How to build an AI agent

In this tutorial, we will use prebuilt LangChain tools for an agentic ReAct agent to showcase its ability to differentiate appropriate use cases for each tool. We will primarily be using the open source LangChain Python package.

What is tool calling?

Tool calling, otherwise known as function calling, is the interface that allows artificial intelligence (AI) agents to work on specific tasks that require up-to-date information, otherwise unavailable to the trained large language models (LLMs). LLMs such as IBM® Granite™ models or OpenAI's GPT (generative pre-trained transformer) models have access only to information used in training. There are many default tools accessible through LangChain including a tool for working with SQL database queries, accessing information on Wikipedia and much more. We encourage you to read the LangChain documentation for a comprehensive list of prebuilt tools.

Custom tools can be defined by using various methods including using the @tool decorator or LangChain Runnables which we'll cover in this tutorial. Async tools can be created by using the StructuredTool or the BaseTool classes. For distinctions between each approach, we encourage you to reference the official LangChain documentation. Refer to the IBM function calling tutorial for examples of custom tools.

We encourage you to check out our AI Agents explainer for an in-depth overview of the various AI agent types and how they differ from traditional LLM chatbots.

Prerequisites

You need an IBM Cloud® account.

Steps

Step 1. Set up your environment

While you can choose from several tools, this tutorial walks you through how to set up an IBM account to use a Jupyter Notebook.

1. Log in to watsonx.ai™ using your IBM Cloud account.
2. Create a watsonx.ai project.

   You can get your project ID from within your project. Click the **Manage** tab. Then, copy the project ID from the **Details** section of the **General** page. You need this ID for this tutorial.
3. Create a Jupyter Notebook.

This step will open a Jupyter Notebook environment where you can copy the code from this tutorial. Alternatively, you can download this notebook to your local system and upload it to your watsonx.ai project as an asset. To view more Granite tutorials, check out the IBM Granite Community. This tutorial is also available on Github.

Step 2. Set up a watsonx.ai Runtime instance and API key

1. Create a watsonx.ai Runtime service instance (select your appropriate region and choose the Lite plan, which is a free instance).

2. Generate an API Key.

3. Associate the watsonx.ai Runtime service instance to the project that you created in watsonx.ai.

Step 3. Install and import relevant libraries and set up your credentials

We need a few libraries and modules for this tutorial. Make sure to import the following ones and if they're not installed, you can resolve this with a quick pip installation. LangChain and LangGraph will be the frameworks and developer toolkits used.

Note: This tutorial was built using Python 3.11.9 and is also compatible with Google Colab which uses Python 3.10.12. To check your python version, you can run the !python --version command in a code cell.

```
#installations !pip install -q langchain \     "langchain_community<0.3.0" \     langgraph \
    youtube_search \     pyowm \     ionic-langchain \     python-dotenv !pip install -qU langchain-ibm
```

Powered by Granite

```
#imports import os from langchain_ibm import ChatWatsonx from langgraph.prebuilt import
create_react_agent from langchain_core.tools import Tool from langchain_core.messages import
HumanMessage from langchain_community.tools import YouTubeSearchTool from
langchain_community.utilities import OpenWeatherMapAPIWrapper from ionic_langchain.tool import
IonicTool from dotenv import load_dotenv load_dotenv(os.getcwd()+"/.env", override=True)
```

Powered by Granite

To set our credentials, we need the WATSONX_APIKEY and WATSONX_PROJECT_ID you generated in step 1. You can either store them in an .env file in your directory or replace the placeholder text. We will also set the URL serving as the API endpoint.

```
WATSONX_APIKEY = os.getenv('WATSONX_APIKEY', "<WATSONX_APIKEY_HERE>")
WATSONX_PROJECT_ID = os.getenv('WATSONX_PROJECT_ID', "
<WATSONX_PROJECT_ID_HERE>") URL = "https://us-south.ml.cloud.ibm.com"
```

Powered by Granite

The weather tool used in this tutorial with require an OpenWeather API key. To generate one, create an OpenWeather account. Upon creating an account, select the "API Keys" tab to display your free key.

```
OPENWEATHERMAP_API_KEY = os.getenv('OPENWEATHERMAP_API_KEY', "
<OPEN_WEATHERMAP_API_KEY_HERE>")
```

Powered by Granite

Step 4. Initialize the LLM

For this tutorial, we will be using the ChatWatsonx wrapper to set our chat model. This wrapper simplifies the integration of tool calling and chaining. We will be using the granite-3-8b-instruct model. We encourage you to use the API references in the ChatWatsonx documentation for further information.

