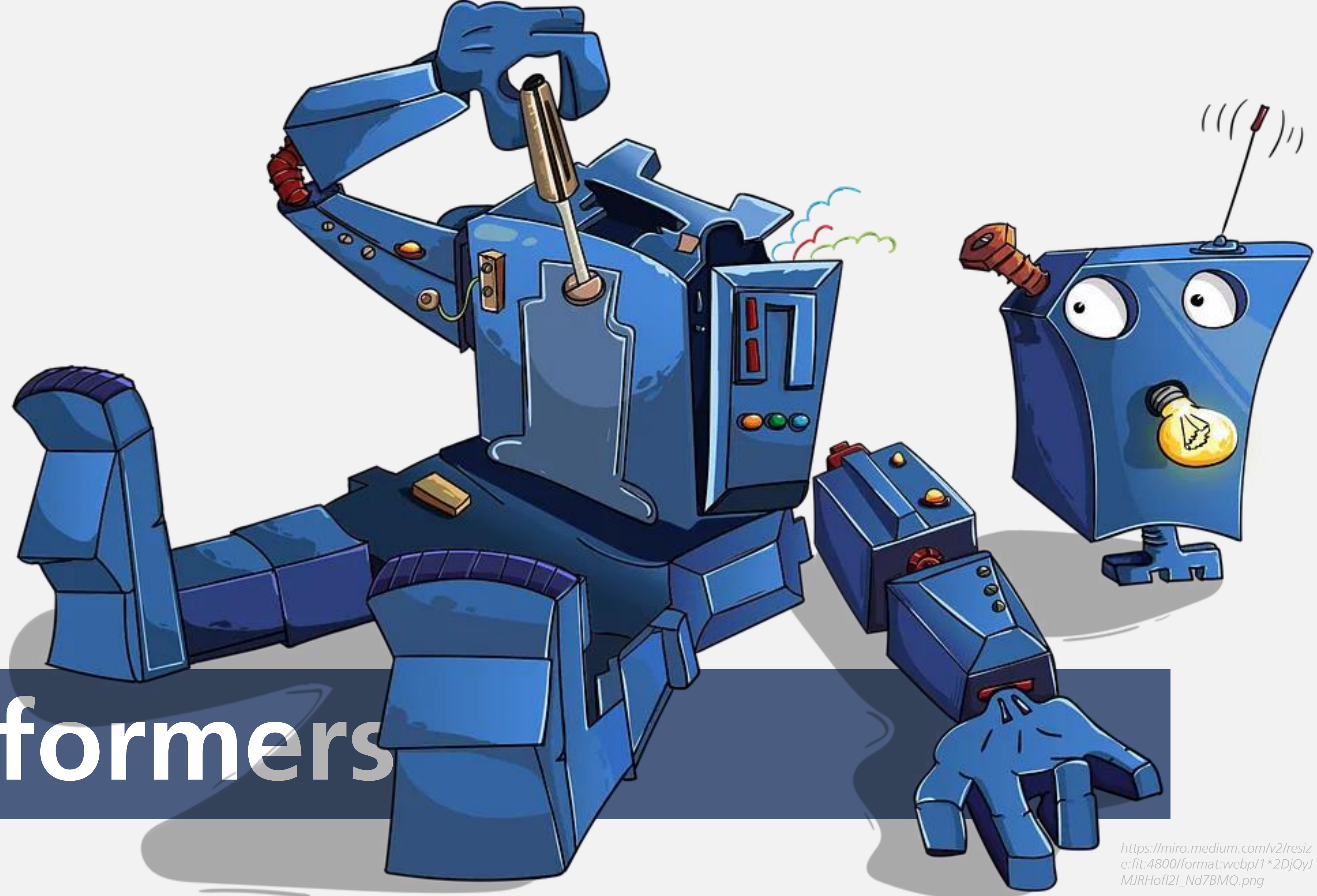
Toolformers

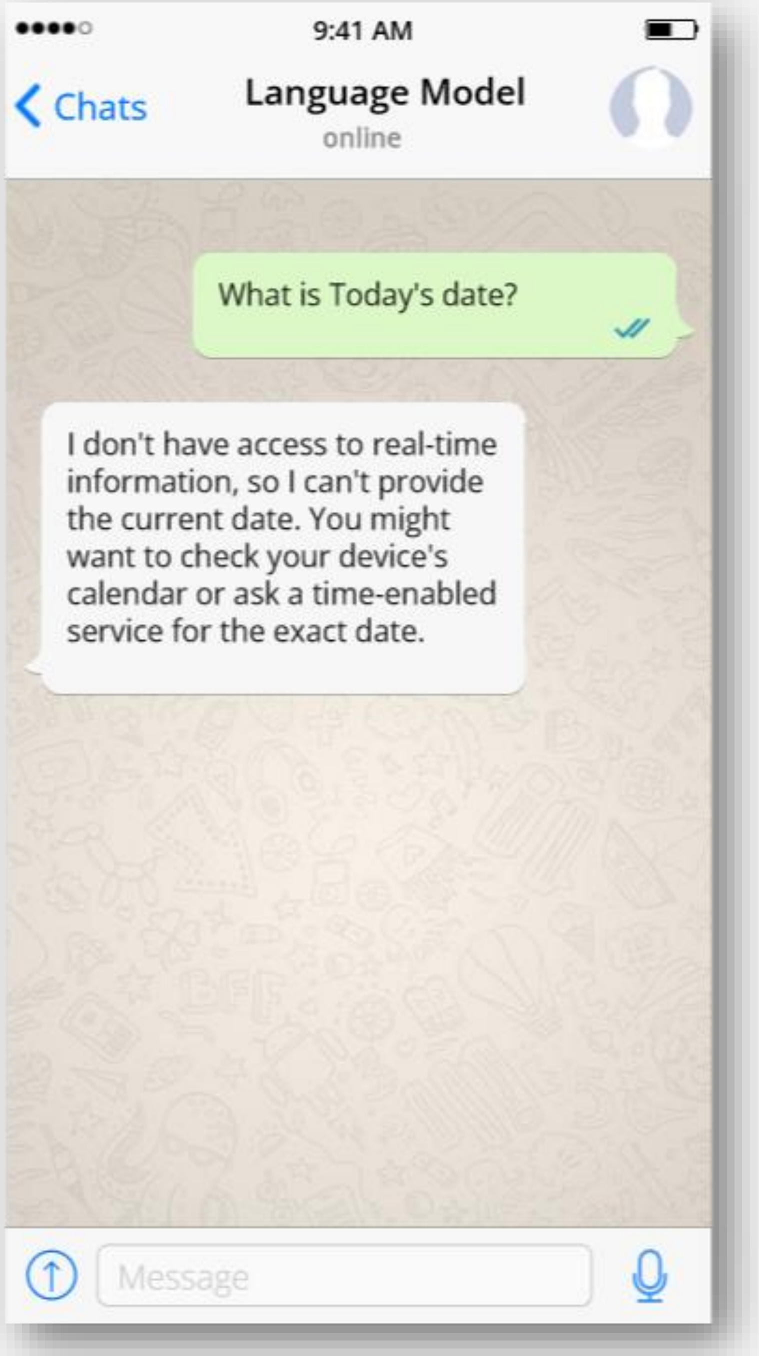


https://miro.medium.com/v2/resize:fit:4800/format:webp/1*2DjQyJMJRHofl2I_Nd7BMQ.png

What can LLMs NOT do?

