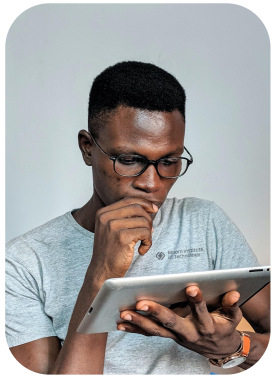
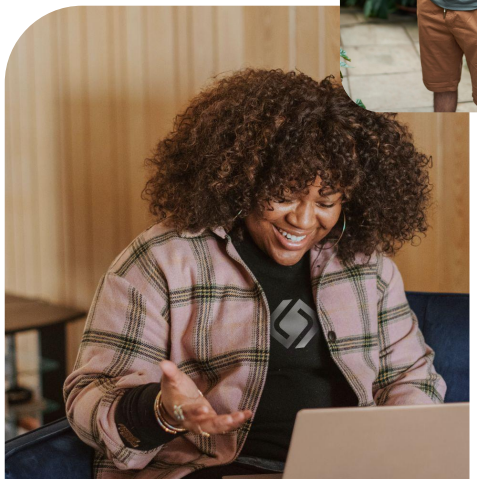
Bloom Institute of Technology

Tool Calling

Gain insights into how tool calling transforms interactions with LLMs, enabling structured outputs like dictionaries for enhanced API interaction, code execution, and JSON object creation.





Core Competencies

The student must demonstrate...

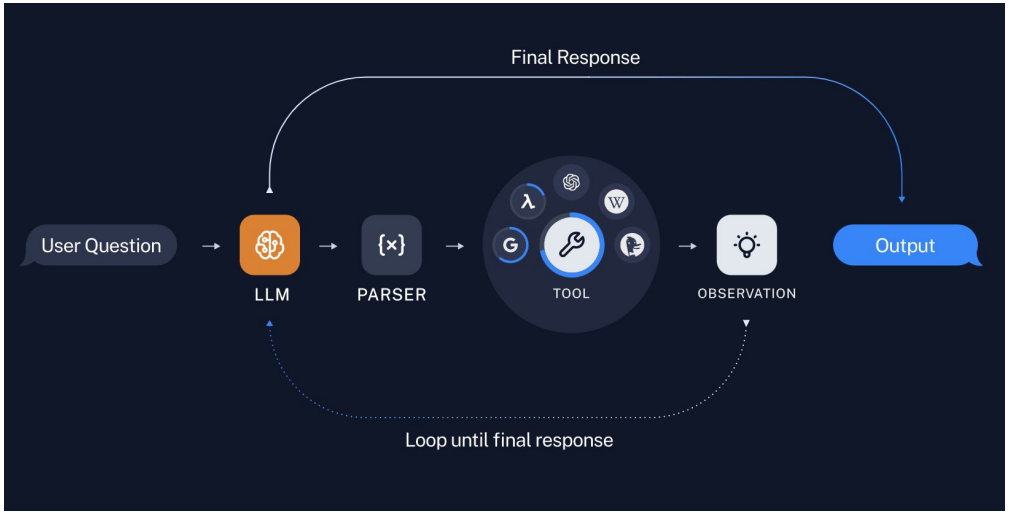
1. Understanding of function/tool calling (20 minutes)
2. Ability to implement tool calling using LangChain (20 minutes)
3. Familiarity with viewing tool calling in LangSmith (5 minutes)

What is tool calling?

Tool calling with LLMs refers to the process where a model generates responses that conform to a predefined user schema, suggesting actions rather than taking them directly.

- The terms `tool calling` and `function calling` are generally used interchangeably, but `function calling` sometimes refers specifically to the invocation of a single function.
- A tool call is defined by its name, a dictionary of arguments, and an optional identifier, where the dictionary is formatted as `{argument_name: argument_value}`.
- LLMs can be configured to recommend multiple tool or function calls in a single interaction. This setup allows the models to act as decision-making engines, determining the most appropriate actions and their sequence.

How agents use tools to answer queries:



Setup Function Call With OpenAI's API

1. **Initialize Client**: Import modules and create an OpenAI client.
2. **Define Weather Function**: Setup a function to return weather data for specific cities.
3. **Setup Conversation**: Prepare a query and specify callable functions.
4. **Generate Response**: Send the query, check for function calls.
5. **Execute Function**: Call the function with arguments, record the response.
6. **Update Conversation**: Append function outputs and send for further processing.
7. **Output Response**: Print the complete conversation including model and function responses.

Try coding the function call first and check your answer here.

Tool Calling Inside LangChain

Only certain language models, like GPT-3.5, support tool calling, allowing the integration of external functionalities. In Langchain, "tools" are external functions integrated using the `@tool` annotation, enabling models to interact with them as part of their capabilities.

1. **Define Tool with Annotation**: Use the `@tool` annotation to define fetch_weather so that Langchain recognizes it as a function that can be converted into OpenAI's function calling structure.

2. **Docstring Explanation**: Include a docstring in fetch_weather to explain that it retrieves the current weather condition for a specific city.

3. **Initialize Language Model**: Create an instance of ChatOpenAI with model `gpt-3.5-turbo-0125`, suitable for integrating with your tool.

4. **Bind Tool to LLM**: Bind the `fetch_weather` function to the language model, allowing it to make function calls.

Tools On LangSmith

LangSmith not only tracks the output arguments from the large language model (LLM), but it also **records the results of functions called by various tools**. This way, you get comprehensive insights into both the LLM's outputs and the tool functionalities.

Execute tools, trace the tools on LangSmith, and check your understanding:

1. What are the two rows in LangSmith related to tool calls?
2. Which of these rows represents the actual tool execution?
3. How would you add this tool call information to an annotation queue and dataset? Consider the simplicity of our example; how might this differ in more complex scenarios?
4. Analyze the function executed by the tool. Did the tool utilize a LLM?
5. Discuss why scoring the tool's function is irrelevant in this scenario, where we're simply calling an API. What might make scoring relevant in other contexts?

Hands-on Homework.

Design and implement functions that can be supplied to a LLM to perform complex mathematical operations and calculations.

Here are some to consider:

1. **Quadratic Equation Solver**: Create a function to solve quadratic equations given coefficients a, b, and c. The function should handle real and complex roots and return them in a user-friendly format.
2. **Fibonacci Sequence Generator**: Write a function to generate the Fibonacci sequence up to a given number n using either iterative or recursive methods. Optimize for large values of n.
3. **Currency Converter**: Create a function that converts an amount from one currency to another using real-time exchange rates. This function may require calling an external API to fetch the latest rates.