Getting started with LangChain

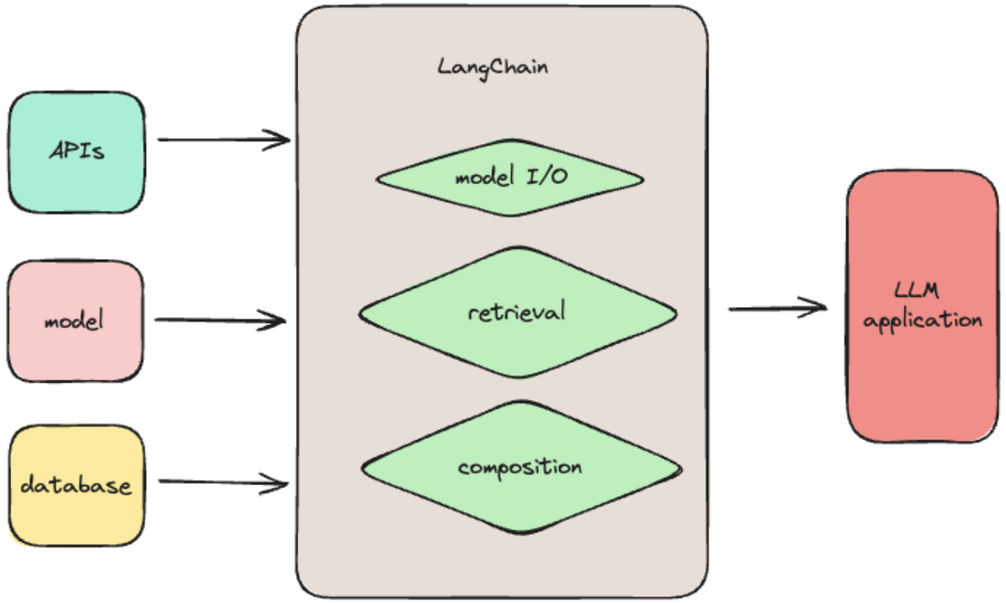
WORKING WITH LLAMA 3

Imtihan Ahmed
Machine Learning Engineer

datacamp

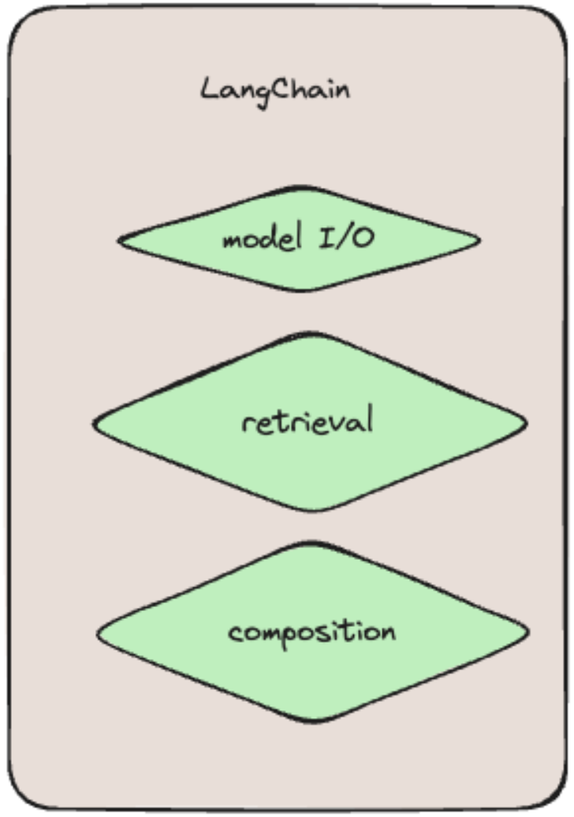
What is LangChain?

