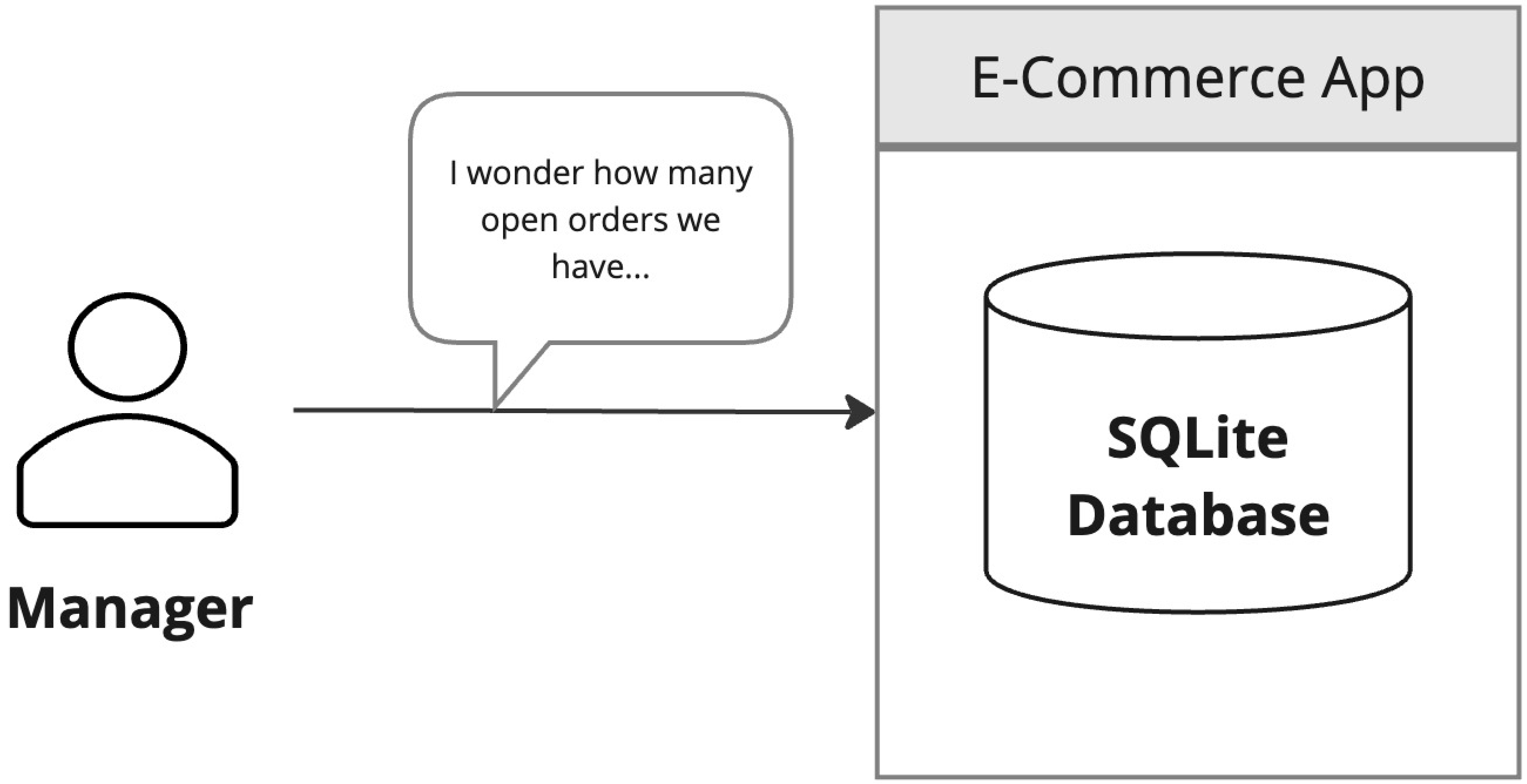
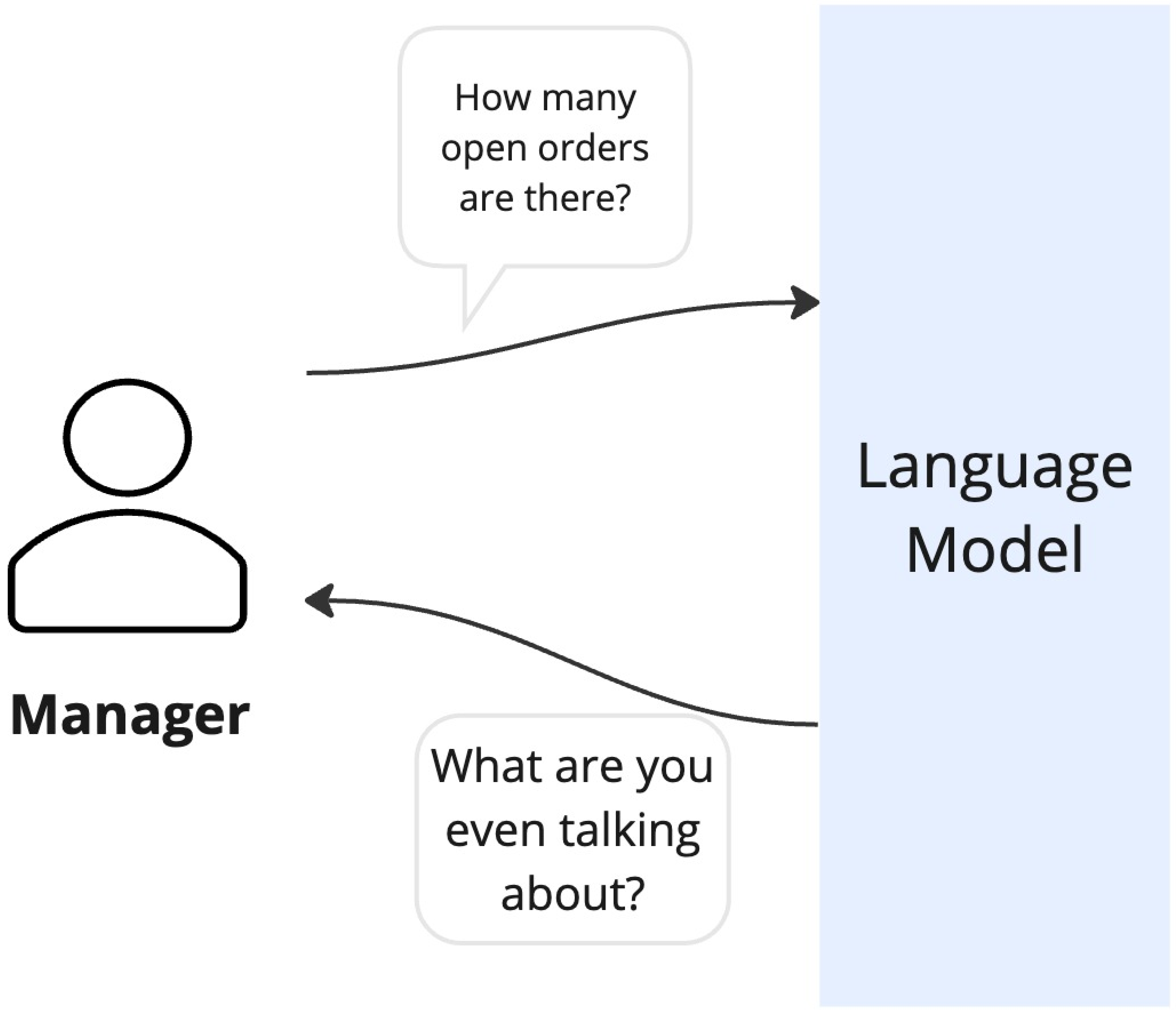
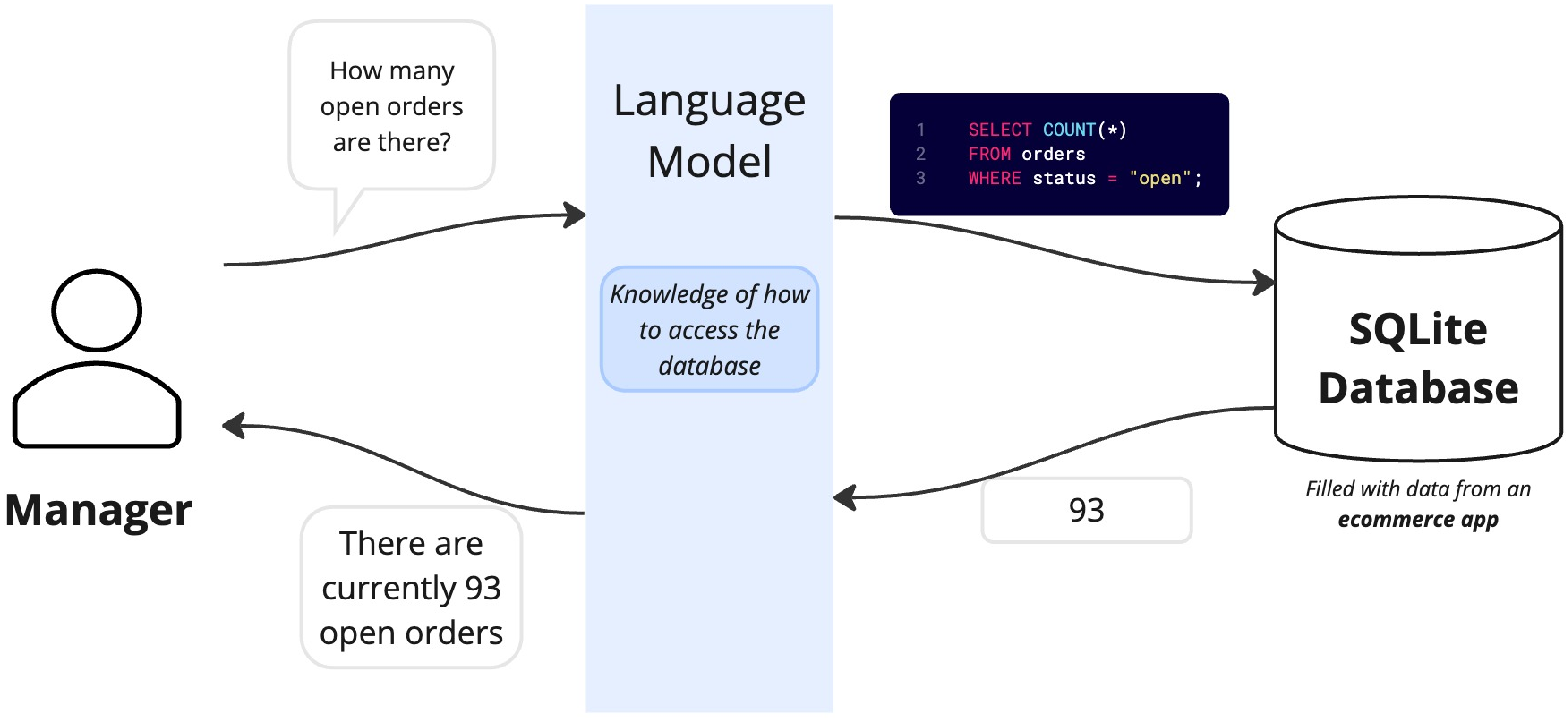
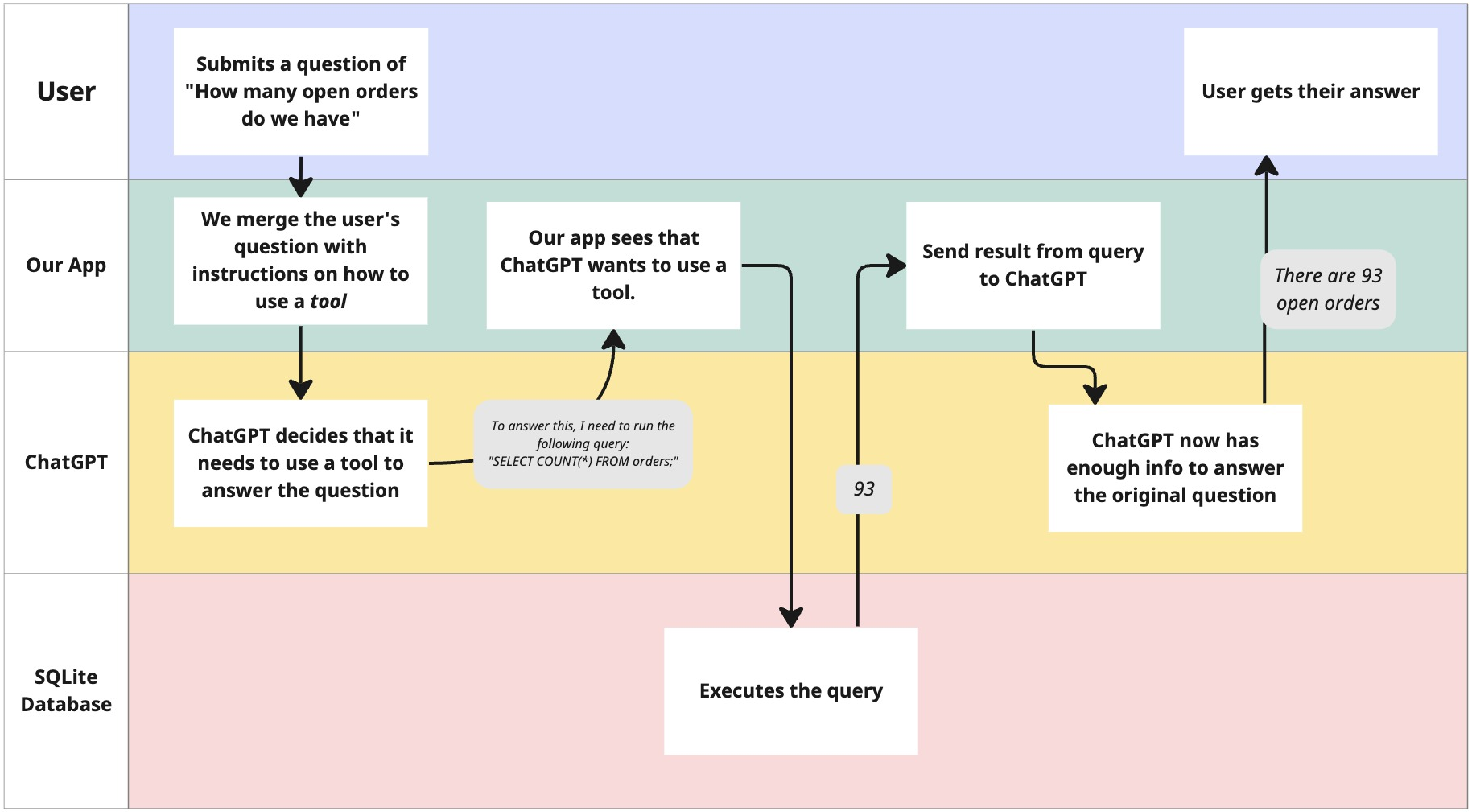