- Emotions and Intentions
- Verification of Facts
- Ethical and Moral Judgments

- Access Real-time or Up-to-date Information
- Executing Tasks Requiring Physical Actions or Sensory Input



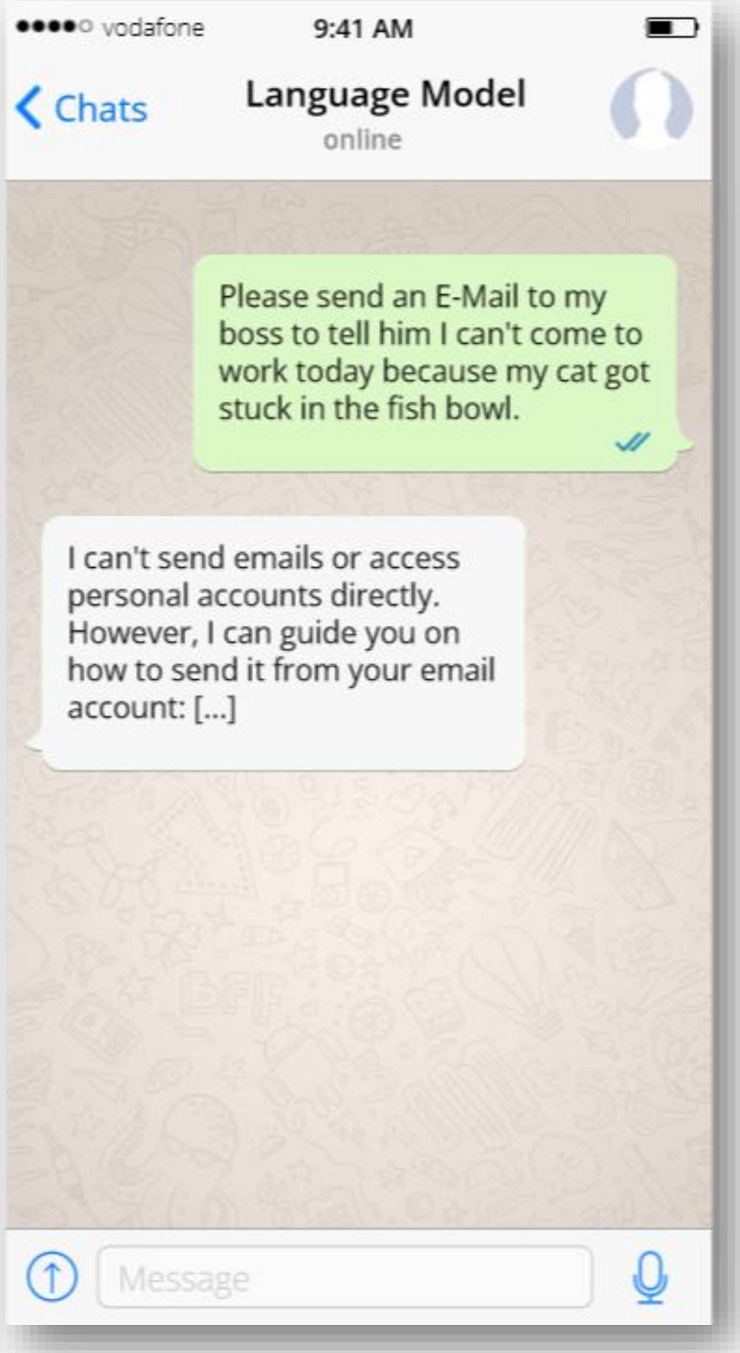
What is Today's date?

I don't have access to real-time information, so I can't provide the current date. You might want to check your device's calendar or ask a time-enabled service for the exact date.

What can LLMs NOT do?

- Emotions and Intentions
- Verification of Facts
- Ethical and Moral Judgments

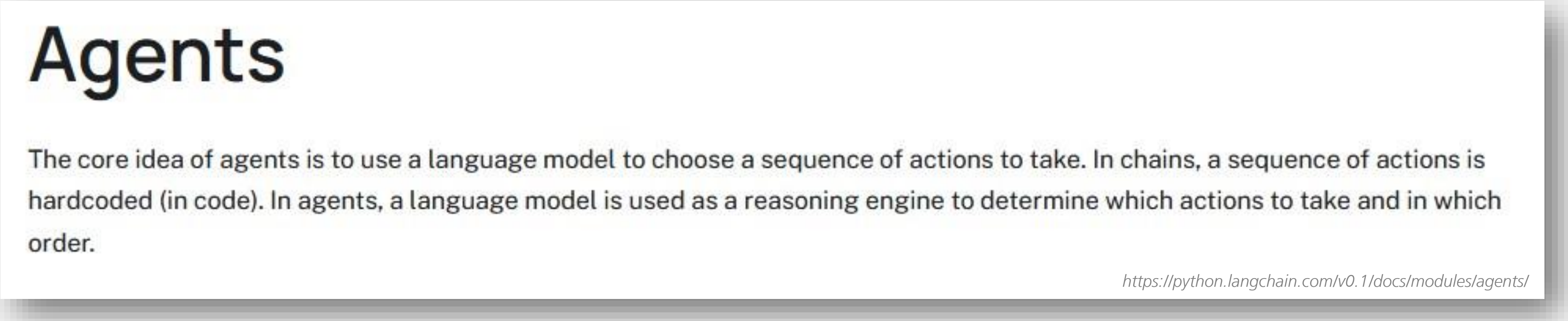
- **Access Real-time or Up-to-date Information**
- **Executing Tasks Requiring Physical Actions or Sensory Input**



Please send an E-Mail to my boss to tell him I can't come to work today because my cat got stuck in the fish bowl.

I can't send emails or access personal accounts directly. However, I can guide you on how to send it from your email account: [...]

