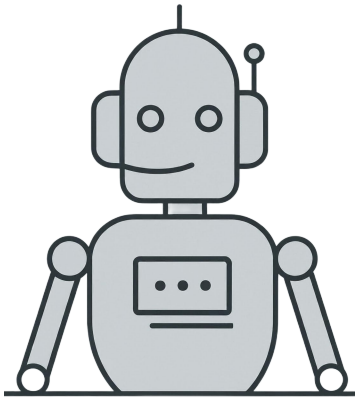
Planning



AI = Intelligent computer programs that can perceive, reason, learn, and act in complex environments (Russell & Norvig, 2022)

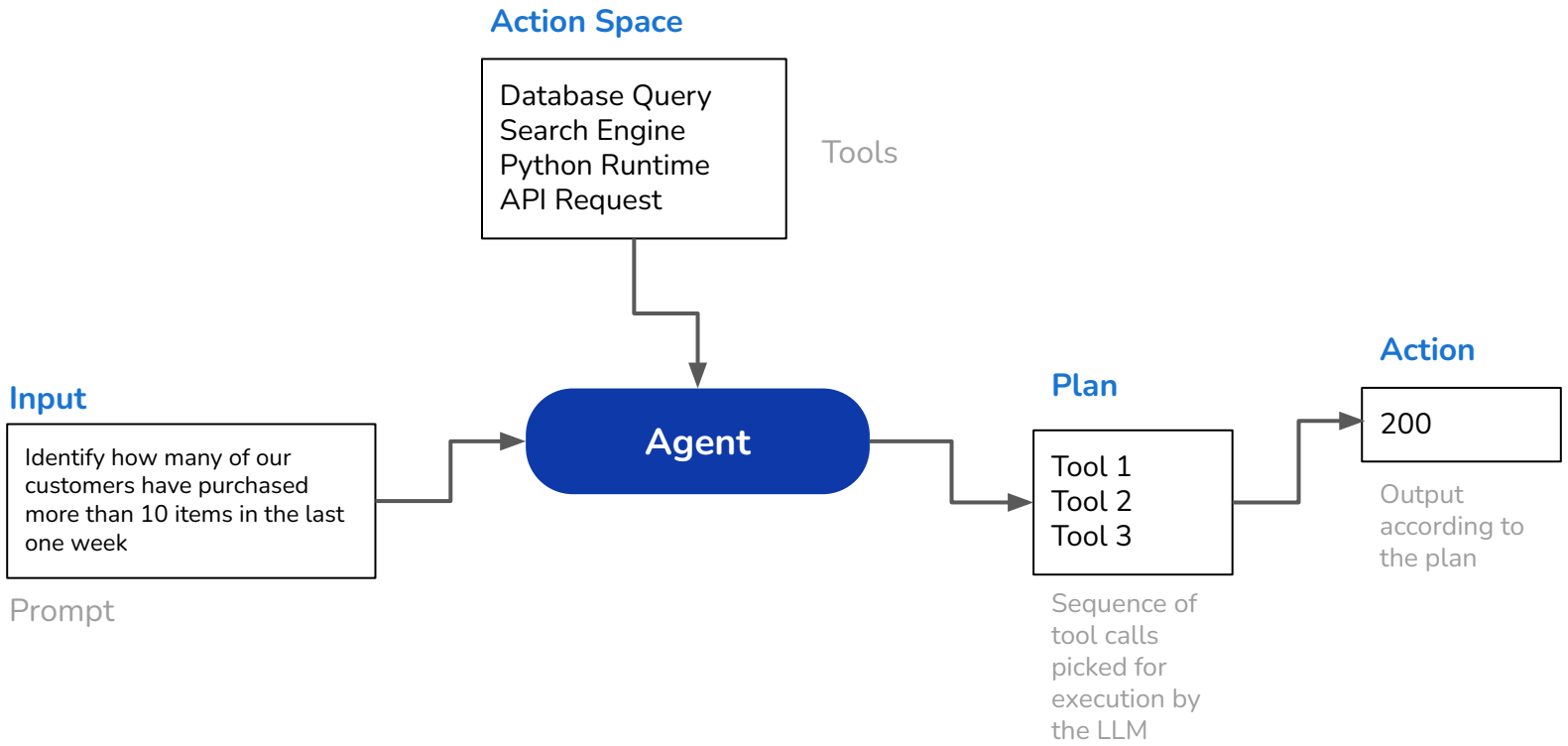
Agenda

In this session, we will discuss:

- Implementation of the ReAct Framework
- Understanding LangChain Abstractions for Planning Agents
- Creating Custom Agents with LangChain

The ReAct Framework

Notation

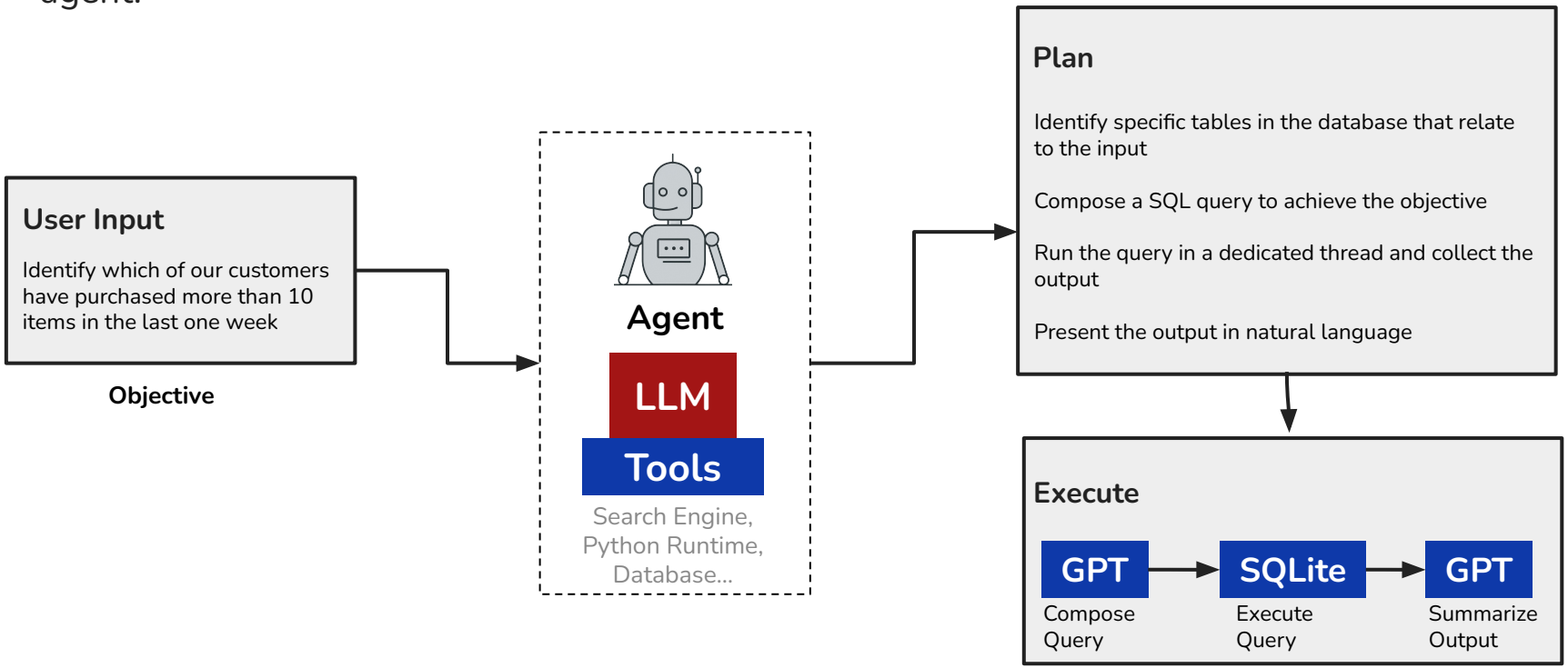


Notation

An Agent operates in an action space defined by the tools it has access to. Given an input prompt, the Agent uses an LLM as a reasoning engine to prepare a plan. The plan is a sequence of tool calls that needs to be executed. The Agent acts by executing the plan and presents the final output back to the user.

Reasoning and Action (ReAct)

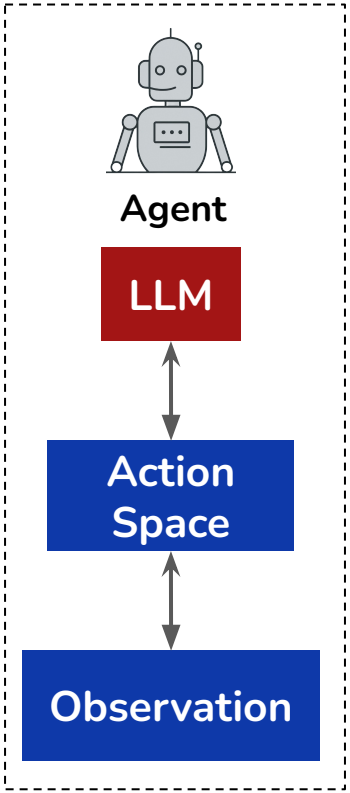
ReAct agents are explicitly primed to reason out a plan before answering user queries. The plan is saved as an inner monologue that is executed using tools available to the agent.