I wonder how many open orders we have...
Manager
E-Commerce App
SQLite Database


How many open orders are there?
Manager
Language Model
What are you even talking about?


How many open orders are there?
Language Model
Knowledge of how to access the database
SQLite Database
Filled with data from an ***ecommerce app***
93
There are currently 93 open orders
Manager

```sql
SELECT COUNT(*)
FROM orders
WHERE status = "open";
```


User
Our App
ChatGPT
SQLite Database
Submits a question of "How many open orders do we have"
We merge the user's question with instructions on how to use a *tool*
ChatGPT decides that it needs to use a tool to answer the question
To answer this, I need to run the following query: "SELECT COUNT(*) FROM orders;"
Our app sees that ChatGPT wants to use a tool.
Executes the query
93
Send result from query to ChatGPT
ChatGPT now has enough info to answer the original question
There are 93 open orders
User gets their answer

platform.openai.com/playground

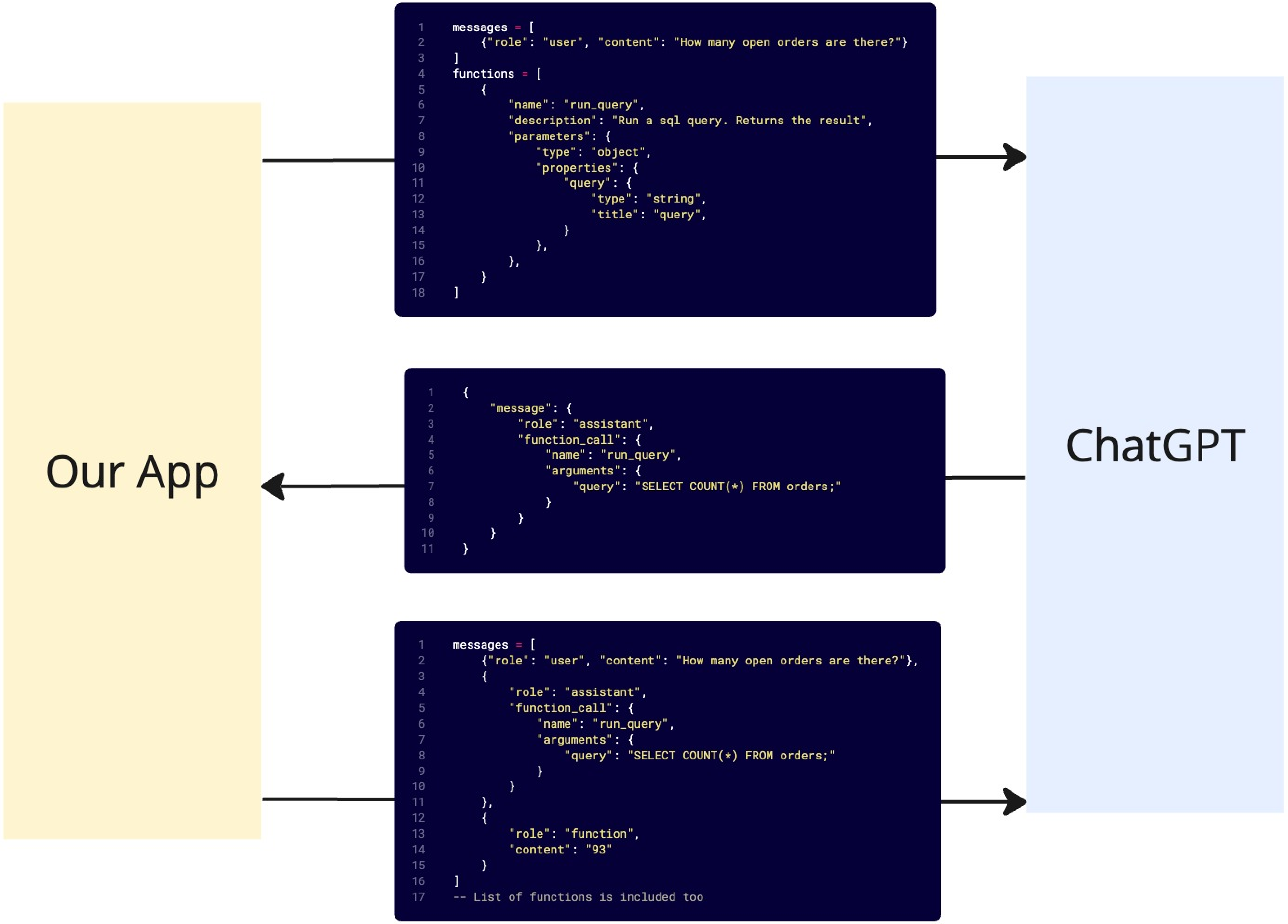
OpenAI saw people doing this a lot

Uses up extra tokens in the prompt!

ChatGPT wouldn't always respond with properly formatted JSON

```
You have access to the following tools:

- run_query: runs a sqlite query and returns the
result. Accepts an argument of a sql query as a string

To use a tool always respond with the following
format:

{
  "name": <name of tool to use>,
  "argument": <argument to pass to the tool>
}
```

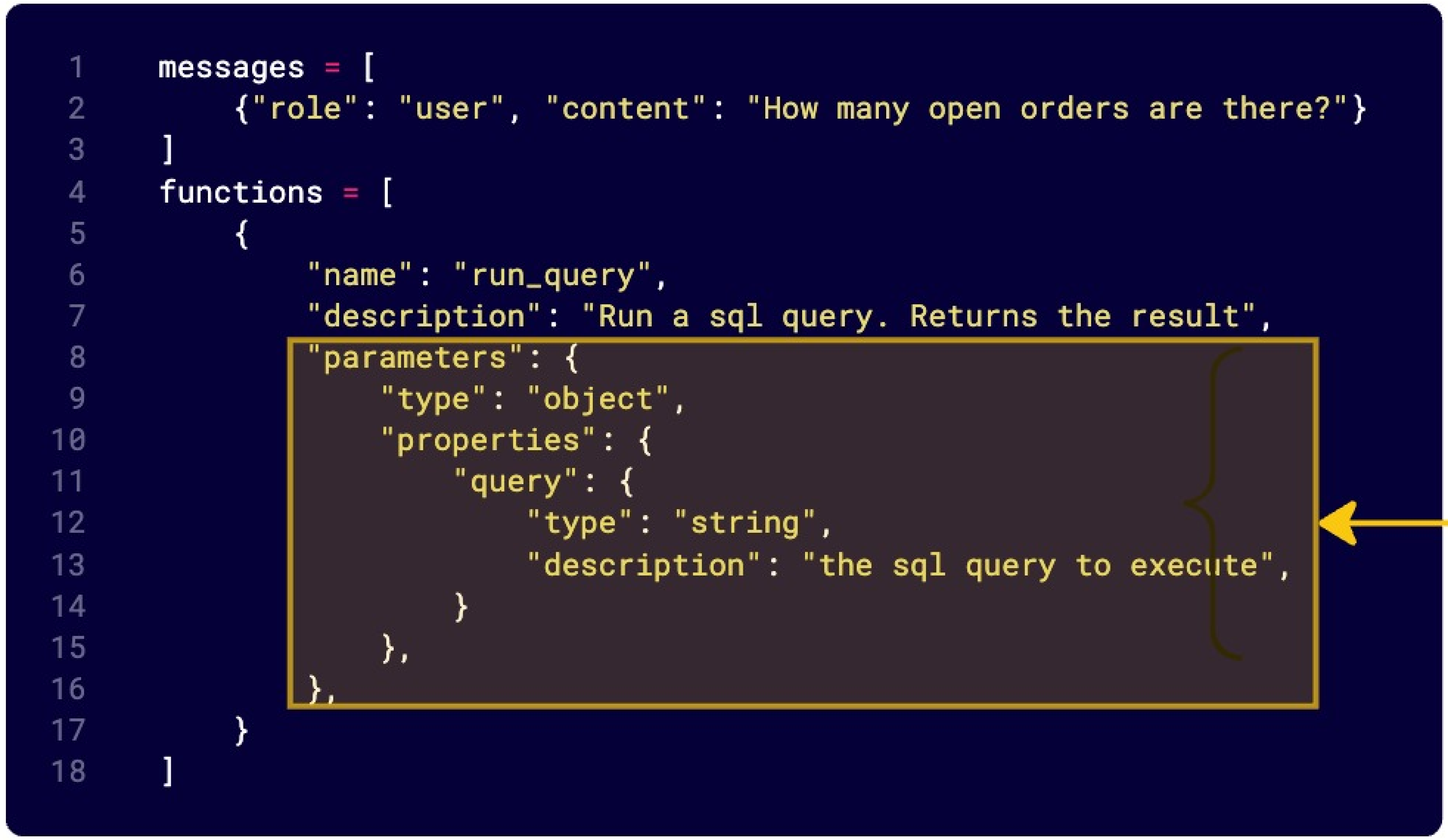
How many open orders do we have?



```
messages = [
    {"role": "user", "content": "How many open orders are there?"}
]
functions = [
    {
        "name": "run_query",
        "description": "Run a sql query. Returns the result",
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "string",
                    "title": "query",
                }
            },
        },
    }
]
```

```
{
    "message": {
        "role": "assistant",
        "function_call": {
            "name": "run_query",
            "arguments": {
                "query": "SELECT COUNT(*) FROM orders;"
            }
        }
    }
}
```

