FUNCTIONS

Topics

- Why do we need functions?
- How LLMs Process Functions
- How LLM’s process functions:
  Flow of Execution when functions are invoked
- Implementing functions in OpenAI
- Implementing functions in Claude
- File Search function with OpenAI

Why Functions (1)

- Improved RAG
- Access to external data / information
- Parsing results

Challenge with RAGs

- How can the RAG bot (project 2) mess up - group brainstorming, searching,…

Challenge with RAGs and Context

Two questions to the bot

1) prompt: “which 3 courses are best for scripting” → RAG results:
...Syllabus might be helpful: ./dataForLab4/IST 652 Syllabus.pdf
...Syllabus might be helpful: ./dataForLab4/IST736-Text-Mining-Syllabus.pdf
...Syllabus might be helpful: ./dataForLab4/IST 644 Syllabus-July24.pdf

2) prompt: “tell me more about the first course” → RAG results:
...Syllabus might be helpful: ./dataForLab4/IST 644 Syllabus-July24.pdf
...Syllabus might be helpful: ./dataForLab4/IST 782 Syllabus.pdf
...Syllabus might be helpful: ./dataForLab4/IST691 Deep Learning in Practice Syllabus.pdf

→ **Why did the RAG not work?**

Challenge with RAGs and Context

What to use for the ‘search’ in a vector database?

RAG typically uses the current prompt to do the information search

- but might need more context to search appropriately.
- Have the bot ask for more information

→ “tell me more about it” or “Expand section 3” – RAG won’t work

Why Functions (2) : Parsing Results

This Code

```
import openai

recipe = 'Fish and chips'
query = f"""What is the recipe for {recipe}?
Return the ingredients list and steps separately."""

response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo-0613",
    messages=[{"role": "user", "content": query}])

response_message = response["choices"][0]["message"]
print(response_message['content'])
```

https://towardsdatascience.com/llm-output-parsing-function-calling-vs-langchain-63b80545b3a7

Why Functions(2): Parsing Results

Generates:

Parsing answers can be difficult

Also, output can vary each time…

```
Ingredients for fish and chips:
- 1 pound white fish fillets (such as cod or haddock)
- 1 cup all-purpose flour
- 1 teaspoon baking powder
- 1 teaspoon salt
- 1/2 teaspoon black pepper
- 1 cup cold beer
- Vegetable oil, for frying
- 4 large russet potatoes
- Salt, to taste

Steps to make fish and chips:

1. Preheat the oven to 200°C (400°F).
2. Peel the potatoes and cut them into thick, uniform strips. Rinse the potato s
3. In a large pot or deep fryer, heat vegetable oil to 175°C (350°F). Ensure the
4. In a mixing bowl, combine the flour, baking powder, salt, and black pepper. W
5. Take the dried potato strips and fry them in batches for about 5-6 minutes or
6. Dip each fish fillet into the prepared batter, ensuring it is well coated. Le
7. Fry the fish fillets for 4-5 minutes on each side or until they turn golden b
8. Season the fish and chips with salt while they are still hot.
9. Serve the fish and chips hot with tartar sauce, malt vinegar, or ketchup as d

Enjoy your homemade fish and chips!
```

https://towardsdatascience.com/llm-output-parsing-function-calling-vs-langchain-63b80545b3a7

