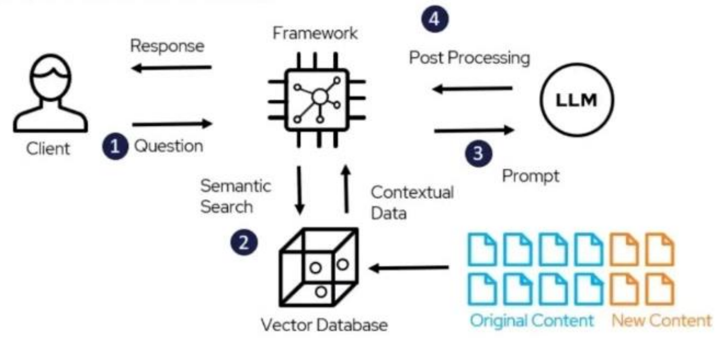


Figure 1 Retrieval Augmented Generation Architecture

Building a Retrieval Augmented Generation (RAG) system was an enlightening journey that began with the relatively straightforward use of Open WebUI from the second assignment and culminated in coding a RAG system from scratch. This process provided a comprehensive understanding of how RAG systems work and their potential applications.

The initial challenge was setting up the required libraries. Dependency conflicts, particularly with tokenizers and transformers versions, caused some headaches. Troubleshooting pip install commands and configuring the environment properly took considerable effort, teaching valuable lessons in package management and version compatibility.

When it came to coding, setting up the ChromaDB vector database was relatively simple. However, getting the embedding model to work smoothly required some trial and error. Error messages about missing modules and request timeouts were common obstacles that needed to be overcome through careful parameter adjustments. This process deepened the understanding of how vector databases and embedding models function within a RAG system.

Creating the knowledge base with multiple PDF documents was another significant task. Selecting and processing the right files took time but ultimately contributed to a more robust system. This step highlighted the importance of curating relevant information for effective RAG performance.

One of the more complex aspects was customizing prompts and adjusting the top-k parameter. While not overly complicated, it required thorough research of documentation and experimentation to achieve optimal results. This experience provided insights into fine-tuning RAG systems for specific use cases.

Transitioning from an ephemeral to a persistent database was surprisingly smooth, which was a relief given the complexity of other parts of the project. This step demonstrated the importance of data persistence in practical RAG applications.

The assignment also introduced key concepts such as vector databases, embedding models, and the advantages they offer over traditional keyword-based searches. Learning about semantic similarity and how it enhances the retrieval process was particularly enlightening.

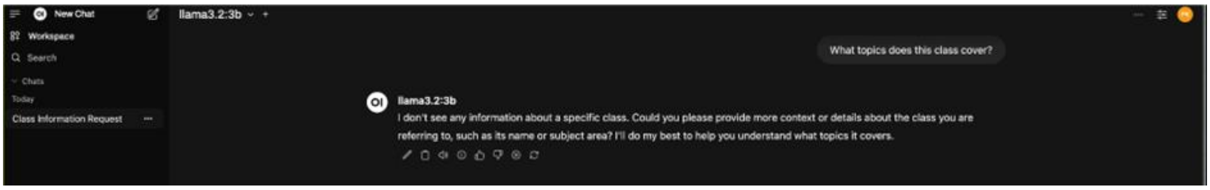
Working with libraries like llama-index and Hugging Face's transformers provided hands-on experience with state-of-the-art NLP tools. Understanding how these components interact to create a functional RAG system was both challenging and rewarding.

Despite the challenges, seeing the RAG system come together and produce relevant answers based on the custom knowledge base was incredibly satisfying. The project provided valuable hands-on experience in working with cutting-edge AI technologies and demonstrated the power of combining large language models with custom knowledge bases.

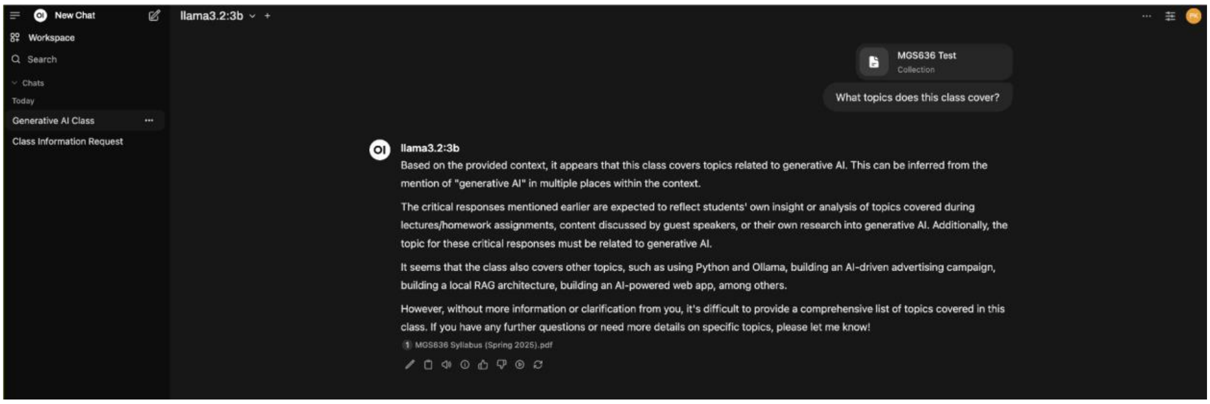
Overall, this assignment offered a comprehensive learning experience in AI system development, from troubleshooting technical issues to fine-tuning system parameters for optimal performance. It highlighted the potential of RAG systems in enhancing AI applications with domain-specific knowledge and provided a solid foundation for further exploration in this rapidly evolving field.

