AI VIET NAM
@aivietnam.edu.vn

AI Agents with LLMs: Using Tool Callings

AI Agent Series

Nguyen-Thuan Duong – TA

Yen-Linh Vu – STA

Anh-Khoi Nguyen– STA

Agency Level

Agency Level	Description	How that's called	Example
–	LLM output has no impact on program flow	Simple processor	process_output(llm_response)
⋆	LLM output determines basic control flow	Router	if llm_decision(): path_a() else: path_b()
⋆⋆	LLM output determines function execution	Tool call	run_function(llm_chosen_tool, llm_chosen_args)
⋆⋆⋆	LLM output controls iteration and program continuation	Multi-step Agent	while llm_should_continue(): execute_next_step()
⋆⋆⋆	One agentic workflow can start another agentic workflow	Multi-Agent	if llm_trigger(): execute_agent()

Ref: https://huggingface.co/blog/smolagents

Introduction

❖ Overview of AI Agent Series

Day 01: AI Agent Libraries	Day 02: LLMs Tools Calling	Day 03: ReAct AI Agent
• High-level understanding of AI Agent. • Investigate how to build AI Agent applications through popular libraries.	• First low-level understading of AI Agent. • Investigate the simplest and early form of AI Agent: Make LLMs using tools to solve tasks.	• Second low-level understanding of AI Agent. • Investigate the general baseline of current AI Agent: Though-Action-Observe through ReAct AI Agent.

