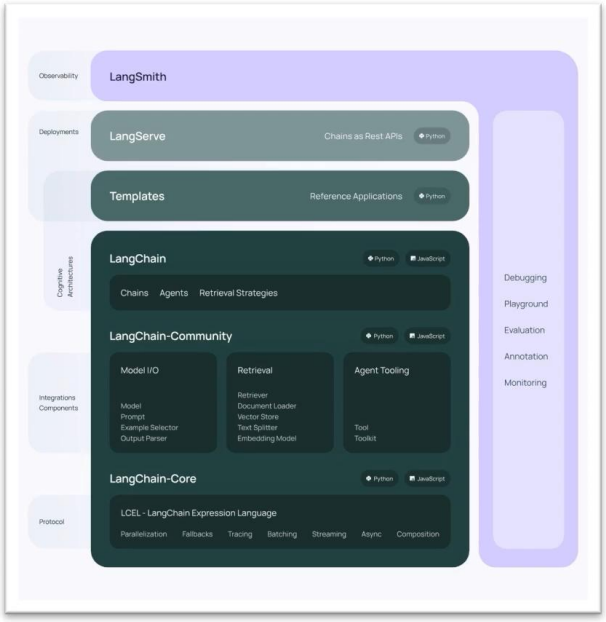
LangChain Ecosystem:

LangChain ecosystem

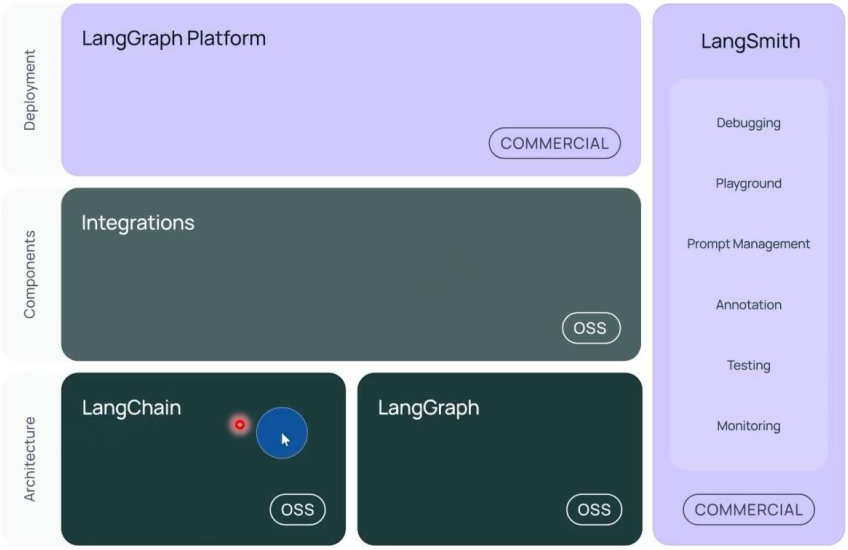
The LangChain ecosystem is a rich set of tools, modules, integrations, and frameworks designed to accelerate LLM-powered application development. It connects seamlessly with vector databases, APIs, cloud platforms, and agent frameworks to create dynamic, intelligent workflows.

LangChain ecosystem

- **LangChain** helps you build LLM applications
- **LangGraph** provides advanced control for complex workflows
- **LangSmith** ensures your applications run smoothly and efficiently.



LangChain ecosystem



LangChain Open-source Libraries

Concretely, the framework consists of the following open-source libraries:

LangChain Documentation

- `langchain-core`: Base abstractions and LangChain Expression Language.
- `langchain-community`: Third party integrations.
  - Partner packages (e.g. `langchain-openai`, `langchain-anthropic`, etc.): Some integrations have been further split into their own lightweight packages that only depend on `langchain-core`.
- `langchain`: Chains, agents, and retrieval strategies that make up an application's cognitive architecture.
- langgraph: Build robust and stateful multi-actor applications with LLMs by modeling steps as edges and nodes in a graph.
- langserve: Deploy LangChain chains as REST APIs.

The broader ecosystem includes:

- LangSmith: A developer platform that lets you debug, test, evaluate, and monitor LLM applications and seamlessly integrates with LangChain.