Why Functions (2): Parsing Results

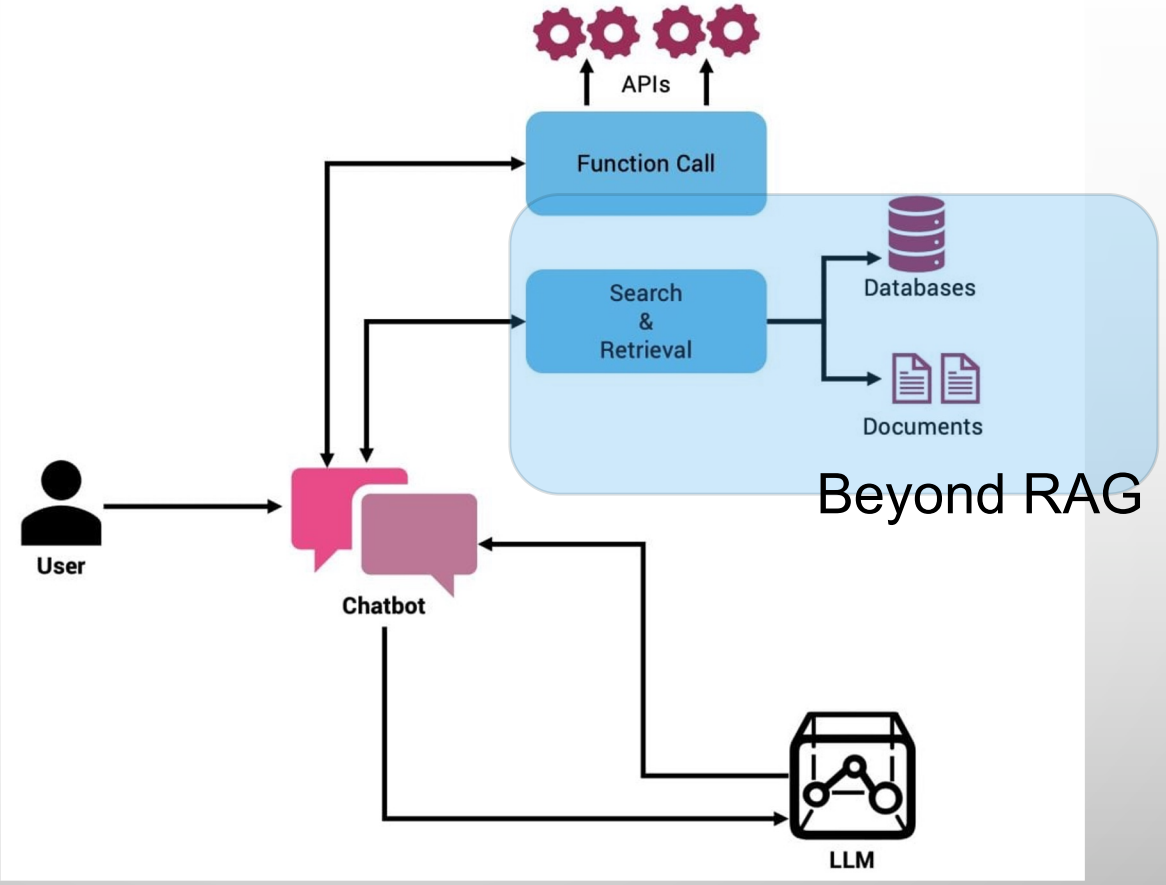
Functions allow structured answers (via the parameters

Note some LLMs (ex. OpenAI) allow you to force the LLM to output JSON (which is a structured answer).