RAG Using Open-WebUI

1. Without the Knowledge Base

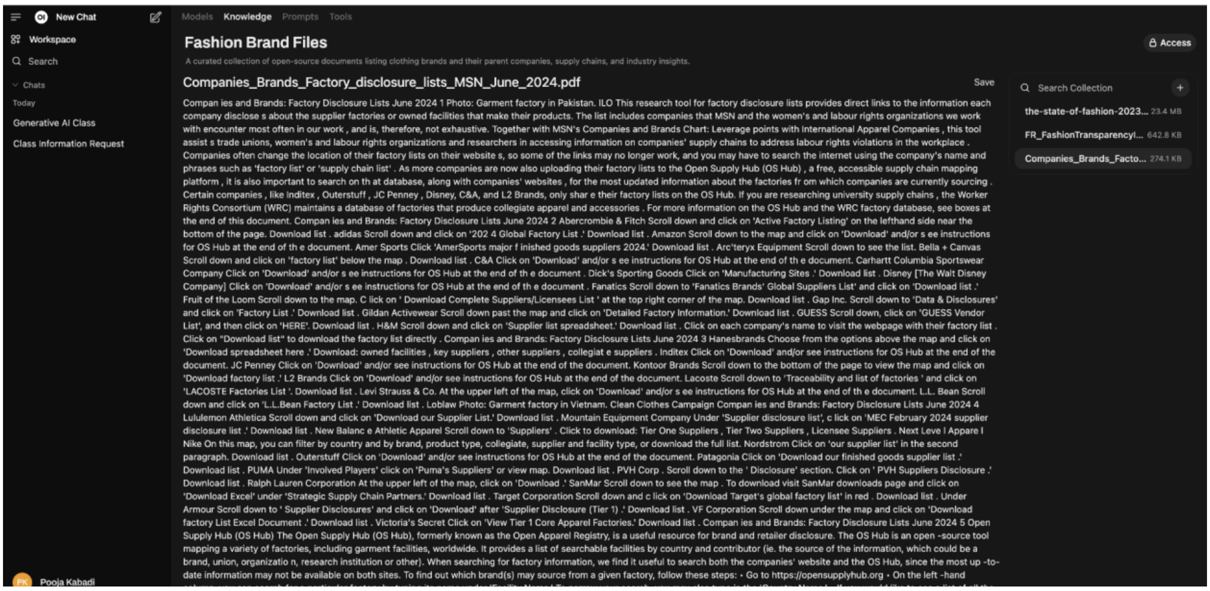


2. With the Knowledge Base

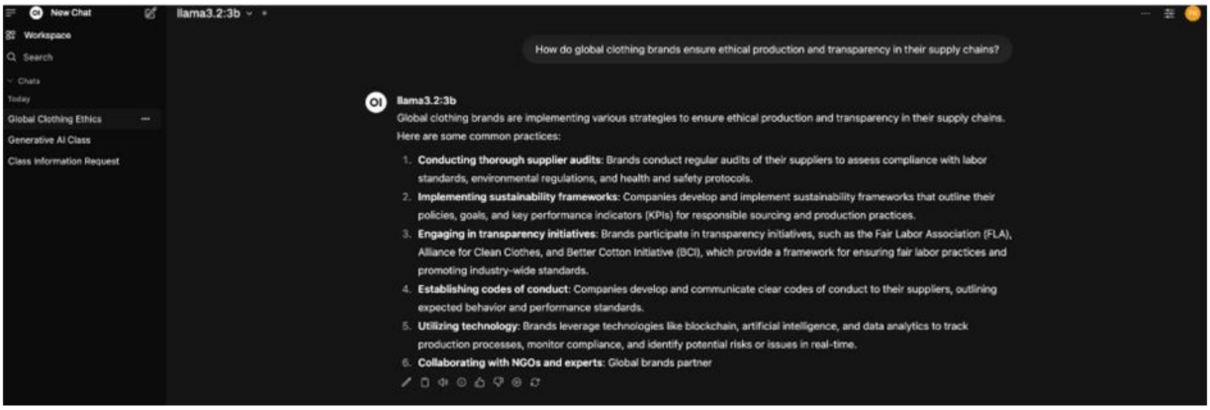


Aspect	Without Collection	With Collection
Response Content	No information provided about specific class	Detailed information about class topics (generative AI, Python, Ollama, AI projects)
Response Length	Very brief	Longer and more informative
Confidence	Uncertain, asks for more context	Confident, provides specific details
Source Indication	None	Shows "MG5636 Test" collection tag
Performance	Poor - unable to answer query	Good - provides relevant and specific information