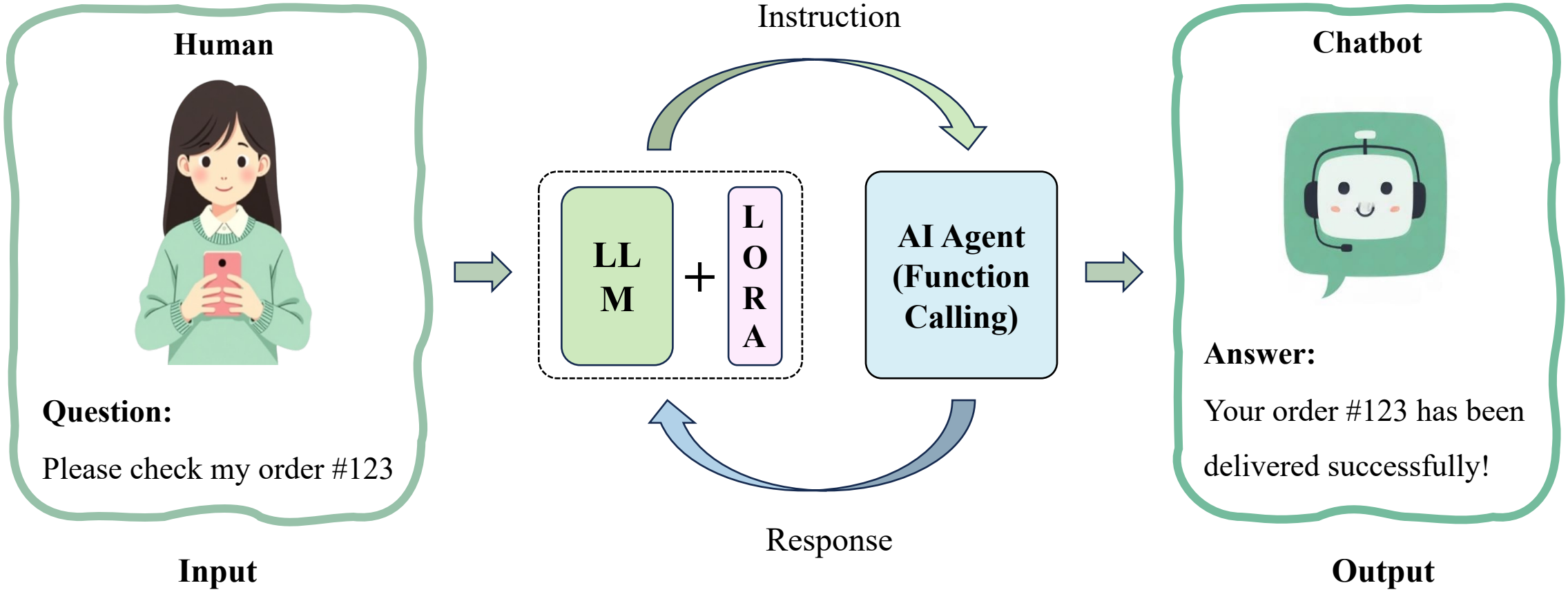
Outline

- Introduction
- Tool Calling in LLMs
- Demo
- Question

Introduction

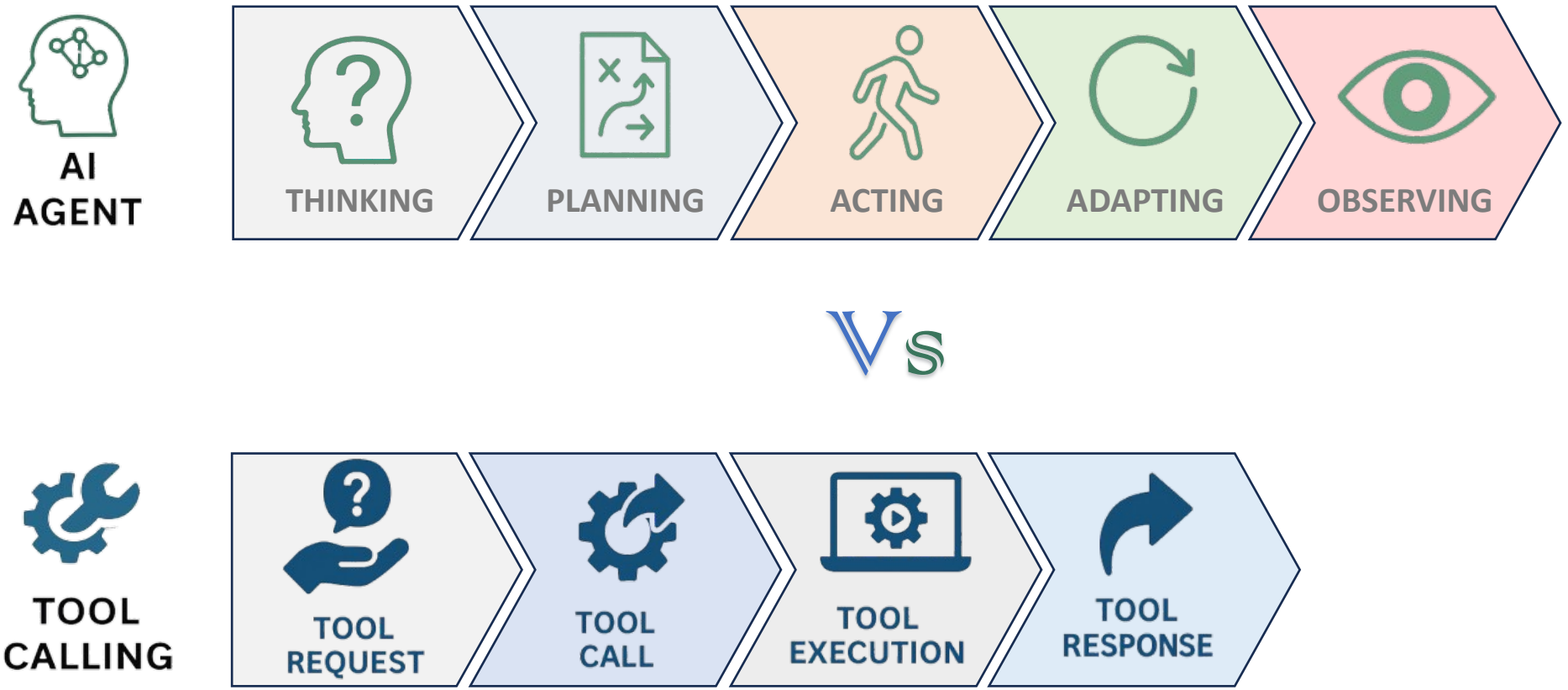
Introduction

❖ Function/Tool Calling Chatbot



Introduction

❖ Agent & Tool Calling



Introduction

❖ Tool Calling Timeline

HuggingGPT 03/2023

LLM lần đầu đóng vai "bộ não" điều phối: phân tích yêu cầu, lập kế hoạch, và kích hoạt các mô hình AI chuyên sâu (text, image, speech...) từ Hugging Face. Đây là nền tảng cho việc tích hợp công cụ (tool use) vào LLM.