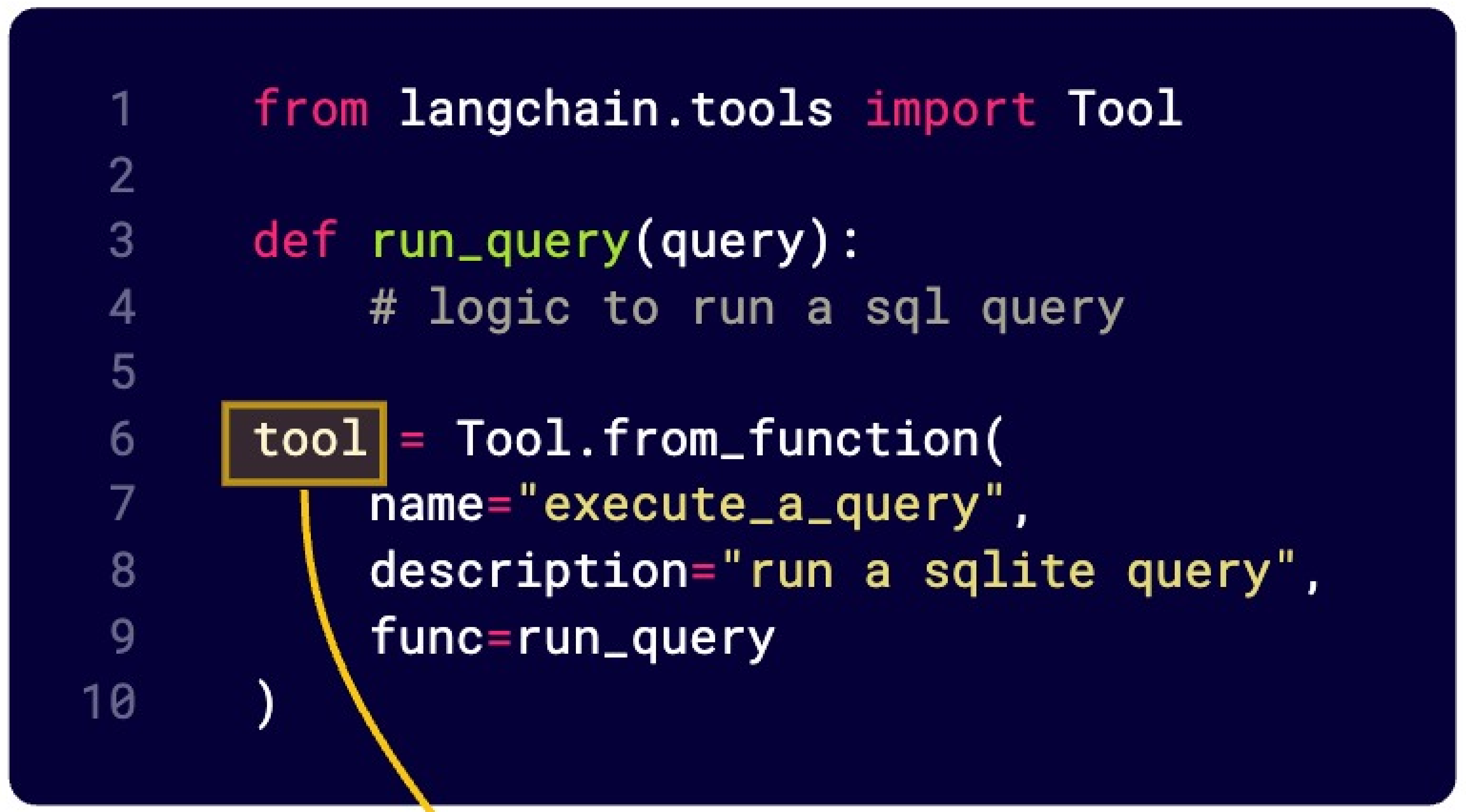
```
messages = [
    {"role": "user", "content": "How many open orders are there?"},
    {
        "role": "assistant",
        "function_call": {
            "name": "run_query",
            "arguments": {
                "query": "SELECT COUNT(*) FROM orders;"
            }
        }
    },
    {
        "role": "function",
        "content": "93"
    }
]
-- List of functions is included too
```



```
messages = [
    {"role": "user", "content": "How many open orders are there?"}
]
functions = [
    {
        "name": "run_query",
        "description": "Run a sql query. Returns the result",
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "string",
                    "description": "the sql query to execute",
                }
            },
        },
    }
]
```

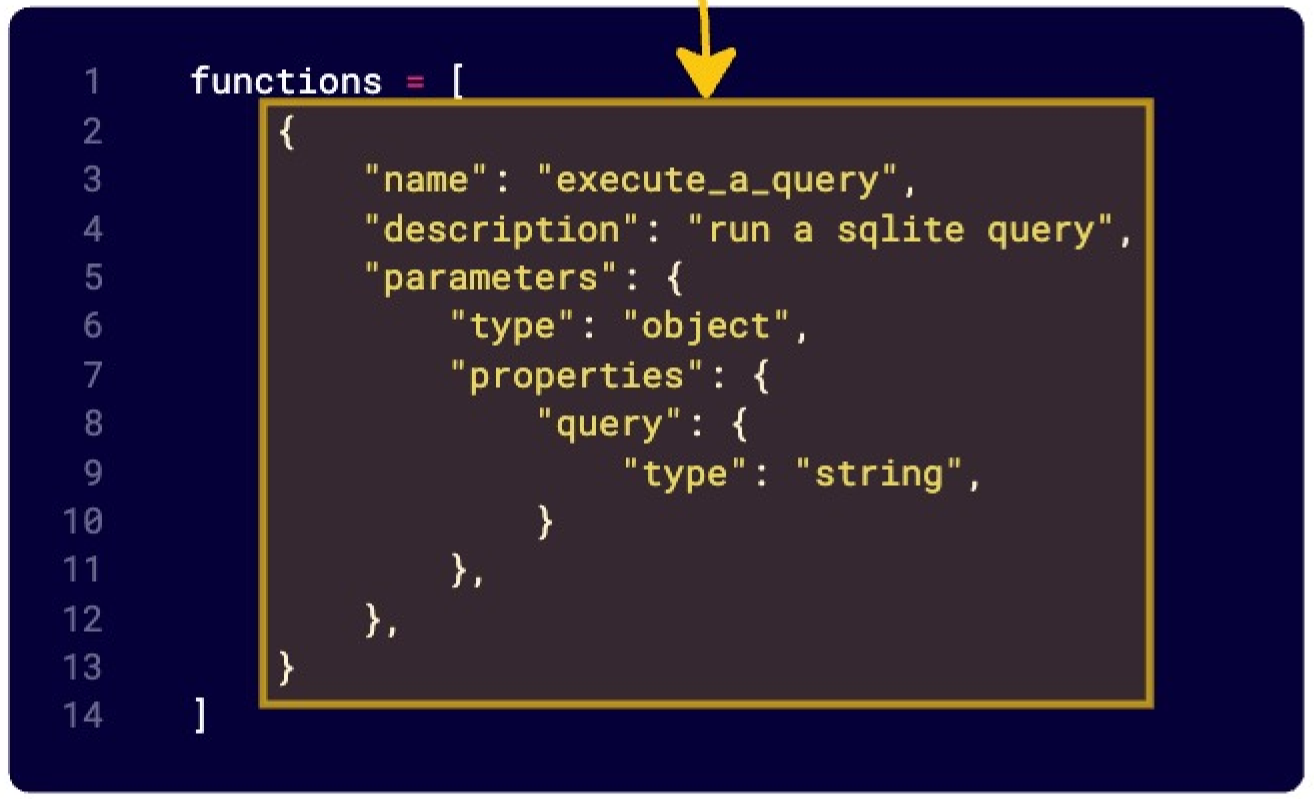
Arguments are described using **JSON schema** transform.tools/json-to-json-schema

Tool to help you
generate a JSON schema

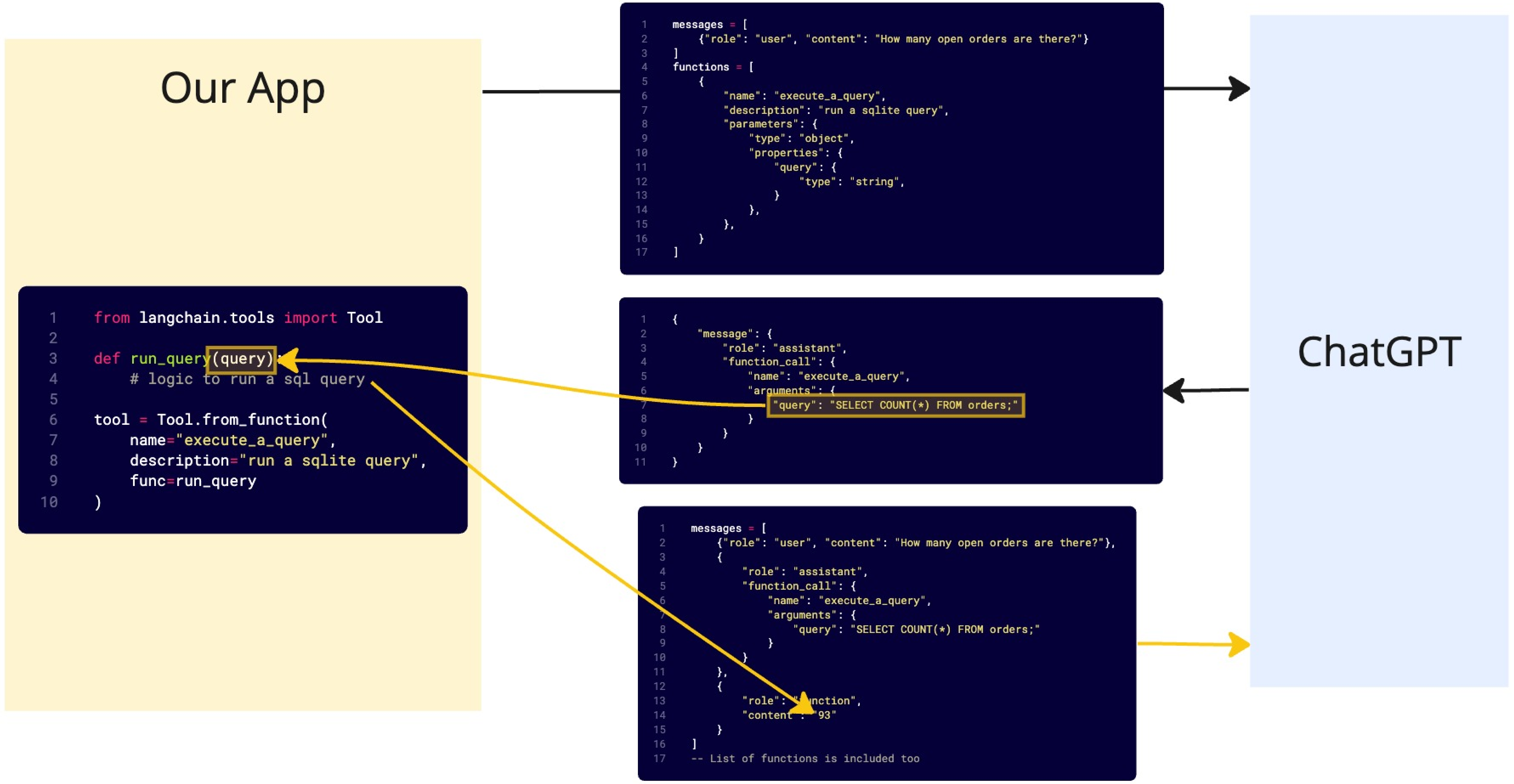


```python
from langchain.tools import Tool

def run_query(query):
    # logic to run a sql query

tool = Tool.from_function(
    name="execute_a_query",
    description="run a sqlite query",
    func=run_query
)
```

Langchain will convert the "tool" object into a function description for OpenAI



```python
functions = [
    {
        "name": "execute_a_query",
        "description": "run a sqlite query",
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "string",
                }
            },
        },
    }
]
```



```python
from langchain.tools import Tool

def run_query(query):
    # logic to run a sql query

tool = Tool.from_function(
    name="execute_a_query",
    description="run a sqlite query",
    func=run_query
)
```

```
messages = [
    {"role": "user", "content": "How many open orders are there?"}
]
functions = [
    {
        "name": "execute_a_query",
        "description": "run a sqlite query",
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "string",
                }
            },
        },
    }
]
```

```
{
    "message": {
        "role": "assistant",
        "function_call": {
            "name": "execute_a_query",
            "arguments": {
                "query": "SELECT COUNT(*) FROM orders;"
            }
        }
    }
}
```

```
messages = [
    {"role": "user", "content": "How many open orders are there?"},
    {
        "role": "assistant",
        "function_call": {
            "name": "execute_a_query",
            "arguments": {
                "query": "SELECT COUNT(*) FROM orders;"
            }
        }
    },
    {
        "role": "function",
        "content": "93"
    }
]
-- List of functions is included too
```

Agent

A chain that knows how to use tools

Will take that list of tools and convert them into JSON function descriptions

Still has input variables, memory, prompts, etc - all the normal things a chain has

```python
agent = OpenAIFunctionsAgent(
    llm=chat,
    prompt=prompt,
    tools=tools
)
```