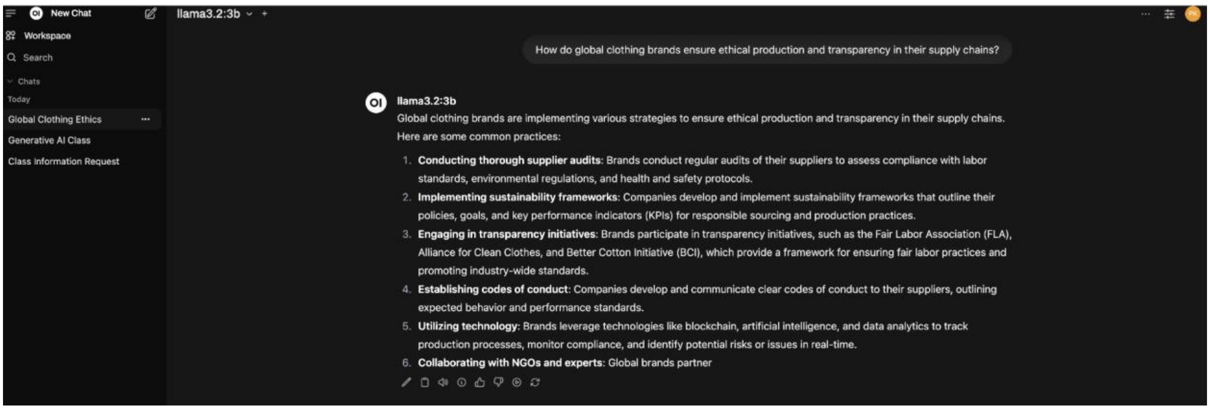
3. Knowledge Collection with 3 Documents