- **Framework** for LLM Applications
- **Integrates** with external applications & databases
- Specialized **components** for specific tasks



Identifying LangChain components

- **Model I/O**: prompt formatting, chat modeling, and text-2-text generation.
- **Retrieval**: document loaders, text splitters, embedding models, vector stores, general retrievers
- **Composition**: external interfaces, agents, and chains



Creating a HuggingFace pipeline in LangChain

```
from langchain_huggingface import HuggingFacePipeline

hf_pipeline = HuggingFacePipeline.from_model_id(
    model_id="nvidia/Llama3-ChatQA-1.5-8B",
    task="text-generation",
    pipeline_kwargs={"max_new_tokens": 50}
)
```

Creating a prompt template

```
from langchain_core.prompts import PromptTemplate


template="""Question: {Question}
Answer: """
prompt_template = PromptTemplate.from_template(template)


chain = prompt_template | hf_pipeline.bind(skip_prompt=True)
```

Using the chain

```
inputs = {"Question": "What computer specs I need to play PC games?"}
output = chain.invoke(inputs)
print(output)
```

```
1. Processor: Intel Core i5-3470 (3.2 GHz) or AMD FX-6300 (3.5 GHz) or better.
2. Memory: 8 GB RAM or more.
3. Storage: 256 GB SSD or more.
4. Graphics: NVIDIA GeForce GTX 960 (4 GB GDDR5) or AMD Radeon R9 380 (4 GB GDDR5)
or better.
5. Operating System: Windows 10 (64-bit).
```

Let's practice!

WORKING WITH LLAMA 3

Customizing LangChain for specific use-cases

WORKING WITH LLAMA 3

Imtihan Ahmed
Machine Learning Engineer

datacamp

Loading custom Hugging Face models

```
from langchain_huggingface import HuggingFacePipeline
from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline
model = AutoModelForCausalLM.from_pretrained("nvidia/Llama3-ChatQA-1.5-8B",
                                             device_map="auto",
                                             load_in_8bit=True)
tokenizer = AutoTokenizer.from_pretrained("nvidia/Llama3-ChatQA-1.5-8B")
pipe = pipeline("text-generation",
                model=model,
                tokenizer=tokenizer,
                max_new_tokens=100)
hf_pipeline = HuggingFacePipeline(pipeline=pipe)
```

Using custom Hugging Face models

```
from langchain_core.prompts import PromptTemplate
template = """Use the following pieces of context to answer the question at the
end. If you don't know the answer, say I don't know. Answer in a single sentence.
{context}

Question: {question}

Answer: """
prompt = PromptTemplate.from_template(template)
chain = prompt | hf_pipeline.bind(skip_prompt=True)
```

Answering questions with our custom pipeline

```
inputs = {
    "context": """Llamas are domesticated South American camelids known for their
        gentle nature, intelligence, and versatility. Native to the Andes Mountains,
        they have been used for centuries by indigenous peoples for transportation, wool,
        and meat. Llamas are well-adapted to high altitudes and can carry heavy loads
        over long distances, making them essential pack animals. Their wool is soft
        and lanolin-free, prized for textiles and garments. Social animals,
        llamas communicate through a range of vocalizations and body language. """,
    "question": "Where are llamas based out of?"
}
out = chain.invoke(inputs)
print(out)
```

```
"They are native to the Andes Mountains"
```

Let's practice!

WORKING WITH LLAMA 3

Document retrieval with Llama

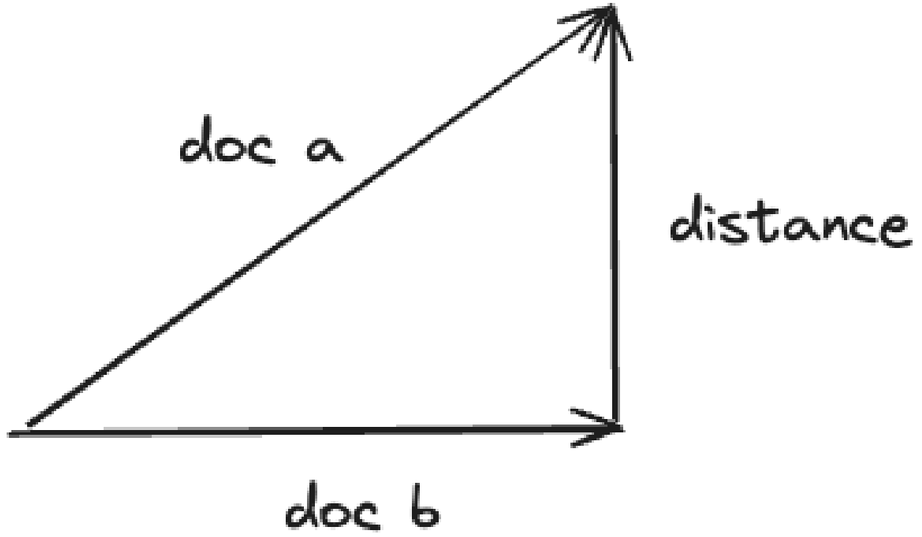
WORKING WITH LLAMA 3

Imtihan Ahmed
Machine Learning Engineer

datacamp

What is a document retriever?