Agent Executor

Takes an *agent* and runs it until the response is **not** a function call

Essentially a fancy while loop

```python
agent_executor = AgentExecutor(
    agent=agent,
    verbose=True,
    tools=tools
)
```

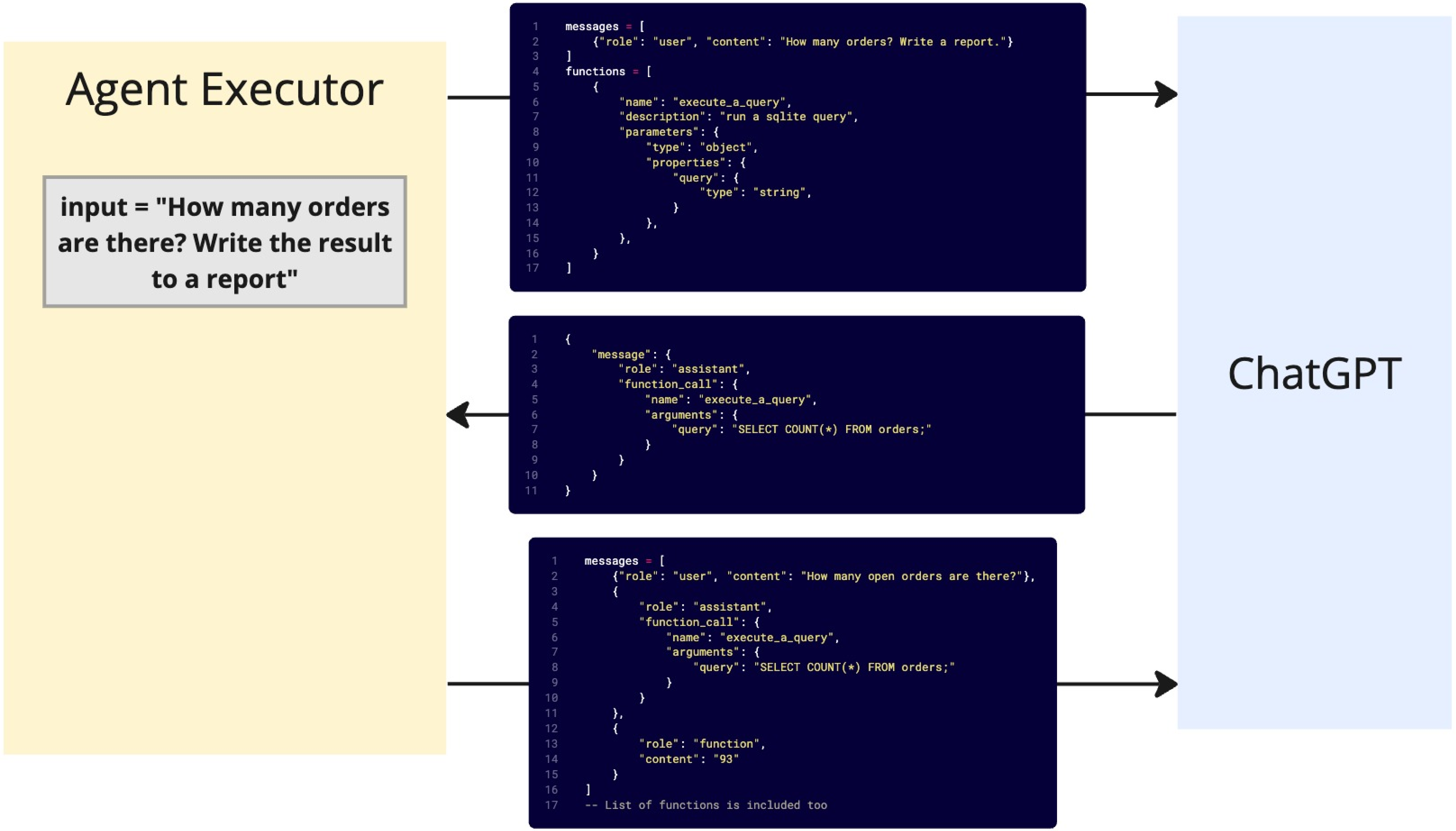
```python
# Fake implementation
class AgentExecutor:
    def __call__(self, input):
        while True:
            result = self.agent(input)

            if result == RequestToCallATool:
                call_tool(result)
            else:
                return result
```

Important

The docs show several different ways of creating an Agent + AgentExecutor

They are all doing the same thing behind the scenes!

```python
from langchain.agents import initialize_agent, AgentType

executor = initialize_agent(
    llm=chat,
    tools=tools,
    agent=AgentType.OPENAI_FUNCTIONS,
    verbose=True,
)
```

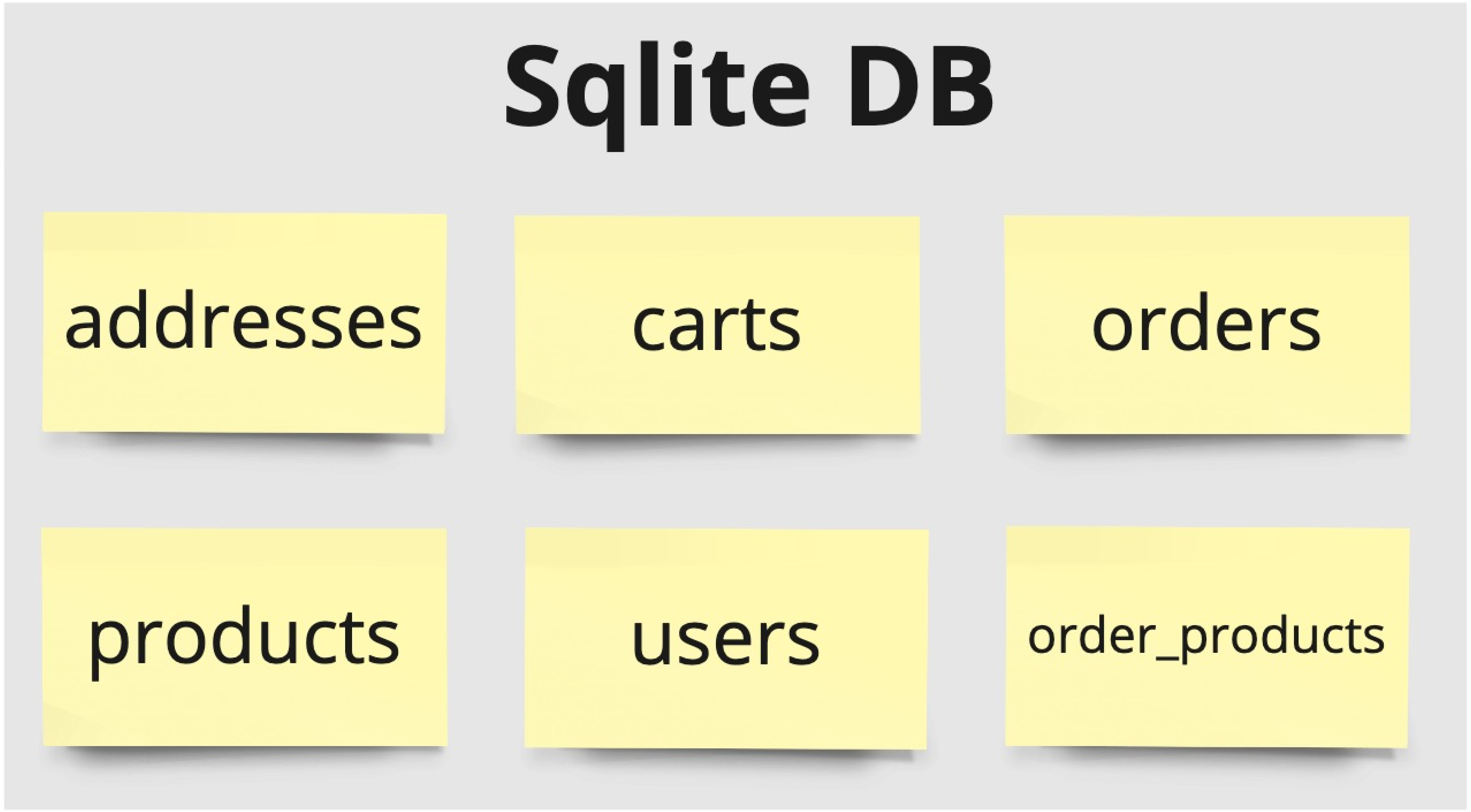


input = "How many orders are there? Write the result to a report"

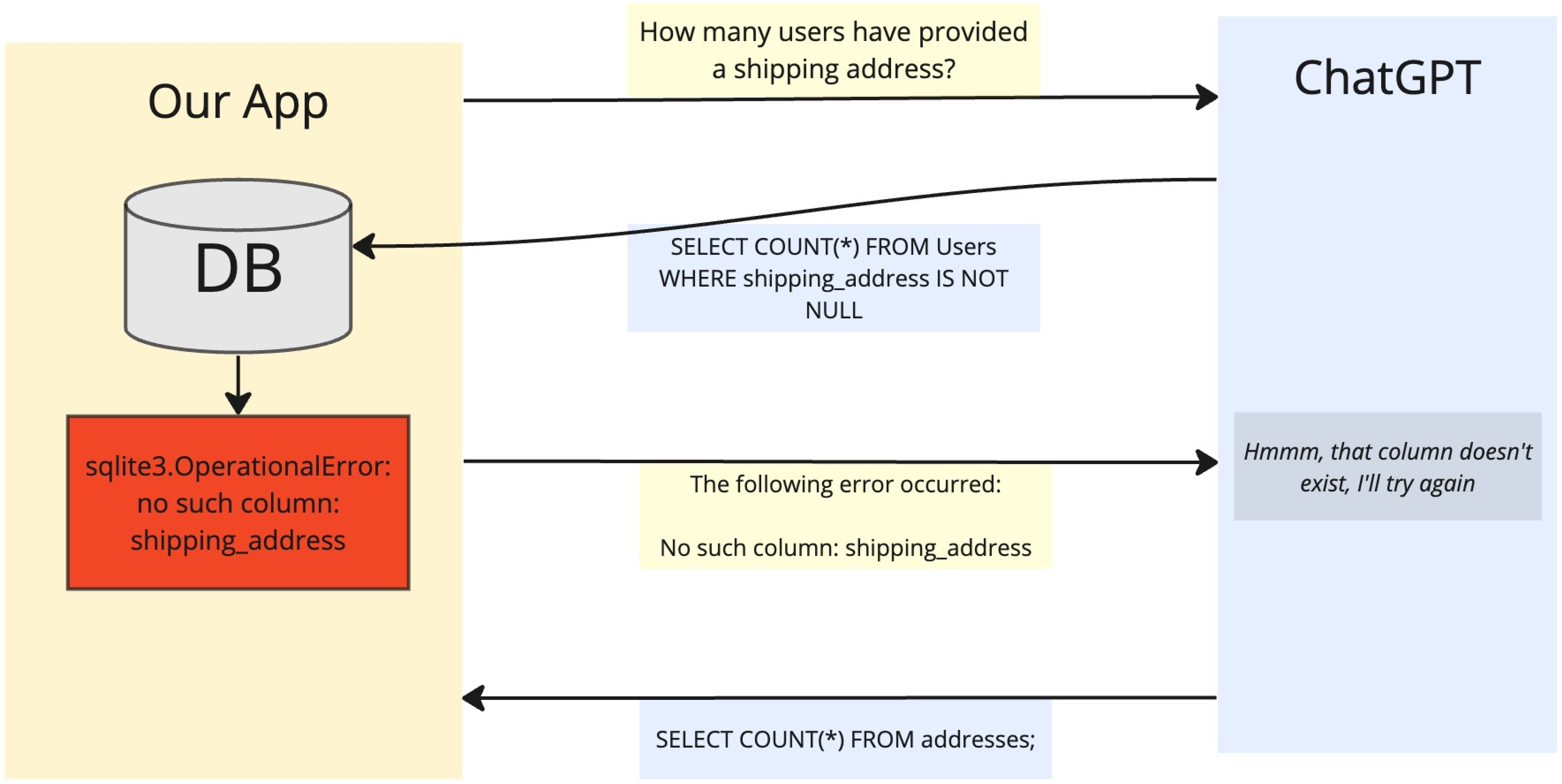
```
messages = [
    {"role": "user", "content": "How many orders? Write a report."}
]
functions = [
    {
        "name": "execute_a_query",
        "description": "run a sqlite query",
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "string",
                }
            },
        },
    }
]
```

```
{
    "message": {
        "role": "assistant",
        "function_call": {
            "name": "execute_a_query",
            "arguments": {
                "query": "SELECT COUNT(*) FROM orders;"
            }
        }
    }
}
```

```
messages = [
    {"role": "user", "content": "How many open orders are there?"},
    {
        "role": "assistant",
        "function_call": {
            "name": "execute_a_query",
            "arguments": {
                "query": "SELECT COUNT(*) FROM orders;"
            }
        }
    },
    {
        "role": "function",
        "content": "93"
    }
]
-- List of functions is included too
```