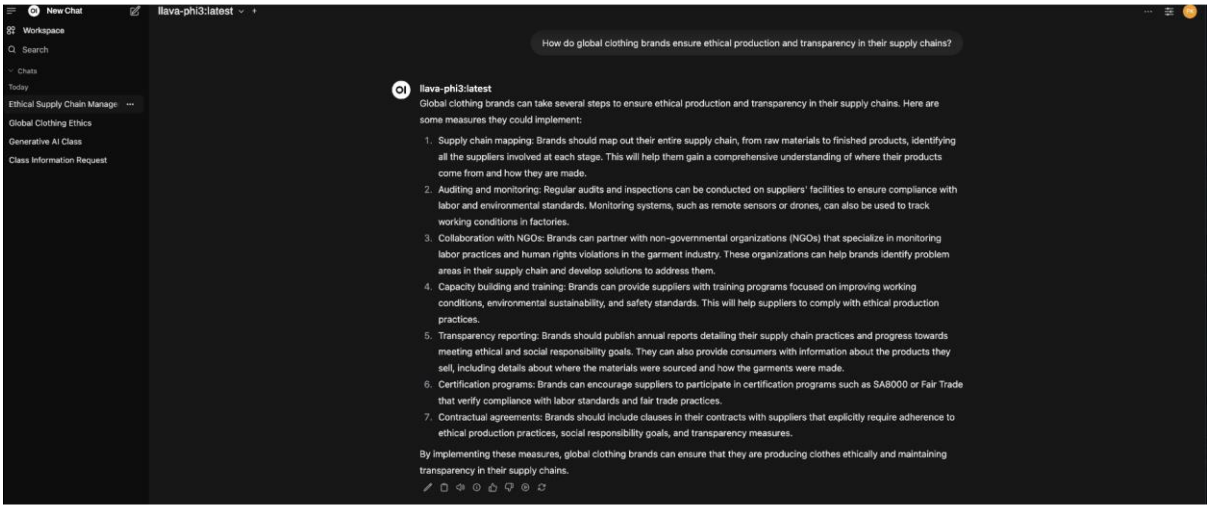
4. Without Knowledge Collection



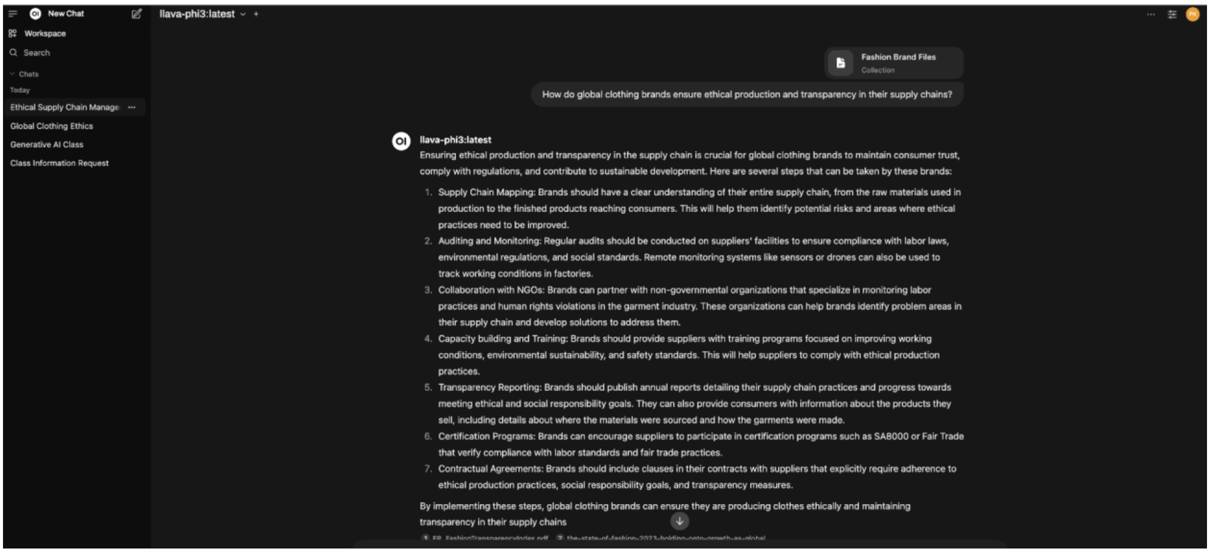
5. With Knowledge Collection



6. Trying the Same Response with Different LLM without Knowledge Collection



7. Trying the Same Response with Different LLM with Knowledge Collection

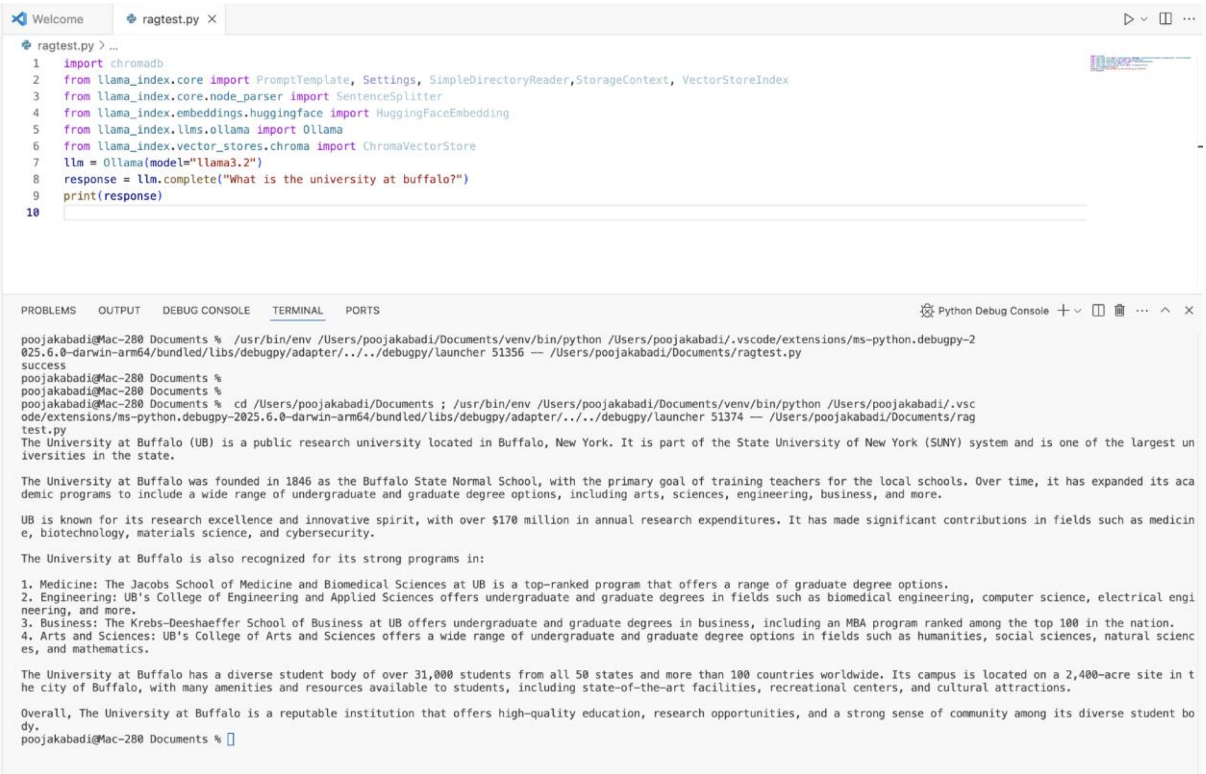


Looking at the outputs the differences suggest that the Llama-3.2 model may be more sensitive to additional context provided by knowledge collections, while Phi 3 model maintains a more consistent output regardless of additional knowledge sources for this prompt.

