The right way to approach RAG (Retrieval-Augmented Generation). Before diving into RAG architecture, you must understand its **building blocks** in LangChain:

🧩 1. Document Loaders in LangChain:

Purpose

Document Loaders are used to **load data** from various sources (PDFs, text files, URLs, databases, etc.) into LangChain as Document objects.

Document {page_content = "actual text",
metadata =information like filename, page number, URL, etc.}

Loader	Description
PyPDFLoader	Loads text from PDF files
TextLoader	Loads plain text files
UnstructuredURLLoader	Loads from websites
DirectoryLoader	Loads all files from a folder
Docx2txtLoader	Loads text from Word documents

✂️ 2. Text Splitters in LangChain

Purpose

Large documents must be split into smaller chunks so that embeddings and retrieval work effectively.
Text splitting is the process of breaking large chunks of text (like articles, PDFs, HTML pages or books) into smaller, manageable pieces (chunks) than LLM can handle effectively.
LLM have context length, if LLM have 50k token (words) it can read. In this case LLM cannot beyond this situation.PDF have 1L words so handle this situation.

Why?

- Embedding models have token limits.
- Smaller chunks improve semantic search and context matching.

Splitter	Description
CharacterTextSplitter	Splits by characters or paragraphs
RecursiveCharacterTextSplitter	Splits intelligently by paragraph → sentence → word
TokenTextSplitter	Splits by tokens (for LLM-specific contexts)

🧠 3. Vector Stores in LangChain

Purpose

A **Vector Store** stores embeddings (numerical vector representations of text chunks) and allows **semantic search** (finding similar text by meaning).

Vector store is a data database or storage system that holds embeddings (numerical vector representations) of text, images and allows you to efficiently search for semantically similar content. LLM cannot directly search text they need numerical representation. eg. chroma

key feature

1. Similarity search.
2. Indexing → generally used for fast similarity searches.
3. CRUD operation → Add, update, delete.

Popular Vector Stores

Store	Description
FAISS	Lightweight, local vector store (Facebook AI)
Chroma	Local persistent store, easy to use
Pinecone	Cloud-based, scalable vector DB
Weaviate, Milvus, Qdrant	Other cloud-native options

🔍 4. Retrievers in LangChain

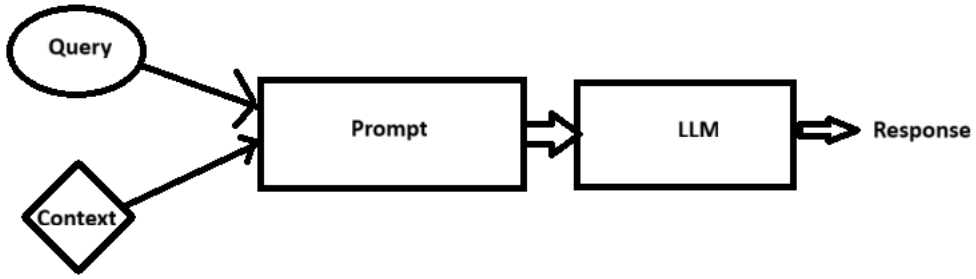
Purpose

Retrievers is a component in LangChain that fetches relevant documents from a data source In response to a user query.

What is RAG? RAG (Retrieval-Augmented Generation)

RAG is a way to make a language model smarter by giving it extra information at the time user ask question.

In simple terms:
It *retrieves* relevant information from external data → gives it to the LLM → the LLM *generates* a better, factual answer.



🗣️Query- Prompt

This is the **user's question or instruction** given to the LLM. It defines what the model needs to answer or perform.

📚Context – Knowledge

This is the **relevant information retrieved** from your own data sources (PDFs, databases, websites, etc.) to help the LLM give a factual, grounded response.

A **RAG (Retrieval-Augmented Generation)** system connects external knowledge with a language model to produce accurate, context-aware answers.
It operates through **four key components**:

1️⃣ **Indexing (Knowledge Preparation Phase)**

2️⃣ **Retrieval (Context Search Phase)**

3️⃣ **Augmentation (Context Injection Phase)**

4️⃣ **Generation (Answer Creation Phase)**

Now we can take more look each **component**,

1️⃣ **Indexing (Knowledge Preparation Phase)**

Convert your raw data (PDFs, text, websites, databases, etc.) into a **searchable knowledge base** of embeddings.

Indexing refers to the process of organizing and preparing a knowledge base to enable efficient retrieval during query time. It typically involves four key steps: document loading, text splitting, embedding generation, and vector storage.

Step	Description	LangChain Module
📥 **Document Loading**	Load files or URLs into the system/memory e.g PDF report, word documents, Youtube transcipts, blog pages, Github Repos, SQL Records etc.	Document Loaders Tools: PyPDFLoader, YoutubeLoader, WebBaseLoader, GitLoader
✂️ **Text Splitting**	Break text into small, meaningful chunks	Text Splitters Tools: RecursiveCharacterTextSplitter, RecursiveCharacterTextSplitter, MarkdownTextSplitter, HTMLHeaderTextSplitter, PythonCodeTextSplitter, TokenTextSplitter
🔢 **Embedding Creation**	Convert text chunks → numerical vectors	Embeddings Tools: OpenAIEmbeddings, HuggingFaceEmbeddings, OllamaEmbeddings
💾 **Vector Store Creation**	Store embeddings in a database for semantic search	Vector Stores (FAISS, Chroma, Pinecone, etc.)

