LangChain Introduction

LangChain is a framework for developing applications powered by language models. It enables applications that:

- **Are context-aware**: connect a language model to sources of context (prompt instructions, few shot examples, content to ground its response in, etc.)
- **Reason**: rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.)

This framework consists of several parts.

- **LangChain Libraries**: The Python and JavaScript libraries. Contains interfaces and integrations for a myriad of components, a basic run time for combining these components into chains and agents, and off-the-shelf implementations of chains and agents.
- **LangChain Templates**: A collection of easily deployable reference architectures for a wide variety of tasks.
- **LangServe**: A library for deploying LangChain chains as a REST API.
- **LangSmith**: A developer platform that lets you debug, test, evaluate, and monitor chains built on any LLM framework and seamlessly integrates with LangChain.

LangChain Libraries

The main value props of the LangChain packages are:

1. **Components**: composable tools and integrations for working with language models. Components are modular and easy-to-use, whether you are using the rest of the LangChain framework or not
2. **Off-the-shelf chains**: built-in assemblages of components for accomplishing higher-level tasks

Off-the-shelf chains make it easy to get started. Components make it easy to customize existing chains and build new ones.

The LangChain libraries themselves are made up of several different packages.

- `langchain-core`: Base abstractions and LangChain Expression Language.
- `langchain-community`: Third party integrations.

- `langchain`: Chains, agents, and retrieval strategies that make up an application's cognitive architecture.

Retrieval

Many LLM applications require user-specific data that is not part of the model's training set. The primary way of accomplishing this is through Retrieval Augmented Generation (RAG). In this process, external data is *retrieved* and then passed to the LLM when doing the *generation* step.

LangChain provides all the building blocks for RAG applications - from simple to complex. This section of the documentation covers everything related to the *retrieval* step - e.g. the fetching of the data. Although this sounds simple, it can be subtly complex. This encompasses several key modules.

Document loaders

Document loaders load documents from many different sources. LangChain provides over 100 different document loaders as well as integrations with other major providers in the space, like AirByte and Unstructured. LangChain provides integrations to load all types of documents (HTML, PDF, code) from all types of locations (private S3 buckets, public websites).

Text Splitting

A key part of retrieval is fetching only the relevant parts of documents. This involves several transformation steps to prepare the documents for retrieval. One of the primary ones here is splitting (or chunking) a large document into smaller chunks. LangChain provides several transformation algorithms for doing this, as well as logic optimized for specific document types (code, markdown, etc).

Text embedding models

Another key part of retrieval is creating embeddings for documents. Embeddings capture the semantic meaning of the text, allowing you to quickly and efficiently find other pieces of a text that are similar. LangChain provides integrations with over 25 different embedding providers and methods, from open-source to proprietary API,

allowing you to choose the one best suited for your needs. LangChain provides a standard interface, allowing you to easily swap between models.

Vector stores

With the rise of embeddings, there has emerged a need for databases to support efficient storage and searching of these embeddings. LangChain provides integrations with over 50 different vectorstores, from open-source local ones to cloud-hosted proprietary ones, allowing you to choose the one best suited for your needs. LangChain exposes a standard interface, allowing you to easily swap between vector stores.

Retrievers

Once the data is in the database, you still need to retrieve it. LangChain supports many different retrieval algorithms and is one of the places where we add the most value. LangChain supports basic methods that are easy to get started - namely simple semantic search. However, we have also added a collection of algorithms on top of this to increase performance. These include:

- Parent Document Retriever: This allows you to create multiple embeddings per parent document, allowing you to look up smaller chunks but return larger context.
- Self Query Retriever: User questions often contain a reference to something that isn't just semantic but rather expresses some logic that can best be represented as a metadata filter. Self-query allows you to parse out the *semantic* part of a query from other *metadata filters* present in the query.
- Ensemble Retriever: Sometimes you may want to retrieve documents from multiple different sources, or using multiple different algorithms. The ensemble retriever allows you to easily do this.
- And more!

Indexing

The LangChain **Indexing API** syncs your data from any source into a vector store, helping you:

- Avoid writing duplicated content into the vector store
- Avoid re-writing unchanged content
- Avoid re-computing embeddings over unchanged content

All of which should save you time and money, as well as improve your vector search results.

Agents

The core idea of agents is to use a language model to choose a sequence of actions to take. In chains, a sequence of actions is hardcoded (in code). In agents, a language model is used as a reasoning engine to determine which actions to take and in which order.

Quickstart

For a quick start to working with agents, please check out this getting started guide. This covers basics like initializing an agent, creating tools, and adding memory.

Concepts

There are several key concepts to understand when building agents: Agents, AgentExecutor, Tools, Toolkits. For an in depth explanation, please check out this conceptual guide

Agent Types

There are many different types of agents to use. For a overview of the different types and when to use them, please check out this section.

Tools

Agents are only as good as the tools they have. For a comprehensive guide on tools, please see this section.

How To Guides

Agents have a lot of related functionality! Check out comprehensive guides including:

- Building a custom agent
- Streaming (of both intermediate steps and tokens
- Building an agent that returns structured output
- Lots functionality around using AgentExecutor, including: using it as an iterator, handle parsing errors, returning intermediate steps, capping the max number of iterations, and timeouts for agents