Ensure that the "TheBloke/Mistral-7B-Instruct-v0.2-GGUF" model is running on LM Studio at "http://localhost:1234/v1"

In [1]:
```
#!pip install openai>=1.43.0
```

In [2]:
```
#!pip install langchain==0.2.15
```

In [3]:
```
#!pip install langchain-core==0.2.35
```

In [4]:
```
from toolformer import Toolformer
from datetime import datetime
from langchain.agents import Tool
from typing import List, Optional, Callable
from utils import text_red, text_underline, remove_non_ascii_and_newline
import smtplib
import re
import ast
```

TOOLS

Define tools that the LLM can trigger. A tool is a Python function that takes a string as an input and returns a string as output.

In [5]:
```
tools = {}
def get_tools() -> list[Tool]:
    """
    Returns a list of all available Tools.
    """
    tool_list = list()
    for tool_name in tools.keys():
        t:dict = tools[tool_name]
        tool = Tool(
            name=tool_name,
            func=t['func'],
            description=t['description']
        )
        tool_list.append(tool)
    return tool_list
```

In [6]:
```
# TOOL - Get current date and time

def get_current_date(dummy:str):
    """
    Returns the current date and time formatted as a string.

    """

    from datetime import datetime
    print("\n" + text_underline(text_red(f" > Tool 'get_current_datetime' was called with inpu

    # Get and format date and time
    now = datetime.now()
    datetime = now.strftime("%A, %B %d, %Y")

    # Return date and time to LLM as a string
    return datetime

# Add this tool to the tool list
tools["Todays Date"] = {
    "func": get_current_date, # This is the function name
    "description": "Returns today's date."
}
```

In [7]:
```
# TOOL - Get the E-Mail adress of a person based on their name

def find_email_address(person:str) -> str:
    """
    Builds a DLR E-Mail adress from the persons firstname and last name.
    Firstname and lastname are passed as a string in the format of a tuple, as such: "(first,
    This is because in our case, tools only take strings as an input.
    """

    # Print a log for ourselves
    print("\n" + text_underline(text_red(f" > Tool 'find_email_address' was called with input

    # Preprocess the input string so that it can be parsed as a tuple
    person = person.strip()
    firstname, lastname = ast.literal_eval(person)

    # Build E-Mail adress
    email = f"{firstname.lower()}.{lastname.lower()}@dlr.de"

    # Return E-Mail adress to the LLM as a string
    return f"The e-mail adress of {firstname} {lastname} is: {email}"

# Add this tool to the tool list
tools["Find Email"] = {
    "func": find_email_address, # This is the function name
    "description": "Returns the E-Mail adress for a person. Input: a tuple of two strings in
}
```

In [8]:
```python
# TOOL - Send an E-Mail (!) WARNING: This actually send an email using our DLR mail service!
# E-Mail Setup
SERVER = smtplib.SMTP("smtprelay.dlr.de")
FROM_MAIL = "dominik.opitz@dlr.de"
DISCLAIMER = "\n\n---\nDisclaimer: This is an automated e-mail sent by a Language Model."
ENABLE_EMAIL_SERVICE = True
def send_mail(data:str) -> str:
    """
    Sends an actual E-Mail using the DLR mail service.
    The data object is a tuple (passed as a string) which contains:
    - email adress, subject, message
    Please use with care.
    """

    # Print a log for ourselves
    print("\n" + text_underline(text_red(f" > Tool 'send_mail' was called with input '{data}'

    # Do some data cleaning as special characters could lead to issues here
    data = data.strip()
    email, subject, message = ast.literal_eval(data)
    subject = remove_non_ascii_and_newline(subject)
    message = remove_non_ascii_and_newline(message)

    # Compose and send E-Mail
    output = ""
    try:
        msg = f"Subject: {subject}\n\n{message}{DISCLAIMER}"
        if ENABLE_EMAIL_SERVICE:
            SERVER.sendmail(FROM_MAIL, email, msg)
            SERVER.quit()
        output = f"E-Mail successfully sent to {email}!"
    except Exception as e:
        output = f"Could not send E-Mail to {email} due to the following error: {e}"

    # Return response to the LLM as a string
    return output

# Add this tool to the tool list
tools["Send Mail"] = {
    "func": send_mail,
    "description": "Sends an E-Mail to another person. Input: A tuple of strings in the follo
}
```

DEFINE YOUR OWN TOOL

Below are some templates to help you design your own tool. Play around with the functions and see what happens when you execute the pipeline. Here are some unexpected situations you might encounter:

- The LLM might decide not to use your tool because it think it can respond to the user input without any tool. Observe when this happens - what could be the reason?
- The LLM might question the validity of the response of a tool. Observe when this happens - how could you counter this behaviour?
- The LLM might provide the wrong input to a tool or call the tool by the wrong name (e.g. incorrect syntax). What could be done to avoid this?