To initialize the LLM, we need to set the model parameters. It is important to configure the model's temperature here in order to limit agent's hallucinations.

```
llm = ChatWatsonx(     model_id="ibm/granite-3-8b-instruct",     url = URL,     apikey =
WATSONX_APIKEY,     project_id = WATSONX_PROJECT_ID,     params = {
        "decoding_method": "greedy",        "temperature": 0,        "min_new_tokens": 5,
        "max_new_tokens": 2000     } )
```

Powered by Granite

Step 5. Establish the built-in tools

We can use the Tool class to make our tools callable. A clear and simple description of the tool is also important. Additionally, the return_direct boolean attribute determines whether the tool response should be returned directly to the user. Lastly, the optional args_schema attribute of pydantic.BaseModel type is used to provide additional information or validation to the model.

Let's imagine you are planning your next vacation to Greece and are eager to learn more about it and prepare for the trip. First, let's set up the built-in LangChain weather tool. The tool uses the OpenWeatherMapAPIWrapper that uses the OPENWEATHERMAP_API_KEY we generated earlier.

```
weather =
OpenWeatherMapAPIWrapper(openweathermap_api_key=OPENWEATHERMAP_API_KEY)
weather_search = Tool(     name="weather_search",     description="Get weather for a city and country
code, e.g. Athens, GR",     func=weather.run, )
```

Powered by Granite

Next, let's set up the prebuilt YouTube tool by using the YouTube Search package available through LangChain. This step will be helpful for finding videos about your travel destination.

```
youtube = YouTubeSearchTool() youtube_search = Tool(     name="youtube_search",
    description="Search YouTube for video links.",     func=youtube.run, )
```

Powered by Granite

Lastly, let's set up an online shopping tool using Ionic. This tool returns items relevant to the user query that are on sale through the e-commerce marketplace.

```
ionic_search = IonicTool().tool()
```

Powered by Granite

Let's set our list of the multiple tools we provide to the LLM. We can also print the list to see how it loaded. For an extended list of available LangChain tools, refer to the **LangChain documentation**.

```
tools = [weather_search, youtube_search, ionic_search] tools
```

Powered by Granite

Output:

```
[Tool(name='weather_search', description='Get weather for a city and country code, e.g. Athens, GR',
func=<bound method OpenWeatherMapAPIWrapper.run of OpenWeatherMapAPIWrapper(owm=
<pyowm.owm.OWM - API key=************************611450cc, subscription type=free,
PyOWM version=(3, 3, 0)>, openweathermap_api_key='******************')>),
```

Tool(name='youtube_search', description='Search YouTube for video links.', func=<bound method BaseTool.run of YouTubeSearchTool()>),

Tool(name='ionic_commerce_shopping_tool', description='\nIonic is an e-commerce shopping tool... [abbreviated]', verbose=True, func=<bound method Ionic.query of <ionic_langchain.tool.Ionic object at 0x17f6e5510>>)]

Step 6. Tool calling

Tool calling typically refers to an LLM returning the name of the tool to call and its arguments. We can either use the extracted information for other purposes or we can call the tool with these arguments. For more examples of this, refer to our function calling tutorial.

Actually, running the tool and retrieving its output is not always implied. In this tutorial, we will explore both approaches.

Return the relevant tool and arguments

To accomplish traditional tool calling, we can simply provide a user query and use the prebuilt bind_tools method to pass the list of tools to the LLM upon each iteration.
llm_with_tools = llm.bind_tools(tools) response = llm_with_tools.invoke([("human", "What are some youtube videos about greece")]) response
Powered by Granite

Output:

***AIMessage**(content='', additional_kwargs={'tool_calls': [{'id': 'chatcmpl-tool-7a15abba7d3c4419970d807ac0c8d353', 'type': 'function', 'function': {'name': 'youtube_search', 'arguments': '{"query": "greece"}'}}]}, response_metadata={'token_usage': {'completion_tokens': 21, 'prompt_tokens': 578, 'total_tokens': 599}, 'model_name': 'ibm/granite-3-8b-instruct', 'system_fingerprint': '', 'finish_reason': 'tool_calls'}, id='chat-5fe7a26b8f954c179c4995e873bff91e', tool_calls=[{'name': 'youtube_search', 'args': {'query': 'greece'}, 'id': 'chatcmpl-tool-7a15abba7d3c4419970d807ac0c8d353', 'type': 'tool_call'}], usage_metadata={'input_tokens': 578, 'output_tokens': 21, 'total_tokens': 599})*
response.additional_kwargs
Powered by Granite