1

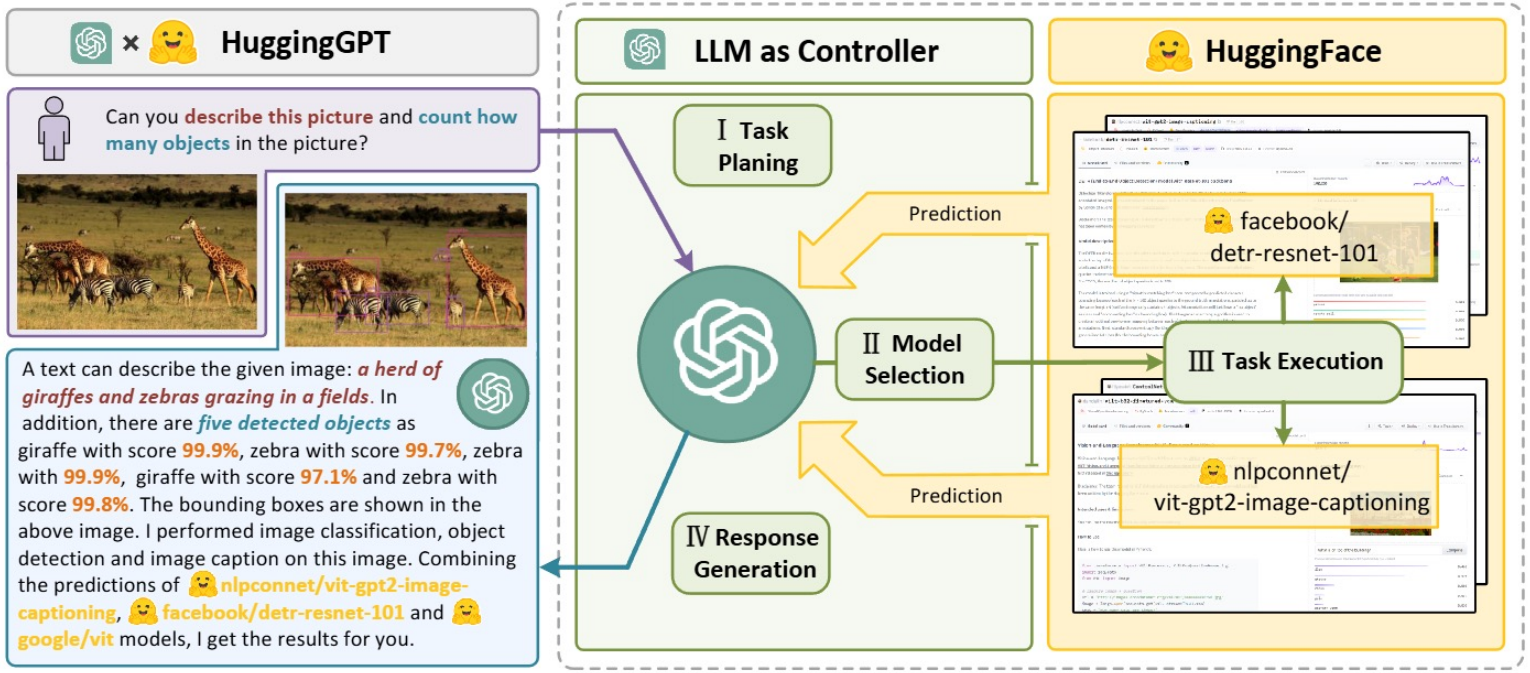


Figure 1: *Language serves as an interface for LLMs (e.g., ChatGPT) to connect numerous AI models (e.g., those in Hugging Face) for solving complicated AI tasks.* In this concept, an LLM acts as a controller, managing and organizing the cooperation of expert models. The LLM first plans a list of tasks based on the user request and then assigns expert models to each task. After the experts execute the tasks, the LLM collects the results and responds to the user.

https://arxiv.org/pdf/2303.17580

Introduction

❖ Tool Calling Timeline

GPT4Tools
LLMs as Tool Makers 05/2023

Phát triển từ ý tưởng trên, GPT-4 không chỉ gọi công cụ (tool calling) mà còn tự động tạo lệnh (function generation) dựa trên mô tả. Bước nhảy vọt này giúp LLM chuyển từ thao tác thủ công sang tự học cách ứng dụng công cụ.

