What is LangChain?

LangChain overview

LangChain is an open source orchestration framework for application development using large language models (LLMs). Available in both Python- and Javascript-based libraries, LangChain's tools and APIs simplify the process of building LLM-driven applications like chatbots and AI agents.

LangChain serves as a generic interface for nearly any LLM, providing a centralized development environment to build LLM applications and integrate them with external data sources and software workflows. LangChain's module-based approach allows developers and data scientists to dynamically compare different prompts and even different foundation models with minimal need to rewrite code. This modular environment also allows for programs that use multiple LLMs: for example, an application that uses one LLM to interpret user queries and another LLM to author a response.

Launched by Harrison Chase in October 2022, LangChain enjoyed a meteoric rise to prominence: as of June 2023, it was the single fastest-growing open source project on Github.[1] Coinciding with the momentous launch of OpenAI's ChatGPT the following month, LangChain has played a significant role in making generative AI (genAI) more accessible to enthusiasts and startups in the wake of its widespread popularity. Advancements in accessibility for agentic AI are currently enabling a revolution in automation.

LangChain can facilitate most use cases for LLMs and natural language processing (NLP), like chatbots, intelligent search, question-answering, summarization services or even AI agents capable of robotic process automation.

Integrations with LLMs

LLMs are not standalone applications: they are pre-trained statistical models that must be paired with an application (and, in some cases, specific data sources) in order to meet their purpose.

For example, Chat-GPT is not an LLM: it is a chatbot application that, depending on the version you've chosen, uses the GPT-3.5 or GPT-4 language model. While it's the GPT model that interprets the user's input and composes a natural language response, it's the application that (among other things) provides an interface for the user to type and read and a UX design that governs the chatbot experience. Even at the enterprise level, Chat-GPT is not the only application using the GPT model: Microsoft uses GPT-4 to power Bing Chat.

Furthermore, though foundation models (like those powering LLMs) are pre-trained on massive datasets, they are not omniscient. If a particular task requires access to specific contextual information, like internal documentation or domain expertise, LLMs must be connected to those external data sources. Even if you simply want your model to reflect real-time awareness of current events, it requires external information: a model's internal data is only up-to-date through the time period during which it was pre-trained.

Likewise, if a given generative AI task requires access to external software workflows—for example, if you wanted your virtual agent to integrate with Slack—then you will need a way to integrate the LLM with the API for that software.

While these integrations can generally be achieved with fully manual code, orchestration frameworks such as LangChain and the IBM watsonx portfolio of artificial intelligence products greatly simplify the process. They also make it much easier to experiment with different LLMs to compare results, as different models can be swapped in and out with minimal changes to code.

How does LangChain work?

At LangChain's core is a development environment that streamlines the programming of LLM applications through the use of *abstraction*: the simplification of code by representing one or more complex processes as a named component that encapsulates all of its constituent steps.

Abstractions are a common element of everyday life and language. For example, "π" allows us to represent the ratio of the length of a circle's circumference to that of its diameter without having to write out its infinite digits. Similarly, a thermostat allows us to control the temperature in our home without needing to understand the complex circuitry this entails—we only need to know how different thermostat settings translate to different temperatures.

LangChain is essentially a library of abstractions for Python and Javascript, representing common steps and concepts necessary to work with language models. These modular components—like functions and object classes—serve as the building blocks of generative AI programs. They can be "*chained*" together to create applications, minimizing the amount of code and fine understanding required to execute complex NLP tasks. Though LangChain's abstracted approach may limit the extent to which an expert programmer can finely customize an application, it empowers specialists and newcomers alike to quickly experiment and prototype.

Importing language models

Nearly any LLM can be used in LangChain. Importing language models into LangChain is easy, provided you have an API key. The *LLM* class is designed to provide a standard interface for all models.

Most LLM providers will require you to create an account in order to receive an API key. Some of these APIs—particularly those for proprietary closed-source models, like those offered by OpenAI or Anthropic—may have associated costs.

Many open source models, like Meta AI's LLaMa, Deepseek's Deepseek-LLM, IBM's Granite and Google's Flan-T5, can be accessed through Hugging Face. IBM watsonx, through its partnership with Hugging Face, also offers a curated suite of open source models. Creating an account with either service will allow you to generate an API key for any of the models offered by that provider.

LangChain is not limited to out-of-the-box foundation models: the *CustomLLM* class allows for custom LLM wrappers. Likewise, you can use the IBM watsonx APIs and Python SDK, which includes a LangChain integration, to build applications in LangChain with models that you've already trained or fine-tuned for your specific needs using the *WatsonxLLM* class (and that model's specific project ID).