Tables that the DB has


Our App
ChatGPT
DB
How many users have provided a shipping address?
Hmmm, that column doesn't exist, I'll try again

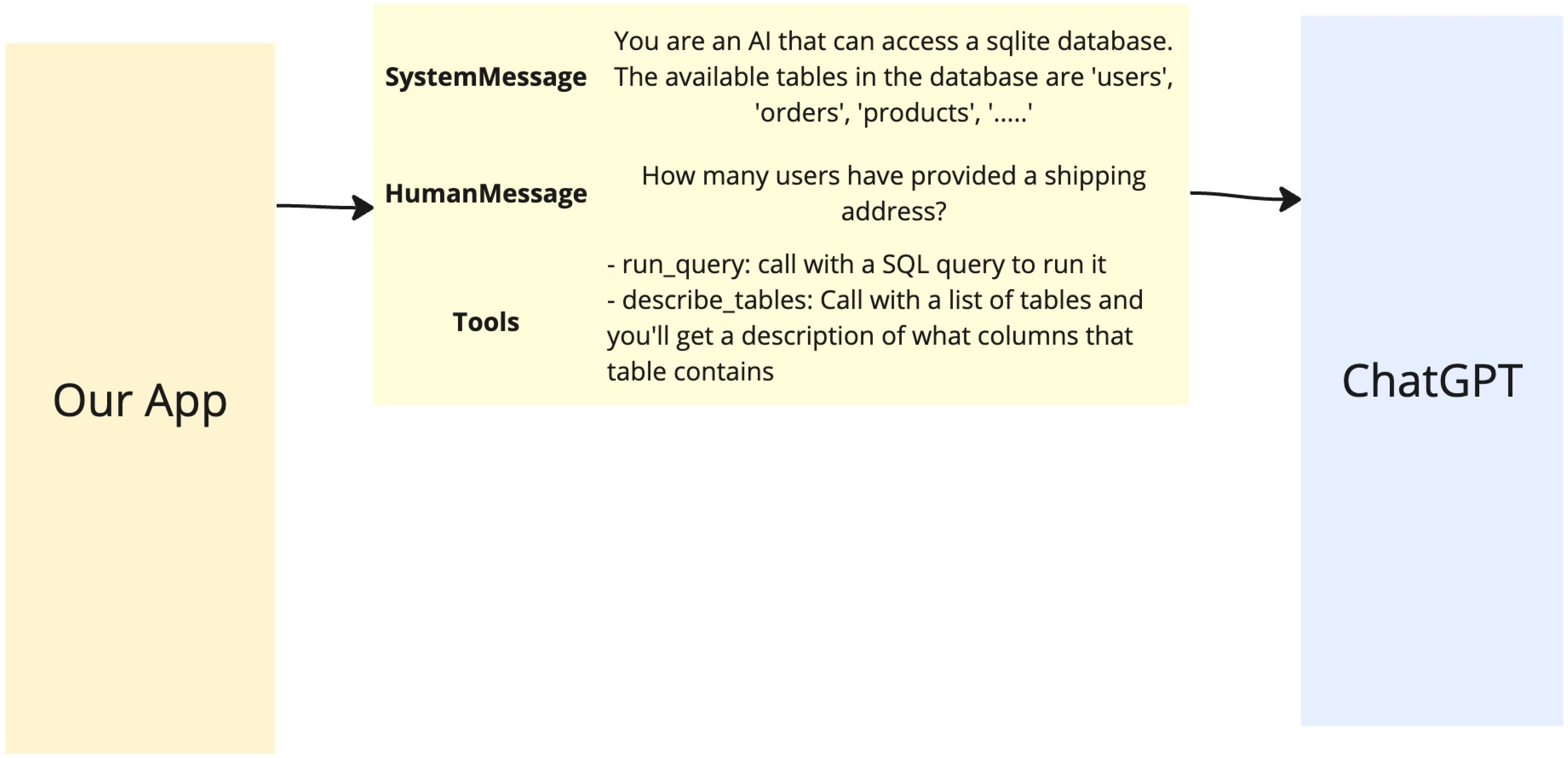
```
SELECT COUNT(*) FROM Users
WHERE shipping_address IS NOT
NULL
```

```
sqlite3.OperationalError:
no such column:
shipping_address
```

```
The following error occurred:

No such column: shipping_address
```

```
SELECT COUNT(*) FROM addresses;
```

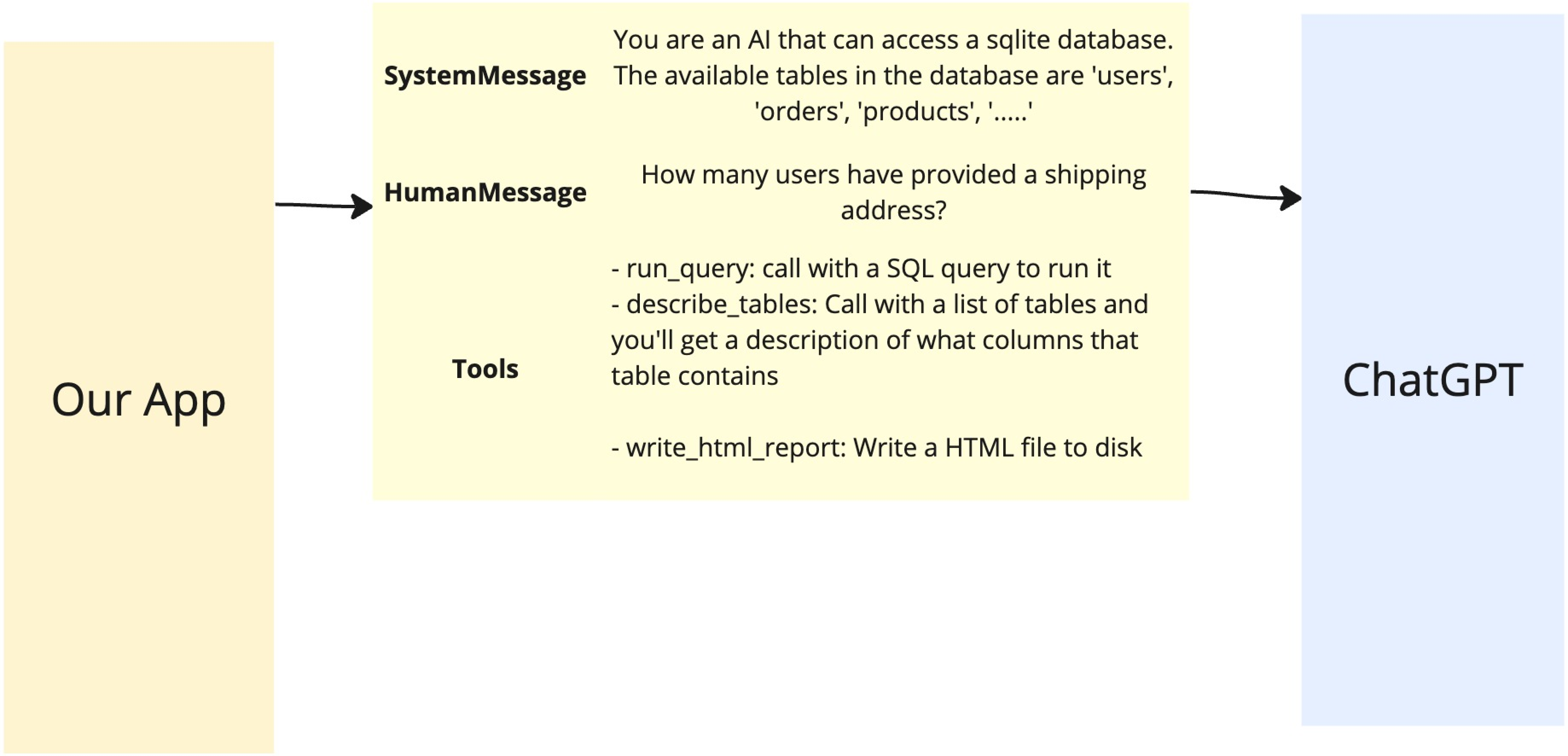
Different query that will (hopefully) work



You are an AI that can access a sqlite database. The available tables in the database are 'users', 'orders', 'products', '.....'

How many users have provided a shipping address?

- run_query: call with a SQL query to run it
- describe_tables: Call with a list of tables and you'll get a description of what columns that table contains

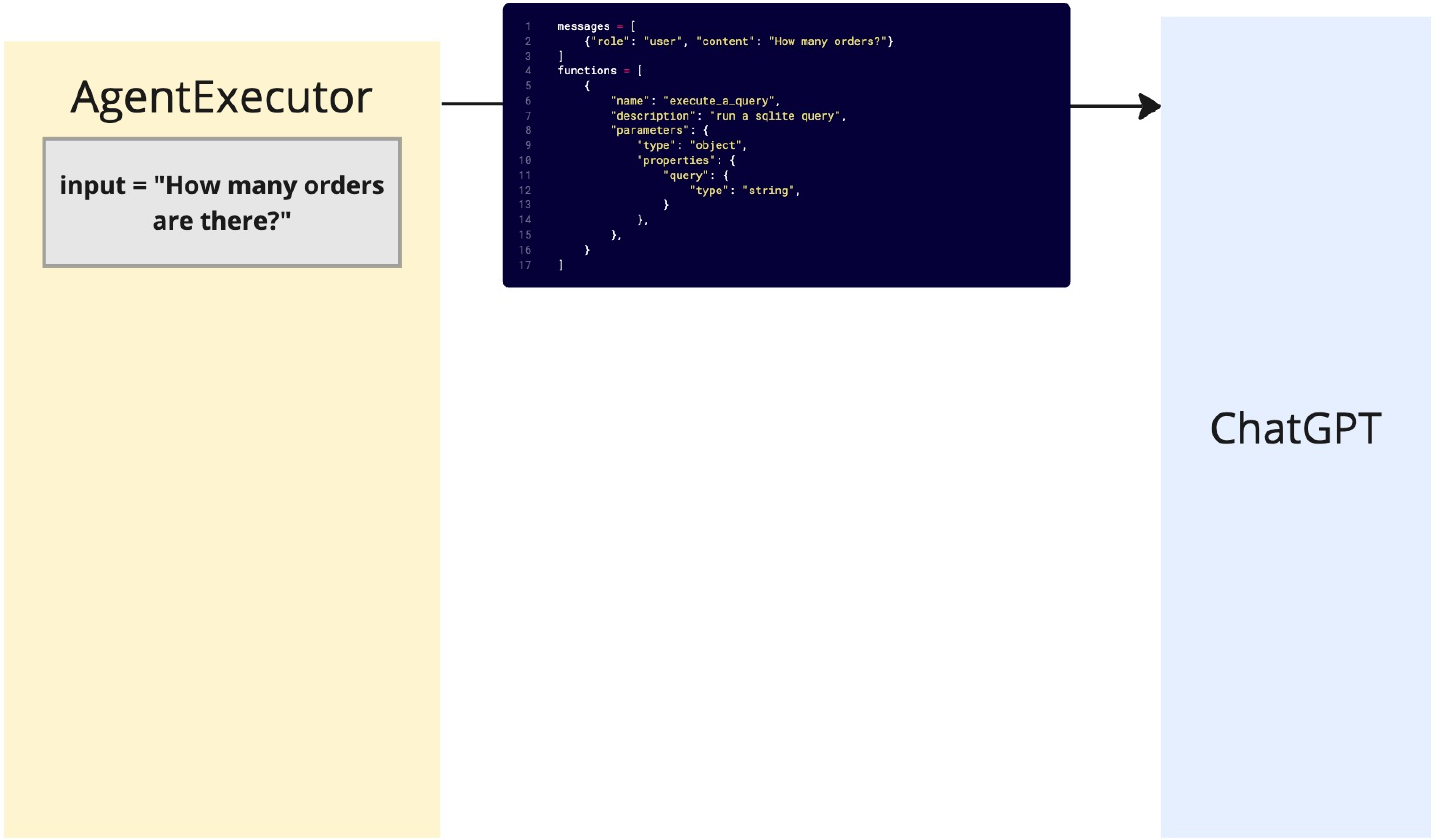


You are an AI that can access a sqlite database. The available tables in the database are 'users', 'orders', 'products', '.....'