LangChain Core

GUVI | HCL
Skill Up. Level Up

- **What it is**: The foundational library or toolkit for LangChain. It contains the core functionalities required to build and manage chains for large language models (LLMs).
- **Key Features**:
  - Tools to create chains (step-by-step processes involving prompts, models, and logic).
  - Core APIs for handling LLMs, memory, tools, and agents.
- **Example**: If you need a basic workflow like asking a model a question and getting a response, LangChain Core handles that.
- **Analogy**: Think of it as the "engine" of a car — it powers everything LangChain does.

The foundation — has the basic tools and rules for building with LangChain.

LangChain Core

GUVI | HCL
Skill Up. Level Up

```python
from langchain_core.prompts import PromptTemplate
from langchain_core.runnables import RunnableLambda, RunnableMap
from langchain_core.output_parsers import StrOutputParser
from openai import OpenAI

# Define your LLM (OpenAI client)
from langchain_openai import ChatOpenAI
llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)

# Define a prompt template
prompt = PromptTemplate.from_template("What is the capital of {country}?")

# Define output parser
output_parser = StrOutputParser()

# Create a simple chain: input → prompt → LLM → parse output
chain = (
    {"country": lambda x: x["country"]}  # extract input
    | prompt                             # format it
    | llm                                # get LLM output
    | output_parser                      # parse to string
)

# Run the chain
response = chain.invoke({"country": "France"})
print(response)

```

LangChain

❑ **What it is**: A broader framework or ecosystem that includes LangChain Core along with additional integrations and tools.

❑ **Key Features**:

- Combines LangChain Core with integrations for databases, APIs, and external tools (like OpenAI, Hugging Face, SQL, etc.).
- Allows you to build more complex applications, such as AI chatbots, data pipelines, or reasoning systems.

❑ **Example**: You use LangChain to integrate an LLM with a database to answer questions about the data.

❑ **Analogy**: It's the **"fully assembled car"** with wheels, a steering wheel, and everything else you need to drive.

LangChain

```python
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
import os

# Set your OpenAI API key
os.environ["OPENAI_API_KEY"] = "your-api-key"

# 1. Initialize the LLM (e.g., GPT-3.5)
llm = OpenAI(temperature=0)

# 2. Create a PromptTemplate
prompt = PromptTemplate(
    input_variables=["topic"],
    template="Explain {topic} in simple terms."
)

# 3. Create an LLMChain
chain = LLMChain(llm=llm, prompt=prompt)

# 4. Run the chain with an input
response = chain.run("quantum computing")
print(response)

```

LangChain Community

- **What it is**: A collaborative space where developers, contributors, and users of LangChain share ideas, tools, and projects.
- **Key Features**:

- Open-source contributions, such as community-built modules or extensions.
- Discussions, tutorials, and shared resources for learning and improving LangChain.

- **Example**: A community-built extension to integrate LangChain with a new AI model or a guide on using LangChain for a chatbot.
- **Analogy**: It's the "community garage" where people discuss upgrades, share spare parts, and help each other fix their cars.

LangChain Community

```python
from langchain_community.document_loaders import WebBaseLoader
from langchain_community.vectorstores import FAISS
from langchain_community.tools import SerpAPIWrapper

from langchain.embeddings import OpenAIEmbeddings
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI

import os

# Set your OpenAI and SerpAPI keys
os.environ["OPENAI_API_KEY"] = "your-openai-api-key"
os.environ["SERPAPI_API_KEY"] = "your-serpapi-key"

# 1. Load content from a webpage
loader = WebBaseLoader("https://en.wikipedia.org/wiki/LangChain")
docs = loader.load()

# 2. Convert to vector store using OpenAI embeddings
embeddings = OpenAIEmbeddings()
vectorstore = FAISS.from_documents(docs, embeddings)

# 3. Create a retriever
retriever = vectorstore.as_retriever()

# 4. Set up QA chain
llm = ChatOpenAI(temperature=0)
qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever)

# 5. Ask a question about the webpage
query = "Who created LangChain?"
response = qa_chain.run(query)
print("Answer:", response)

# 6. BONUS: Use a community search tool (SerpAPI)
search = SerpAPIWrapper()
print("Live search result:", search.run("LangChain Python framework"))
```