Prompt templates

Prompts are the instructions given to an LLM. The "art" of composing prompts that effectively provide the context necessary for the LLM to interpret input and structure output in the way most useful to you is often called prompt engineering.

The *PromptTemplate* class in LangChain formalizes the composition of prompts without the need to manually hard code context and queries. Important elements of a prompt are likewise entered as formal classes, like *input_variables*. A prompt template can thus contain and reproduce context, instructions (like "do not use technical terms"), a set of examples to guide its responses (in what is called "few-shot prompting"), a specified output format or a standardized question to be answered. You can save and name an effectively structured prompt template and easily reuse it as needed.

Though these elements can all be manually coded, *PromptTemplate* modules empower smooth integration with other LangChain features, like the eponymous *chains*.

Chains

As its name implies, *chains* are the core of LangChain's workflows. They combine LLMs with other components, creating applications by executing a sequence of functions.

The most basic chain is LLMChain. It simply calls a model and prompt template for that model. For example, imagine you saved a prompt as "ExamplePrompt" and wanted to run it against Flan-T5. You can import LLMChain from langchain.chains, then define chain_example = LLMChain(llm = flan-t5, prompt = ExamplePrompt). To run the chain for a given input, you simply call chain_example.run("input").

To use the output of one function as the input for the next function, you can use *SimpleSequentialChain*. Each function could utilize different prompts, different tools, different parameters or even different models, depending on your specific needs.

Indexes

To achieve certain tasks, LLMs will need access to specific external data sources not included in its training dataset, such as internal documents, emails or datasets. LangChain collectively refers to such external documentation as "indexes".

Document loaders

LangChain offers a wide variety of document loaders for third party applications. This allows for easy importation of data from sources like file storage services (like Dropbox, Google Drive and Microsoft OneDrive), web content (like YouTube, PubMed or specific URLs), collaboration tools (like Airtable, Trello, Figma and Notion), databases (like Pandas, MongoDB and Microsoft), among many others.

Vector databases

Unlike "traditional" structured databases, vector databases represent data points by converting them into *vector embeddings*: numerical representations in the form of vectors with a fixed number of dimensions, often clustering related data points using unsupervised learning methods. This enables low latency queries, even for massive datasets, which greatly increases efficiency. Vector embeddings also store each vector's metadata, further enhancing search possibilities.

LangChain provides integrations for over 25 different embedding methods, as well as for over 50 different vector stores (both cloud-hosted and local).

Text splitters

To increase speed and reduce computational demands, it's often wise to split large text documents into smaller pieces. LangChain's *TextSplitters* split text up into small, semantically meaningful chunks that can then be combined using methods and parameters of your choosing.

Retrieval

Once external sources of knowledge have been connected, the model must be able to quickly retrieve and integrate relevant information as needed. Like watsonx, LangChain offers retrieval augmented generation (RAG): its *retriever* modules accept a string query as an input and return a list of *Document*'s as output.

With LangChain, we can also build agentic RAG systems. In traditional RAG applications, the LLM is provided with a vector database to reference when forming its responses. In contrast, agentic AI applications are not restricted to only data retrieval. Agenic RAG can also encompass tools for tasks such as solving mathematical calculations, writing emails, performing data analysis and more.

Memory

LLMs, by default, do not have any long-term memory of previous interactions (unless that chat history is used as input for a query). LangChain solves this problem with simple utilities for adding memory to a system, with options ranging from retaining the entirety of all conversations to retaining a summarization of the conversation thus far to retaining the *n* most recent exchanges.

Tools

Despite their heralded power and versatility, LLMs have important limitations: namely, a lack of up-to-date information, a lack of domain-specific expertise and a general difficulty with math.

LangChain *tools* are a set of functions that empower LangChain agents to interact with real-world information in order to expand or improve the services it can provide. Examples of prominent pre-built LangChain tools include:

- **Wolfram Alpha:** provides access to powerful computational and data visualization functions, enabling sophisticated mathematical capabilities.
- **Google Search:** provides access to Google Search, equipping applications and agents with real-time information.
- **OpenWeatherMap:** fetches weather information.
- **Wikipedia**: provides efficient access to information from Wikipedia articles.

Your subscription will be delivered in English. You will find an unsubscribe link in every newsletter. You can manage your subscriptions or unsubscribe here. Refer to our IBM Privacy Statement for more information.

LangChain agents

We can build an agent with the LangChain framework to give an LLM the ability to make decisions, use tools and complete complex tasks step-by-step, rather than just generating a single text response. Unlike a simple prompt-response interaction with just an LLM, an agent powered by LangChain can think, plan, execute a sequence of actions, learn and adapt.

