Cheat Sheet: Using Built-in Agents in LangChain

Estimated time needed: 20 minutes

I. Understanding built-in agents

LangChain provides built-in agents that enable LLMs to make decisions, use tools, access memory, and interact with external systems. These agents are typically accessed in two ways:

Prebuilt agent creators	Utility functions that help quickly set up ready-to-use agents for common use cases (for example, SQL, CSV, Pandas).
LangGraph ReActive	Graph-structured agents with explicit reasoning steps and control flow logic.

II. Exploring core agent types

These agent types define the basic way an agent thinks, chooses tools, and performs actions. They are foundational and can be used to build more specialized agents.

Agent type	Description
`ZERO_SHOT_REACT_DESCRIPTION`	Performs reasoning before acting. Picks the right tool using only its description.
`REACT_DOCSTORE`	Zero-shot agent with access to a document store for retrieving info (for example, Wikipedia).
`SELF_ASK_WITH_SEARCH`	Breaks complex questions into simpler ones and answers using a search tool.
`CONVERSATIONAL_REACT_DESCRIPTION`	Maintains conversation history while reasoning and acting.
`CHAT_ZERO_SHOT_REACT_DESCRIPTION`	Like a zero-shot agent but optimized for chat models like GPT-4.
`CHAT_CONVERSATIONAL_REACT_DESCRIPTION`	Chat-based version of conversational ReAct agent.
`STRUCTURED_CHAT_ZERO_SHOT_REACT_DESCRIPTION`	Optimized for chat models and structured tools with multiple inputs.
`OPENAI_FUNCTIONS`	Designed to work with OpenAI's function calling schema.
`OPENAI_MULTI_FUNCTIONS`	Handles multiple tools using OpenAI's multi-function architecture.

III. Model compatibility

Some LLMs may not fully support structured output parsing required by certain agents (for example, structured-chat-zero-shot-react-description). If the tool returns a dictionary or complex output, these models might fail with parsing or validation errors.

Description	Code

You can create an agent using `initialize_agent`, where you pass the agent type, so it behaves according to the agent type. The agent in this code will perform reasoning before acting.	`from langchain.agents import initialize_agent`
	`from langchain.agents.agent_types import AgentType`
	
	`agent = initialize_agent([tools],`
	`llm,`
	`agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,`
	`verbose=True,`
	`handle_parsing_errors=True)`

IV. Prebuilt agents (task-specific utilities)

These utilities use the core agent types behind the scenes but make it easy to set up agents for specific types of data or tasks.

Function	Purpose	Agent type (internally used by default)
`create_pandas_dataframe_agent()`	Natural language to DataFrame analysis and visualization	`zero-shot-react-description`
`create_csv_agent()`	Query CSV files conversationally	`zero-shot-react-description`
`create_sql_agent()`	Natural language SQL queries	`zero-shot-react-description`
`create_openai_functions_agent()`	Agent with OpenAI function-calling tools	`openai-functions`
`create_tool_calling_agent()`	Generic agent that invokes structured tools	`structured-chat-zero-shot-react-description`

Description	Code

This code shows an example of create_pandas_dataframe_agent where you pass in the LLM, the dataframe you are working on, the agent type, and verbose. You can use this to perform dataframe analysis using natural language.	`from langchain_experimental.agents import create_pandas_dataframe_agent`
	
	`agent_executor = create_pandas_dataframe_agent(`
	`  llm,`
	`  dataframe,`
	`  agent_type="zero-shot-react-description",`
	`  verbose=True`
	`)`

V. LangGraph agents

LangGraph supports building multi-step reasoning agents using a directed graph structure. These agents are ideal for advanced use cases where you want full control over reasoning steps, looping, or conditional logic.

Description	Code

The `create_react_agent()utility` constructs a graph agent that performs tool usage in a loop with reasoning at each step until a stopping condition is met.	`from langgraph.prebuilt import create_react_agent`
	`agent_exec = create_react_agent(model=llm, tools=[tool_list])`
	`msgs = agent_exec.invoke({"messages": [("human", "Add the numbers -10, -20, -30")]})`

Author(s)

IBM Skills Network Team