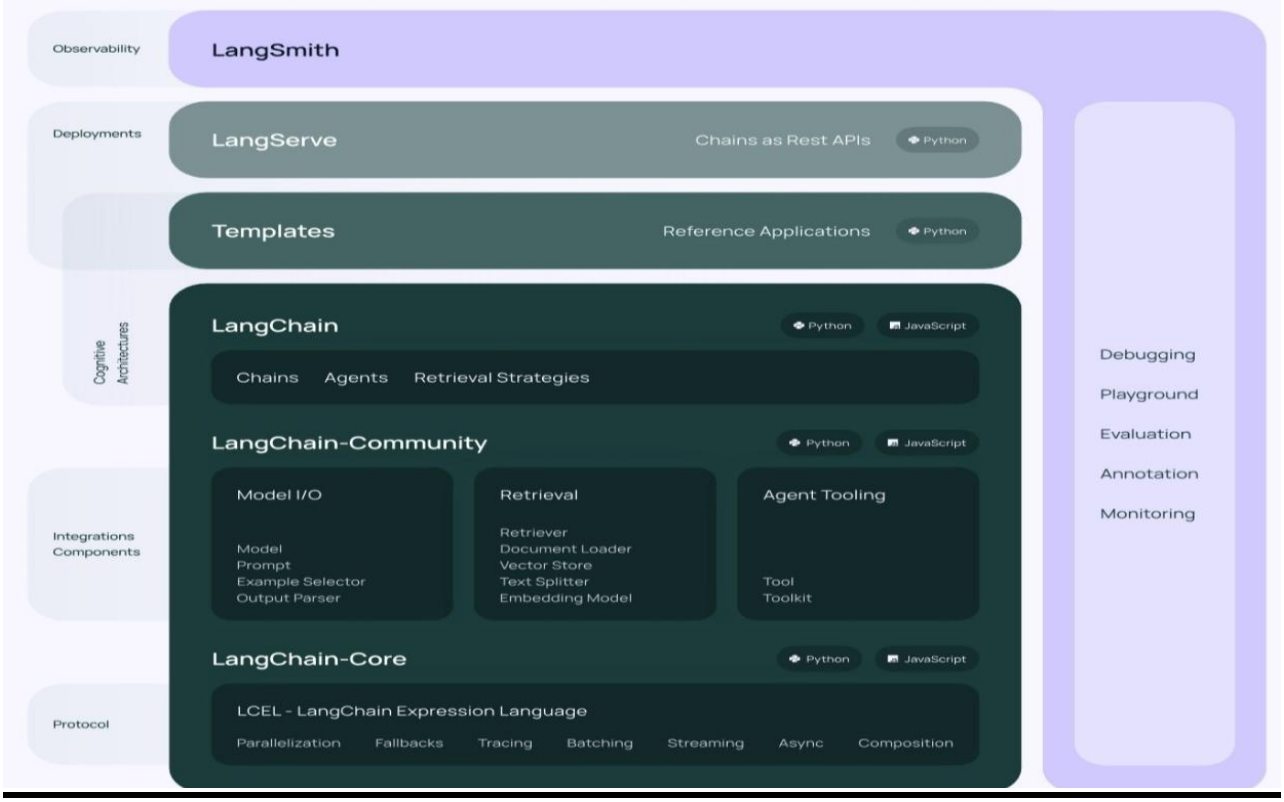
LangChain Architecture



What is LangChain?

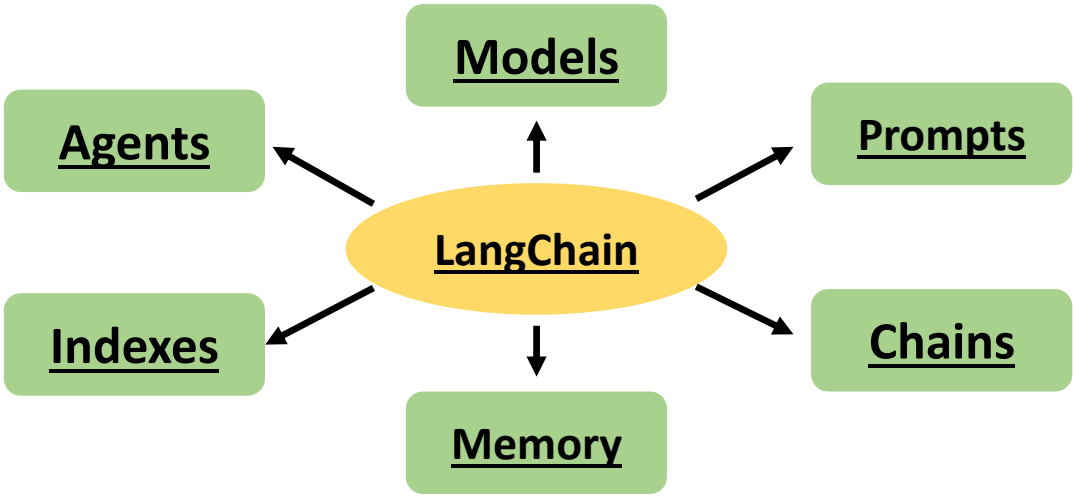
LangChain is an open-source framework designed to help developers build powerful applications using **large language models (LLMs)** and making it easy to connect them with tools, data, and workflows.

