Advanced Generative AI: Building LLM Applications

RAG with LangChain

Quick Recap

- Understanding how Vector stores store and retrieve vectorized text is crucial, as they are key to retrieving relevant information during the RAG process.
- LangChain Retrievers fetch relevant documents or data, which RAG models use to generate accurate and context-aware responses.

Engage and Think

Imagine you're building a system that needs to provide instant, accurate answers—like a super-smart search engine for your users. Traditional models often struggle to keep up, but with RAG, you can change that.

RAG uses advanced techniques to quickly break down complex information and retrieve the most relevant details, ensuring users get precise, context-aware responses. Want to see how RAG can solve real-world challenges? Discover how it can power anything from smart assistants to dynamic content generation, making your systems faster, smarter, and more effective. How will you use RAG to transform user experiences.

Learning Objectives

By the end of this lesson, you will be able to:

- Apply knowledge of traditional generative models to compare their limitations with RAG-based approaches
- Utilize text-splitting techniques to prepare documents for efficient processing in RAG models
- Implement vector stores and embedding models to facilitate contextually relevant information retrieval in RAG systems
- Integrate retrievers and LLMs in RAG pipelines to enhance question-answering tasks with accurate and context-rich responses

Challenges with Traditional Generative Models

Traditional Generative Models

Large pre-trained language models, such as GPT-3, generate text by leveraging patterns and information from the data on which they are trained.

Limitations

- **Hallucination**: May generate plausible but incorrect information
- **Accuracy issues:** Responses may be outdated or lack specificity
- **Static knowledge:** Knowledge is fixed after training and cannot be updated dynamically

RAG-Based Model

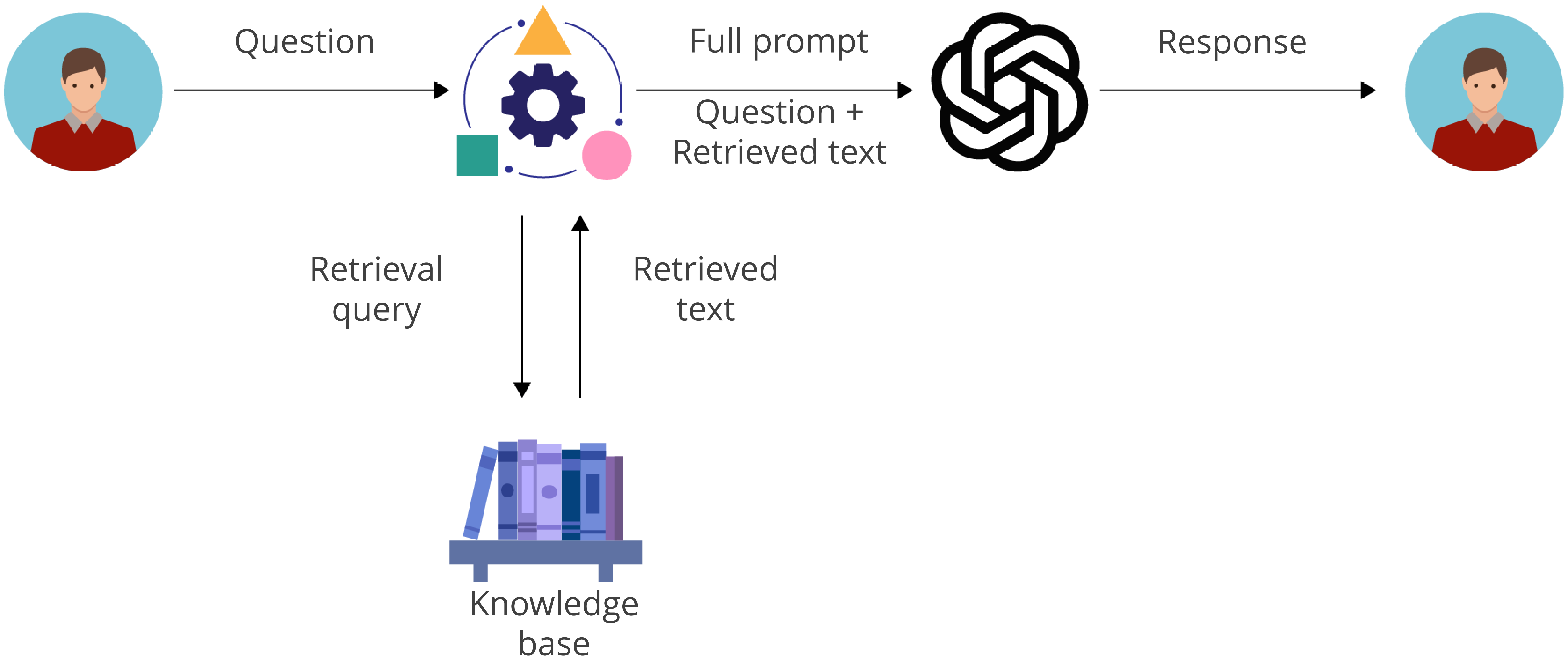
These models combine the retrieval of relevant documents with generative capabilities to produce more accurate and contextually rich responses.