2

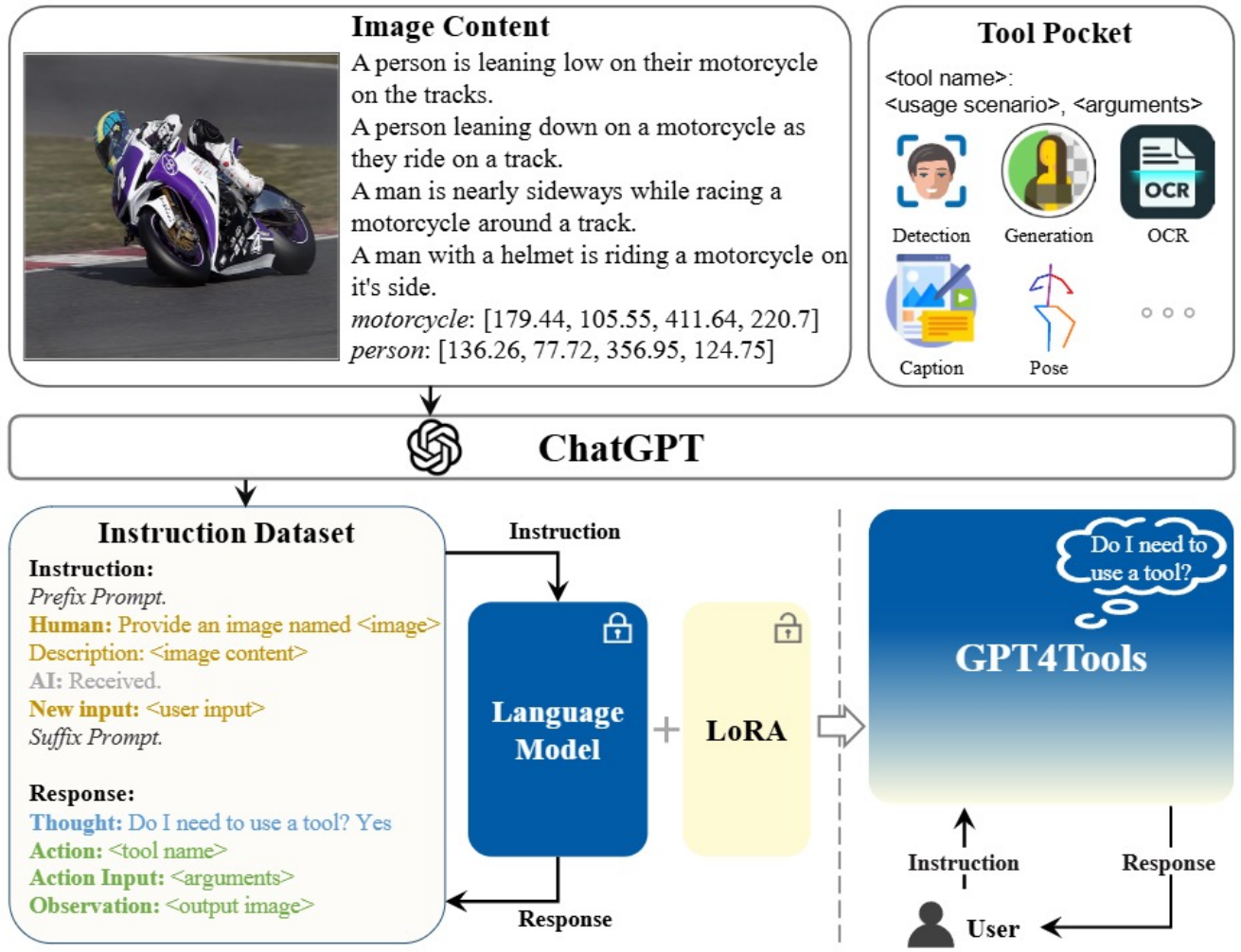


Figure 1: Diagram of the GPT4Tools. We prompt the ChatGPT with image content and definition of tools in order to obtain a tool-related instruction dataset. Subsequently, we employ LoRA [38] to train an open source LLM on the collected instruction dataset, thus adapting the LLM to use tools.