LangChain turns language models into **smart, interactive applications** by combining them with structured logic, memory, external tools, and live data access.

- Common Use Cases
- Chatbots with memory
- Document Q&A (RAG)
- AI agents (autonomous task-solving)
- Data extraction from unstructured sources
- Multi-step workflows (e.g., summarization → classification)

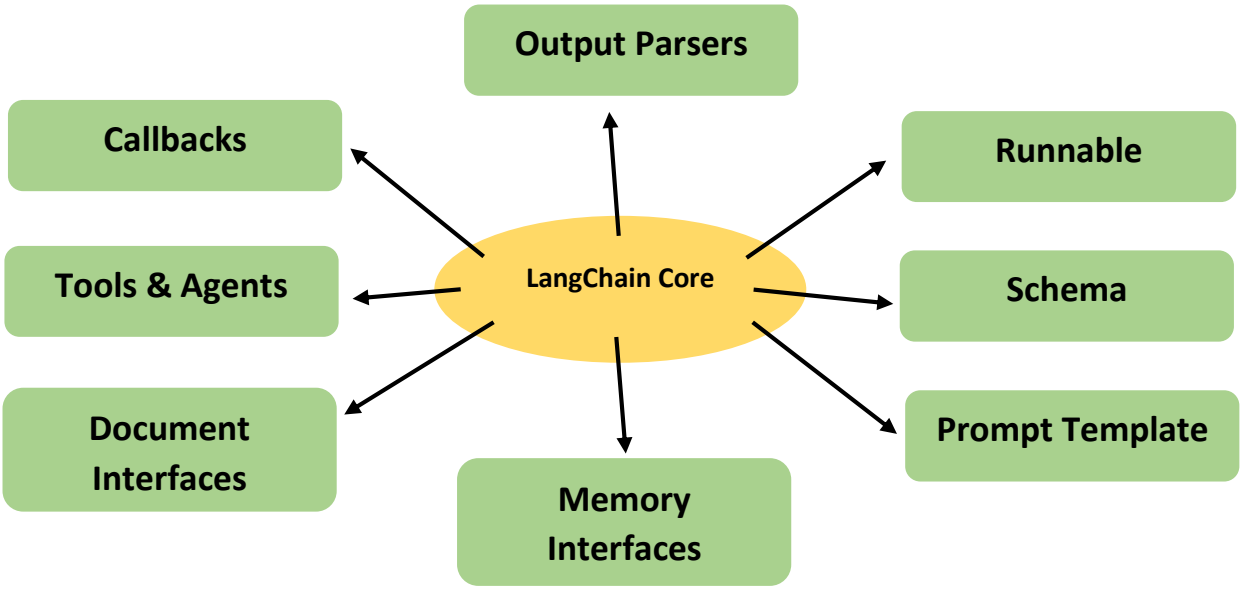
LangChain Components:

The main components of the **LangChain** package are high-level abstractions built on top of **LangChain-core** and **LangChain-community.**



LangChain Core Components:

These components define **how LangChain workflows are structured, executed, and extended**, without being tied to any specific model or service.



1.Runnables

The heart of LangChain Core: composable building blocks for any LLM pipeline.

- **Runnable**: Base interface for any computation unit (e.g., prompt → model → parser).
- **RunnableMap, RunnableLambda, RunnableSequence, RunnableParallel**: Used to compose complex chains.

2. Schema

Defines the data structures for interacting with LLMs and tools.

- **HumanMessage, AIMessage, SystemMessage**: Message types for chat models.
- **PromptValue, PromptTemplate**: Input formatting.
- **LLMResult, ChatResult**, Generation: Model outputs.
- **OutputParser**: Parses model output into structured formats.

3. Prompt Templates

Reusable templates for constructing prompts dynamically.

- **PromptTemplate**: Fills in variables using a Jinja-like format.

- **ChatPromptTemplate**: For multi-turn chat conversations.
- **MessagesPlaceholder**: For inserting dynamic message lists (e.g., memory/history).

4. Memory Interfaces

Defines how past interactions are remembered.

- **BaseMemor**y: Abstract base for memory components.
- **Message memory schemas**: For storing and replaying chat history.

5. Document Interfaces

Standardizes how documents are loaded and processed.

- **Document:** A wrapper with page_content and metadata.
- **BaseRetriever:** Interface for search/retrieval systems.