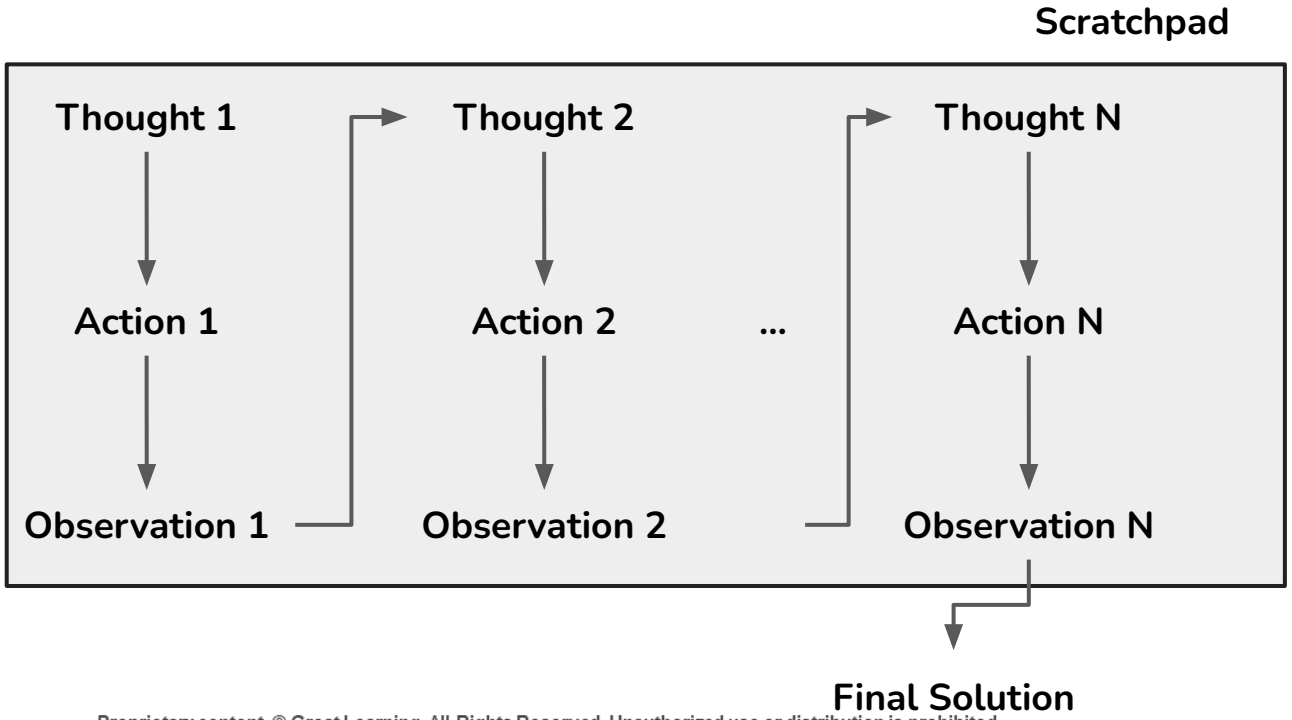
Reasoning and Action (ReAct)

Note:
A specific way to implement planning agents is the ReAct framework. The planning stage relies on the reasoning capability of the LLM to compose a sequence of steps based on the tools available. This plan is executed in a separate thread and the output from each tool execution is presented back to the LLM. The Agent presents the final answer looking at all the interim outputs of the plan.