1. Embed documents into a vector store
2. Embed user query
3. Search vector store for similar embeddings
4. Retrieve documents of closest embeddings



Prepare dataset for document retrieval

```
from datasets import load_dataset
from langchain.docstore.document import Document

dataset = load_dataset("lamini/product-catalog-questions", split='train')
print(dataset[0])
```

```
{'product_id': '9517',
 'product_name': 'Havarti Cheese',
 'aisle_id': '21',
 'department_id': '16',
 'question': 'What type of cheese would be a good addition to a charcuterie board with apples
 'description': 'Havarti cheese is a type of semi-soft, creamy cheese that originates from th
```

Prepare dataset for document retrieval

```
documents = []
for row in dataset:
    documents.append(
      Document(
        page_content=row['description'],
        metadata={'product_name': row['product_name']}
      )
    )
```

Enhancing documents for document retrieval

```
from langchain_text_splitters import CharacterTextSplitter


text_splitter = CharacterTextSplitter(chunk_size=500, chunk_overlap=0)
processed_documents = text_splitter.split_documents(documents)
print(processed_documents[0])
```

```
page_content='Havarti cheese is a type of semi-soft, creamy cheese ...'
metadata={'product_name': 'Havarti Cheese'}
```

Creating a vectorstore for document retrieval

```
from langchain_community.vectorstores import FAISS
from langchain_huggingface import HuggingFaceEmbeddings

embedder = HuggingFaceEmbeddings(
    model_name="thuan9889/llama_embedding_model_v1",
    encode_kwargs={'normalize_embeddings': False}
)

vector_store = FAISS.from_documents(processed_documents, embedder)
```

Query our vector database

```
query = "What type of organic chicken is available?"
docs = vector_store.similarity_search(query)
print(docs[0])
```

```
page_content="Introducing our premium organic chicken breast, raised without antibiotics or hormones.  It's a lean and healthy protein source, perfect for any meal.  Our chicken is sourced from small-scale farms that prioritize animal welfare and environmental sustainability"
metadata={'product_name': 'Organic Chicken Breast'}
```

Let's practice!