Strengths

- **Reduced hallucination:** Uses retrieved documents to ground responses in information
- **Improved accuracy:** Ensures more accurate responses with access to up-to-date and specific information
- **Dynamic knowledge:** Integrates new information dynamically from external sources

Introduction to RAG

RAG is a technique that enables a large language model (LLM) to generate enriched responses by augmenting a user's prompt with supporting data retrieved from an external knowledge base.



About Retrieval Augmentation Generation (RAG)

RAG combines elements of retrieval-based and generative models to enhance AI's capabilities.

RAG leverages both retrieval and generation components as a framework.

RAG integrates the strengths of models like BERT for retrieval and GPT for generation.

RAG aims to address the limitations of purely generative or retrieval-based models.

Quick Check

Which of the following is a key challenge associated with traditional generative models?

A. Inability to generate coherent text

B. Lack of factual accuracy in responses

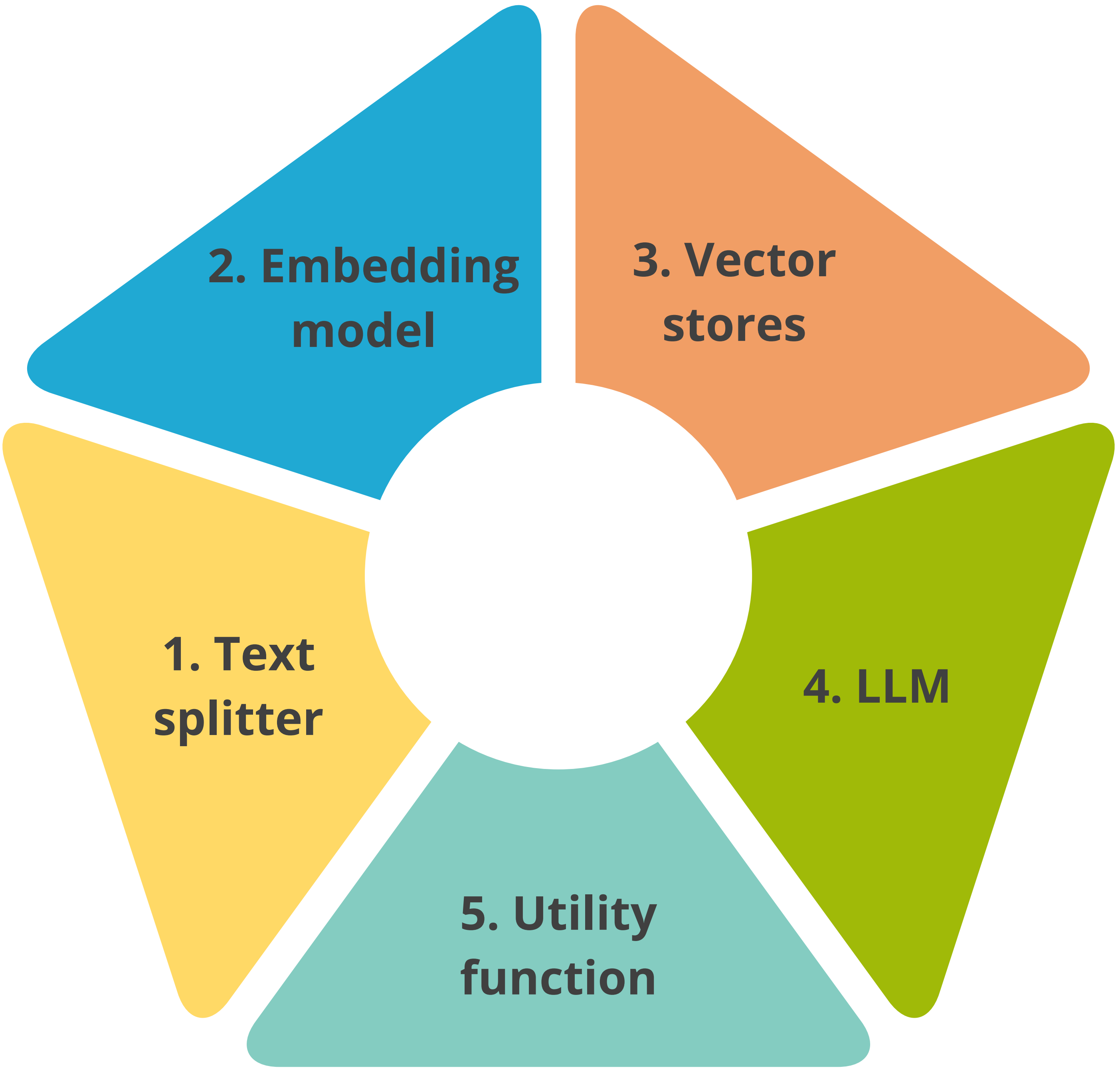
C. Difficulty in processing text chunks

D. Integration with vector stores

Components of RAG

Components of RAG

RAG consists of three main components:



Components of RAG

1. Text splitter

Splits documents into smaller sections to fit the context windows of large language models (LLMs)

2. Embedding model

Uses deep learning to generate embeddings for documents

Components of RAG

3. Vector stores

Stores and queries document embeddings and their metadata in databases

4. LLM

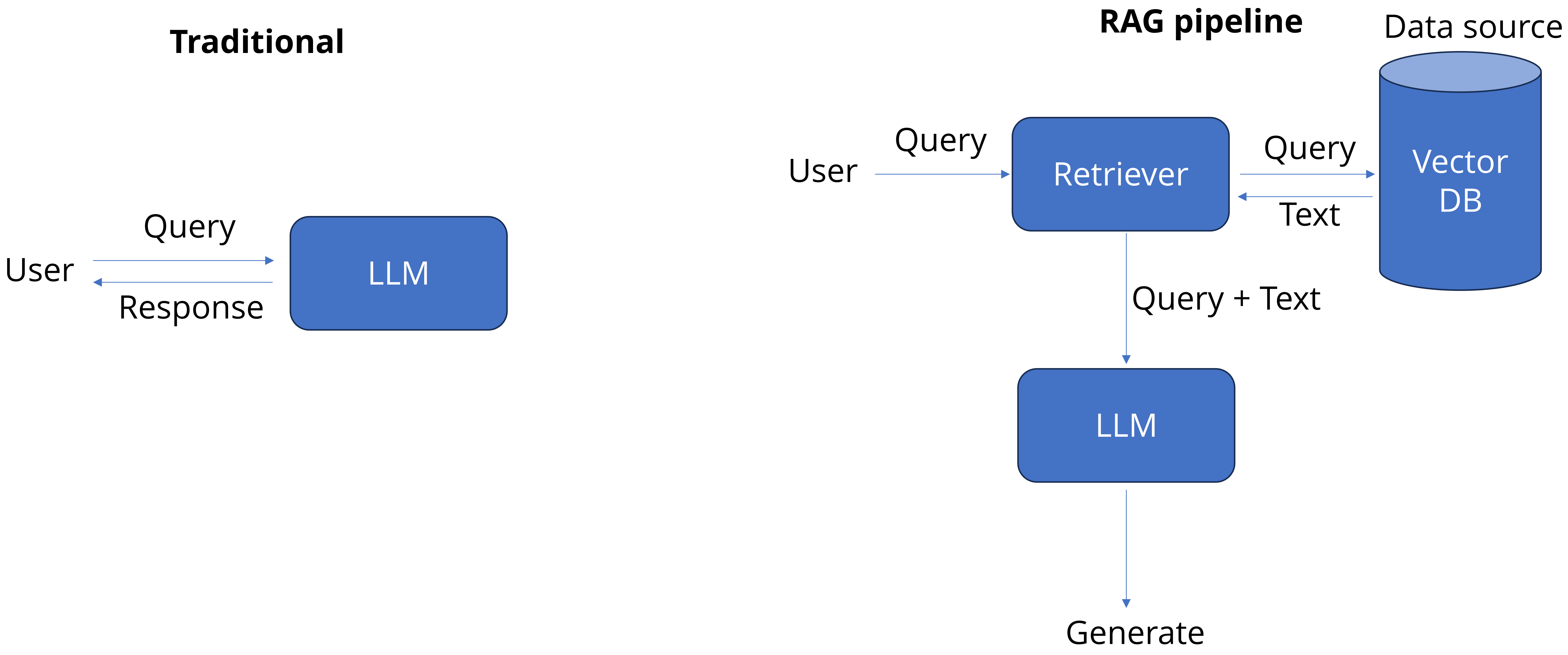
Generates responses based on the retrieved information