Why Functions (3): Access to External Data

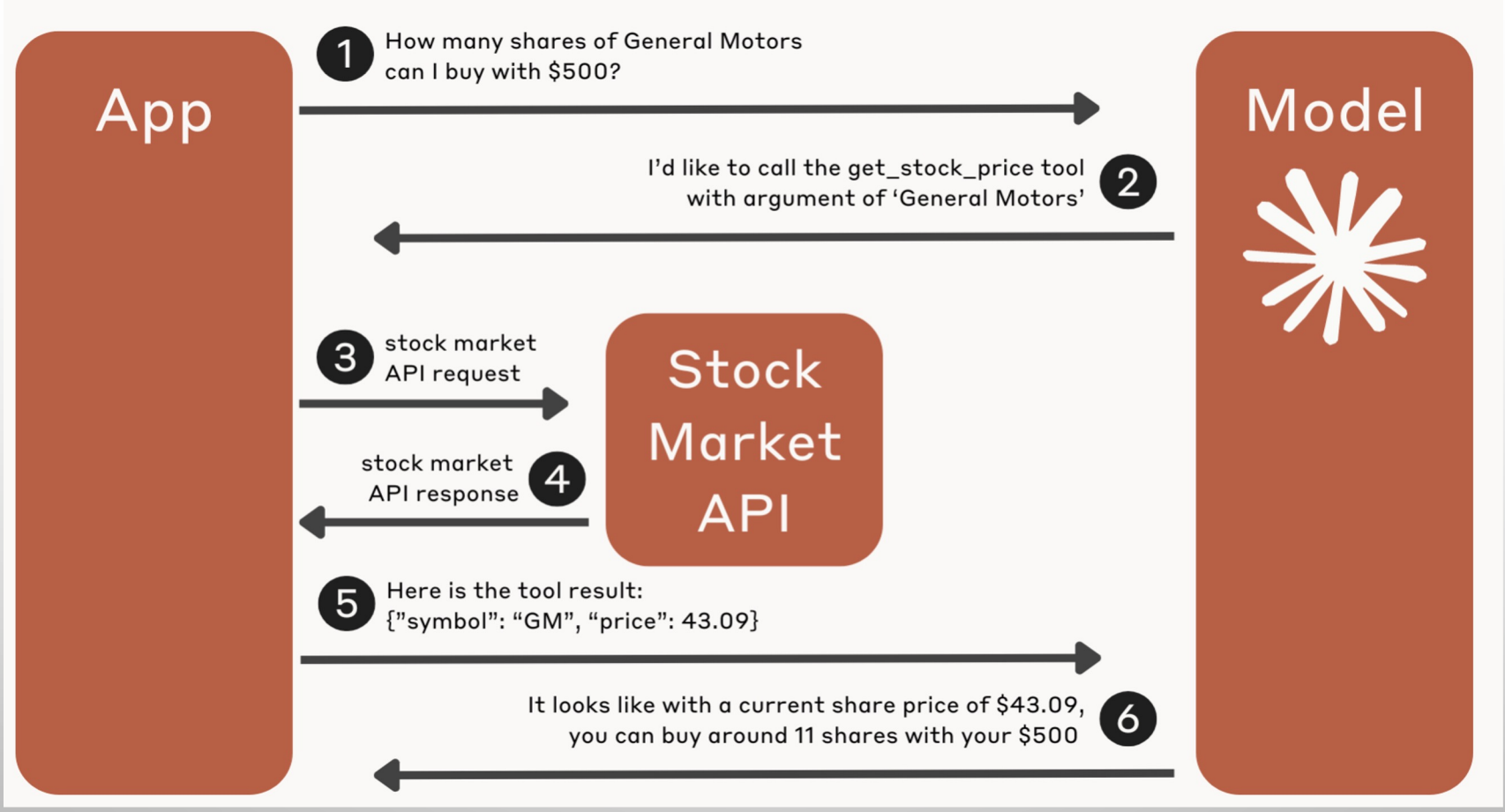
Examples:

- Today's weather
- Price of a flight



https://thenewstack.io/a-comprehensive-guide-to-function-calling-in-llms/

Functions: Flow of Execution



https://github.com/anthropics/courses/blob/master/ToolUse/01_tool_use_overview.ipynb