Agent

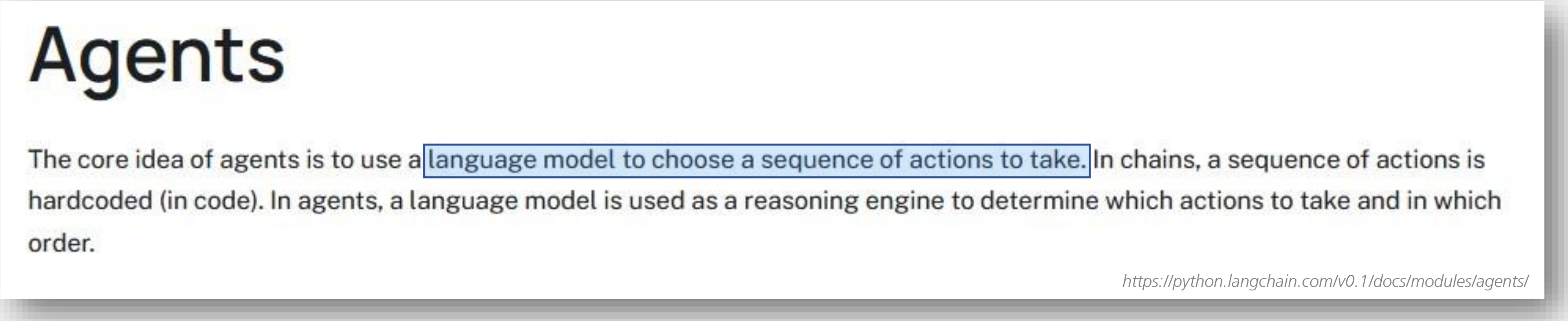


Agents

The core idea of agents is to use a language model to choose a sequence of actions to take. In chains, a sequence of actions is hardcoded (in code). In agents, a language model is used as a reasoning engine to determine which actions to take and in which order.

https://python.langchain.com/v0.1/docs/modules/agents/

Agent



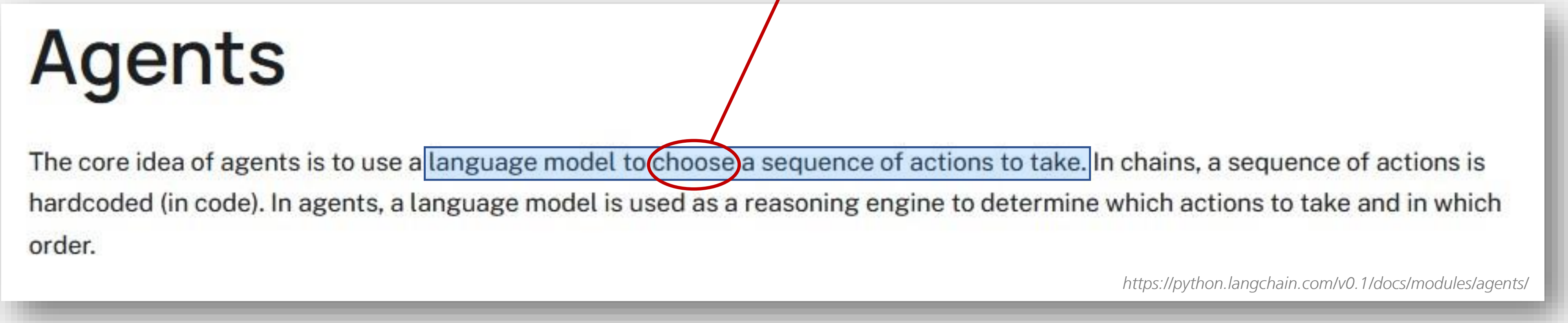
Agents

The core idea of agents is to use a language model to choose a sequence of actions to take. In chains, a sequence of actions is hardcoded (in code). In agents, a language model is used as a reasoning engine to determine which actions to take and in which order.

https://python.langchain.com/v0.1/docs/modules/agents/

Agent

LLMs don't perform the action themselves, they only trigger an external service to do it



Agents

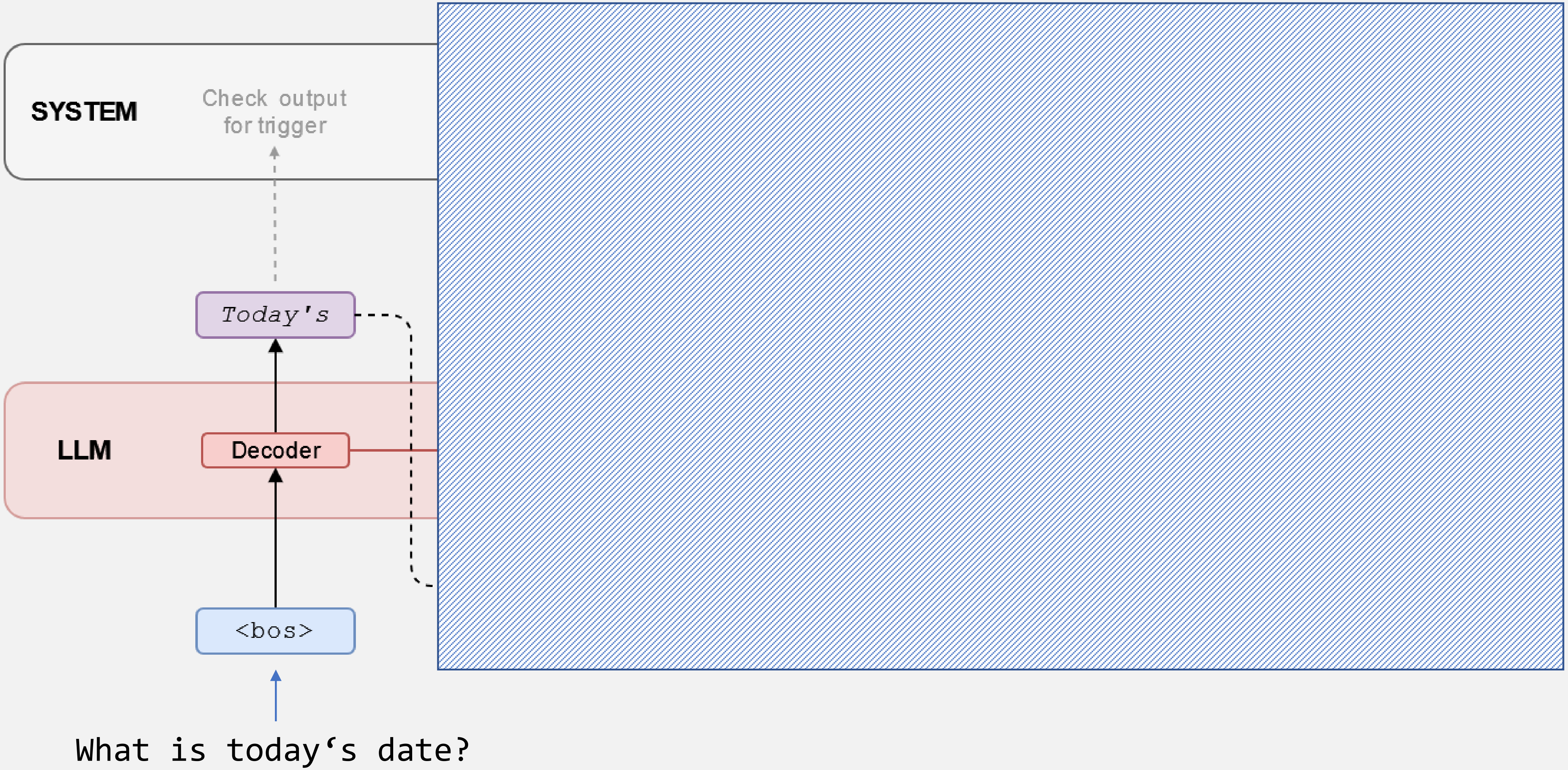
The core idea of agents is to use a language model to choose a sequence of actions to take. In chains, a sequence of actions is hardcoded (in code). In agents, a language model is used as a reasoning engine to determine which actions to take and in which order.