5. Utility functions

Includes tools such as web retrievers and document parsers to retrieve and preprocess files

Traditional vs. RAG Pipeline

Unlike traditional generative models, RAG-based models uniquely refer to a knowledge base to generate contextually rich responses.



Quick Check

In a Retrieval-Augmented Generation (RAG) model, what is the role of the Vector Store?

A. To store and retrieve vectorized representations of text
B. To split text into manageable chunks
C. To generate language model responses
D. To evaluate the accuracy of the model

Creating a Simple RAG Model

Load Document

LLMs lack up-to-date knowledge of the world and access to internal documents, so relevant information from various knowledge sources must be provided. These sources include:

CSVs

Google sheets

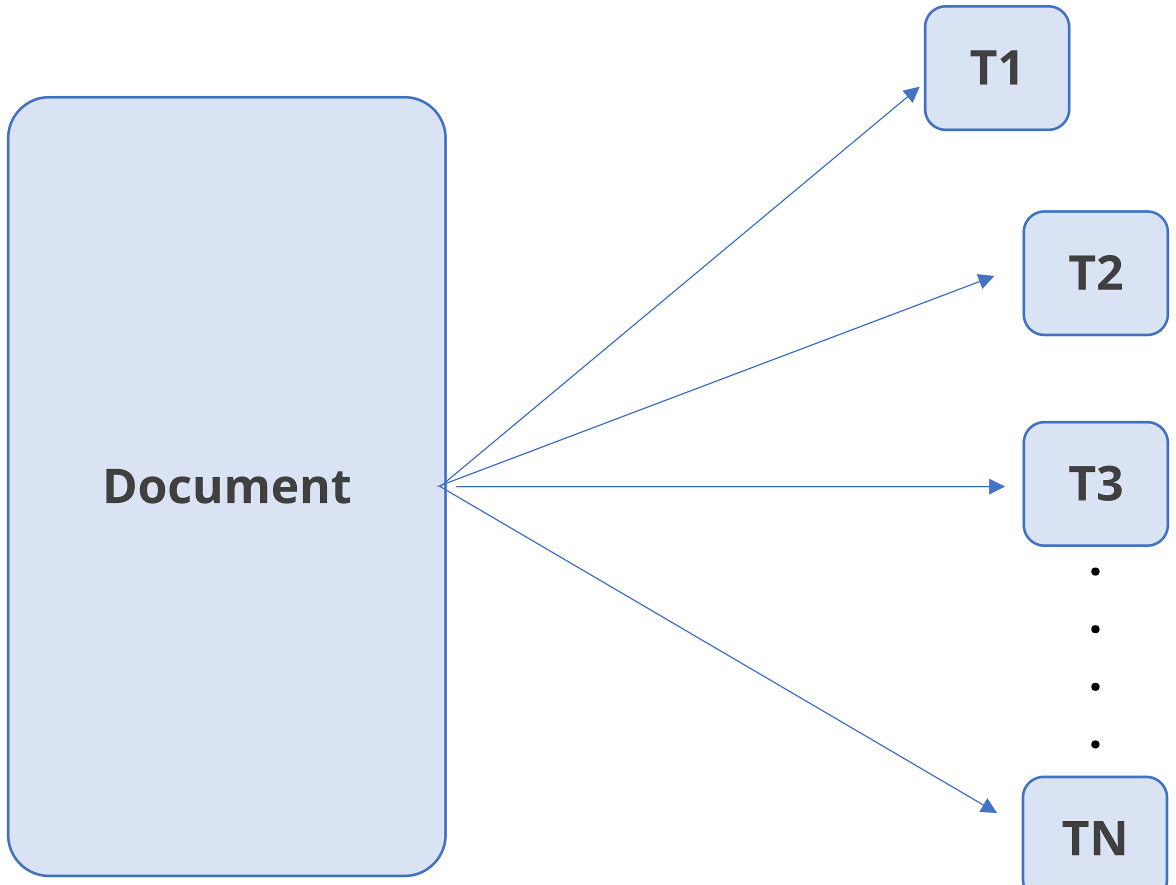
Text documents

PDF documents

Creating Text Chunks

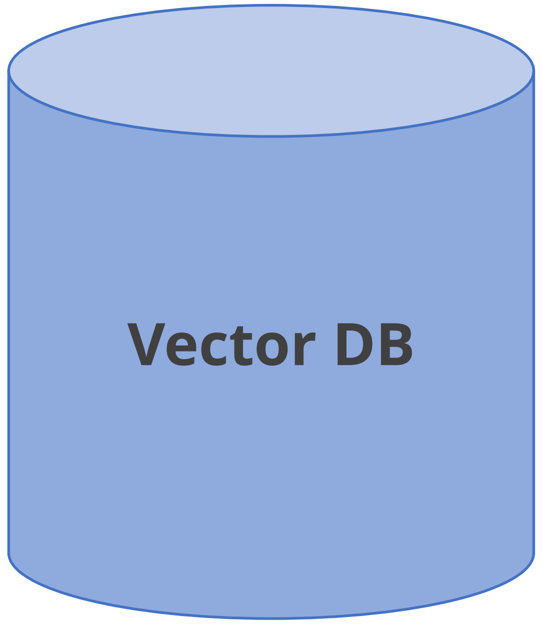
Data from knowledge sources often exceeds LLMs' context window, causing the ChatGPT API to truncate and potentially omit crucial information.

Text chunking, which divides lengthy texts into smaller segments, effectively addresses this issue.



Building Knowledge Bases

In RAG-based applications, data embeddings are stored instead of raw texts. These embeddings are floating-point numbers representing data in a high-dimensional vector space.



- Vector databases are used to store and manage these embeddings.
- These databases are specialized data stores designed specifically for storing and querying vectors.

Retriever

RAG utilizes a retriever that acts as a knowledge scout. This retriever searches external knowledge bases, such as databases, to find relevant documents or information.

The system retrieves relevant documents as context from the vector database based on the question.

Prompt and LLM

The retrieved information is sent along with the prompt that guides the large language model (LLM)

The LLM uses the context and prompt, generates a coherent and contextually relevant answer.

Quick Check

Which component in a RAG model is responsible for breaking down documents into smaller, manageable pieces?

A. Embedding Model

B. Text Splitter

C. LLM (Language Model)

D. Retriever

Building Advanced RAG with LangChain

Import Necessary Libraries

Step 1: This code imports essential libraries for setting up a RAG system. It includes tools for vector storage, document loading, text splitting, and embedding, all powered by OpenAI's advanced LLMs.

