Conceptual guide

Open on GitHub

This guide provides explanations of the key concepts behind the LangChain framework and AI applications more broadly.

We recommend that you go through at least one of the Tutorials before diving into the conceptual guide. This will provide practical context that will make it easier to understand the concepts discussed here.

The conceptual guide does not cover step-by-step instructions or specific implementation examples — those are found in the How-to guides and Tutorials. For detailed reference material, please see the API reference.

High level

- **Why LangChain?**: Overview of the value that LangChain provides.
- **Architecture**: How packages are organized in the LangChain ecosystem.

Concepts

- **Chat models**: LLMs exposed via a chat API that process sequences of messages as input and output a message.
- **Messages**: The unit of communication in chat models, used to represent model input and output.
- **Chat history**: A conversation represented as a sequence of messages, alternating between user messages and model responses.
- **Tools**: A function with an associated schema defining the function's name, description, and the arguments it accepts.
- **Tool calling**: A type of chat model API that accepts tool schemas, along with messages, as input and returns invocations of those tools as part of the output message.
- **Structured output**: A technique to make a chat model respond in a structured format, such as JSON that matches a given schema.
- **Memory**: Information about a conversation that is persisted so that it can be used in future conversations.
- **Multimodality**: The ability to work with data that comes in different forms, such as text, audio, images, and video.
- **Runnable interface**: The base abstraction that many LangChain components and the LangChain Expression Language are built on.
- **Streaming**: LangChain streaming APIs for surfacing results as they are generated.
- **LangChain Expression Language (LCEL)**: A syntax for orchestrating LangChain components. Most useful for simpler applications.
- **Document loaders**: Load a source as a list of documents.

- **Retrieval**: Information retrieval systems can retrieve structured or unstructured data from a datasource in response to a query.
- **Text splitters**: Split long text into smaller chunks that can be individually indexed to enable granular retrieval.
- **Embedding models**: Models that represent data such as text or images in a vector space.
- **Vector stores**: Storage of and efficient search over vectors and associated metadata.
- **Retriever**: A component that returns relevant documents from a knowledge base in response to a query.
- **Retrieval Augmented Generation (RAG)**: A technique that enhances language models by combining them with external knowledge bases.
- **Agents**: Use a language model to choose a sequence of actions to take. Agents can interact with external resources via tool.
- **Prompt templates**: Component for factoring out the static parts of a model "prompt" (usually a sequence of messages). Useful for serializing, versioning, and reusing these static parts.
- **Output parsers**: Responsible for taking the output of a model and transforming it into a more suitable format for downstream tasks. Output parsers were primarily useful prior to the general availability of tool calling and structured outputs.
- **Few-shot prompting**: A technique for improving model performance by providing a few examples of the task to perform in the prompt.
- **Example selectors**: Used to select the most relevant examples from a dataset based on a given input. Example selectors are used in few-shot prompting to select examples for a prompt.
- **Async programming**: The basics that one should know to use LangChain in an asynchronous context.
- **Callbacks**: Callbacks enable the execution of custom auxiliary code in built-in components. Callbacks are used to stream outputs from LLMs in LangChain, trace the intermediate steps of an application, and more.
- **Tracing**: The process of recording the steps that an application takes to go from input to output. Tracing is essential for debugging and diagnosing issues in complex applications.
- **Evaluation**: The process of assessing the performance and effectiveness of AI applications. This involves testing the model's responses against a set of predefined criteria or benchmarks to ensure it meets the desired quality standards and fulfills the intended purpose. This process is vital for building reliable applications.
- **Testing**: The process of verifying that a component of an integration or application works as expected. Testing is essential for ensuring that the application behaves correctly and that changes to the codebase do not introduce new bugs.

Glossary

- **AIMessageChunk**: A partial response from an AI message. Used when streaming responses from a chat model.
- **AIMessage**: Represents a complete response from an AI model.

- astream_events: Stream granular information from LCEL chains.
- BaseTool: The base class for all tools in LangChain.
- batch: Use to execute a runnable with batch inputs.
- bind_tools: Allows models to interact with tools.
- Caching: Storing results to avoid redundant calls to a chat model.
- Chat models: Chat models that handle multiple data modalities.
- Configurable runnables: Creating configurable Runnables.
- Context window: The maximum size of input a chat model can process.
- Conversation patterns: Common patterns in chat interactions.
- Document: LangChain's representation of a document.
- Embedding models: Models that generate vector embeddings for various data types.
- HumanMessage: Represents a message from a human user.
- InjectedState: A state injected into a tool function.
- InjectedStore: A store that can be injected into a tool for data persistence.
- InjectedToolArg: Mechanism to inject arguments into tool functions.
- input and output types: Types used for input and output in Runnables.
- Integration packages: Third-party packages that integrate with LangChain.
- Integration tests: Tests that verify the correctness of the interaction between components, usually run with access to the underlying API that powers an integration.
- invoke: A standard method to invoke a Runnable.
- JSON mode: Returning responses in JSON format.
- langchain-community: Community-driven components for LangChain.
- langchain-core: Core langchain package. Includes base interfaces and in-memory implementations.
- langchain: A package for higher level components (e.g., some pre-built chains).
- langgraph: Powerful orchestration layer for LangChain. Use to build complex pipelines and workflows.
- langserve: Used to deploy LangChain Runnables as REST endpoints. Uses FastAPI. Works primarily for LangChain Runnables, does not currently integrate with LangGraph.
- LLMs (legacy): Older language models that take a string as input and return a string as output.
- Managing chat history: Techniques to maintain and manage the chat history.
- OpenAI format: OpenAI's message format for chat models.
- Propagation of RunnableConfig: Propagating configuration through Runnables. Read if working with python 3.9, 3.10 and async.
- rate-limiting: Client side rate limiting for chat models.
- RemoveMessage: An abstraction used to remove a message from chat history, used primarily in LangGraph.

- role: Represents the role (e.g., user, assistant) of a chat message.
- RunnableConfig: Use to pass run time information to Runnables (e.g., `run_name`, `run_id`, `tags`, `metadata`, `max_concurrency`, `recursion_limit`, `configurable`).
- Standard parameters for chat models: Parameters such as API key, `temperature`, and `max_tokens`.
- Standard tests: A defined set of unit and integration tests that all integrations must pass.
- stream: Use to stream output from a Runnable or a graph.
- Tokenization: The process of converting data into tokens and vice versa.
- Tokens: The basic unit that a language model reads, processes, and generates under the hood.
- Tool artifacts: Add artifacts to the output of a tool that will not be sent to the model, but will be available for downstream processing.
- Tool binding: Binding tools to models.
- @tool: Decorator for creating tools in LangChain.
- Toolkits: A collection of tools that can be used together.
- ToolMessage: Represents a message that contains the results of a tool execution.
- Unit tests: Tests that verify the correctness of individual components, run in isolation without access to the Internet.
- Vector stores: Datastores specialized for storing and efficiently searching vector embeddings.
- with_structured_output: A helper method for chat models that natively support tool calling to get structured output matching a given schema specified via Pydantic, JSON schema or a function.
- with_types: Method to overwrite the input and output types of a runnable. Useful when working with complex LCEL chains and deploying with LangServe.