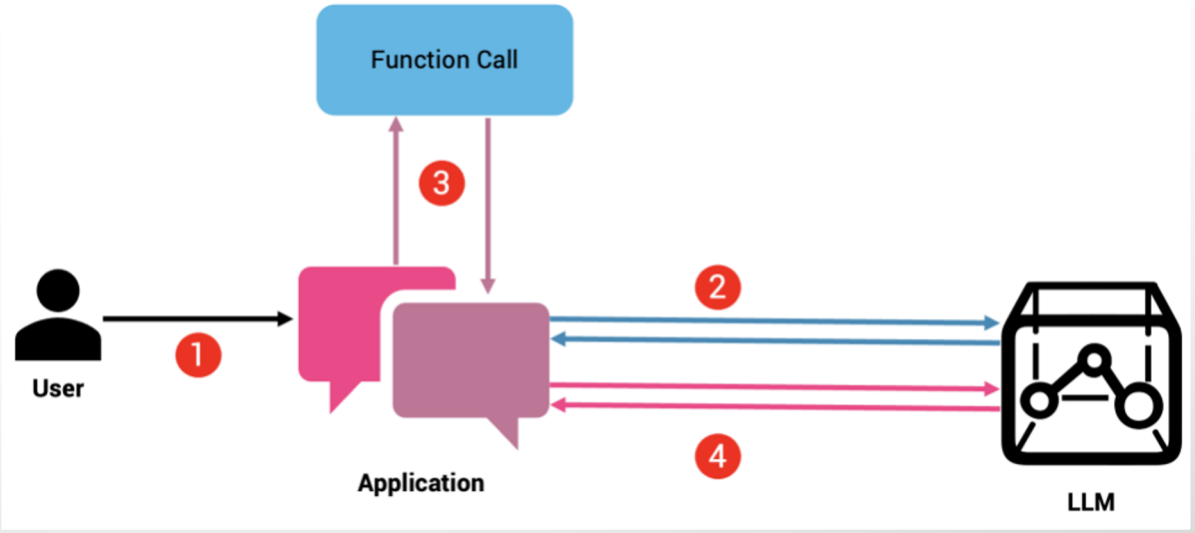
Functions: Flow of Execution

1) User question: “what is today’s weather”

2) Question sent to LLM
(along with possible tools)

2a) LLM wants results
from a function

3) App calls function
(ex. get_weather)