https://arxiv.org/pdf/2305.18752

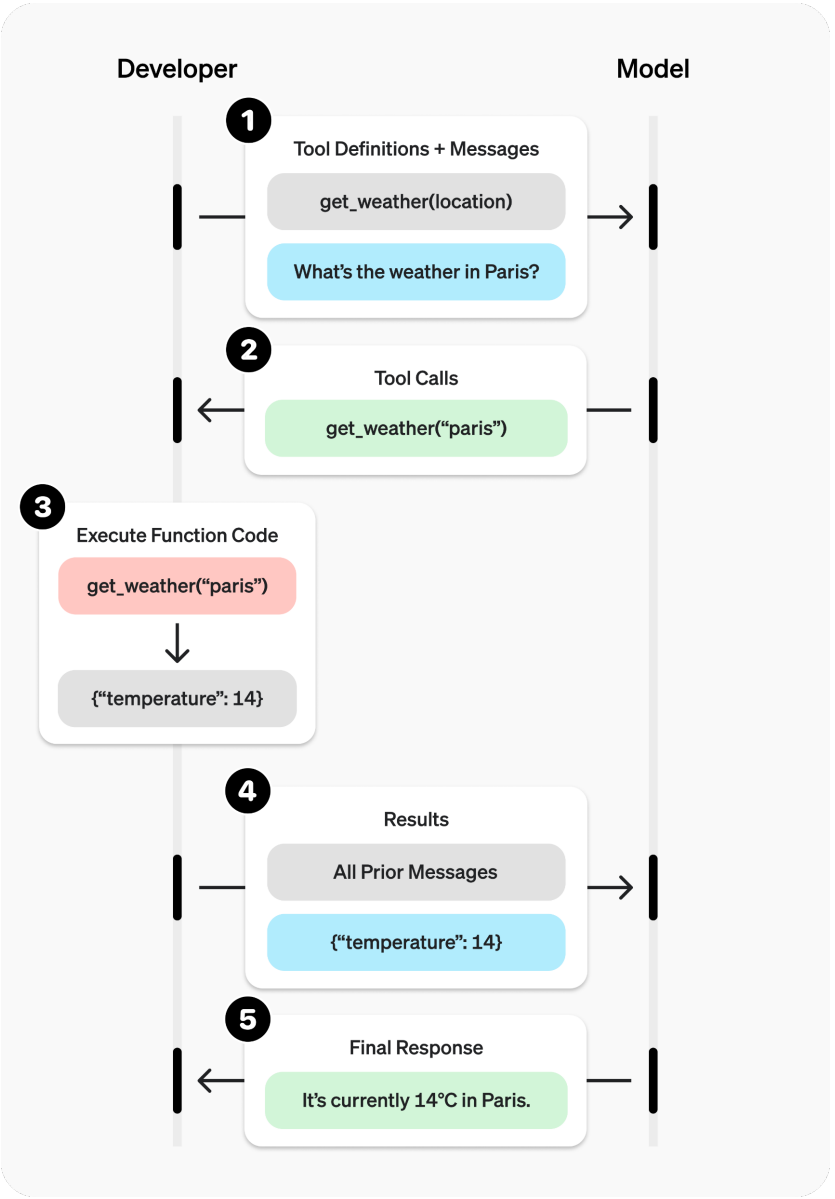
Introduction

❖ Tool Calling Timeline

OpenAI
Function Calling API 06/2023

Kế thừa xu hướng, OpenAI ra mắt API cho phép LLM sinh lệnh gọi hàm (function calls) dạng JSON. Công cụ này kết nối mô hình với hệ thống backend, cơ sở dữ liệu... một cách chính xác và an toàn, mở đường cho tích hợp đa nền tảng.

3



https://platform.openai.com/docs/guides/function-calling

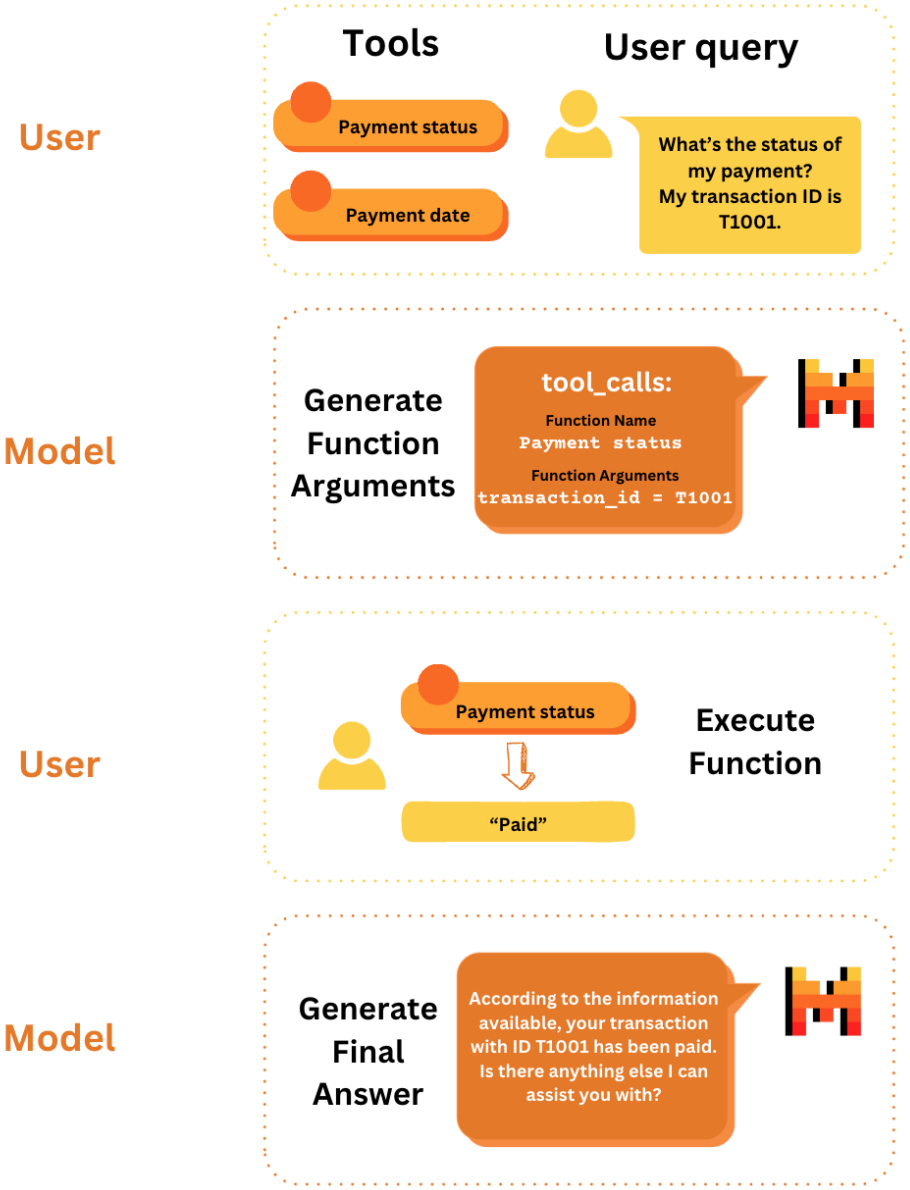
Introduction

❖ Tool Calling Timeline

Function Calling in AI Agents 10/2023

Tiến thêm một bước, Function Calling trở thành phần cốt lõi trong vòng lặp agent (agent loop): ví dụ Mistral 7B thực hiện chuỗi suy nghĩ → chọn công cụ → thực thi → cải thiện (planning → acting → observing → adapting). Quy trình cho phép xử lý tác vụ đa bước (multi-step tasks) trong môi trường phản hồi.