https://python.langchain.com/v0.1/docs/modules/agents/

What is a „Trigger“? -> The name of the tool that the agent chose to execute

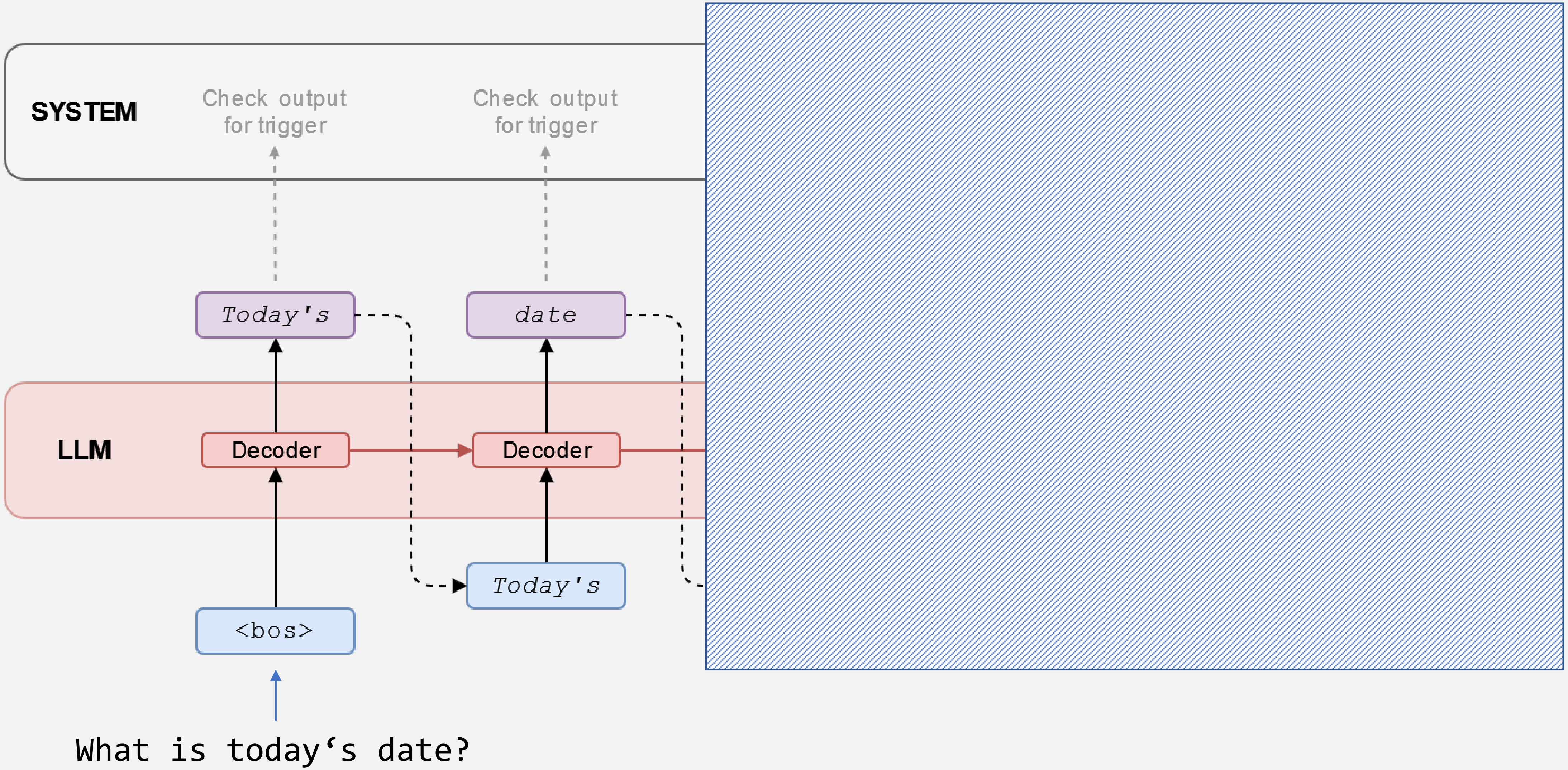
Agent

LLMs don't perform the action themselves, they only trigger an external service to do it.