🧭 **Retrieval:**

Retrieval is the process of **searching and fetching the most relevant pieces of information** from a knowledge base (vector store) that can help the LLM answer a user's query accurately.

Retrieval bridges the gap between the user's question (Query) and your stored knowledge (Context).

Purpose of Retrieval

- To find **semantically similar text chunks** to the user's query.
- To ensure the **LLM has accurate and grounded context** before generating a response.
- To make the system **factual, explainable, and domain-specific.**

Steps in the Retrieval Process

Step	Name
1️⃣ **Query Embedding**	Convert the user's question into a **vector representation** using the same embedding model used during indexing.
2️⃣ **Similarity Search**	Compare the query vector with all document vectors in the **vector store (FAISS, Chroma etc.)** using cosine similarity or dot product.
3️⃣ **Top-K Selection**	Retrieve the **top K most relevant** document chunks (e.g., top 3 or 5) with the highest similarity scores.
4️⃣ **Context Construction**	Combine the selected chunks into a single **context string** that will be sent to the LLM along with the user query.

💡 **Augmentation:**

Augmentation means **enhancing the original user query** by adding the **retrieved context (knowledge)** from your data sources before sending it to the language model.

Why Augmentation Is Important

Without RAG	With RAG
The model relies only on what it was trained on (static knowledge).	The model gets *updated, domain-specific, and factual* context.
May hallucinate or produce outdated info.	Produces grounded, verifiable answers.
No data privacy control.	Context retrieved only from your approved sources.

🧩 Steps in the Augmentation Process

Step	Description
1️⃣ **Retrieve Relevant Chunks**	Get top-k document chunks from the vector store based on query similarity.
2️⃣ **Build the Augmented Prompt**	Combine the user query and retrieved chunks into a structured input for the model.
3️⃣ **Context Injection**	Insert the retrieved knowledge into the model's input (prompt engineering).
4️⃣ **Send to LLM for Generation**	Pass the augmented input to the model to generate a final, grounded response.

💡 Generation:

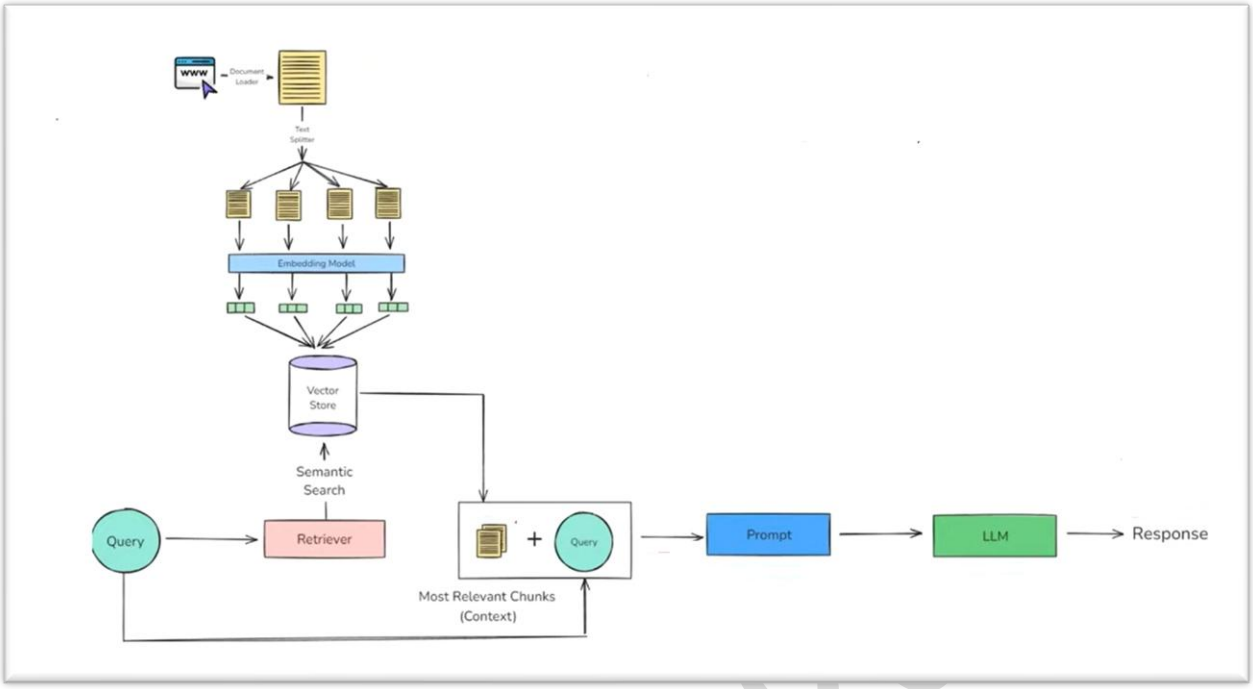
Generation is the stage where the **Large Language Model (LLM)** takes the **augmented input** (user query + retrieved context) and **produces the final answer**.

🧩 Purpose of the Generation Step

Natural Language Output: Convert structured data + context into a readable, fluent answer.

Input & Output of Generation Stage

Component	Description
Input	Augmented Prompt → (User Query + Retrieved Context)
Process	LLM processes both inputs to generate a relevant answer
Output	Final text answer or structured output (summary, report, etc.)



Overall RAG System