Reasoning and Action (ReAct)



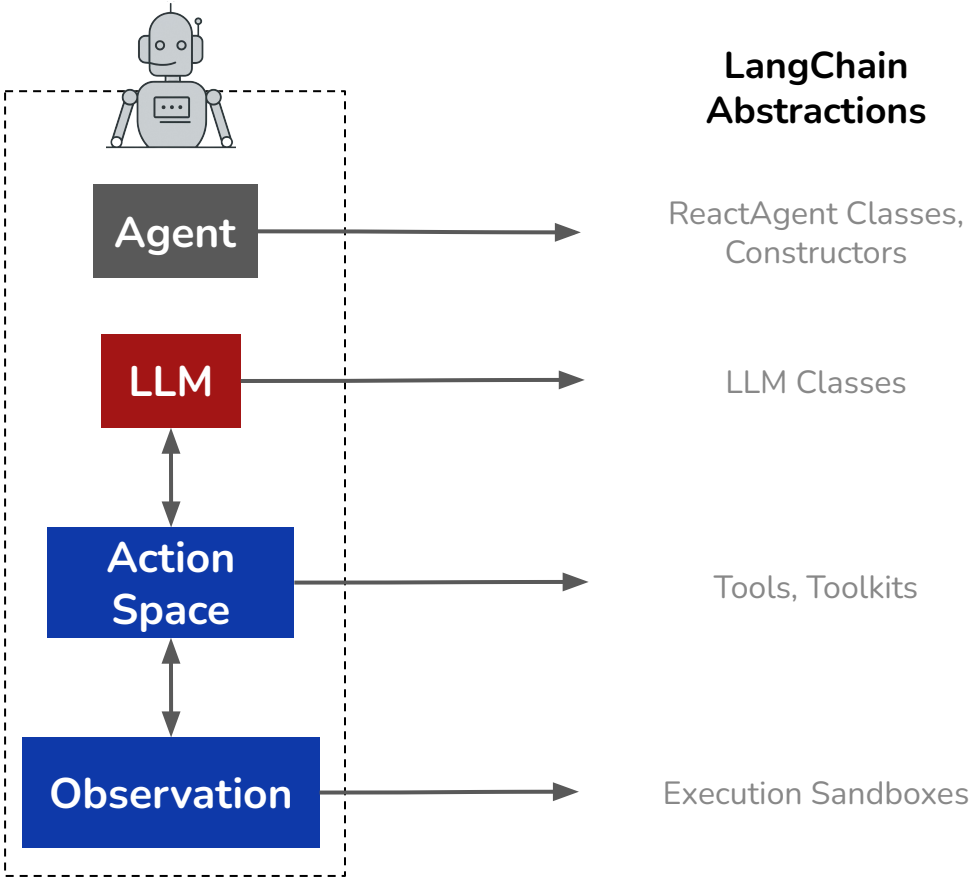
ReAct agents use Thought-Action-Observation chains to organize their planning effort.



Reasoning and Action (ReAct)

Note:
The ReAct framework specifies that the planning stage should use Thought-Action-Observation chains where each thought involves one tool and the output from this tools' execution (action). The output from the tool execution creates an observation and this chain of observations is used to derive the final answer presented to the user. ReAct forces the agent to go into an inner monologue of T-A-O chains instead of directly jumping to an answer.

ReAct x LangChain



LangChain provides convenient constructors to build ReAct Agents. LLM Classes wrap API clients. Tool Classes wrap functions that can be bound to LLM Classes. Finally, Tool Call executions are run in a thread and outputs appended to the prompt.

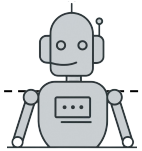
ReAct x LangChain

Note:
Since the ReAct pattern is so popular, LangChain provides several abstractions that implement this pattern. These are available as constructors that operate at different levels of the agentic workflow as depicted in the figure.

Hands-on

Creating a SQL Database ReAct Agent

Customizing ReAct x LangChain



	LangChain Abstractions	Custom Agents
Agent	ReactAgent Classes, Constructors	Custom Classes
LLM	LLM Classes	LlamaCpp
Action Space	Tools, Toolkits	LangChain Tools
Observation	Execution Sandboxes	Custom methods

Defaults of LangChain can be overridden at all levels of abstraction. LLM Classes could be replaced with open LLM Classes. Default tools could be modified with custom functions.

Customizing ReAct x LangChain

Notes:

While LangChain defaults allow convenient abstractions, classes at all the levels of abstraction can be modified. Agent constructors could be replaced with custom Classes (for e.g., wording the ReAct prompt differently). LLM Classes could be replaced with open LLM servers (e.g., LlamaCpp). Built-in tools and toolkits could be augmented or replaced with custom tools that inherit from LangChain classes. Finally, the Agent execution itself could be overridden by custom environments where the execution happens.

Hands-on

Customizing the SQL Database ReAct Agent

Thank You