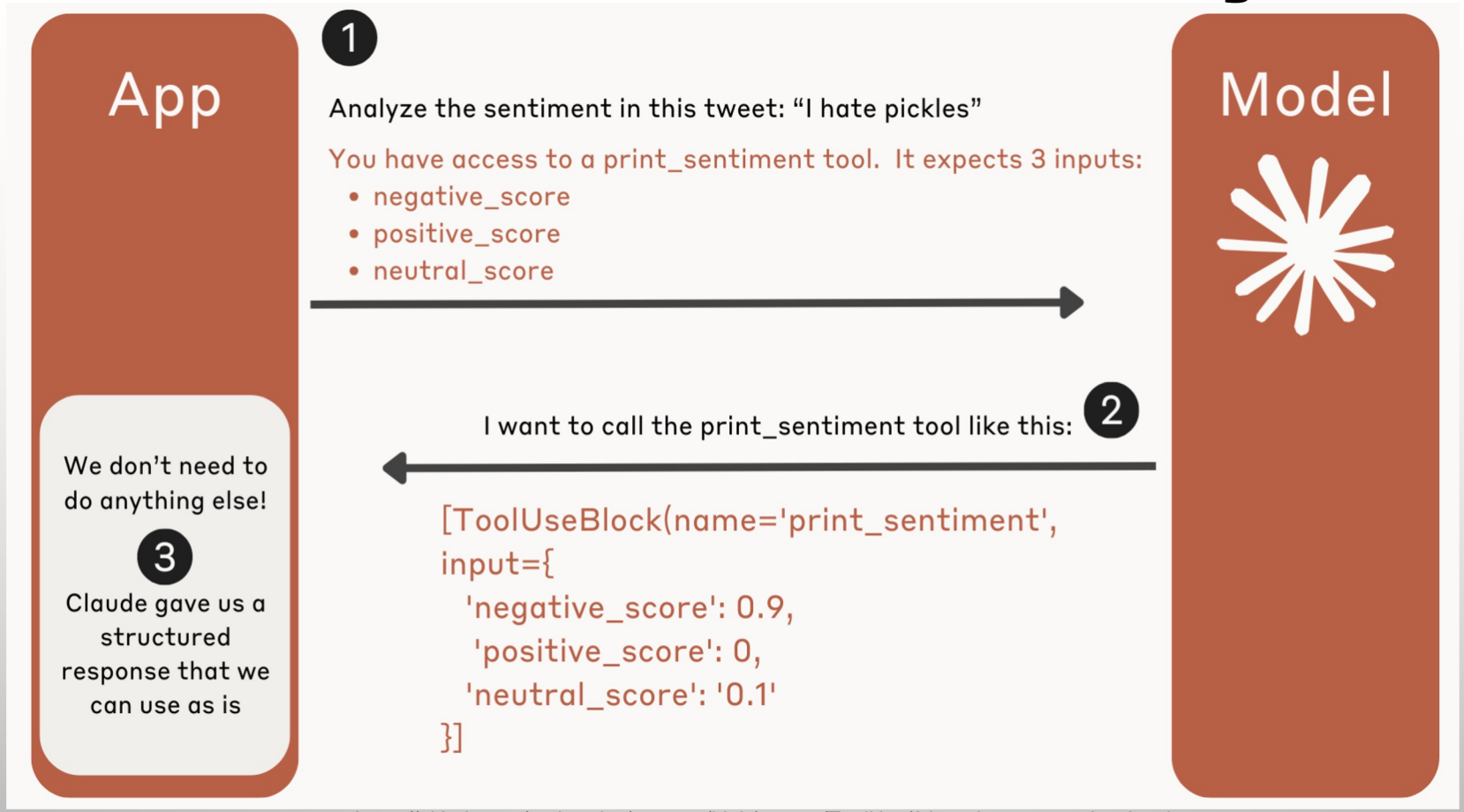


https://thenewstack.io/a-comprehensive-guide-to-function-calling-in-llms/

4) Results given to LLM

4a) Answer to initial question provided back to application

Functions: Flow of Execution for Formatting



https://github.com/anthropics/courses/blob/master/ToolUse/01_tool_use_overview.ipynb

Cost of Functions

Functions are injected into the system message (prompt) in a syntax the model has been trained to understand.

- **Functions count against the model's context limit and are billed as input tokens**

- **So, cost increases, time to process increases…**

If running into context limits, try to limit the number of functions or the length of documentation you provide for function parameters.

Defining the function

```
{
    "type": "function",
    "function": {
        "name": "get_current_weather",
        "description": "Get the current weather",
        "parameters": {
            "type": "object",
            "properties": {
                "location": {
                    "type": "string",
                    "description": "The city and state, e.g. San Francisco, CA",
                },
                "format": {
                    "type": "string",
                    "enum": ["celsius", "fahrenheit"],
                    "description": "The temperature unit to use. Infer this from the users location.
                },
            },
            "required": ["location", "format"],
        },
    }
},
```

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Defining the function

```
{
    "type": "function",
    "function": {
        "name": "get_current_weather",
        "description": "Get the current weather",
        "parameters": {
            "type": "object",
            "properties": {
                "location": {
                    "type": "string",
                    "description": "The city and state, e.g. San Francisco, CA",
                },
                "format": {
                    "type": "string",
                    "enum": ["celsius", "fahrenheit"],
                    "description": "The temperature unit to use. Infer this from the users location.
                },
            },
            "required": ["location", "format"],
        },
    }
},
```

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Defining the function

```
{
    "type": "function",
    "function": {
        "name": "get_current_weather",
        "description": "Get the current weather",
        "parameters": {
            "type": "object",
            "properties": {
                "location": {
                    "type": "string",
                    "description": "The city and state, e.g. San Francisco, CA",
                },
                "format": {
                    "type": "string",
                    "enum": ["celsius", "fahrenheit"],
                    "description": "The temperature unit to use. Infer this from the users location.
                },
            },
            "required": ["location", "format"],
        },
    }
},
```

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Defining the function

```
{
    "type": "function",
    "function": {
        "name": "get_current_weather",
        "description": "Get the current weather",
        "parameters": {
            "type": "object",
            "properties": {
                "location": {
                    "type": "string",
                    "description": "The city and state, e.g. San Francisco, CA",
                },
                "format": {
                    "type": "string",
                    "enum": ["celsius", "fahrenheit"],
                    "description": "The temperature unit to use. Infer this from the users location.
                },
            },
            "required": ["location", "format"],
        },
    }
},
```

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Defining the function

```
{
    "type": "function",
    "function": {
        "name": "get_current_weather",
        "description": "Get the current weather",
        "parameters": {
            "type": "object",
            "properties": {
                "location": {
                    "type": "string",
                    "description": "The city and state, e.g. San Francisco, CA",
                },
                "format": {
                    "type": "string",
                    "enum": ["celsius", "fahrenheit"],
                    "description": "The temperature unit to use. Infer this from the users location.
                },
            },
            "required": ["location", "format"],
        },
    }
},
```