4



https://docs.mistral.ai/capabilities/function_calling/

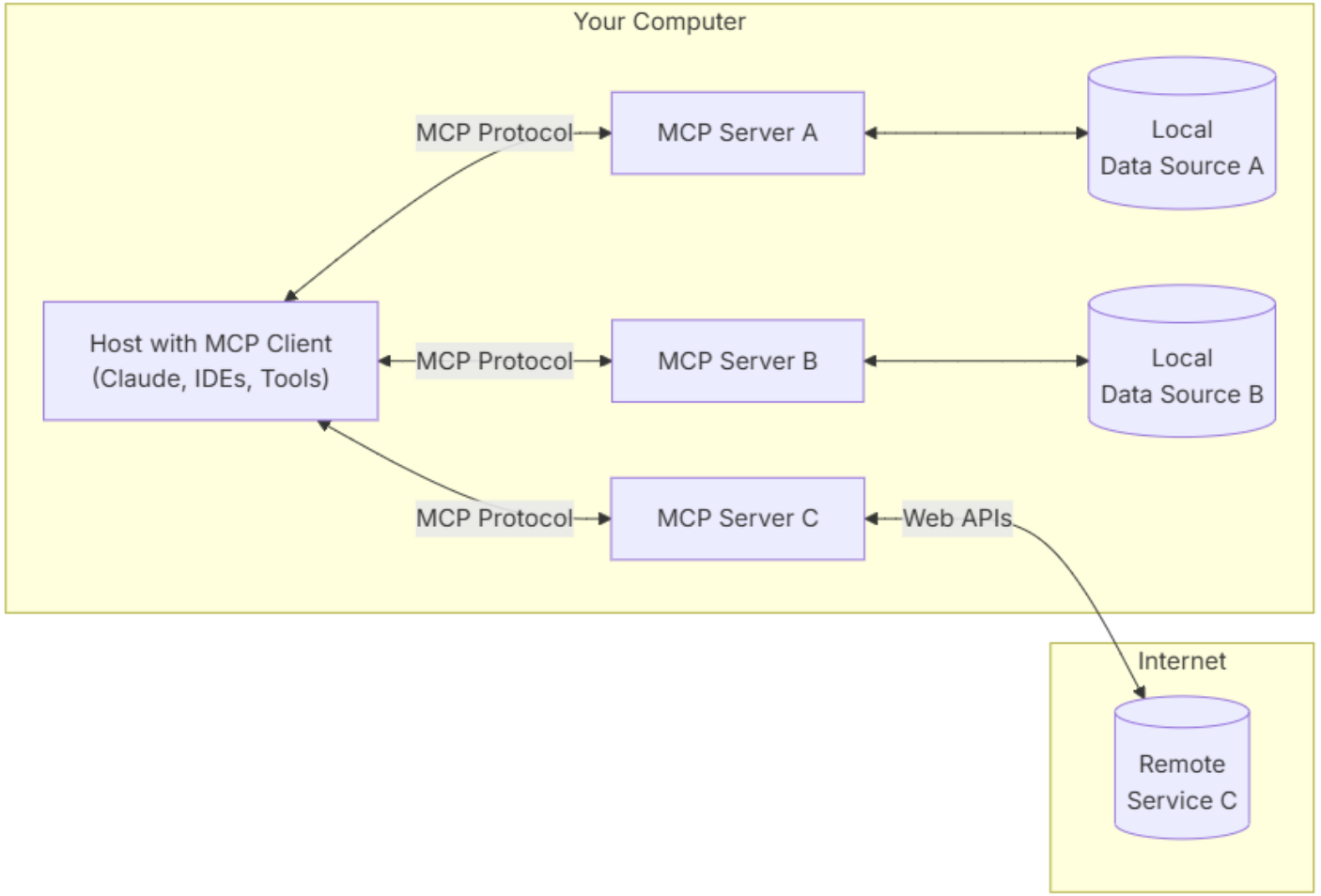
Introduction

❖ Tool Calling Timeline

Anthropic
Model Context Protocol **11/2024**

Đến cuối năm 2024, MCP cho phép nhiều LLM chia sẻ ngữ cảnh (context sharing) và phối hợp như một hệ đa agent (multi-agent). Đây là bước tiến lớn cho các hệ thống AI phức tạp và linh hoạt hơn.