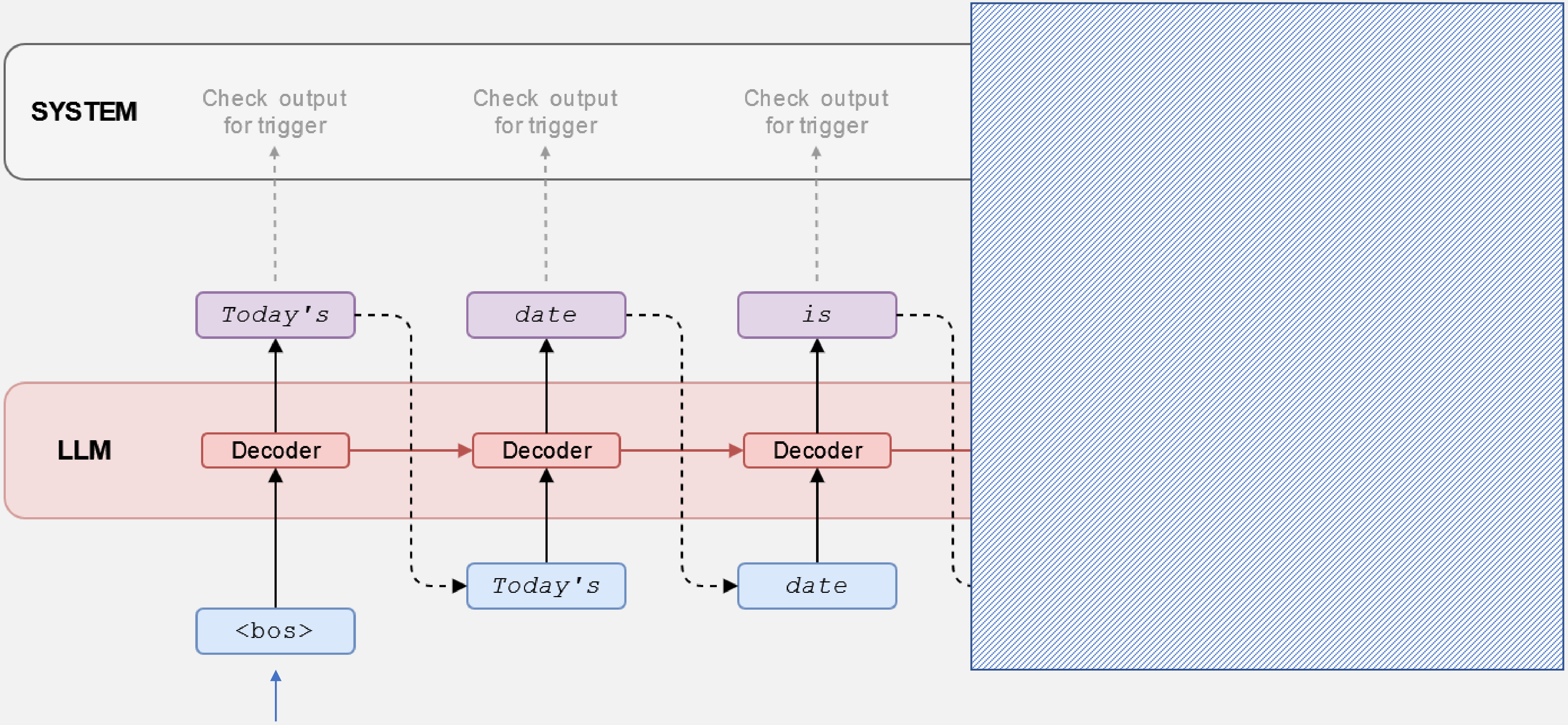
Agent

LLMs don't perform the action themselves, they only trigger an external service to do it.



LLMs don‘t perform the action themselves, they only trigger an external service to do it.

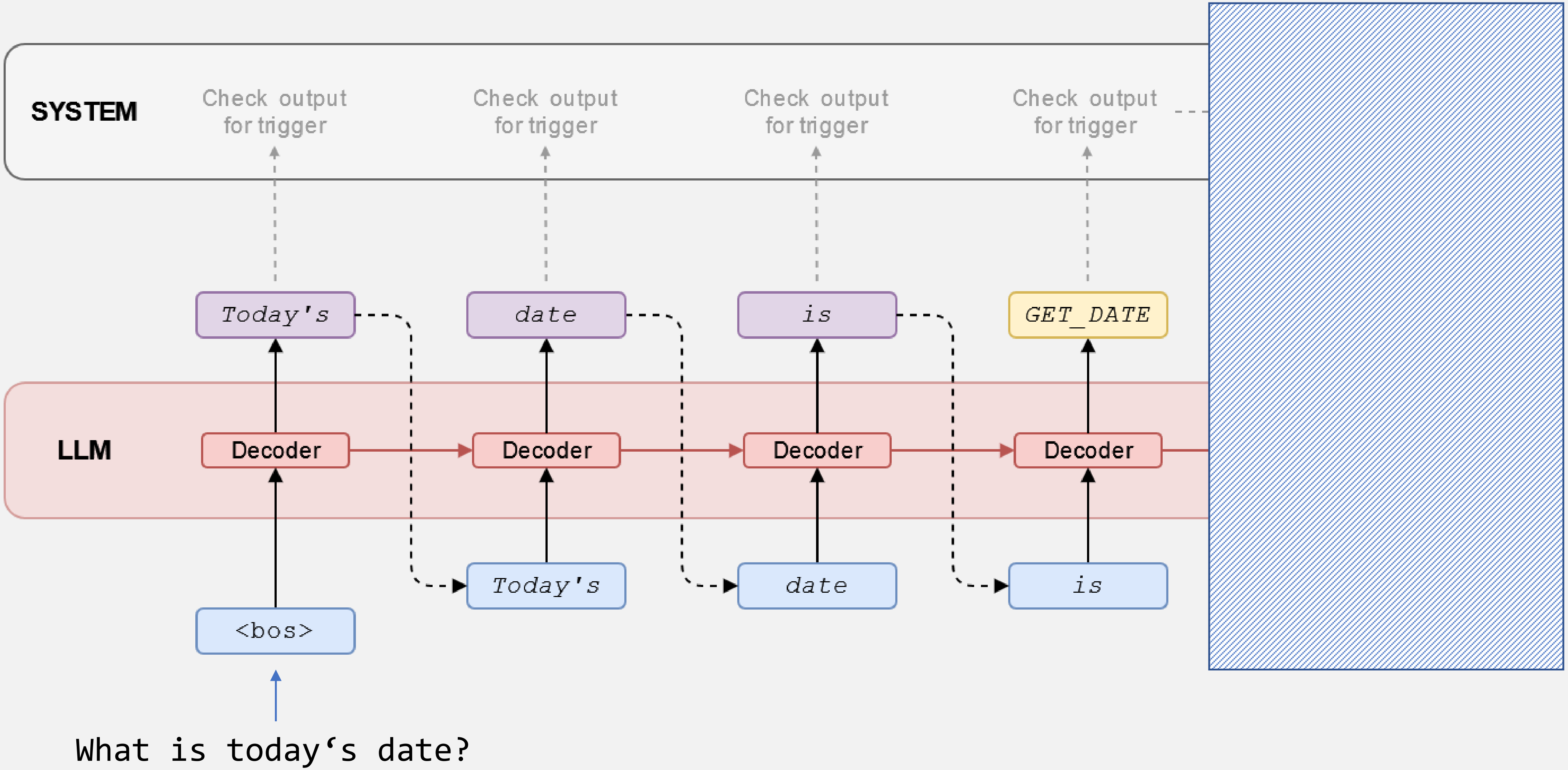
Agent



What is today‘s date?

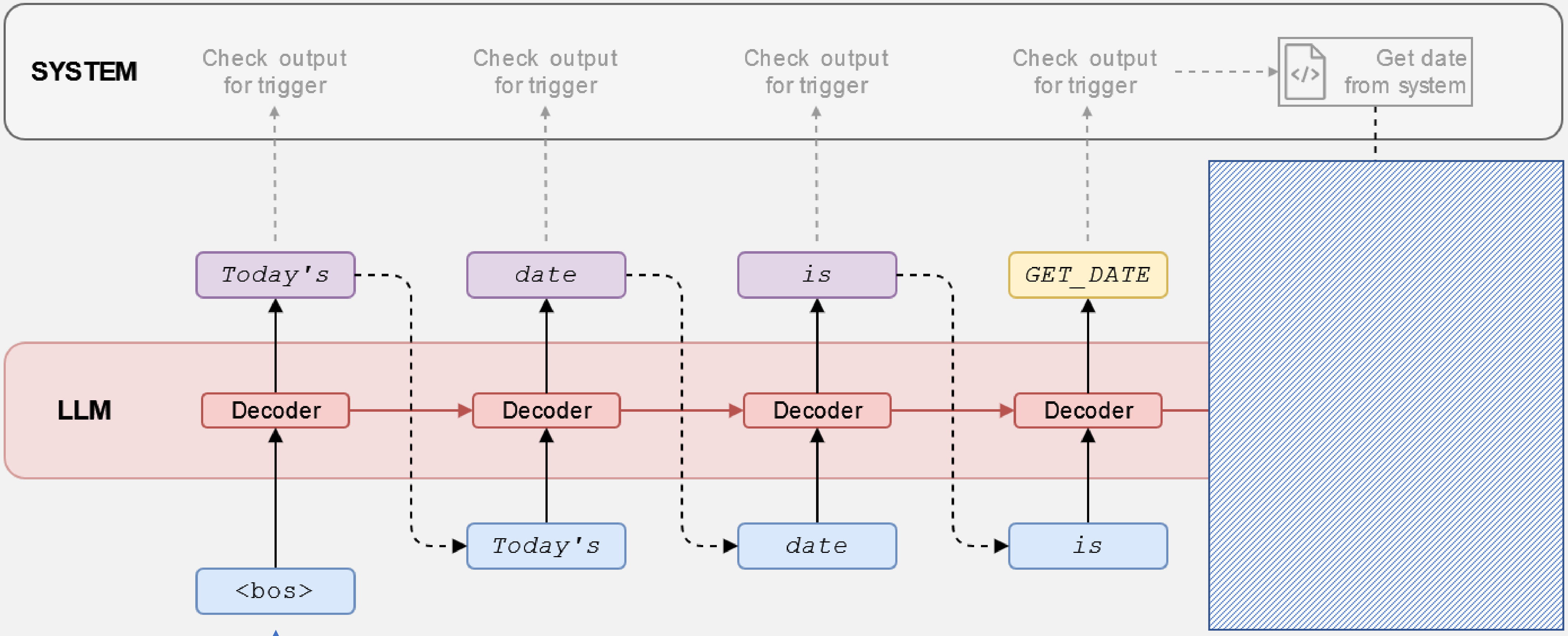
LLMs don‘t perform the action themselves, they only trigger an external service to do it.