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

All Tools

```
tools = [
    {
        "type": "function",
        "function": {
            "name": "get_current_weather",
            "description": "Get the current weather",
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {
                        "type": "string",
                        "description": "The city and state, e.g. San Francisco, CA",
                    },
                    "format": {
                        "type": "string",
                        "enum": ["celsius", "fahrenheit"],
                        "description": "The temperature unit to use. Infer this from the users location.
                    },
                },
                "required": ["location", "format"],
            },
        }
    },
    {
        "type": "function",
        "function": {
            "name": "get_n_day_weather_forecast",
            "description": "Get an N-day weather forecast",
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {
                        "type": "string",
                        "description": "The city and state, e.g. San Francisco, CA",
                    },
                    "format": {
                        "type": "string",
                        "enum": ["celsius", "fahrenheit"],
                        "description": "The temperature unit to use. Infer this from the users location.
                    },
                    "num_days": {
                        "type": "integer",
                        "description": "The number of days to forecast",
                    }
                },
                "required": ["location", "format", "num_days"]
            },
        }
    },
]
```

https://cookbook.openai.com/examp

Functions (tools) in OpenAI

A Utility: do a chat completion

```python
def chat_completion_request(messages, tools=None, tool_choice=None, model=GPT_MODEL):
    try:
        response = client.chat.completions.create(
            model=model,
            messages=messages,
            tools=tools,
            tool_choice=tool_choice,
        )
        return response
    except Exception as e:
        print("Unable to generate ChatCompletion response")
        print(f"Exception: {e}")
        return e
```

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Using the Function

LLM asks the user for clarification

```
messages = []
messages.append({"role": "system", "content": "Don't make assumptions about what values to plug into fun
messages.append({"role": "user", "content": "What's the weather like today"})
chat_response = chat_completion_request(
    messages, tools=tools
)
assistant_message = chat_response.choices[0].message
messages.append(assistant_message)
assistant_message
```

ChatCompletionMessage(content="**I need to know your location to provide you with the current weather. Could you please specify the city and state (or country) you're in?**", role='assistant', function_call=None, tool_calls=None)

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Using the Function

LLM “invokes” the function, by returning a function request to the app

```
messages.append({"role": "user", "content": "I'm in Glasgow, Scotland."})
chat_response = chat_completion_request(
    messages, tools=tools
)
assistant_message = chat_response.choices[0].message
messages.append(assistant_message)
assistant_message
```

ChatCompletionMessage(content=None, role='assistant', function_call=None, tool_calls=[ChatCompletionMessageToolCall(id='call_Dn2RJJSxzDm49vlVTehseJ0k', **function=Function(arguments='{"location":"Glasgow, Scotland","format":"celsius"}', name='get_current_weather'), type='function')])**

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Using the Function

LLM asks the user for clarification (for a different function)

```
messages = []
messages.append({"role": "system", "content": "Don't make assumptions about what values to plug into fun
messages.append({"role": "user", "content": "what is the weather going to be like in Glasgow, Scotland o
chat_response = chat_completion_request(
    messages, tools=tools
)
assistant_message = chat_response.choices[0].message
messages.append(assistant_message)
assistant_message
```

ChatCompletionMessage(content='**Please specify the number of days (x) for which you want the weather forecast for Glasgow, Scotland**.', role='assistant', function_call=None, tool_calls=None)

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Using the Function

LLM “invokes” the function, by returning a function request to the app
(for a different function)

```
messages.append({"role": "user", "content": "5 days"})
chat_response = chat_completion_request(
    messages, tools=tools
)
chat_response.choices[0]
```

Choice(finish_reason='tool_calls', index=0, logprobs=None,
message=ChatCompletionMessage(content=None, role='assistant', function_call=None,
tool_calls=[ChatCompletionMessageToolCall(id='call_Yg5ydH9lHhLjjYQyXbNvh004',
function=Function(arguments='{"location":"Glasgow,Scotland","format":"celsius","num_days":5}', name='get_n_day_weather_forecast'), type='function')]))

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Accessing the function request

A simple chat.completions request – using a set of tools

```
# Step #1: Prompt with content that may result in function call. In this case the model can identify the
messages = [{
    "role":"user",
    "content": "What is the name of the album with the most tracks?"
}]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages,
    tools= tools,
    tool_choice="auto"
)

# Append the message to messages list
response_message = response.choices[0].message
messages.append(response_message)

print(response_message)
```