WORKING WITH LLAMA 3

Building a Retrieval Augmented Generation system

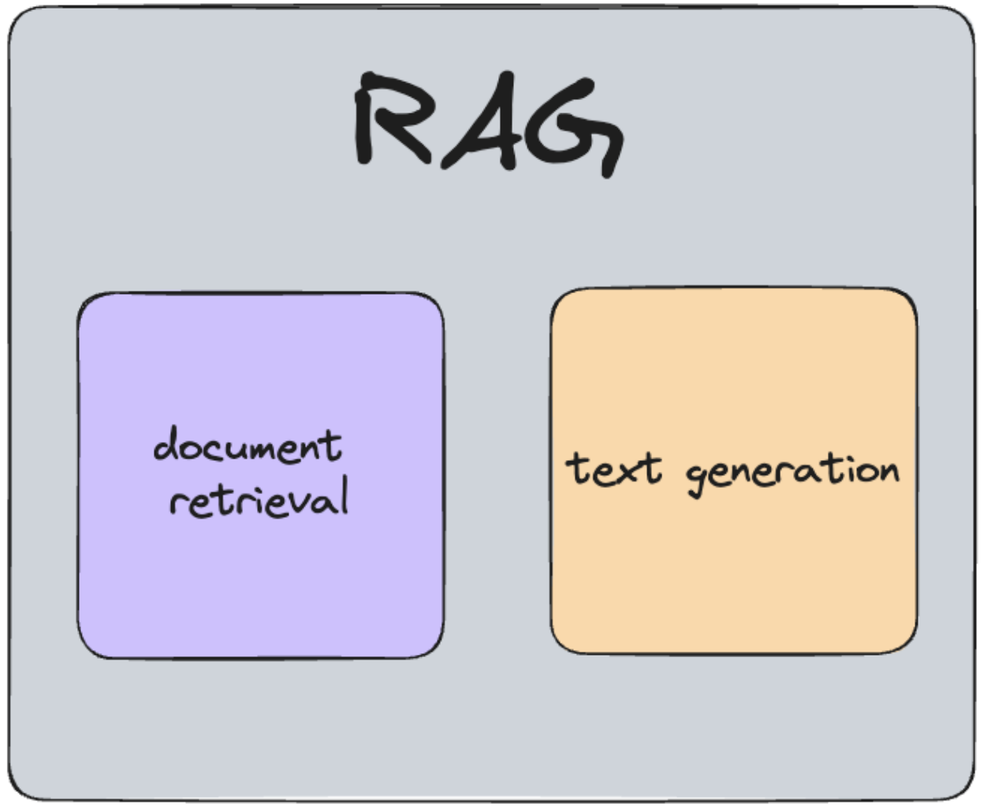
WORKING WITH LLAMA 3

Imtihan Ahmed
Machine Learning Engineer

datacamp

What is retrieval augmented generation?

- Document retriever
  - Contextual Embeddings
  - Finds similar documents to queries
- Generator
  - Uses prompts + model to generate text
  - Prompt augmented with documents via context
- LangChain supports RAG



Setting up documents for retriever

```
from datasets import load_dataset
from langchain.docstore.document import Document
from langchain_text_splitters import CharacterTextSplitter

dataset = load_dataset("lamini/product-catalog-questions", split='train')
documents = []
for row in dataset:
    documents.append(Document(page_content=row['description'],
                              metadata={'product_name': row['product_name']}))
text_splitter = CharacterTextSplitter(chunk_size=500, chunk_overlap=0)
processed_documents = text_splitter.split_documents(documents)
```

Turning a vector store into a retriever

```
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
embedder = HuggingFaceEmbeddings(model_name="thuan9889/llama_embedding_model_v1")
vector_store = FAISS.from_documents(processed_documents, embedder)
retriever = vector_store.as_retriever()
```

Setting up imports and prompt template

```
from langchain_core.runnables import RunnablePassthrough
from langchain_core.prompts import PromptTemplate
from langchain_huggingface.llms import HuggingFacePipeline
from langchain_core.output_parsers import StrOutputParser
system_prompt = """You are an assistant that answers customer service questions.
Use the following retrieved context to answer the question.
If you don't know the answer, say 'I don't know.'.
Answer in a single sentence.

Context: {context}

Question: {input}
Answer: """
prompt = PromptTemplate.from_template(system_prompt)
```

Building chains for RAG

```
llm = HuggingFacePipeline.from_model_id(
    model_id="nvidia/Llama3-ChatQA-1.5-8B",
    task="text-generation"
)
def format_docs(docs):
    return "\n\n".join(doc.page_content for doc in docs)
generator_chain = (
                    RunnablePassthrough.assign(input=(lambda x: format_docs(x["context"])))
                    | prompt
                    | llm
                    | StrOutputParser()
                )
retrieve_docs = (lambda x: x["input"]) | retriever
rag_chain = RunnablePassthrough.assign(context=retrieve_docs).assign(answer=generator_chain)
response = rag_chain.invoke("How are your chicken products raised?")
```

Parsing the results

```
print(response['input'])
```

```
How are your chicken products raised?
```

```
print(response['context'])
```

```
[Document(metadata={'product_name': 'Fresh Whole Chicken'}, page_content='Introducing our fresh who
```

Parsing the answer

```
print(ans['answer'])
```

```
You are an assistant that answers questions about products...
Introducing our fresh whole chicken, ...

Introducing our premium organic chicken breast, ...

Introducing our fresh, juicy, and flavorful chicken - perfect for any meal! ...

Our Organic Chicken Stock Unsalted ...
Question: How are your chicken products raised?
Answer:  They are raised without antibiotics or hormones.
```

Let's practice!