RAG Using Python

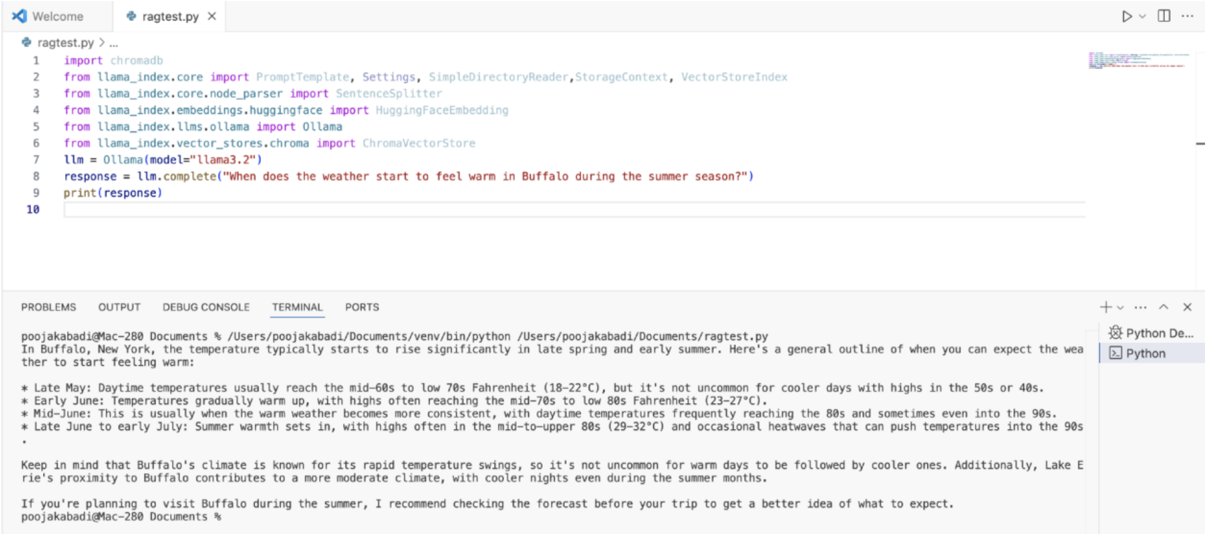
To production-ready for an enterprise, consumer facing application, including RAG give us more flexibility for which we would be creating our own RAG to meet our needs.

1. Basic RAG run for the 1st time using Llama3.2 model

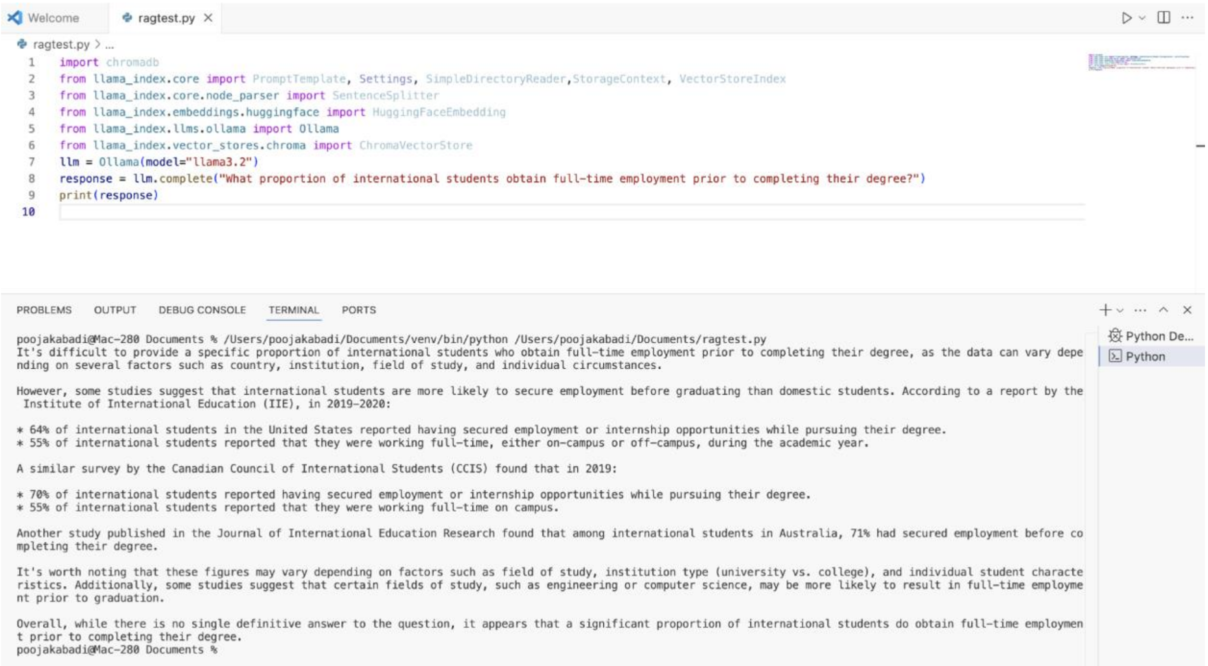


2. Custom prompt 1

3. Custom prompt 2



4. Custom prompt 3



To gain a comprehensive understanding of various Large Language Models (LLMs), conducted experiments using different models through Open-WebUI. Building on our previous experience with the Phi-3 LLM, we now aim to evaluate its performance in a scripted environment. This approach allows us to compare the capabilities of different models, including Phi-3, in both interactive web interfaces and programmatic implementations. By doing so, we can assess how these models perform across different contexts and use cases, providing valuable insights into their strengths, limitations, and potential applications in various scenarios.