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Accessing the function request

Determine if a tool has been requested

```python
# Step 2: determine if the response from the model includes a tool call.
tool_calls = response_message.tool_calls
if tool_calls:
    # If true the model will return the name of the tool / function to call and the argument(s)
    tool_call_id = tool_calls[0].id
    tool_function_name = tool_calls[0].function.name
    tool_query_string = eval(tool_calls[0].function.arguments)['query']
```

...

```python
else:
    # Model did not identify a function to call, result can be returned to the user
    print(response_message.content)
```

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Accessing the function request

Do the function call if needed

```
# Step 3: Call the function and retrieve results. Append the results to the messages list.
if tool_function_name == 'ask_database':
    results = ask_database(conn, tool_query_string)

    messages.append({
        "role":"tool",
        "tool_call_id":tool_call_id,
        "name": tool_function_name,
        "content":results
    })

    # Step 4: Invoke the chat completions API with the function response appended to the messages li
    # Note that messages with role 'tool' must be a response to a preceding message with 'tool_calls
    model_response_with_function_call = client.chat.completions.create(
        model="gpt-4o",
        messages=messages,
    )  # get a new response from the model where it can see the function response
    print(model_response_with_function_call.choices[0].message.content)
else:
    print(f"Error: function {tool_function_name} does not exist")
```

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

Accessing the function request

Do the function call if needed

```
# Step 3: Call the function and retrieve results. Append the results to the messages list.
if tool_function_name == 'ask_database':
    results = ask_database(conn, tool_query_string)

    messages.append({
        "role":"tool",
        "tool_call_id":tool_call_id,
        "name": tool_function_name,
        "content":results
    })

    # Step 4: Invoke the chat completions API with the function response appended to the messages li
    # Note that messages with role 'tool' must be a response to a preceding message with 'tool_calls
    model_response_with_function_call = client.chat.completions.create(
        model="gpt-4o",
        messages=messages,
    )  # get a new response from the model where it can see the function response
    print(model_response_with_function_call.choices[0].message.content)
else:
    print(f"Error: function {tool_function_name} does not exist")
```

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

How about Claude?

How about Claude?

Define a simple ‘calculator’ function

```
def calculator(operation, operand1, operand2):
    if operation == "add":
        return operand1 + operand2
    elif operation == "subtract":
        return operand1 - operand2
    elif operation == "multiply":
        return operand1 * operand2
    elif operation == "divide":
        if operand2 == 0:
            raise ValueError("Cannot divide by zero.")
        return operand1 / operand2
    else:
        raise ValueError(f"Unsupported operation: {operation}")
```

https://github.com/anthropics/courses/blob/master/ToolUse/01_tool_use_overview.ipynb

How about Claude?

Define a simple ‘calculator’ tool

```
calculator_tool = {
    "name": "calculator",
    "description": "A simple calculator that performs basic arithmetic operations.",
    "input_schema": {
        "type": "object",
        "properties": {
            "operation": {
                "type": "string",
                "enum": ["add", "subtract", "multiply", "divide"],
                "description": "The arithmetic operation to perform.",
            },
            "operand1": {"type": "number", "description": "The first operand."},
            "operand2": {"type": "number", "description": "The second operand."},
        },
        "required": ["operation", "operand1", "operand2"],
    },
}
```

https://github.com/anthropics/courses/blob/master/ToolUse/01_tool_use_overview.ipynb

How about Claude?

Now use the tool within Claude

```
def prompt_claude(prompt):
    messages = [{"role": "user", "content": prompt}]
    response = client.messages.create(
        model="claude-3-haiku-20240307",
        system="You have access to tools, but only use them when necessary. ]
        messages=messages,
        max_tokens=500,
        tools=[calculator_tool],
    )

    if response.stop_reason == "tool_use":
        tool_use = response.content[-1]
        tool_name = tool_use.name
        tool_input = tool_use.input
                                    ...
    elif response.stop_reason == "end_turn":
        print("Claude didn't want to use a tool")
        print("Claude responded with:")
        print(response.content[0].text)
```

https://github.com/anthropics/courses/blob/master/ToolUse/01_tool_use_overview.ipynb

How about Claude?