5



https://modelcontextprotocol.io/introduction

Tool Calling in LLMs

Tool Calling

- LLM Router
- Built in tools
- Langchain ToolCalling
- ToolCalling Instruction FT
- ToolCalling Vision Task

LLM Router

LLM Router

❖ Getting Started

```python
def path_a():
    print("PATH A  → send a welcome e-mail.")

def path_b():
    print("PATH B  → log a support ticket.")
```

```python
messages = [
    {
        "role": "system",
        "content": (
            "You are a decision router. "
            "Reply with *exactly one* capital letter: **A** or **B**."
        ),
    },
    {
        "role": "user",
        "content": (
            "A customer wrote: “The product arrived damaged and I need help.”\n"
            "If this is a *simple* inquiry, answer **A**.\n"
            "If it is a *problem* requiring human assistance, answer **B**.\n"
        ),
    },
]
```

LLM Router

❖ Model

```python
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM


model_name = "meta-llama/Llama-3.2-1B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.bfloat16,
    device_map="auto"
)
```

```python
prompt = tokenizer.apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True,
)
```

LLM Router

❖ Decision

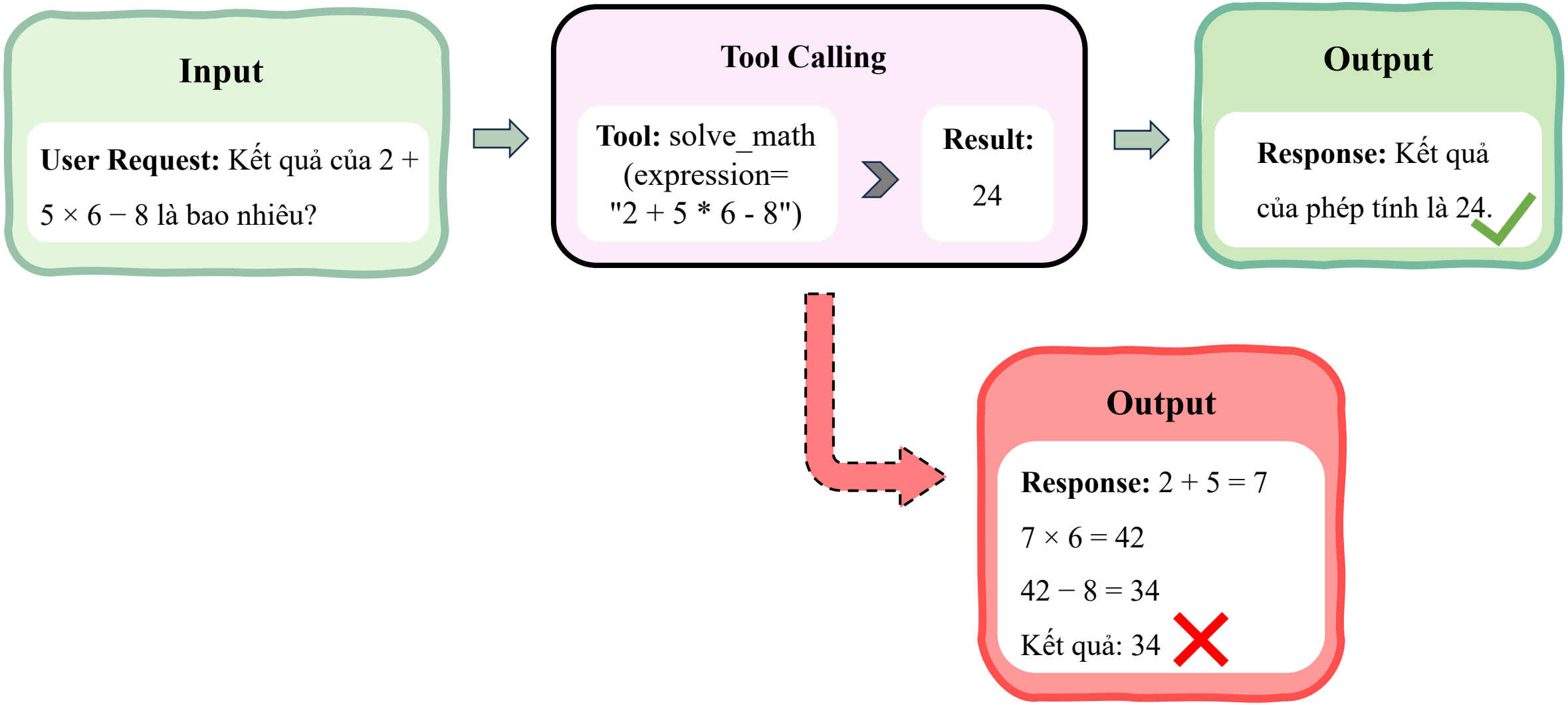
```python
inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

with torch.no_grad():
    output_ids = model.generate(
        **inputs,
        max_new_tokens=128,
        pad_token_id=tokenizer.eos_token_id,
    )

output = tokenizer.decode(output_ids[0], skip_special_tokens=True).strip()
print(f"Decision: {output}")
```

```
[7]
    decision = output.split()[-1]
    print(f"Model decision → {decision}")

...  Model decision → A
```