Agent



Agent

LLMs don‘t perform the action themselves, they only trigger an external service to do it.

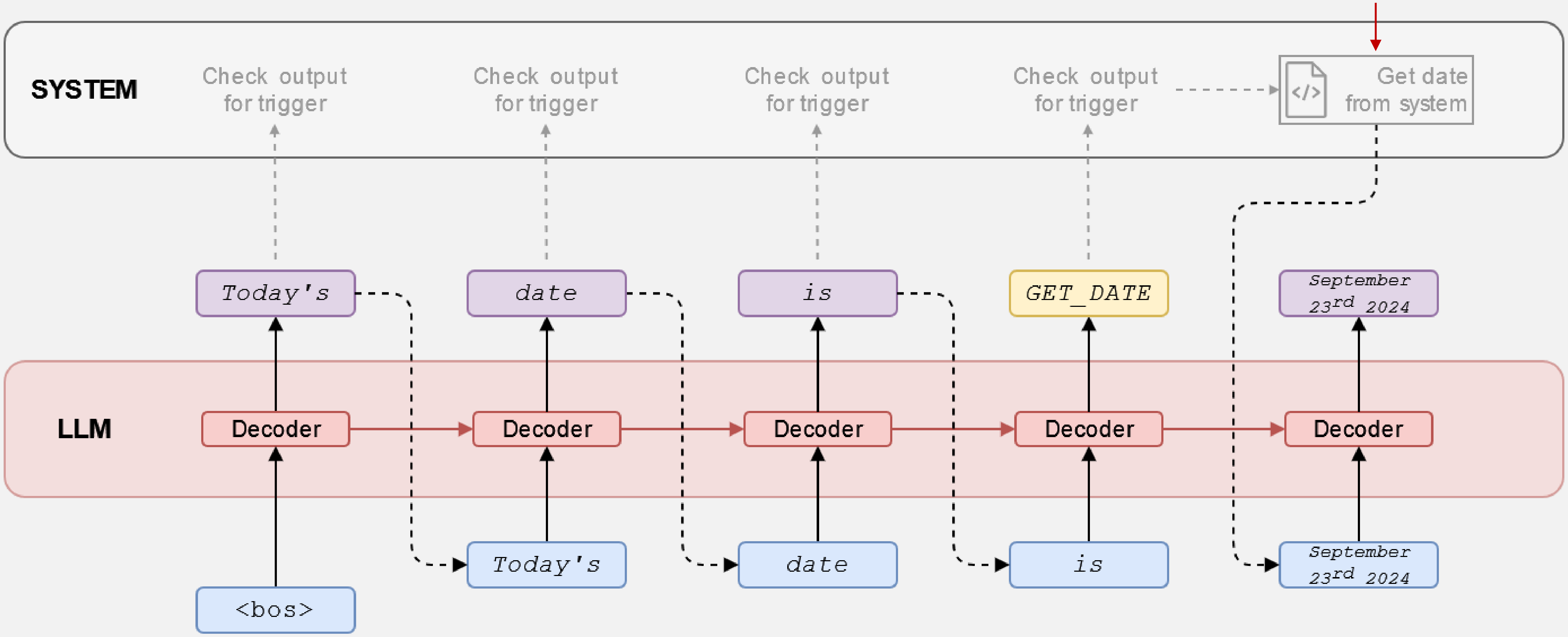


What is today‘s date?

Agent

LLMs don't perform the action themselves, they only trigger an external service to do it.

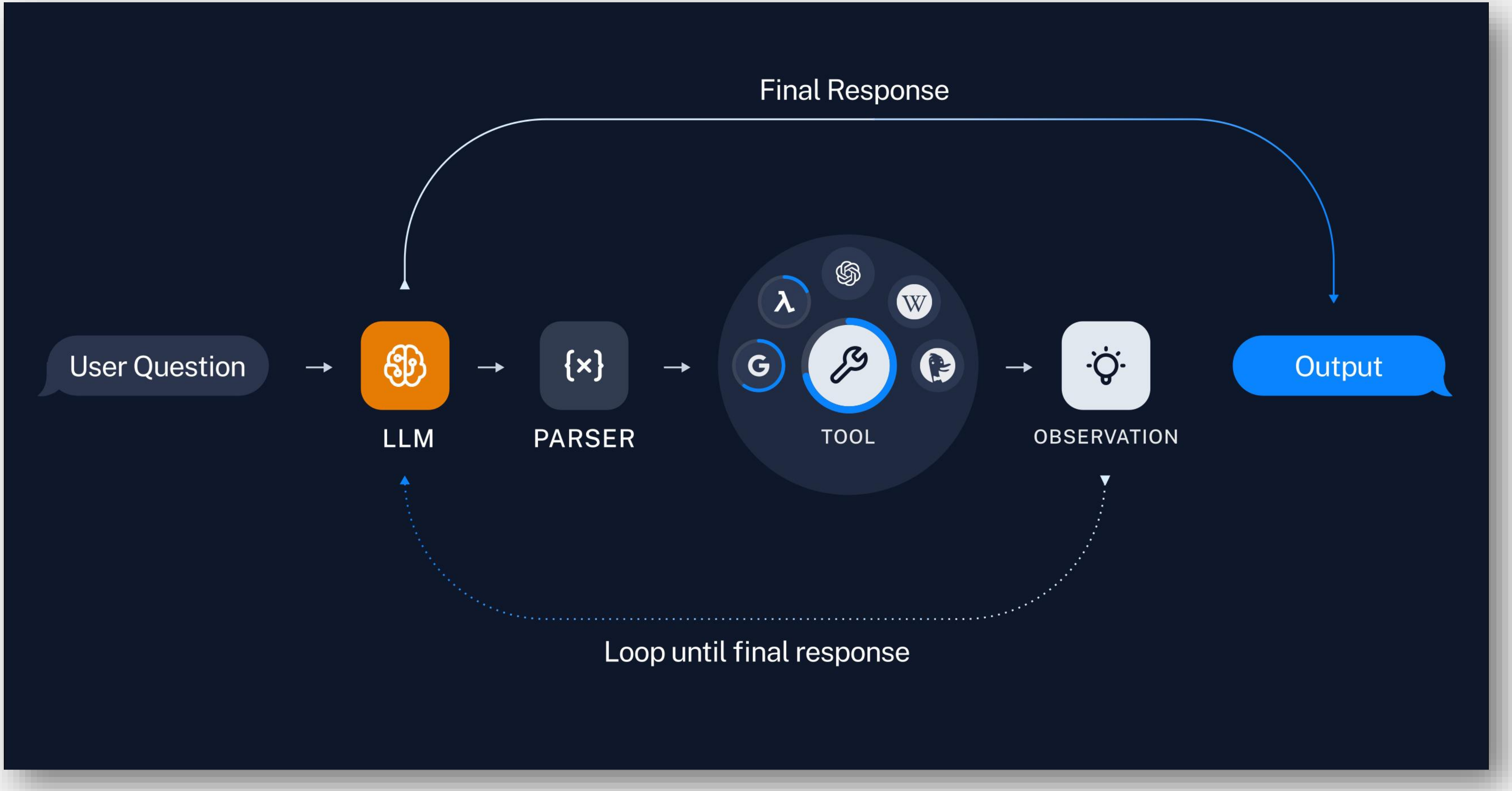
! Implemented Manually !



