Concepts

Concept

The core element of any language model application is...the model. LangChain gives you the building blocks to interface with any language model. Everything in this section is about making it easier to work with models. This largely involves a clear interface for what a model is, helper utils for constructing inputs to models, and helper utils for working with the outputs of models.

Models

There are two main types of models that LangChain integrates with: LLMs and Chat Models. These are defined by their input and output types.

- **LLMs**
  LLMs in LangChain refer to pure text completion models. The APIs they wrap take a string prompt as input and output a string completion. OpenAI's GPT-3 is implemented as an LLM.

- **Chat Models**
  Chat models are often backed by LLMs but tuned specifically for having conversations. Crucially, their provider APIs use a different interface than pure text completion models. Instead of a single string, they take a list of chat messages as input and they return an AI message as output. See the section below for more details on what exactly a message consists of. GPT-4 and Anthropic's Claude-2 are both implemented as chat models.

- **Considerations**
  These two API types have pretty different input and output schemas. This means that best way to interact with them may be quite different. Although LangChain makes it possible to treat them interchangeably, that doesn't mean you **should**. In particular, the prompting strategies for LLMs vs ChatModels may be quite different. This means that you will want to make sure the prompt you are using is designed for the model type you are working with.

  Additionally, not all models are the same. Different models have different prompting strategies that work best for them. For example, Anthropic's models work best with XML while OpenAI's work best with JSON. This means that the prompt you use for one model may not transfer to other ones. LangChain provides a lot of default prompts, however these are not guaranteed to work well with the model are you using. Historically speaking, most prompts work well with OpenAI but are not heavily tested on other models. This is something we are working to address, but it is something you should keep in mind.

Messages

ChatModels take a list of messages as input and return a message. There are a few different types of messages. All messages have a role and a content property.
The role describes WHO is saying the message. LangChain has different message classes for different roles.
The content property describes the content of the message.
This can be a few different things:

- A string (most models are this way)
- A List of dictionaries (this is used for multi-modal input, where the dictionary contains information about that input type and that input location)

In addition, messages have an additional_kwargs property. This is where additional information about messages can be passed. This is largely used for input parameters that are *provider specific* and not general. The best known example of this is function_call from OpenAI.

- **HumanMessage**
  This represents a message from the user. Generally consists only of content.

- **AIMessage**
  This represents a message from the model. This may have additional_kwargs in it - for example functional_call if using OpenAI Function calling.

- **SystemMessage**
  This represents a system message. Only some models support this. This tells the model how to behave. This generally only consists of content.

- **FunctionMessage**
  This represents the result of a function call. In addition to role and content, this message has a name parameter which conveys the name of the function that was called to produce this result.

- **ToolMessage**
  This represents the result of a tool call. This is distinct from a FunctionMessage in order to match OpenAI's function and tool message types. In addition to role and content, this message has a tool_call_id parameter which conveys the id of the call to the tool that was called to produce this result.

Prompts

The inputs to language models are often called prompts. Oftentimes, the user input from your app is not the direct input to the model. Rather, their input is transformed in some way to produce the string or list of messages that does go into the model. The objects that take user input and transform it into the final string or messages are known as "Prompt Templates". LangChain provides several abstractions to make working with prompts easier.

- **PromptValue**
  ChatModels and LLMs take different input types. PromptValue is a class designed to be interoperable between the two. It exposes a method to be cast to a string (to work with LLMs) and another to be cast to a list of messages (to work with ChatModels).

- **PromptTemplate**
  This is an example of a prompt template. This consists of a template string. This string is then formatted with user inputs to produce a final string.

- **MessagePromptTemplate**
  This is an example of a prompt template. This consists of a template **message** - meaning a specific role and a PromptTemplate. This PromptTemplate is then formatted with user inputs to produce a final string that becomes the content of this message.

- **HumanMessagePromptTemplate**
  This is MessagePromptTemplate that produces a HumanMessage.

- **AIMessagePromptTemplate**
  This is MessagePromptTemplate that produces an AIMessage.

- **SystemMessagePromptTemplate**
  This is MessagePromptTemplate that produces a SystemMessage.

- **MessagesPlaceholder**
  Oftentimes inputs to prompts can be a list of messages. This is when you would use a MessagesPlaceholder. These objects are parameterized by a variable_name argument. The input with the same value as this variable_name value should be a list of messages.

- **ChatPromptTemplate**
  This is an example of a prompt template. This consists of a list of MessagePromptTemplates or MessagePlaceholders. These are then formatted with user inputs to produce a final list of messages.

Output Parsers

The output of models are either strings or a message. Oftentimes, the string or messages contains information formatted in a specific format to be used downstream (e.g. a comma separated list, or JSON blob). Output parsers are responsible for taking in the output of a model and transforming it into a more usable form. These generally work on the content of the output message, but occasionally work on values in the additional_kwargs field.

- **StrOutputParser**
  This is a simple output parser that just converts the output of a language model (LLM or ChatModel) into a string. If the model is an LLM (and therefore outputs a string) it just passes that string through. If the output is a ChatModel (and therefore outputs a message) it passes through the .content attribute of the message.

- **OpenAI Functions Parsers**
  There are a few parsers dedicated to working with OpenAI function calling. They take the output of the function_call and arguments parameters (which are inside additional_kwargs) and work with those, largely ignoring content.

- **Agent Output Parsers**
  Agents are systems that use language models to determine what steps to take. The output of a language model therefore needs to be parsed into some schema that can represent what actions (if any) are to be taken. AgentOutputParsers are responsible for taking raw LLM or ChatModel output and converting it to that schema. The logic inside these output parsers can differ depending on the model and prompting strategy being used.