Output:

*{'tool_calls': [{'id': 'chatcmpl-tool-7a15abba7d3c4419970d807ac0c8d353',
'type': 'function',
'function': {'name': 'youtube_search',
'arguments': '{"query": "greece"}'}}]}*

As seen in the tool_calls output, the LLM correctly identifies the appropriate tool call and arguments. The LLM does not run the tool itself. We will do this in the next step.

Run the tool call and retrieve its output

To run the tool calls, we first need to create a ReAct agent by using the prebuilt LangGraph create_react_agent helper method. This function creates a graph that serves as the bridge between the chat model and the available tools, thus enabling agentic tool calling. This graph is represented in the following diagram.

```
agent_executor = create_react_agent(llm, tools)
```

Powered by Granite

We are now able to ask the agent questions that require tool calling. First, we can ask the model to return URLs to YouTube videos about Greece. We can use the HumanMessage class to pass the user query to the LLM.

```
user_query = "What are some YouTube videos about Greece" response = agent_executor.invoke({"messages": user_query}) response["messages"]
```

Powered by Granite

Output:

```
[HumanMessage(content='What are some YouTube videos about Greece', additional_kwargs={}, response_metadata={}, id='1adba6c0-32e6-4bbd-92a6-7d21b0177439'), AIMessage(content='', additional_kwargs={'tool_calls': [{'id': 'chatcmpl-tool-b4b5bf452404424ba4d6d9c26e53c6ce', 'type': 'function', 'function': {'name': 'youtube_search', 'arguments': '{"query": "Greece"}'}}]}, response_metadata={'token_usage': {'completion_tokens': 22, 'prompt_tokens': 578, 'total_tokens': 600}, 'model_name': 'ibm/granite-3-8b-instruct', 'system_fingerprint': '', 'finish_reason': 'tool_calls'}, id='chat-5f41aee6736842749285aa7fbff50f65', tool_calls=[{'name': 'youtube_search', 'args': {'query': 'Greece'}, 'id': 'chatcmpl-tool-b4b5bf452404424ba4d6d9c26e53c6ce', 'type': 'tool_call'}], usage_metadata={'input_tokens': 578, 'output_tokens': 22, 'total_tokens': 600}), ToolMessage(content="['https://www.youtube.com/watch?v=waQY2Ucpbd0&pp=ygUGR3JlZWNl', 'https://www.youtube.com/watch?v=NMlBB2pK5qo&pp=ygUGR3JlZWNl']", name='youtube_search', id='1ccf3137-2c10-495e-86ad-a548a3434243', tool_call_id='chatcmpl-tool-b4b5bf452404424ba4d6d9c26e53c6ce'), AIMessage(content='Here are some YouTube videos about Greece:\n\n1. [Greece Travel Guide | Top 10 Tourist Attractions](https://www.youtube.com/watch?v=waQY2Ucpbd0&pp=ygUGR3JlZWNl)\n2. [Greece Travel Guide | Top 10 Tourist Attractions](https://www.youtube.com/watch?v=NMlBB2pK5qo&pp=ygUGR3JlZWNl)', additional_kwargs={}, response_metadata={'token_usage': {'completion_tokens': 117, 'prompt_tokens': 677, 'total_tokens': 794}, 'model_name': 'ibm/granite-3-8b-instruct', 'system_fingerprint': '', 'finish_reason': 'stop'}, id='chat-801e3b596a174ac88246b507c93e5869', usage_metadata={'input_tokens': 677, 'output_tokens': 117, 'total_tokens': 794})]
```

Great! As seen in the AIMessage , the model correctly identified the appropriate tool call. In the ToolMessage , we see the model successfully returned the expected output by using the built-in LangChain YouTube tool. Finally, the AIMessage shows the LLM synthesized the tool response.