Command Prompt

```
! pip install langchain
from langchain.vectorstores import FAISS
from langchain.document_loaders import WebBaseLoader
from langchain.embeddings import OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter
from langchain.llms import OpenAI
from langchain.chains import VectorDBQA, RetrievalQA
from langchain.chat_models import ChatOpenAI
import openai
import os
```

Web Data Loading for the RAG Knowledge Base

Step 2: The code utilizes LangChain's "WebBaseLoader. It is for developing the knowledge base utilized in RAG, facilitating the retrieval and integration of contextually relevant and accurate information into language model responses.

Command Prompt

```
yolo_loader = WebBaseLoader("https://blog.paperspace.com/yolo-nas/").load()
```

Split the Data into Chunks

Step 3: The code below splits a document into smaller chunks for processing, particularly for tasks such as information retrieval or language model input.

Command Prompt

```
text_splitter = CharacterTextSplitter(
    separator="\n\n",
    chunk_size=10000,
    chunk_overlap=200,
    is_separator_regex=False,)


yolo_nas_chunks = text_splitter.split_documents(yolo_loader)
yolo_nas_chunks
```

Output

```
[Document(metadata={'source': 'https://blog.paperspace.com/yolo-nas/', 'title': 'YOLO-NAS: The Next Frontier
in Object Detection in Computer Vision', 'description': 'In this article we will explore a cutting-edge objec
t detection model,YOLO-NAS which has marked a huge advancement in YOLO series.', 'language': 'en'}, page_cont
ent='YOLO-NAS: The Next Frontier in Object Detection in Computer Vision\n\n\nPaperspace joins DigitalOcean.\n
\n Read More\n\nProducts\n\nPRoduct\n\n\nGradient\n\nBuild, train, deploy, and manage AI models.\n\n\nNoteboo
```

Embedding and Vector Store Setup

Step 4: The code sets up embeddings for the Retrieval-Augmented Generation (RAG) process using OpenAIEmbeddings and stores them efficiently with CacheBackedEmbeddings.

Command Prompt

```
from langchain.embeddings.cache import CacheBackedEmbeddings
from langchain.storage import LocalFileStore

store = LocalFileStore("./cachce/")

# create an embedder
core_embeddings_model = OpenAIEmbeddings()

embedder = CacheBackedEmbeddings.from_bytes_store(
    core_embeddings_model,
    store,
    namespace = core_embeddings_model.model
)
```

Embedding and Vector Store Setup

Step 5: The code creates a FAISS vector store from preprocessed data chunks and instantiates a retriever for quick similarity-based retrieval and efficient document access during RAG.

Command Prompt

```
# store embeddings in vector store
vectorstore = FAISS.from_documents(yolo_nas_chunks, embedder)

# instantiate a retriever
retriever = vectorstore.as_retriever()
```

Establishing the Retrieval System

Step 6: The code configures a retrieval system for RAG using LangChain, initializing a ChatPromptTemplate and setting up a chat-based LLM with "ChatOpenAI." It prepares the system for efficient question-answering tasks

Command Prompt

```
# Install the latest version of LangChain
!pip install langchain --upgrade

# Import the ChatPromptTemplate object from the langchain.prompts module
from langchain.prompts import ChatPromptTemplate

def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)

# prompt to send to the 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:"""

prompt_template = ChatPromptTemplate.from_template(prompt)
llm = ChatOpenAI(temperature=0.7, model_name="gpt-3.5-turbo")
```

Establishing the Retrieval System

Step 7: The rag_chain_from_docs integrates context, prompt, and LLM processing. The rag_chain_with_source combines a retriever with rag_chain_from_docs to perform retrieval-based question-answering, providing source documents for added context.