What is today‘s date?

Agent

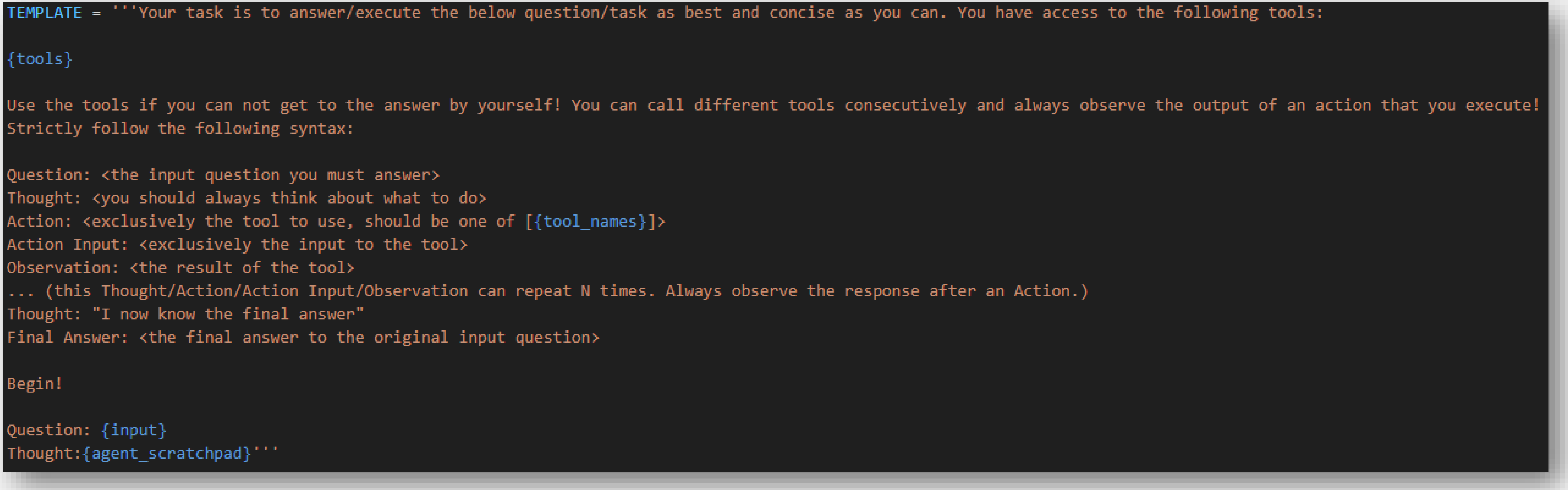
LLMs don't perform the action themselves, they only trigger an external service to do it.