5. Basic RAG run for the 1st time using Phi3 model

ragtest.py > ...

```python
import chromadb
from llama_index.core import PromptTemplate, Settings, SimpleDirectoryReader,StorageContext, VectorStoreIndex
from llama_index.core.node_parser import SentenceSplitter
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.llms.ollama import Ollama
from llama_index.vector_stores.chroma import ChromaVectorStore
llm = Ollama(model="phi3")
response = llm.complete("What is the university at buffalo?"")
print(response)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
poojakabadi@Mac-280 Documents % /Users/poojakabadi/Documents/venv/bin/python /Users/poojakabadi/Documents/ragtest.py
The University of Buffalo, also known as SUNY Buffalo or UB for short, is a public research university in Amherst and Buffalo, New York. It was established on April
17th, 1962 following the merger between Adelphi College (founded in 1874) and University of Buffalo Academy of Fine Arts (established in 1937). With campuses located
 across both Amherst County and Erie County near Niagara Falls, UB offers a wide array of undergraduate programs as well as graduate degrees. Notably known for its C
ollege of Arts & Sciences with strengths in business administration, social welfare and policy studies among other disciplines. The university also boasts the Newhou
se School which is dedicated to journalism education along with various research centers including those focused on cancer treatment and environmental sustainability
. As an institution within the State University of New York system (SUNY), UB offers affordable tuition rates, generous financial aid opportunities as well as many s
tudy-abroad programs worldwide. The university's alumni include a range of professionals across various fields including academia, entertainment and technology secto
rs among others.

It's important to note that while the University of Buffalo is sometimes referred to in shorthand or colloquial terms as UB or SUNY Buffalo within specific contexts
especially local ones (such as sports), when referring specifically to its academic programs and research, it's best known by its full name.
poojakabadi@Mac-280 Documents %
```

6. Simple RAG pipeline

ragtest.py > ...

```python
import chromadb
from llama_index.core import PromptTemplate, Settings, SimpleDirectoryReader,StorageContext, VectorStoreIndex
from llama_index.core.node_parser import SentenceSplitter
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.llms.ollama import Ollama
from llama_index.vector_stores.chroma import ChromaVectorStore
llm =None
Settings.llm=Ollama(model="llama3.2", request_timeout=360.0)
chroma_client = chromadb.EphemeralClient()
chroma_collection = chroma_client.create_collection("mgs636test")
embed_model = HuggingFaceEmbedding(model_name="BAAI/bge-small-en")
Settings.embed_model = embed_model

documents = SimpleDirectoryReader("./data/").load_data()
vector_store =ChromaVectorStore(chroma_collection=chroma_collection)
storage_context =StorageContext.from_defaults(vector_store=vector_store)
index = VectorStoreIndex.from_documents(documents,storage_context=storage_context, embed_model=embed_model)
query_engine = index.as_query_engine(llm=Settings.llm)
response = query_engine.query("What topics are covered in this class?")
print(response)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
poojakabadi@Mac-280 Documents % /Users/poojakabadi/Documents/venv/bin/python /Users/poojakabadi/Documents/ragtest.py
The class covers Generative AI, with topics including local AI, LLMs (Large Language Models), RAG architecture for knowledge augmentation, and building AI-powered we
b apps.
poojakabadi@Mac-280 Documents %
```

7. Efficient RAG pipeline and generated folder contents

ragcreate.py > ...

```python
import chromadb
from llama_index.core import PromptTemplate, Settings, SimpleDirectoryReader,StorageContext, VectorStoreIndex
from llama_index.core.node_parser import SentenceSplitter
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.llms.ollama import Ollama
from llama_index.vector_stores.chroma import ChromaVectorStore
from chromadb import PersistentClient
llm =None
Settings.llm=Ollama(model="llama3.2", request_timeout=360.0)
chroma_client = chromadb.PersistentClient(path="./chroma_db")
chroma_collection = chroma_client.create_collection("mgs636test")
embed_model = HuggingFaceEmbedding(model_name="BAAI/bge-small-en")
Settings.embed_model = embed_model

documents = SimpleDirectoryReader("./data/").load_data()
vector_store =ChromaVectorStore(chroma_collection=chroma_collection)
storage_context =StorageContext.from_defaults(vector_store=vector_store)
index = VectorStoreIndex.from_documents(documents,storage_context=storage_context, embed_model=embed_model)
query_engine = index.as_query_engine(llm=Settings.llm)
response = query_engine.query("What topics are covered in this class?")
print(response)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
poojakabadi@Mac-280 Documents % /Users/poojakabadi/Documents/venv/bin/python /Users/poojakabadi/Documents/ragmodifytest.py
Generative AI is a primary focus of this class, with topics including local AI, building AI-driven advertising campaigns, building a local RAG architecture for knowledge augmentation, and AI-powered web apps. Additionally, students will be expected to complete one critical response on topics related to generative AI throughout the semeste
r.
poojakabadi@Mac-280 Documents %
```