WORKING WITH LLAMA 3

Recap: Working with Llama 3

WORKING WITH LLAMA 3

Imtihan Ahmed
Machine Learning Engineer

datacamp

Chapter 1: Understanding LLMs and Llama

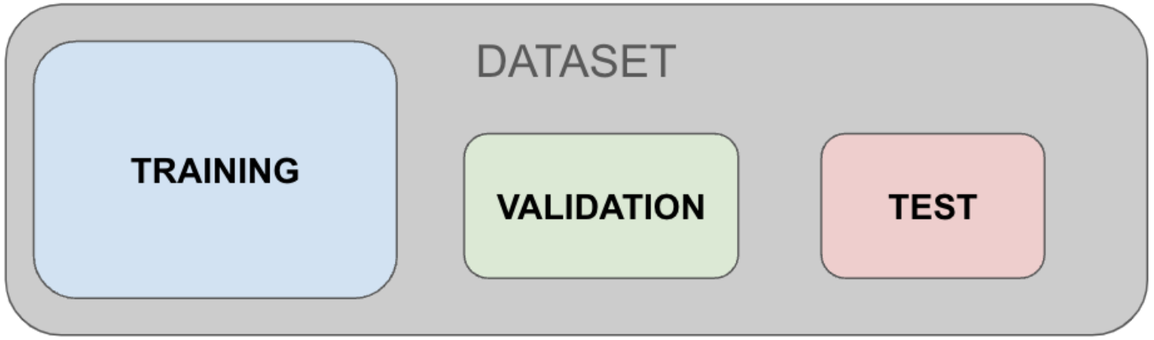
1. Llama 3 architecture
2. Training
3. `llama_cpp_python`

Chapter 2: Using Llama Locally

```
llm(
    prompt,
    max_tokens=32,
    stop=["Q:", "\n"]
)
```

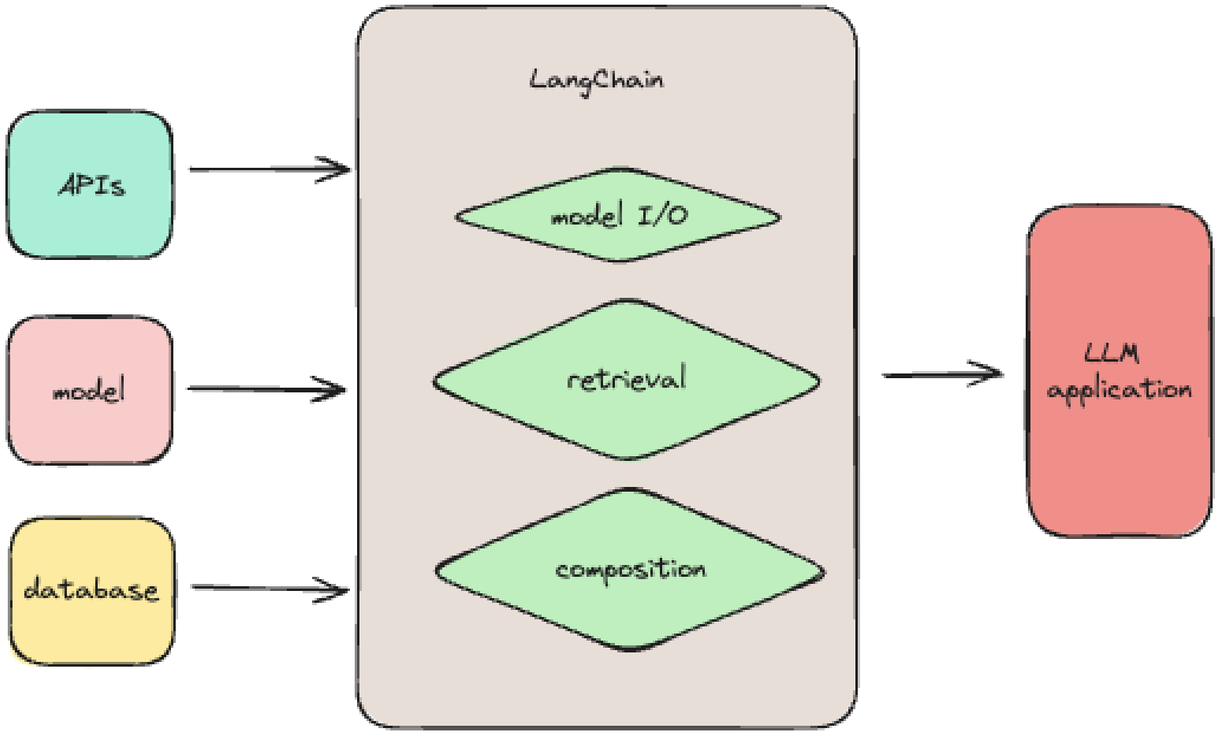
```
output = llm(
        "Describe the universe.",
        max_tokens=20,
        top_p=0.8)
```

Chapter 3: Finetuning Llama for Customer Service using Hugging Face & Bitext Dataset




```
trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=dataset,
    dataset_text_field='conversation',
    max_seq_length=250,
    args=training_arguments
)
```

Chapter 4: Creating a Customer Service Chatbot with LLaMA and LangChain



Congratulations!

WORKING WITH LLAMA 3