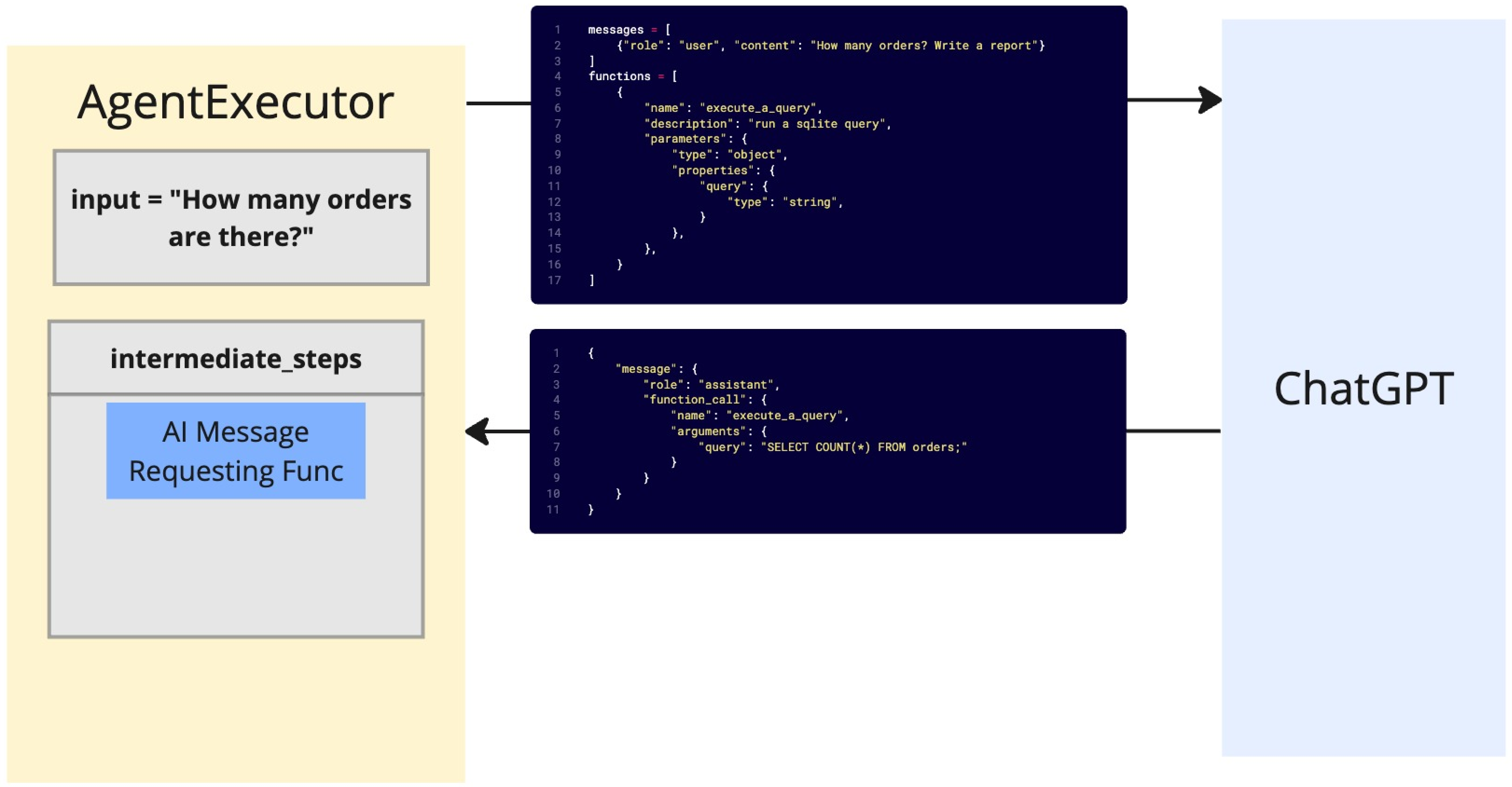
How many users have provided a shipping address?

- run_query: call with a SQL query to run it
- describe_tables: Call with a list of tables and you'll get a description of what columns that table contains

- write_html_report: Write a HTML file to disk

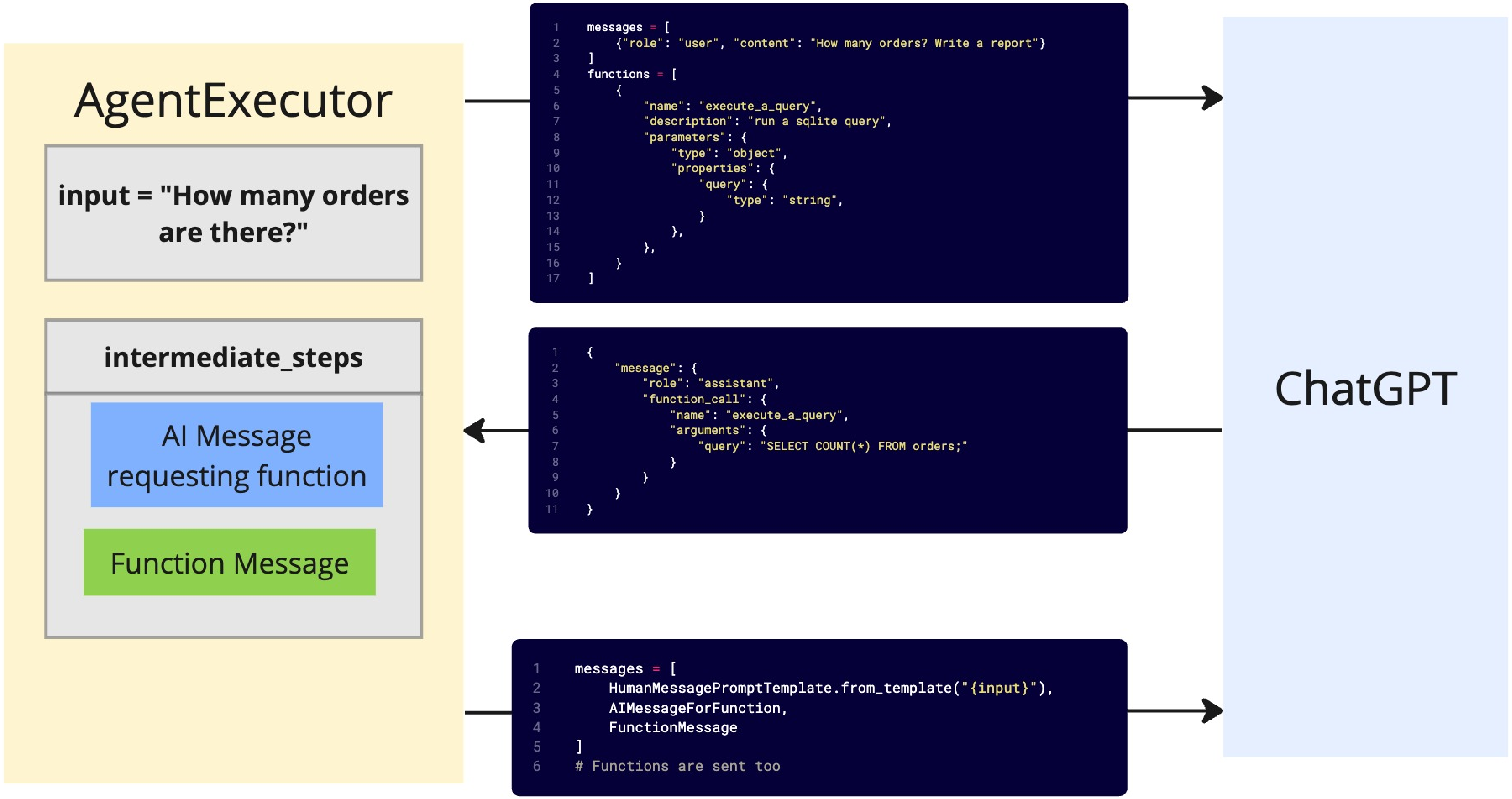


```
messages = [
    {"role": "user", "content": "How many orders?"}
]
functions = [
    {
        "name": "execute_a_query",
        "description": "run a sqlite query",
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "string",
                }
            },
        },
    }
]
```


AgentExecutor
input = "How many orders are there?"
intermediate_steps
AI Message Requesting Func
ChatGPT

```
messages = [
    {"role": "user", "content": "How many orders? Write a report"}
]
functions = [
    {
        "name": "execute_a_query",
        "description": "run a sqlite query",
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "string",
                }
            },
        },
    }
]
```

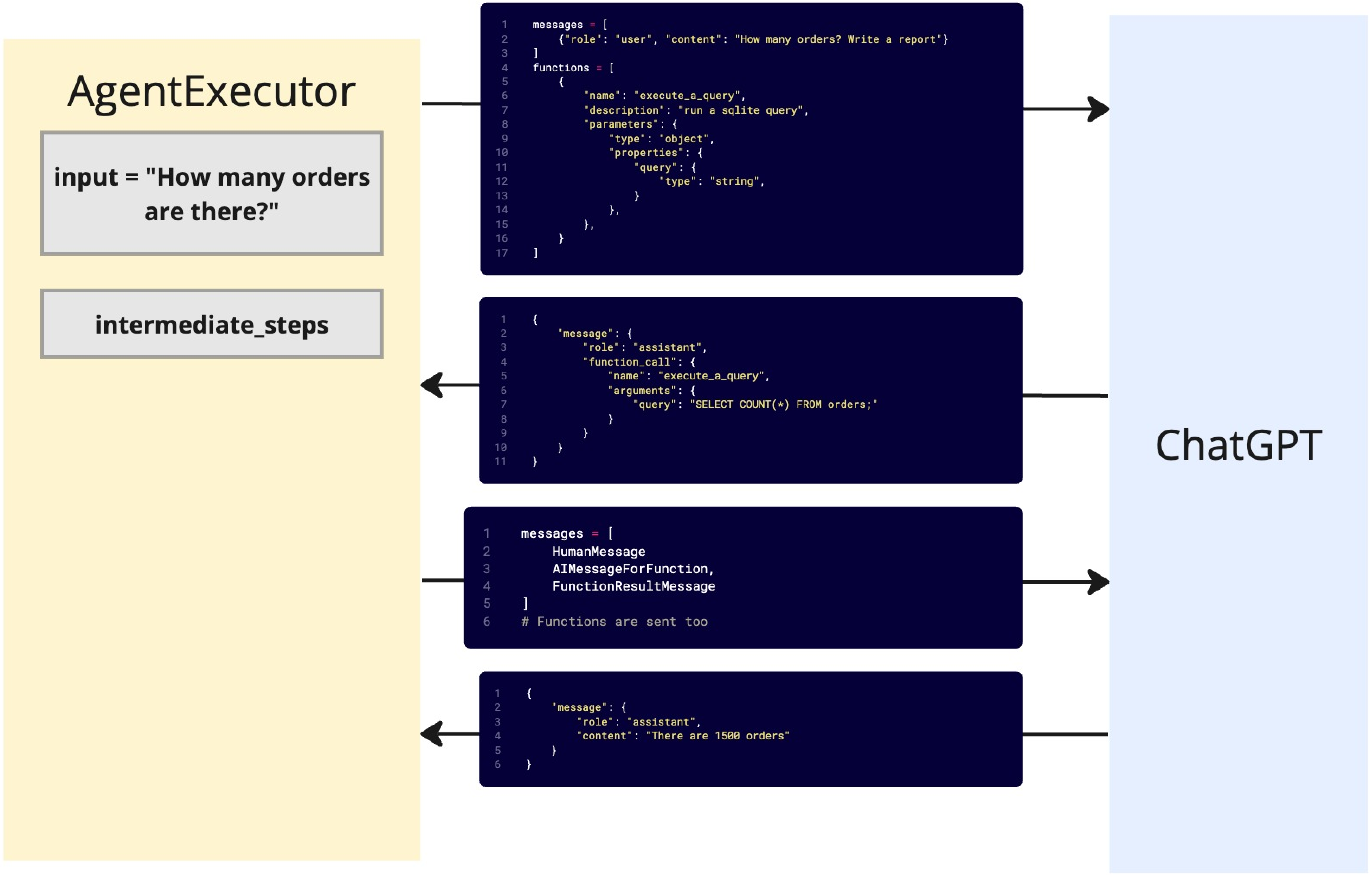
```
{
    "message": {
        "role": "assistant",
        "function_call": {
            "name": "execute_a_query",
            "arguments": {
                "query": "SELECT COUNT(*) FROM orders;"
            }
        }
    }
}
```