6. Tools & Agents (Core Interfaces)

Defines abstract behaviors for tool usage and agent decision-making.

- **Tool:** An interface for any callable external function or API.
- **AgentAction, AgentFinish:** Describes reasoning steps and final answers.
- **AgentStep:** Used for multi-step agent workflows.

7. Callbacks / Tracing / Events

Used to monitor and control execution steps used to **track, log, debug, and observe** the execution of chains, agents, tools, and models (like logging and debugging).

- **CallbackManager, Tracer, Run:** For tracking steps in a chain.
- **CallbackHandlers:** Hook into lifecycle events (start, end, error, etc.)
- **Tags, Metadata:** Attach information for tracing or filtering runs.

8. Output Parsers

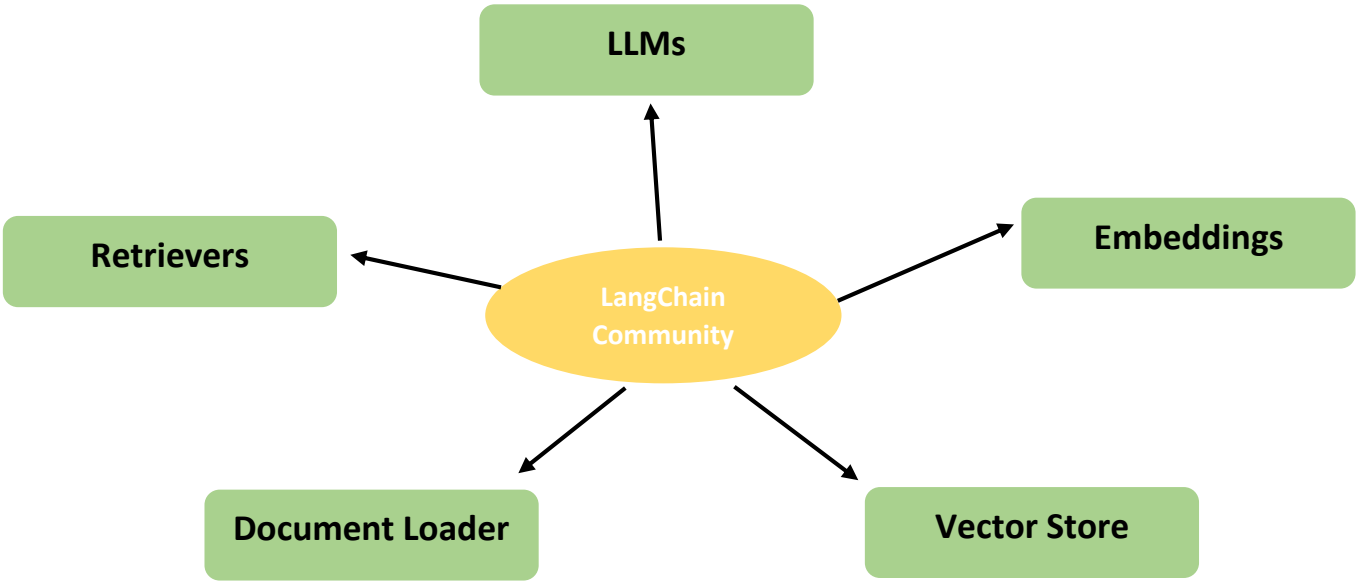
For converting raw LLM outputs into structured formats.

- **OutputParser**: Base class for parsers (JSON, Pydantic, custom).
- **StrOutputParser, JsonOutputParser, PydanticOutputParser**

LangChain-community

The **LangChain-community** package contains **integrations** for tools, models, databases, and utilities commonly used in LangChain applications

Main Components in LangChain-community



1.LLMs (Large Language Models)

Interfaces to real-world LLM providers.

- OpenAI, Anthropic, Cohere, Google Palm, HuggingFace, Replicate, etc.
- Compatible with LLM and ChatModel interfaces.
- Offers text and chat generation.

2. Embeddings

Support for embedding generation using external APIs or local models.

- **OpenAIEmbeddings, HuggingFaceEmbeddings,**etc.
- Used in vector stores for semantic search.

Example: LangChain_community.embeddings

3. Vector Stores

Integrations for storing and querying vectorized documents.

- FAISS, Chroma, Pinecone, Weaviate, Milvus, Qdrant, ElasticSearch, etc.
- Used for Retrieval-Augmented Generation (RAG).

Example: LangChain_community.vectorstores

4. Document Loaders

For reading and parsing documents in various formats.

- **PDFLoader, WebBaseLoader, CSVLoader, UnstructuredFileLoader, NotionLoader**, etc.
- Supports websites, files, Notion, GitHub, etc.

Example: LangChain_community.document_loaders

5. Retrievers

MultiQuery, BM25, Ensemble Contextual Compression, TimeWeighted, Vector Store Retriever