Next, let's ask the model about the weather in Greece to determine whether it calls the weather_search tool as expected.

```
user_query = "What is the weather in Athens, GR" response = agent_executor.invoke({"messages": user_query}) response["messages"]
```

Powered by Granite

Output:

```
[HumanMessage(content='What is the weather in Athens, GR', additional_kwargs={}, response_metadata={}, id='a0c4b69c-988a-4f7d-9b8a-4780305f8e2a'), AIMessage(content='', additional_kwargs={'tool_calls': [{'id': 'chatcmpl-tool-9a0c07a3b35f4c69a351c5540ab663f8', 'type': 'function', 'function': {'name': 'weather_search', 'arguments': '{"__arg1": "Athens, GR"}'}}]}, response_metadata={'token_usage':
```

{'completion_tokens': 26, 'prompt_tokens': 579, 'total_tokens': 605}, 'model_name': 'ibm/granite-3-8b-instruct', 'system_fingerprint': '', 'finish_reason': 'tool_calls'}, id='chat-eeed087050e049f0ad715f3615c7fdda', tool_calls=[{'name': 'weather_search', 'args': {'__arg1': 'Athens, GR'}, 'id': 'chatcmpl-tool-9a0c07a3b35f4c69a351c5540ab663f8', 'type': 'tool_call'}], usage_metadata={'input_tokens': 579, 'output_tokens': 26, 'total_tokens': 605}),
***ToolMessage**(content='In Athens, GR, the current weather is as follows:\nDetailed status: few clouds\nWind speed: 4.47 m/s, direction: 23°\nHumidity: 58%\nTemperature: \n - Current: 15.15°C\n - High: 15.74°C\n - Low: 14.1°C\n - Feels like: 14.24°C\nRain: {}\nHeat index: None\nCloud cover: 20%', name='weather_search', id='587b0230-b667-41de-97b9-3779554d2559', tool_call_id='chatcmpl-tool-9a0c07a3b35f4c69a351c5540ab663f8'),*
***AIMessage**(content='The current weather in Athens, GR is:\n- Detailed status: few clouds\n- Wind speed: 4.47 m/s, direction: 23°\n- Humidity: 58%\n- Temperature:\n - Current: 15.15°C\n - High: 15.74°C\n - Low: 14.1°C\n - Feels like: 14.24°C\n- Rain: None\n- Heat index: None\n- Cloud cover: 20%', additional_kwargs={}, response_metadata={'token_usage': {'completion_tokens': 125, 'prompt_tokens': 733, 'total_tokens': 858}, 'model_name': 'ibm/granite-3-8b-instruct', 'system_fingerprint': '', 'finish_reason': 'stop'}, id='chat-6719a5ca266a439bb10ed410db25c5ef', usage_metadata={'input_tokens': 733, 'output_tokens': 125, 'total_tokens': 858})]*

The model was able to discern the appropriate tool to call, run the tool with the extracted arguments as well as synthesize the tool output. Now, let's ask the LLM for some suitcases under USD100 for your upcoming trip. Note, the tool is designed to search for prices in cents. Hence, we ask for suitcases under 10,000 cents in this scenario, equivalent to USD 100.
user_query = "Find some suitcases for less than 10000" response = agent_executor.invoke({"messages": user_query}) response["messages"]
Powered by Granite

Output:

*[**HumanMessage**(content='Find some suitcases for less than 10000', additional_kwargs={}, response_metadata={}, id='8b207035-150a-4390-aff3-8b09ef85a592'),*
***AIMessage**(content='', additional_kwargs={'tool_calls': [{'id': 'chatcmpl-tool-b011e718b18e41dcbcae2f7786af263d', 'type': 'function', 'function': {'name': 'ionic_commerce_shopping_tool', 'arguments': '{"__arg1": "suitcases, 10, 0, 10000"}'}}]}, response_metadata={'token_usage': {'completion_tokens': 41, 'prompt_tokens': 582, 'total_tokens': 623}, 'model_name': 'ibm/granite-3-8b-instruct', 'system_fingerprint': '', 'finish_reason': 'tool_calls'}, id='chat-e38c8568d1754636a6a92082561180bd', tool_calls=[{'name': 'ionic_commerce_shopping_tool', 'args': {'__arg1': 'suitcases, 10, 0, 10000'}, 'id': 'chatcmpl-tool-b011e718b18e41dcbcae2f7786af263d', 'type': 'tool_call'}], usage_metadata={'input_tokens': 582, 'output_tokens': 41, 'total_tokens': 623}),*
***ToolMessage**(content='[{"products": [{"links": [{"text": "Details", "type": "pdp", "url": "https://go.ionic.click/Ch4CKd"}], "merchant_name": "Walmart", "merchant_product_id": "811277349", "name": "Zimtown Hardside Lightweight Spinner Orange 3 Piece Luggage Set with TSA Lock", "price": "$69.99", "status": "available", "thumbnail": "https://i5.walmartimages.com/asr/b809a274-ccc7-4ca4-b4f1-e848b4412fe6.314144bcd13e5467a33cb99e8dd5237c.jpeg?odnHeight=100&odnWidth=100&odnBg=ffffff", "brand_name": "Zimtown", "upc": "273109526768"}, {"links": [{"text": "Details", "type": "pdp", "url": "https://www.amazon.com/dp/B071HHX6VF?tag=ioniccommer00-20&linkCode=osi&th=1&psc=1"}], "merchant_name": "Amazon", "merchant_product_id": "B071HHX6VF", "name": "Amazon Basics Expandable Hardside Luggage, Suitcase with Wheels, 30-Inch Spinner with Four Spinner Wheels and Scratch-Resistant Surface, Black", "price": "$74.49", "status": "available", "thumbnail": "https://m.media-amazon.com/images/I/41jJcuMYSdL._SL160_.jpg", "brand_name": "Amazon Basics", "upc": "841710177190"}, ...[abbreviated],*

```
AIMessage(content='Here are some suitcases that are less than 10000:\n\n1. [Zimtown Hardside
Lightweight Spinner Orange 3 Piece Luggage Set with TSA Lock](https://go.ionic.click/Ch4CKd) -
$69.99\n2. [Amazon Basics Expandable Hardside Luggage, Suitcase with Wheels, 30-Inch Spinner]
(https://www.amazon.com/dp/B071HHX6VF) - $74.49\n3. [SwissGear Sion Softside Expandable
Luggage, Blue, Carry-On 21-Inch](https://www.amazon.com/dp/B01MFBVKDF) - $80.73\n4.
[Travelers Club Midtown Hardside Luggage Travel, Rose Gold, 4-Piece Set]
(https://www.amazon.com/dp/B07RS4PK3J) - $95.19\n5. [American Tourister Stratum 2.0 Expandable
Hardside Luggage with Spinner Wheels, 28" SPINNER, Slate Blue]
(https://www.amazon.com/dp/B0B2X1BDFH) - $89.97\n6. [Wrangler Smart Luggage Set with Cup
Holder and USB Port, Navy Blue, 20-Inch Carry-On](https://www.amazon.com/dp/B07SLG6WZ2) -
$39.99\n7. [Wrangler Hardside Carry-On Spinner Luggage, Lilac, 20-Inch]
(https://www.amazon.com/dp/B0C7YWMBGP) - $40.00\n8. [Protege 20 inch Hard Side Carry-On
Spinner Luggage, Black Matte Finish (Walmart.com Exclusive)](https://go.ionic.click/qJqBRA) -
$29.87\n9. [Wrangler Wesley Rolling Duffel Bag, Tannin, Large 30-Inch]
(https://www.amazon.com/dp/B07XKWMLJ5) - $44.00\n10. [U.S. Traveler Boren Polycarbonate
Hardside Rugged Travel Suitcase Luggage with 8 Spinner Wheels, Aluminum Handle, Lavender,
Checked-Large 30-Inch](https://www.amazon.com/dp/B085B4D852) - $79.99', additional_kwargs={},
response_metadata={'token_usage': {'completion_tokens': 612, 'prompt_tokens': 2794, 'total_tokens':
3406}, 'model_name': 'ibm/granite-3-8b-instruct', 'system_fingerprint': '', 'finish_reason': 'stop'},
id='chat-d08201ff6ef84c428e7ae44372396926', usage_metadata={'input_tokens': 2794,
'output_tokens': 612, 'total_tokens': 3406})]
```

As seen by the LLM response, the model correctly used the shopping tool to return several suitcases for purchase online under USD 100.

Summary

In this tutorial, you used prebuilt LangChain tools to create a ReAct agent in Python with watsonx using the granite-3-8b-instruct model. You used the youtube_search , weather_search and ionic_search tools. The tutorial showed how to implement traditional tool calling as well as an agentic approach that runs the tools. The sample output is important as it shows the steps the agent took in creating its own agent workflow by using the functions available. The tools granted to the agent were vital for answering user queries.