Check if we need to invoke the ‘calculator’

```
if tool_name == "calculator":
    print("Claude wants to use the calculator tool")
    operation = tool_input["operation"]
    operand1 = tool_input["operand1"]
    operand2 = tool_input["operand2"]

    try:
        result = calculator(operation, operand1, operand2)
        print("Calculation result is:", result)
    except ValueError as e:
        print(f"Error: {str(e)}")
```

https://github.com/anthropics/courses/blob/master/ToolUse/01_tool_use_overview.ipynb

How about Claude?

```
prompt_claude("I had 23 chickens but 2 flew away.  How many are left?")
```

```
Claude want to use the calculator tool
Calculation result is:  21
```

```
prompt_claude("What is 201 times 2")
```

```
Claude want to use the calculator tool
Calculation result is:  402
```

```
prompt_claude("Write me a haiku about the ocean")
```

```
Claude didn't want to use a tool
Claude responded with:
Here is a haiku about the ocean:

Vast blue expanse shines,
Waves crash upon sandy shores,
Ocean's soothing song.
```

https://github.com/anthropics/courses/blob/master/ToolUse/01_tool_use_overview.ipynb

File search function with OpenAI

What is OpenAI file search?

- file_search is a document or file retrieval tool designed by openAI
- Using this tool:
  - Documents are automatically parsed, chunked, converted to embedding
  - Documents are stored by OpenAI (in a vector storage created by OpenAI)
  - Default chunk size is 800 tokens and chunk overlap is 400 tokens
- To retrieve the documents based on user query:
  - OpenAI utilizes both vector & keyword search to retrieve documents.
- Users are provided 1GB of free vector storage space (After 1GB, $0.1/GB/per day

https://platform.openai.com/docs/assistants/tools/file-search

Comparing the options

OpenAI file_search	Embedding based search
Chunk size and chunk overlap is fixed	Provides complete control over chunk size & chunk overlap size
Does not require any upfront cost, time, or energy (to setup in your application).	Requires you to set up vector storage, (define embeddings, store them, …)
Allows you to set the expiration date of the uploaded file	No expiration date of the uploaded files (i.e., needs to be handled by the developer)

File search function with OpenAI

All HW/Labs can NOT use OpenAi File Search.

You can use it in your project (if you want)

Vector storage Example

For vector storage, we first need to create a OpenAI assistant with file search enabled.

Below example assistant is a Financial analyst assistant

```
client = OpenAI(api_key="OPENAI_API_KEY")

assistant = client.beta.assistants.create(
  name="Financial Analyst Assistant",
  instructions="You are an expert financial analyst.\
  Use you knowledge base to answer questions about audited financial statements.",
  model="gpt-4o-mini",
  tools=[{"type": "file_search"}],
)
```

Create a vector storage

```
vector_store = client.beta.vector_stores.create(name="Financial Statements")
```

The vector storage can be further managed from your OpenAI account.

Go to your OpenAI dashboard and you will find assistant named “Financial Analyst Assistant” under Assistant and “Financial Statements” under Storage>Vector stores

https://platform.openai.com/docs/assistants/tools/file-search

Adding files to vector storage

you can either upload the files to vector storage or attach a file to your message using thread to OpenAI API.

Upload files to vector storage

```
file_paths = ["path_to_file1", "path_to_file2"]
file_streams = [open(path, "rb") for path in file_paths]
```

Using the upload_and_poll upload the files to the vector storage. This method poll the status of storage till all the files are out of “in_progress” state when uploading.

```
file_batch = client.beta.vector_stores.file_batches.upload_and_poll(
  vector_store_id=vector_store.id, files=file_streams)
```

Attach file to your message

```
message_file = client.files.create(file=open("path_to_file", "rb"), purpose="assistants")
thread = client.beta.threads.create(messages=[{
        "role": "user",
        "content": "User query",
        # Attach the new file to the message.
        "attachments": [
          { "file_id": message_file.id, "tools": [{"type": "file_search"}] }],}])
```