Command Prompt

```
from langchain.chains.combine_documents import create_stuff_documents_chain
from langchain_core.runnables import RunnablePassthrough, RunnableParallel

# This code defines a chain where input documents are first formatted, then passed
through a prompt template, and finally processed by an LLM.
rag_chain_from_docs = (
    RunnablePassthrough.assign(context=(lambda x: format_docs(x["context"])))
    | prompt_template
    | llm
)
# This code creates a parallel process: one retrieves the context (using a
retriever), and the other passes the question through unchanged. The results are
then combined and assigned to the variable `answer` using the `rag_chain_from_docs`
processing chain.
rag_chain_with_source = RunnableParallel({"context": retriever, "question":
RunnablePassthrough()}
).assign(answer=rag_chain_from_docs)
```

Retrieve Responses

Step 8: After processing the queries, the RAG system generates and returns contextually rich and accurate responses. The responses are printed on the console.

Command Prompt

```
# This is to generate response with RAG system
response = rag_chain_with_source.invoke(
    "What does YOLO-NAS include quantization includes?")

print(response['answer'].content)
```

Output

```
YOLO-NAS includes quantization blocks that involve converting neural network weights, biases, and activations
to integer values (INT8) for enhanced model efficiency. This transition results in minimal precision reductio
n compared to other YOLO models. These quantization blocks contribute to improved model efficiency and perfor
mance in object detection tasks.
```

Model Evaluation and Monitoring

Model Evaluation

The model's performance is evaluated using predefined question-answer pairs. The **QAEvalChain** compares the model's responses with the expected answers and calculates the model's accuracy.

Steps to evaluate model:

- Define a function evaluate_model to evaluate the model using question-answer pairs.
- The evaluate_model function performs the following tasks:
  1. Creates an evaluation chain using the chat model
  2. Generates predictions by processing each question through the agent
  3. Evaluates the predictions against the actual answers

Model Monitoring

Maintaining the quality and performance of a RAG application in a production environment presents challenges. RAG provides essential building blocks for monitoring production quality, offering valuable insights into your application's performance

Key Methods: Model Monitoring

The SimpleModelMonitor class is a basic implementation of the monitoring system used in this course. It includes the following methods:

Loading and Saving Logs

These functions handle the persistence of log data.

Log Interactions

log_interaction function logs each interaction with the model, recording:

- The timestamp
- The query sent to the model
- The execution time of the query

Key Methods: Model Monitoring

The SimpleModelMonitor class is a basic implementation of the monitoring system used in this course. It includes the following methods:

Visualization

The **plot_execution_times** function creates a plot of execution times over timestamps. This provides a visual representation of the model's performance

Performance Metrics

get_average_execution_time calculates and returns the average execution time of all logged interactions.

Real-World Applications of RAG

RAG has practical applications across various fields:

- **Customer support chatbots:** Provide accurate and context-aware responses
- **Search engines:** Understand user queries and generate informative snippets
- **Content generation:** Create news articles and product descriptions

Demo: Implementing RAG from scratch

Duration: 10 minutes

Overview: In this demo, you integrate retrieval with generation by splitting a document into manageable chunks, embedding them, and indexing with FAISS. It retrieves the most relevant context for a given query and uses a custom prompt to guide the process. A quantized language model then generates concise, context-informed answers.

Note

Please download the solution document from the Reference Material Section and follow the Jupyter Notebook for step-by-step execution.

DEMONSTRATION

Quick Check

Which of the following is a key purpose of monitoring and evaluation in the context of a RAG-based model?

A. To reduce the computational cost of generating responses

B. To ensure the model's responses are both accurate and relevant over time

C. To split text into smaller chunks for better processing

D. To retrieve the most relevant documents for generating answers

Key Takeaways

- Traditional generative models struggle with complex tasks, requiring the integration of components like retrievers and vector stores to enhance their capabilities
- RAG-based models improve performance by using tools such as text splitters, embedding models, and utility functions for effective information processing and retrieval.
- LangChain simplifies the creation of advanced RAG models by offering libraries for document handling, chunk creation, and embedding storage, enabling context-aware systems.
- RAG pipelines, using elements like ChatPromptTemplate and vector stores, ensure efficient and accurate question-answering in practical applications.

Q&A