In [ ]:
```python
# TOOL - YOUR TOOL HERE.
def my_tool(dummy_string:str) -> str:
    """
    Write your own tool here. Watch out for the following:
    - Function should take a string as an input. You can describe the required syntax in the 
    - Function should return a meaningfull string, aka the output of the function. Always pro

    Here are some examples:
    - Define a dictionary for fictional characters.
    - Define a simple calculator.
    - Obtain the weather for a given location.
    """

    ### YOUR CODE HERE

### YOUR CODE HERE
tools["<toolname>"] = {
    "func": my_tool, ### YOUR FUNCTION HERE
    "description": "<simple description and input syntax>"
}
```

In [ ]:
```python
# TOOL - COMPLETE THE TOOL
def get_location_iss(dummy_string:str) -> str:
    """
    Template to get the current position of the International Space Station (ISS).
        > The API http://api.open-notify.org/iss-now.json returns a JSON that with the curren
    """

    try:
        # Step 1: Get the location
        import requests
        response = requests.get("http://api.open-notify.org/iss-now.json").json()
        ### YOUR CODE: Parse longitude and latitude from the response


        # Step 2: YOUR CODE Format the location as a string for the LLM

        # Step 3: YOUR CODE: Return a response to the LLM to let it know that it was done suc
        return location

    except Exception as e:
        ### YOUR CODE: Craft a suitable response - does the LLM have to know why the request 

#### YOUR CODE HERE
# Add this tool to the tool list. Remember to state the input format correctly
```

```
In [ ]: # TOOL - COMPLETE THE TOOL
def view_location_on_map(location:str) -> str:
    """
    Template to view a coordinate location on a map by opening a browser.
        > https://nominatim.openstreetmap.org/ui/reverse.html - lets you display coordinates o
        > https://stackoverflow.com/a/31715178 - let's you open a browser and visit a URL stra
    """

    try:
        # Step 1: Process the input
        ### YOUR CODE:
        # Parse `location` input, which is always a string, accordingly.
        # Example: You could tell the LLM to format its input as a tuple: "(latitude, longitu
        # Then use `a, b = ast.literal_eval(location)` to extract the latitude & longitude of


        # Step 2: Define the URL that will be opened in the browser to display the map
        url = f"https://nominatim.openstreetmap.org/ui/reverse.html?lat={latitude}&lon={longi

        # Step 3: YOUR CODE Open the URL in the browser
        import webbrowser


        # Step 4: YOUR CODE Return response to the LLM as a string
        return

    except Exception as e:
        ### YOUR CODE: Craft a suitable response - does the LLM have to know why the action fa
        return


#### YOUR CODE HERE
# Add this tool to the tool list. Remember to state the input format correctly
```

Executing below cell will start the pipeline. Make sure to start your LM Studio Inference Server.

You can view the LLM responses live in the LM Studio Server logs.

Recommended LLM: Mistral Instruct v2 7B Q5KM Port: 1234

```
In [9]: # YOUR INPUT - Provide your task or question here.
#user_input = "Where is the ISS currently located at? Please show me on a map."
user_input = "What's tomorrow's date?" # works fine
#user_input = "Tell me the E-Mail adress of Dominik Opitz."
#user_input = "Send an email to Dominik Opitz and greet him from our event today: the WAW ML 

tool_list = get_tools()
toolformer = Toolformer(tool_list)
toolformer.run(user_input)
```

Available Tools:

```
name='Todays Date' description="Returns today's date." func=<function get_current_date at 0x0
00001EBAD3A6A70>

name='Find Email' description='Returns the E-Mail adress for a person. Input: a tuple of two
strings in the following format: (<firstname>, <lastname>)' func=<function find_email_address
at 0x000001EBCF47DD80>

name='Send Mail' description='Sends an E-Mail to another person. Input: A tuple of strings in
the following format: (<email>, <subject>, <message>)' func=<function send_mail at 0x000001EB
AD3A5120>
```