AgentExecutor
input = "How many orders are there?"
intermediate_steps
AI Message requesting function
Function Message
ChatGPT

```
messages = [
    {"role": "user", "content": "How many orders? Write a report"}
]
functions = [
    {
        "name": "execute_a_query",
        "description": "run a sqlite query",
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "string",
                }
            },
        },
    }
]
```

```
{
    "message": {
        "role": "assistant",
        "function_call": {
            "name": "execute_a_query",
            "arguments": {
                "query": "SELECT COUNT(*) FROM orders;"
            }
        }
    }
}
```

```
messages = [
    HumanMessagePromptTemplate.from_template("{input}"),
    AIMessageForFunction,
    FunctionMessage
]
# Functions are sent too
```

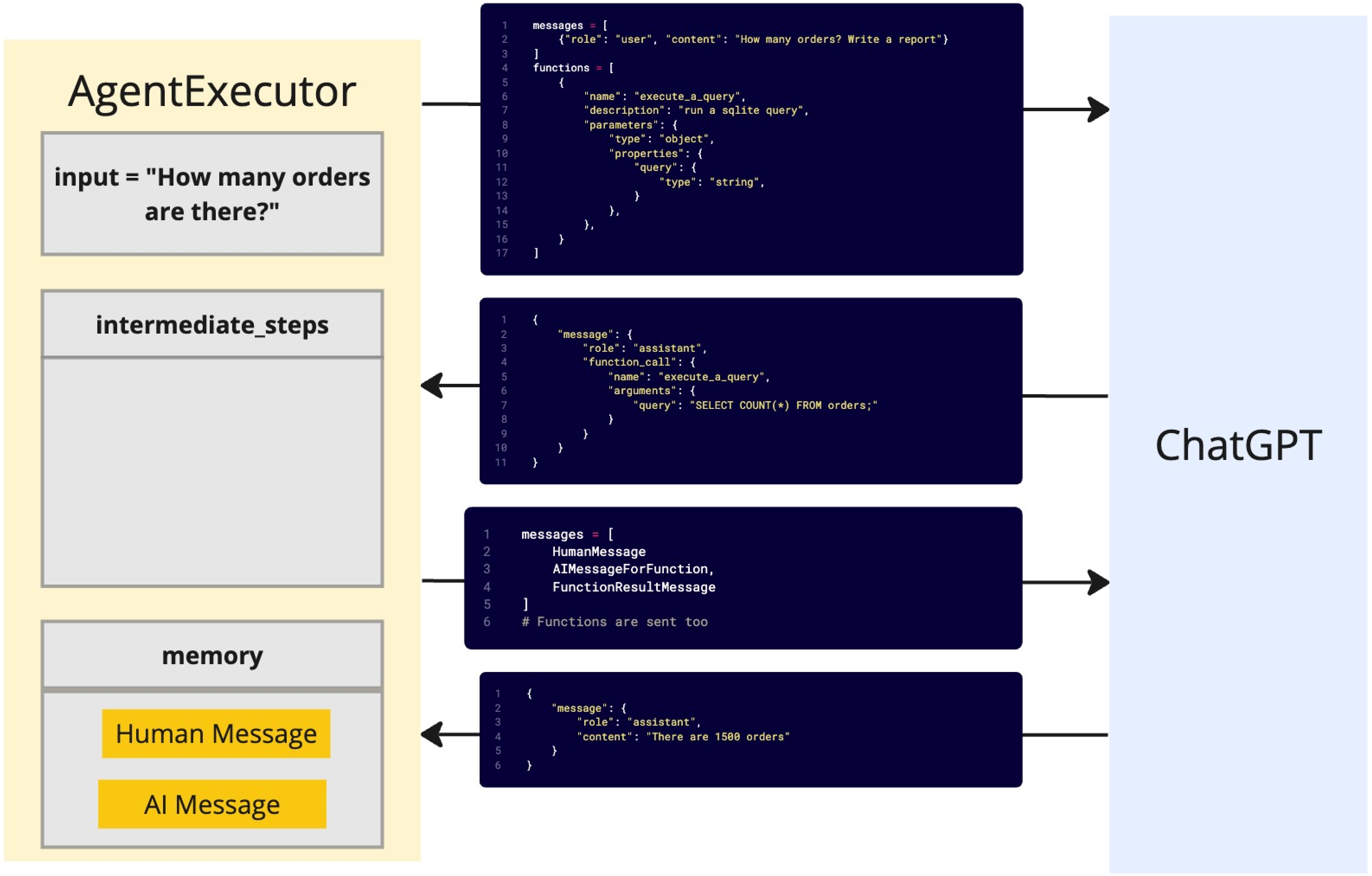


```
messages = [
    {"role": "user", "content": "How many orders? Write a report"}
]
functions = [
    {
        "name": "execute_a_query",
        "description": "run a sqlite query",
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "string",
                }
            },
        },
    }
]
```

```
{
    "message": {
        "role": "assistant",
        "function_call": {
            "name": "execute_a_query",
            "arguments": {
                "query": "SELECT COUNT(*) FROM orders;"
            }
        }
    }
}
```

```
messages = [
    HumanMessage
    AIMessageForFunction,
    FunctionResultMessage
]
# Functions are sent too
```

```
{
    "message": {
        "role": "assistant",
        "content": "There are 1500 orders"
    }
}
```

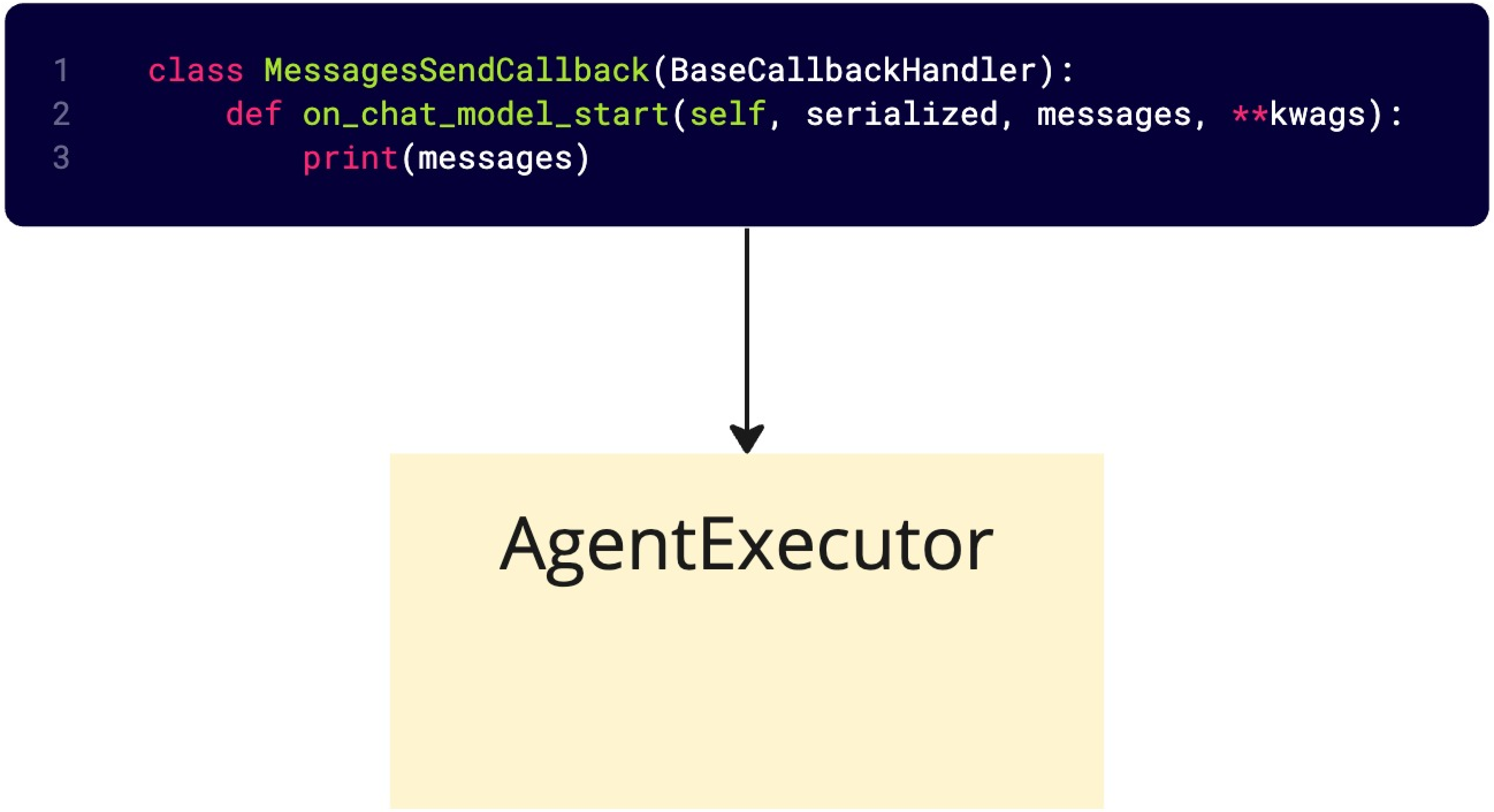


```
messages = [
    {"role": "user", "content": "How many orders? Write a report"}
]
functions = [
    {
        "name": "execute_a_query",
        "description": "run a sqlite query",
        "parameters": {
            "type": "object",
            "properties": {
                "query": {
                    "type": "string",
                }
            },
        },
    }
]
```

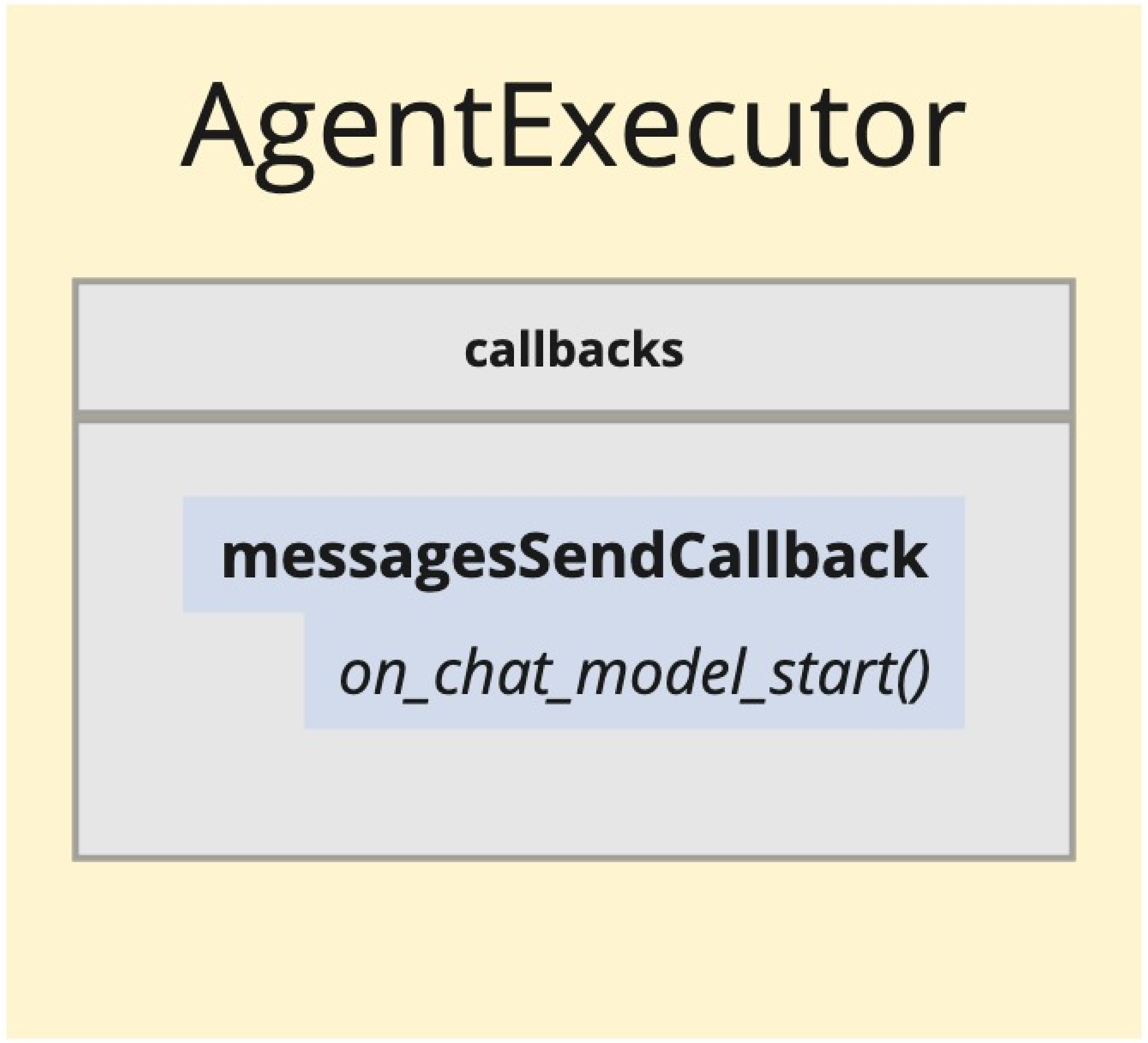
```
{
    "message": {
        "role": "assistant",
        "function_call": {
            "name": "execute_a_query",
            "arguments": {
                "query": "SELECT COUNT(*) FROM orders;"
            }
        }
    }
}
```