poojakabadi

Back/Forward View Group Share Add Tags Action Search

Name	Date Modified	Size	Kind
Documents	Today at 2:31 AM	--	Folder
chroma_db	Today at 5:10 AM	--	Folder
8a073cc3-c630-434c-ac31-7e3d21281e33	Today at 1:34 AM	--	Folder
data_level0.bin	Today at 1:56 AM	16.8 MB	MacBin...archive
header.bin	Today at 1:56 AM	100 bytes	MacBin...archive
length.bin	Today at 1:56 AM	40 KB	MacBin...archive
link_lists.bin	Today at 1:56 AM	Zero bytes	MacBin...archive
39c589b8-908a-43ca-b211-6da3a480bb56	Today at 2:53 AM	--	Folder
data_level0.bin	Today at 5:09 AM	16.8 MB	MacBin...archive
header.bin	Today at 5:09 AM	100 bytes	MacBin...archive
index_metadata.pickle	Today at 5:09 AM	386 KB	Document
length.bin	Today at 5:09 AM	40 KB	MacBin...archive
link_lists.bin	Today at 5:09 AM	36 KB	MacBin...archive
chroma.sqlite3	Today at 5:10 AM	106.2 MB	Document

8. Running the created RAG pipeline

ragrun.py > ...

```python
import chromadb
from llama_index.core import PromptTemplate, Settings, SimpleDirectoryReader,StorageContext, VectorStoreIndex
from llama_index.core.node_parser import SentenceSplitter
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.llms.ollama import Ollama
from llama_index.vector_stores.chroma import ChromaVectorStore
from chromadb import PersistentClient
llm =None
Settings.llm=Ollama(model="llama3.2", request_timeout=360.0)
chroma_client = chromadb.PersistentClient(path="./chroma_db")
chroma_collection = chroma_client.get_or_create_collection("mgs636test")
embed_model = HuggingFaceEmbedding(model_name="BAAI/bge-small-en")
Settings.embed_model = embed_model

vector_store =ChromaVectorStore(chroma_collection=chroma_collection)
storage_context =StorageContext.from_defaults(vector_store=vector_store)
index = VectorStoreIndex.from_vector_store(
    vector_store, storage_context=storage_context
)
query_engine = index.as_query_engine(llm=Settings.llm)
response = query_engine.query("What topics are covered in this class?")
print(response)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
poojakabadi@Mac-280 Documents % /Users/poojakabadi/Documents/venv/bin/python /Users/poojakabadi/Documents/ragrun.py
This class focuses on Generative AI, covering topics such as local AI, using Python and Ollama to build a first local LLM interface, building an AI-driven advertising cam
paign using LLMs, knowledge augmentation, and more. Students will be expected to complete critical responses that reflect their insight or analysis of these topics.
poojakabadi@Mac-280 Documents %
```

9. Custom RAG pipeline

ragcreateown.py > ...

```python
import chromadb
import os
from llama_index.core import PromptTemplate, Settings, SimpleDirectoryReader, StorageContext, VectorStoreIndex
from llama_index.core.node_parser import SentenceSplitter
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.llms.ollama import Ollama
from llama_index.vector_stores.chroma import ChromaVectorStore
from chromadb import PersistentClient
from llama_index.readers.file import PDFReader
pdf_reader = PDFReader()
input_dir = "data"
pdf_files = [f for f in os.listdir(input_dir) if f.endswith(".pdf")]
print(f"Total PDF files: {len(pdf_files)}\n")
doc_summaries = []
documents=[]
for pdf_file in pdf_files:
    file_path = os.path.join(input_dir, pdf_file)
    docs = pdf_reader.load_data(file_path)
    num_pages = len(docs)
    doc_summaries.append((pdf_file, num_pages))
    documents.extend(docs)
for name, pages in doc_summaries:
    print(f"{name}: {pages} page(s)")
Settings.llm = Ollama(model="llama3.2", request_timeout=360.0)
Settings.embed_model = HuggingFaceEmbedding(model_name="BAAI/bge-small-en")
chroma_client = chromadb.PersistentClient(path="./chroma_db")
chroma_collection = chroma_client.get_or_create_collection("mgs636test")
vector_store = ChromaVectorStore(chroma_collection=chroma_collection)
storage_context = StorageContext.from_defaults(vector_store=vector_store)
index = VectorStoreIndex.from_documents(documents, storage_context=storage_context)
query_engine = index.as_query_engine(llm=Settings.llm)
response = query_engine.query("What does each document says? Help us with the summary of each document.")
print(f"\n")
print(response)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
poojakabadi@Mac-280 Documents % /Users/poojakabadi/Documents/venv/bin/python /Users/poojakabadi/Documents/ragcreateown.py
Total PDF files: 3

Companies_Brands_Factory_disclosure_lists_MSN_June_2024.pdf: 5 page(s)
FR_FashionTransparencyIndex.pdf: 20 page(s)
the-state-of-fashion-2023-holding-onto-growth-as-global-clouds-gathers-vf.pdf: 144 page(s)


It seems that the first page (page 20) is a list with two main categories.

The "GLOBAL" section appears to be discussing the overall fashion industry and its current state. Unfortunately, this information cannot be provided as it is not present
in the given context.

The "REGIONAL REALITIES" section has two subheadings: "ECONOMY". The content of this page is unclear without more information.
poojakabadi@Mac-280 Documents %
```