https://python.langchain.com/v0.1/docs/use_cases/tool_use/

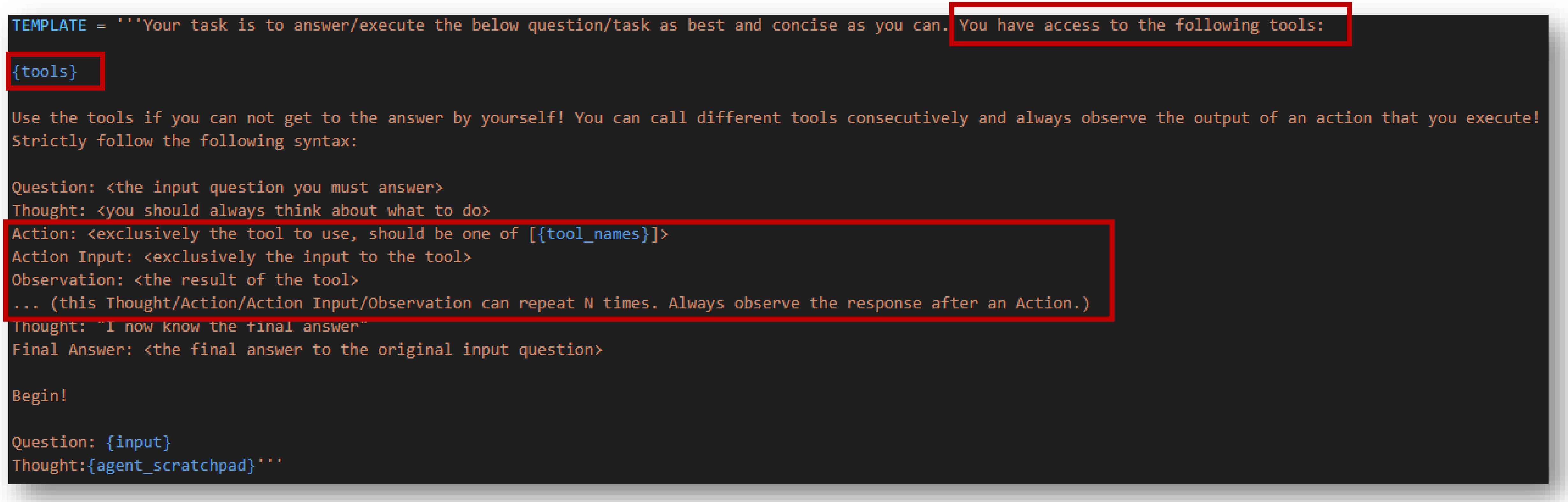
It‘s not that difficult to implement! ☺

Agent - Prompting



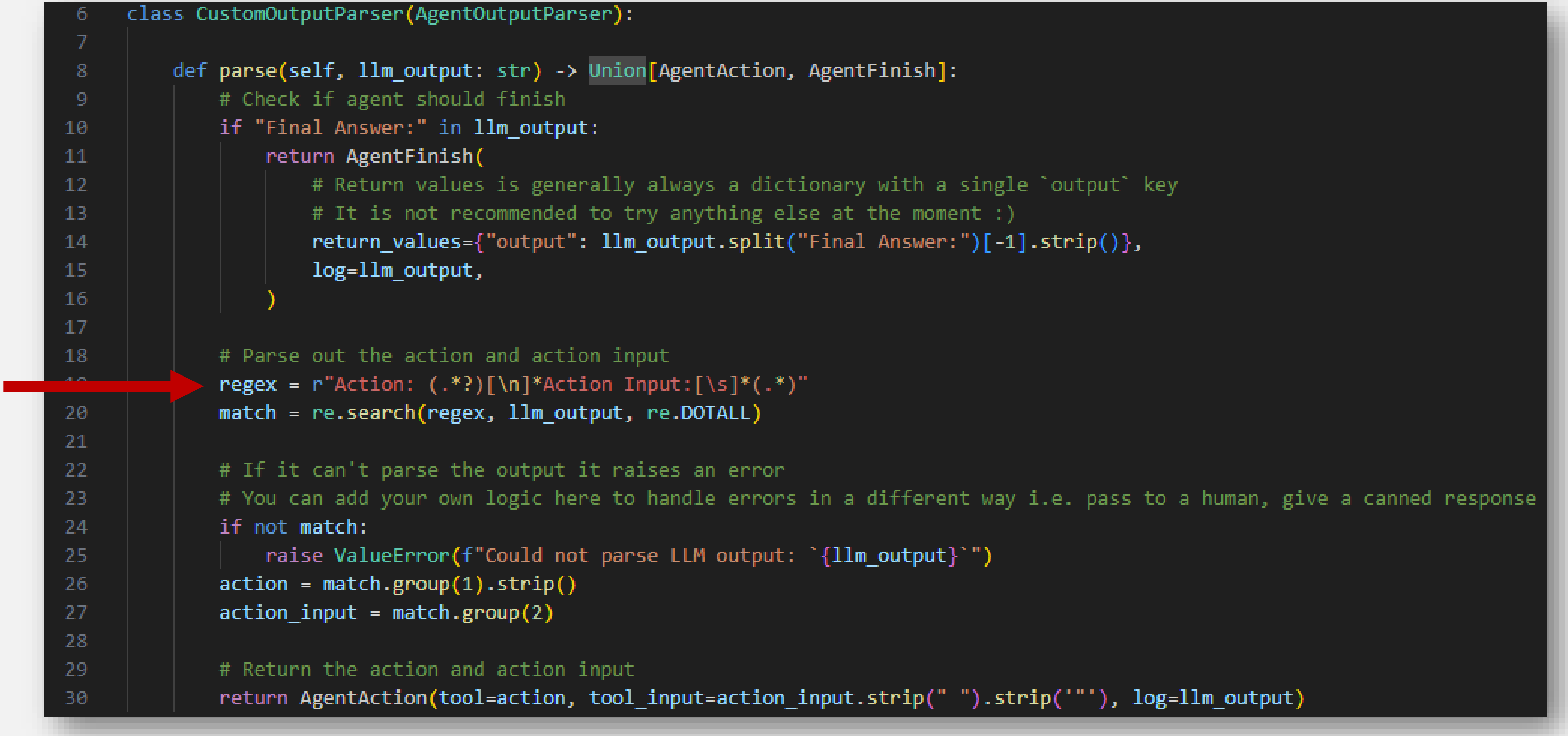
```
TEMPLATE = '''Your task is to answer/execute the below question/task as best and concise as you can. You have access to the following tools:

{tools}

Use the tools if you can not get to the answer by yourself! You can call different tools consecutively and always observe the output of an action that you execute!
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 [{tool_names}]>
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 response after an Action.)
Thought: "I now know the final answer"
Final Answer: <the final answer to the original input question>

Begin!

Question: {input}
Thought:{agent_scratchpad}'''
```

Agent - Prompting



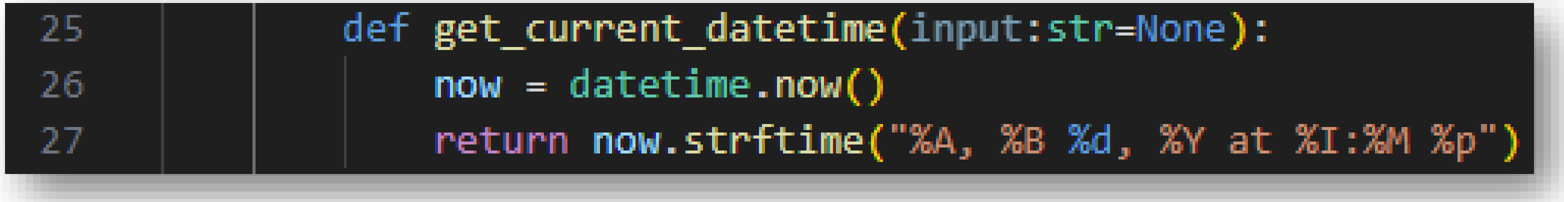
```
TEMPLATE = '''Your task is to answer/execute the below question/task as best and concise as you can. You have access to the following tools:

{tools}

Use the tools if you can not get to the answer by yourself! You can call different tools consecutively and always observe the output of an action that you execute!
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 [{tool_names}]>
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 response after an Action.)
Thought: "I now know the final answer"
Final Answer: <the final answer to the original input question>

Begin!

Question: {input}
Thought:{agent_scratchpad}'''
```

Agent - Parsing



```python
class CustomOutputParser(AgentOutputParser):

    def parse(self, llm_output: str) -> Union[AgentAction, AgentFinish]:
        # Check if agent should finish
        if "Final Answer:" in llm_output:
            return AgentFinish(
                # Return values is generally always a dictionary with a single `output` key
                # It is not recommended to try anything else at the moment :)
                return_values={"output": llm_output.split("Final Answer:")[-1].strip()},
                log=llm_output,
            )

        # Parse out the action and action input
        regex = r"Action: (.*?)[\n]*Action Input:[\s]*(.*)"
        match = re.search(regex, llm_output, re.DOTALL)

        # If it can't parse the output it raises an error
        # You can add your own logic here to handle errors in a different way i.e. pass to a human, give a canned response
        if not match:
            raise ValueError(f"Could not parse LLM output: `{llm_output}`")
        action = match.group(1).strip()
        action_input = match.group(2)

        # Return the action and action input
        return AgentAction(tool=action, tool_input=action_input.strip(" ").strip('"'), log=llm_output)
```

Agent – Tool (Example)



```python
def get_current_datetime(input:str=None):
    now = datetime.now()
    return now.strftime("%A, %B %d, %Y at %I:%M %p")
```