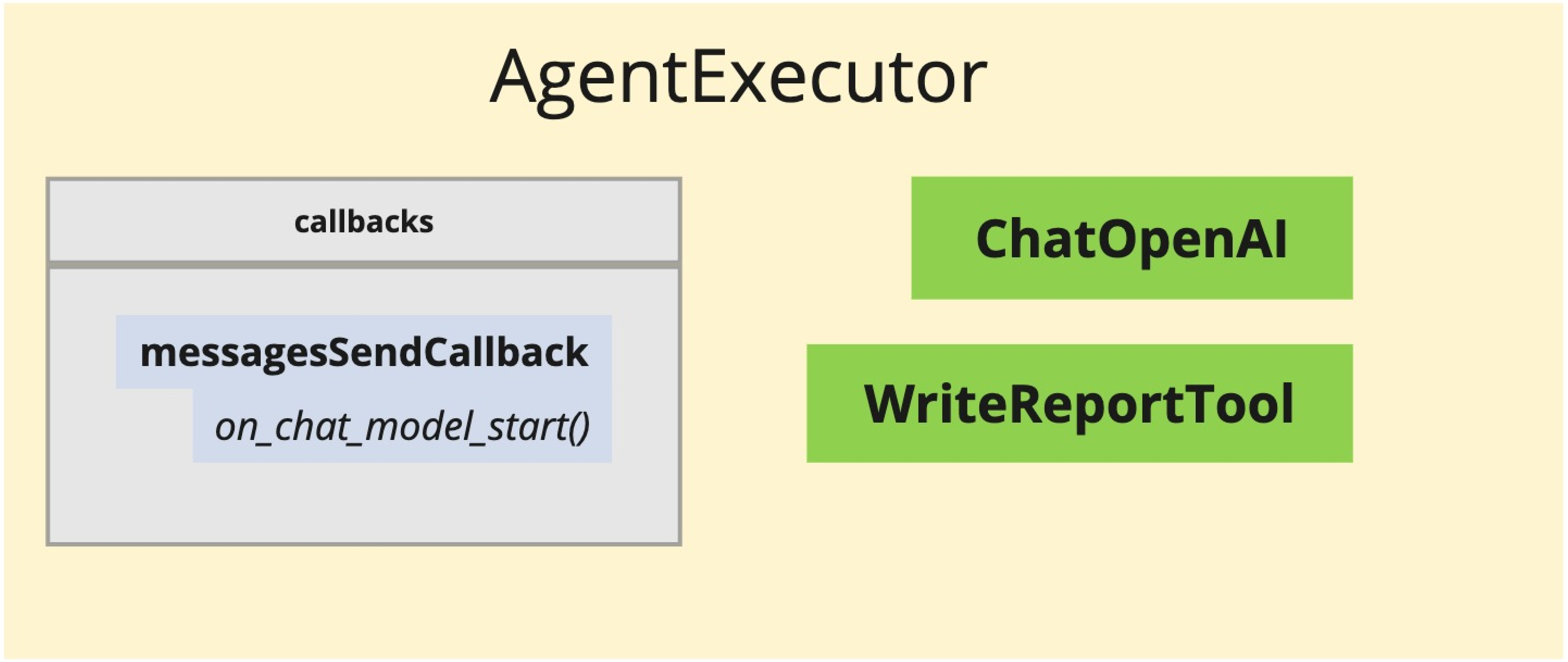
```
messages = [
    HumanMessage
    AIMessageForFunction,
    FunctionResultMessage
]
# Functions are sent too
```

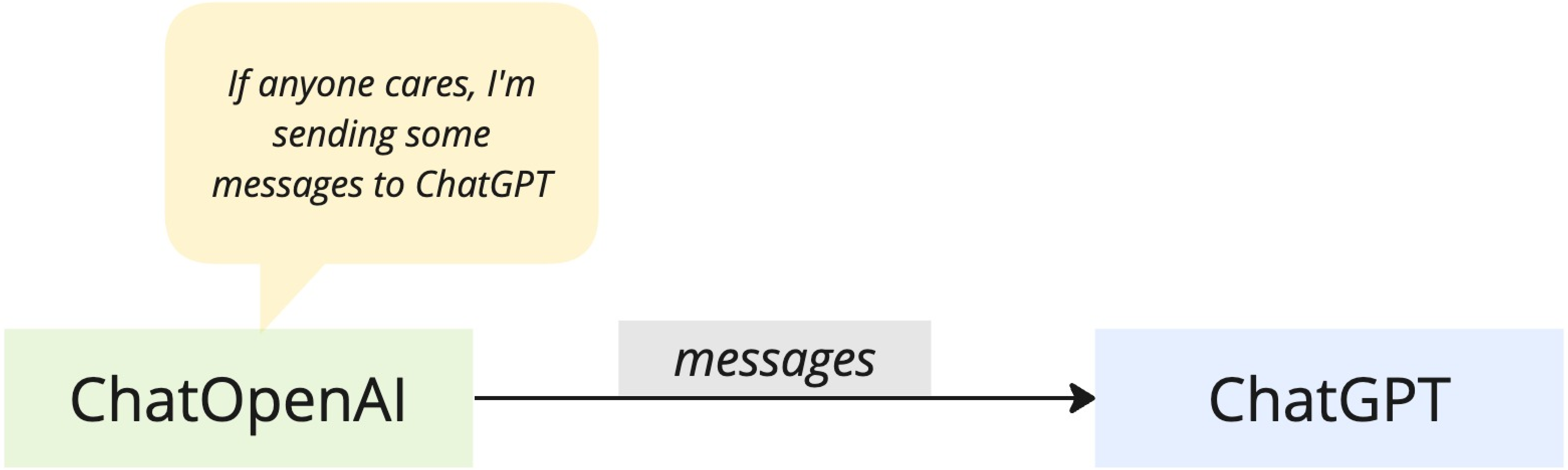
```
{
    "message": {
        "role": "assistant",
        "content": "There are 1500 orders"
    }
}
```

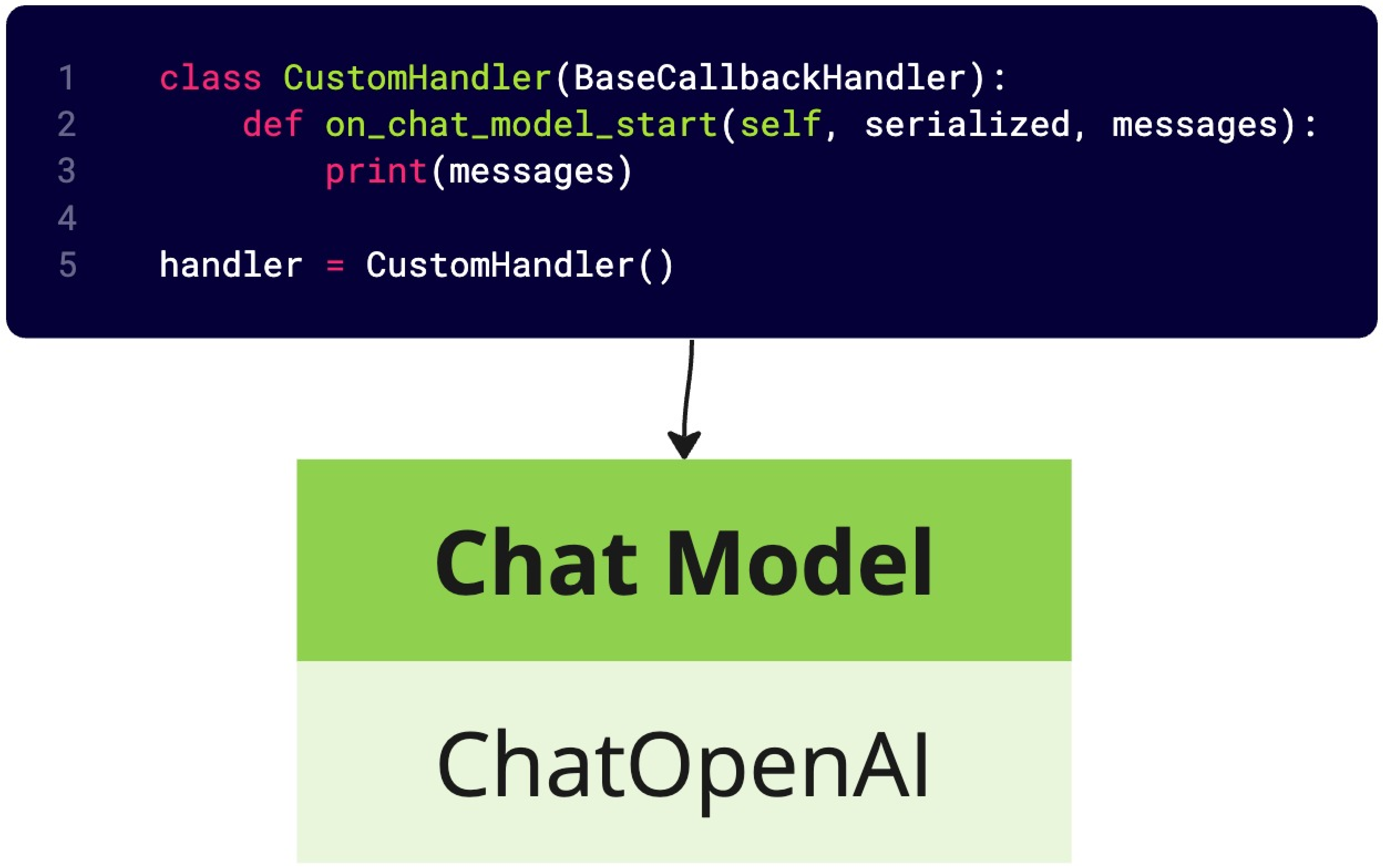


```python
class MessagesSendCallback(BaseCallbackHandler):
    def on_chat_model_start(self, serialized, messages, **kwags):
        print(messages)
```


AgentExecutor
callbacks
messagesSendCallback
on_chat_model_start()


AgentExecutor
callbacks
messagesSendCallback
on_chat_model_start()
ChatOpenAI
WriteReportTool


If anyone cares, I'm sending some messages to ChatGPT
ChatOpenAI
messages
ChatGPT



```python
class CustomHandler(BaseCallbackHandler):
    def on_chat_model_start(self, serialized, messages):
        print(messages)

handler = CustomHandler()
```

New table

Object	Possible Event	Possible Event	Possible Event	Possible Event
LLM's	**on_llm_start()** *Called when LLM starts running*	**on_llm_new_token()** *Called when the model receives a token in stream mode*	**on_llm_end()** *Called when model is done*	**on_llm_error()** *Called when an error occurs*
Chat Models	**on_chat_model_start()** *Called when chat model starts running*	**on_llm_new_token()** *Called when the model receives a token in stream mode*	**on_llm_end()** *Called when model is done*	**on_llm_error()** *Called when an error occurs*
Tools	**on_tool_start()** *Called when the tool starts*	**on_tool_end()** *Called when tool is done*	**on_tool_error()** *Called when tool errors*	
Chains	**on_chain_start()** *Called when chain starts*	**on_chain_end()** *Called when chain is done*	**on_chain_error()** *Called when chain has an error*	
Agents	**on_agent_action()** *Called when the agent receives a message*	**on_agent_finish()** *Called when the agent is done*		