> Entering new AgentExecutor chain...

```
C:\Users\opit_do\Documents\DLR Divers\WAWs\WAW ML Dresden 2024\llm-tutorial-waw-ml-2024\Toolf
ormer\notebook_tutorial\toolformer.py:36: LangChainDeprecationWarning: The class `LLMChain` w
as deprecated in LangChain 0.1.17 and will be removed in 1.0. Use RunnableSequence, e.g., `pr
ompt | llm` instead.
  llm_chain = LLMChain(llm=llm, prompt=prompt)
C:\Users\opit_do\Documents\DLR Divers\WAWs\WAW ML Dresden 2024\llm-tutorial-waw-ml-2024\Toolf
ormer\notebook_tutorial\toolformer.py:42: LangChainDeprecationWarning: Use new agent construc
tor methods like create_react_agent, create_json_agent, create_structured_chat_agent, etc.
  agent = LLMSingleActionAgent(
C:\Users\opit_do\Documents\DLR Divers\WAWs\WAW ML Dresden 2024\llm-tutorial-waw-ml-2024\Toolf
ormer\notebook_tutorial\toolformer.py:53: LangChainDeprecationWarning: The method `Chain.run`
was deprecated in langchain 0.1.0 and will be removed in 1.0. Use invoke instead.
  response = agent_executor.run(input)
```

```
Inferencing LLM with following prompt:
Human: Your task is to answer/execute the below question/task as best and concise as you can.
You have optional access to the following tools:

Todays Date: Returns today's date.
Find Email: Returns the E-Mail adress for a person. Input: a tuple of two strings in the foll
owing format: (<firstname>, <lastname>)
Send Mail: Sends an E-Mail to another person. Input: A tuple of strings in the following form
at: (<email>, <subject>, <message>)

Use the tools if you can not get to the answer by yourself! You can call different tools cons
ecutively and always observe the output of an action that you execute! They might not be nece
ssary to deduct the final answer.
Strictly follow the following syntax:

Question: <the input question you must answer>
Thought: <you should always think about what to do>
Action: <exclusively the tool to use, should be one of [Todays Date, Find Email, Send Mail]>
Action Input: <exclusively the input to the tool>
Observation: <the result of the tool>
... (this Thought/Action/Action Input/Observation can repeat N times. Always observe the resp
onse after an Action.)
Thought: "I now know the final answer"
Final Answer: <the final answer to the original input question>

Begin!

Question: What's tomorrow's date?
Thought:

I need to find tomorrow's date, but today's date is not given directly. I can first get today
's date using the "Todays Date" tool, and then add one day to it to get tomorrow's date.

Action: Todays Date
Action Input: None
 > Tool 'get_current_datetime' was called with input 'None''

Observation:Monday, September 16, 2024

Inferencing LLM with following prompt:
Human: Your task is to answer/execute the below question/task as best and concise as you can.
You have optional access to the following tools:

Todays Date: Returns today's date.
Find Email: Returns the E-Mail adress for a person. Input: a tuple of two strings in the foll
owing format: (<firstname>, <lastname>)
Send Mail: Sends an E-Mail to another person. Input: A tuple of strings in the following form
at: (<email>, <subject>, <message>)

Use the tools if you can not get to the answer by yourself! You can call different tools cons
ecutively and always observe the output of an action that you execute! They might not be nece
ssary to deduct the final answer.
Strictly follow the following syntax:

Question: <the input question you must answer>
Thought: <you should always think about what to do>
Action: <exclusively the tool to use, should be one of [Todays Date, Find Email, Send Mail]>
Action Input: <exclusively the input to the tool>
Observation: <the result of the tool>
... (this Thought/Action/Action Input/Observation can repeat N times. Always observe the resp
onse after an Action.)
Thought: "I now know the final answer"
Final Answer: <the final answer to the original input question>

Begin!

Question: What's tomorrow's date?
Thought:I need to find tomorrow's date, but today's date is not given directly. I can first g
et today's date using the "Todays Date" tool, and then add one day to it to get tomorrow's da
```

```
te.

Action: Todays Date
Action Input: None
Observation: Monday, September 16, 2024
Thought:

I now have today's date. I can add one day to it using the built-in functionality in most pro
gramming languages or calculators to get tomorrow's date. However, since the provided task do
es not allow the use of such functions, I will need to use the "Todays Date" tool again after
finding today's date to add one day and get tomorrow's date.

Action: Todays Date
Action Input: None
 > Tool 'get_current_datetime' was called with input 'None''

Observation:Monday, September 16, 2024

Inferencing LLM with following prompt:
Human: Your task is to answer/execute the below question/task as best and concise as you can.
You have optional access to the following tools:

Todays Date: Returns today's date.
Find Email: Returns the E-Mail adress for a person. Input: a tuple of two strings in the foll
owing format: (<firstname>, <lastname>)
Send Mail: Sends an E-Mail to another person. Input: A tuple of strings in the following form
at: (<email>, <subject>, <message>)

Use the tools if you can not get to the answer by yourself! You can call different tools cons
ecutively and always observe the output of an action that you execute! They might not be nece
ssary to deduct the final answer.
Strictly follow the following syntax:

Question: <the input question you must answer>
Thought: <you should always think about what to do>
Action: <exclusively the tool to use, should be one of [Todays Date, Find Email, Send Mail]>
Action Input: <exclusively the input to the tool>
Observation: <the result of the tool>
... (this Thought/Action/Action Input/Observation can repeat N times. Always observe the resp
onse after an Action.)
Thought: "I now know the final answer"
Final Answer: <the final answer to the original input question>

Begin!

Question: What's tomorrow's date?
Thought:I need to find tomorrow's date, but today's date is not given directly. I can first g
et today's date using the "Todays Date" tool, and then add one day to it to get tomorrow's da
te.

Action: Todays Date
Action Input: None
Observation: Monday, September 16, 2024
Thought: I now have today's date. I can add one day to it using the built-in functionality in
most programming languages or calculators to get tomorrow's date. However, since the provided
task does not allow the use of such functions, I will need to use the "Todays Date" tool agai
n after finding today's date to add one day and get tomorrow's date.

Action: Todays Date
Action Input: None
Observation: Monday, September 16, 2024
Thought:

I now have today's date twice. I can use the first observation as the start date and add one
day to it using the second observation.

Thought: I now know the final answer.
Final Answer: Tuesday, September 17, 2024 (Tomorrow's date)
```

```
> Finished chain.

Response:  Tuesday, September 17, 2024 (Tomorrow's date)
```

```
In [ ]:
```