LangChain provides a streamlined user experience with a ready-made, extensible framework for creating AI agents, so there's no need to build new tool selection logic, reasoning loops (such as for ReAct agents), observation/action tracking or prompt orchestration and formatting.

The specific LangChain packages, classes and methods vary depending on the AI platform you intend to use. Some key components of the *WatsonxLLM* class that allow for communication with watsonx.ai models using LangChain include:

- ***langchain_ibm***: The package responsible for the LangChain IBM integration. It is necessary to install this package to use any of the following classes and methods.
- ***ibm_watsonx_ai***: The library that allows connection to watsonx.ai services like IBM Cloud and IBM Cloud Pak for Data.
- ***APIClient***: The main class of the *ibm_watsonx_ai* library that manages the API service resources. The parameters include the API credentials and endpoint.
- ***WatsonxLLM***: The wrapper for IBM watsonx.ai foundation models. This wrapper provides chain integration and is necessary to import. The parameters include the model ID, watsonx.ai API key, URL endpoint, project ID as well as any LLM parameters.
- ***ModelInference***: The class that instantiates the model interface. The parameters include the model ID, watsonx.ai credentials, project ID, model parameters and more. Once instantiated, the model can then be passed into the class.
- ***invoke***: The method that calls the model directly with a single prompt of string type.
- ***generate***: The method that calls the model with multiple prompts of string type in a list.

Another LangChain class for building AI agents with the integration of tool calling and chaining with watsonx.ai models is *ChatWatsonx*. This class, which is leveraged in many of our tutorials, uses the *bind_tools* method to pass a list of tools to the LLM upon each iteration. These can include both custom and pre-built tools. To retrieve the AI agent response, the *invoke* method can be used. Once the agent is invoked, the *tool_calls* attribute of the response displays the name, arguments, id and type of each tool call made, if any.

LangGraph

LangGraph, created by LangChain, is an open source AI agent framework that supports multi-agent orchestration and enables developers to build agentic workflows where different agents interact, specialize and collaborate.

At its core, LangGraph uses the power of graph-based architectures to model and manage the intricate relationships between various components of an AI agent workflow. Combined with the human-in-the-loop monitoring mechanism and a set of API and tool integrations, LangGraph provides users with a versatile platform for developing AI solutions and workflows including chatbots, state graphs and other agent-based systems.

With the *langchain-mcp-adapters* library, LangGraph agents can also use tools defined on model context protocol (MCP) servers. The *mcp* library allows users to build custom MCP servers as well. Essentially, MCP enables a secure connection between an AI system, such as an AI agent, and external tools. Thus, various LLMs can connect to the same tools and data sources given the standard MCP.

LangSmith

Released in the fall of 2023, LangSmith aims to bridge the gap between the accessible prototyping capabilities that brought LangChain to prominence and building production-quality LLM applications.

LangSmith provides tools to monitor, evaluate and debug applications, including the ability to automatically trace all model calls to spot errors and test performance under different model configurations. The use of LangSmith is not limited to applications built using the LangChain ecosystem. The evaluation of agent performance is done using LLM-as-a-judge evaluators. This observability and these key metrics aim to optimize more robust, cost-efficient applications.

Getting started with LangChain

LangChain is open source and free to use: source code is available for download on Github.

LangChain can also be installed on Python with a simple pip command: *pip install langchain*. To install all LangChain dependencies (rather than only those you find necessary), you can run the command *pip install langchain[all]*.

Many step-by-step tutorials are provided by IBM including LangChain tool calling, agentic RAG, LLM agent orchestration, agentic chunking and more.

LangChain use cases

AI Applications made with LangChain provide great utility for a variety of use cases, from straightforward question-answering and text generation tasks to more complex solutions that use an LLM as a “reasoning engine.”

Chatbots

Chatbots are among the most intuitive uses of LLMs. LangChain can be used to provide proper context for the specific use of a chatbot, and to integrate chatbots into existing communication channels and workflows with their own APIs.

Summarization

Language models can be tasked with summarizing many types of text, from breaking down complex academic articles and transcripts to providing a digest of incoming emails.

Question answering

Using specific documents or specialized knowledge bases (like Wolfram, arXiv or PubMed), LLMs can retrieve relevant information from storage and articulate helpful answers). If fine-tuned or properly prompted, some LLMs can answer many questions even without external information.

Data augmentation

LLMs can be used to generate synthetic data for use in machine learning. For example, an LLM can be trained to generate additional data samples that closely resemble the data points in a training dataset.

Virtual agents

Integrated with the right workflows, LangChain's Agent modules can use an LLM to autonomously determine next steps and take action using robotic process automation (RPA).