LANGCHAIN DOCUMENTATON

LangChain is an open-source framework designed to simplify the development of applications powered by large language models (LLMs). It provides a standard interface for integrating with various LLMs and external tools, enabling the creation of complex, modular, and advanced AI-driven solutions.

The LangChain Ecosystem consists of several key components:

- **LangChain (Core Framework):**

  This is the foundation, providing the core abstractions and interfaces for building LLM applications. It includes functionalities for:

- **Chains:** Combining LLMs with other components (like data sources, tools) into sequences of operations.
- **Agents:** Allowing LLMs to interact with their environment, use tools, and perform multi-step reasoning.
- **Prompt Management:** Tools for creating, managing, and optimizing prompts for LLMs.
- **Memory:** Storing and retrieving conversational history for stateful interactions.
- **Integrations:** Connecting with various LLMs, vector stores, document loaders, and other tools.
- **LangGraph:**

  A library for building robust, stateful, and multi-agent applications. It allows for defining complex, graph-based workflows with features like loops, retries, and human-in-the-loop interactions, which are essential for more sophisticated AI systems.

- **LangSmith:**

  A platform for LLM application development, debugging, testing, and monitoring. It provides observability into LLM runs, enabling developers to trace, evaluate, and optimize the performance of their applications in production.

- **LangServe:**

  Facilitates the deployment of LangChain applications as production-ready web APIs. It simplifies the process of turning a LangChain-powered pipeline into a scalable and accessible service.

- **LangFlow:**

A visual interface for building and experimenting with LangChain applications without extensive coding. It democratizes AI development by providing a drag-and-drop environment for designing LLM workflows.

- **LangChain Hub:**

  A centralized, version-controlled repository for discovering and sharing high-quality prompts and runnable objects, promoting reusability and consistency in prompt engineering.

- **LangChain Community:**
  A package containing a wide range of community-maintained integrations with various models, vector stores, document loaders, and other tools, expanding the capabilities of the core framework.
  In essence, the LangChain ecosystem provides a comprehensive set of tools and platforms to support the entire lifecycle of Generative AI projects, from initial development and experimentation to deployment, monitoring, and continuous improvement.

Component	**Role**	**Example Classes**
Models	The interface to LLMs or Chat Models (like Gemini, GPT, Llama).	ChatGoogleGenerativeAI, OpenAI
Prompts	Templates for generating inputs to models. Allows for dynamic input injection.	PromptTemplate, ChatPromptTemplate, MessagesPlaceholder
Chains (or Runnables)	The structured sequence of calls to models, prompts, and other components.	**LCEL** (`
Retrieval (Indexes)	Tools for interacting with external data (Vector Stores, Document Loaders). Essential for RAG.	FAISS, Chroma, WebBaseLoader
Memory	Stores and manages the state of a conversation across multiple turns.	ConversationBufferMemory, FileChatMessageHistory
Agents	Systems that use an LLM to decide on a sequence of **Actions** using **Tools**.	create_react_agent, AgentExecutor

TYPES OF AGENTS IN LANGCHAIN :

- **Zero-shot ReAct:**
  The agent takes action based on a single input.
- **ReAct:**
  This is similar to zero-shot, but the agent can "think" (reason) about what to do. It's more powerful for complex tasks.
- **Self-Ask with Search:**
  This agent can specifically ask search queries to find information.
- **Conversational:**
  Designed for conversational scenarios, remembering past interactions.
- **Plan-and-execute agents:**
  These agents plan out the entire sequence of actions to take, and then execute the plan.

TYPES OF MEMORY IN LANGCHAIN :

- **ConversationBufferMemory:**
  Simplest form, it stores the entire conversation history.
- **ConversationBufferWindowMemory:**
  Stores a rolling window of the most recent interactions.
- **ConversationSummaryMemory:**
  Summarizes the conversation over time to keep the memory concise.
- **ConversationSummaryBufferMemory:**
  Combines the buffer and summary approaches. It uses a buffer initially, then summa rizes when the buffer gets too large.
- **ConversationKnowledgeGraphMemory:**
  Uses a knowledge graph to store and retrieve information from the conversation.

Map Reduce:

In Langchain, Map Reduce is a type of chain often used for processing large docum ents. It works by:

Mapping: Splitting the document into chunks and processing each chunk independently w ith an LLM.

Reducing: Combining the results from each chunk into a single, final output using another LLM call. This is great for summarizing long texts or answering questions based on a large co rpus of data.

Runnable:

In Langchain, a "Runnable" is a standard interface that represents a callable unit. It 's a core concept that allows you to easily chain together different components like models, prompts, and other runnables. Anything that implements the Runnable interface can be inv oked, composed, and streamed, making it super flexible for building complex workflows.

In short, Map Reduce is a specific type of chain for processing large amounts of data, while Runnable is a more general interface for creating composable and executable units within Langchain. They can be used together to build powerful and modular LLM applications!

TYPES OF CHAINS IN LANGCHAIN (2025)

CORE:

- LLMChain

SEQUENTIAL:

- SimpleSequentialChain
- SequentialChain

RETRIEVAL (RAG):

- RetrievalQA
- RetrievalQAWithSourcesChain
- SelfQueryChain

DOCUMENT COMPRESSION:

- LLMChainExtractor
- EmbeddingsFilter
- DocumentCompressorPipeline

SUMMARIZATION:

- load_summarize_chain (map_reduce, refine, stuff)

ROUTING:

- LLMRouterChain
- MultiRouteChain

AGENT/TOOLS:

- AgentExecutor (replaces old agent chains)

TYPES OF PARSERS

Parser	Purpose
StrOutputParser	Return raw text
JsonOutputParser	Strict JSON
PydanticOutputParser	Typed / validated output

Parser	**Purpose**
StructuredOutputParser	JSON schema-based outputs
EnumOutputParser	Multiple choice classification
OutputFixingParser	Fix broken output
RetryOutputParser	Retry broken parsing
DatetimeOutputParser	Extract/parse dates