Link for the screen recording [Screen Recording of Custom RAG pipeline]

10. Custom RAG pipeline with custom top_k and prompting

ragcreateown.py > ...

```python
import chromadb
import os
from llama_index.core import PromptTemplate, Settings, SimpleDirectoryReader, StorageContext, VectorStoreIndex
from llama_index.core.node_parser import SentenceSplitter
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.llms.ollama import Ollama
from llama_index.vector_stores.chroma import ChromaVectorStore
from chromadb import PersistentClient
from llama_index.readers.file import PDFReader
pdf_reader = PDFReader()
input_dir = "data"
pdf_files = [f for f in os.listdir(input_dir) if f.endswith(".pdf")]
print(f"Total PDF files: {len(pdf_files)}\n")
doc_summaries = []
documents=[]
for pdf_file in pdf_files:
    file_path = os.path.join(input_dir, pdf_file)
    docs = pdf_reader.load_data(file_path)
    num_pages = len(docs)
    doc_summaries.append((pdf_file, num_pages))
    documents.extend(docs)
for name, pages in doc_summaries:
    print(f"{name}: {pages} page(s)")
Settings.llm = Ollama(model="llama3.2", request_timeout=360.0)
Settings.embed_model = HuggingFaceEmbedding(model_name="BAAI/bge-small-en")
chroma_client = chromadb.PersistentClient(path="./chroma_db")
chroma_collection = chroma_client.get_or_create_collection("mgs636test")
vector_store = ChromaVectorStore(chroma_collection=chroma_collection)
storage_context = StorageContext.from_defaults(vector_store=vector_store)
index = VectorStoreIndex.from_documents(documents, storage_context=storage_context)
custom_prompt = PromptTemplate(
    "You are a helpful assistant. Based on the following context, summarize what each document in the knowledge base discusses.Focus on key themes, insigh
) #For custom prompt template
query_engine = index.as_query_engine(llm=Settings.llm,similarity_top_k=5,text_qa_template=custom_prompt) #top_k is set to 5 to give us the most relavant i
response = query_engine.query("What does each document says? Help us with the summary of each document.")
print(f"\n")
print(response)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
poojakabadi@Mac-280 Documents % /Users/poojakabadi/Documents/venv/bin/python /Users/poojakabadi/Documents/ragcreateown.py
Total PDF files: 3

Companies_Brands_Factory_disclosure_lists_MSN_June_2024.pdf: 5 page(s)
FR_FashionTransparencyIndex.pdf: 20 page(s)
the-state-of-fashion-2023-holding-onto-growth-as-global-clouds-gathers-vf.pdf: 144 page(s)


Based on the context, here's a summary of what each document in the knowledge base is expected to discuss:

1. The State of Fashion 2023:
   - Key theme: Global fashion industry trends and challenges.
   - Insights: Analysis of the impact of global economic fluctuations on the fashion industry.
   - Notable statistics or trends: Expected discussion of the current state of the fashion industry, including factors such as sustainability, consumer behavior, and e-co
mmerce growth.

2. GLOBAL FRAGILITY:
   - Key theme: The vulnerabilities of the global fashion supply chain.
   - Insights: Discussion of potential risks to global trade and economic stability in the fashion industry.
   - Notable statistics or trends: Possible mention of disruptions caused by pandemics, natural disasters, or other factors affecting the global economy.

3. REGIONAL REALITIES:
   - Key theme: Regional differences within the global fashion industry.
   - Insights: Analysis of regional market trends, consumer behavior, and supply chain challenges in various parts of the world.
   - Notable statistics or trends: Expected discussion of regional fashion markets, including growth rates, consumer spending patterns, and supply chain performance.

4. GLOBAL ECONOMY:
   - Key theme: The impact of global economic conditions on the fashion industry.
   - Insights: Discussion of how global economic fluctuations affect demand for fashion products, supply chain logistics, and retailer profitability.
   - Notable statistics or trends: Expected mention of GDP growth rates, inflation, interest rates, and other macroeconomic indicators affecting the fashion industry.

Please note that these summaries are based on the provided context and may not be exhaustive.
poojakabadi@Mac-280 Documents %
```

Bonus Point

Upon reviewing the code, it becomes evident that the SentenceSplitter library remains unused. This library, a node parser from LlamaIndex, is designed to break down lengthy documents into smaller, meaningful segments prior to indexing. Such segmentation enhances retrieval accuracy and response quality through several means:

- It creates more semantically coherent chunks
- It prevents token overflow issues
- It ensures the model doesn't conflate unrelated content

To implement this library effectively, we could have employed the following approach:

First, we would define the chunk size:

sentence_splitter = SentenceSplitter(chunk_size=512, chunk_overlap=50)

Next, we would apply this chunking method to our loaded documents:

nodes = sentence_splitter.get_nodes_from_documents(documents)

Finally, we would pass these nodes to the index:

index = VectorStoreIndex(nodes, storage_context=storage_context)

This implementation would potentially improve the overall performance and accuracy of our document processing system.