Build in Tools

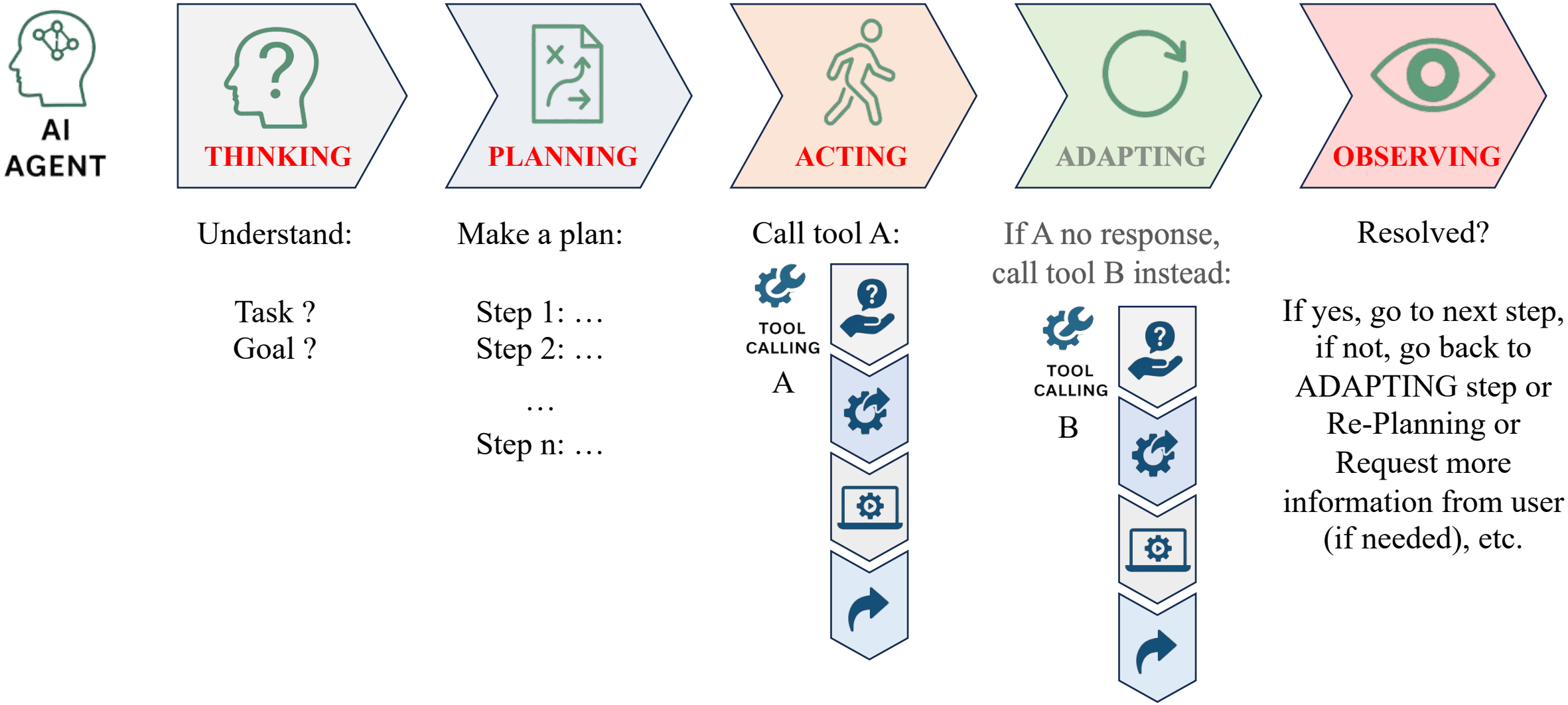
Tool Calling in LLMs

❖ Tool Calling



Tool Calling in LLMs

❖ Tool Calling Workflow



Tool Calling in LLMs

❖ Tool Calling in Llama3

Model calls a tool

```
{
  "name": "get_current_conditions",
  "parameters": {
    "location": "San Francisco, CA",
    "unit": "Fahrenheit“
  }
}
```

Function Calling (JSON Schema)

```
{
  "type": "function",
  "function": {
  "name": "get_current_conditions",
  "description": "Get the current weather conditions for a specific location",
  "parameters": {
                …
  }
}
```

Model calls a tool

```
<function=spotify_trending_songs>{"n":"5"}</function><eom_id>
```

Function Calling (JSON Schema)

```
{
  "type": "function",
  "function": {
  "name": "get_current_conditions",
  "description": "Get the current weather conditions for a specific location",
  "parameters": {
                …
  }
}
If a you choose to call a function ONLY reply in the following format:
<{start_tag}={function_name}>{parameters}{end_tag}
where
start_tag => `<function`
parameters => a JSON dict with the function argument name as key and function argument value as value.
end_tag => `</function>`
```

JSON based tool calling

WolframAlpha

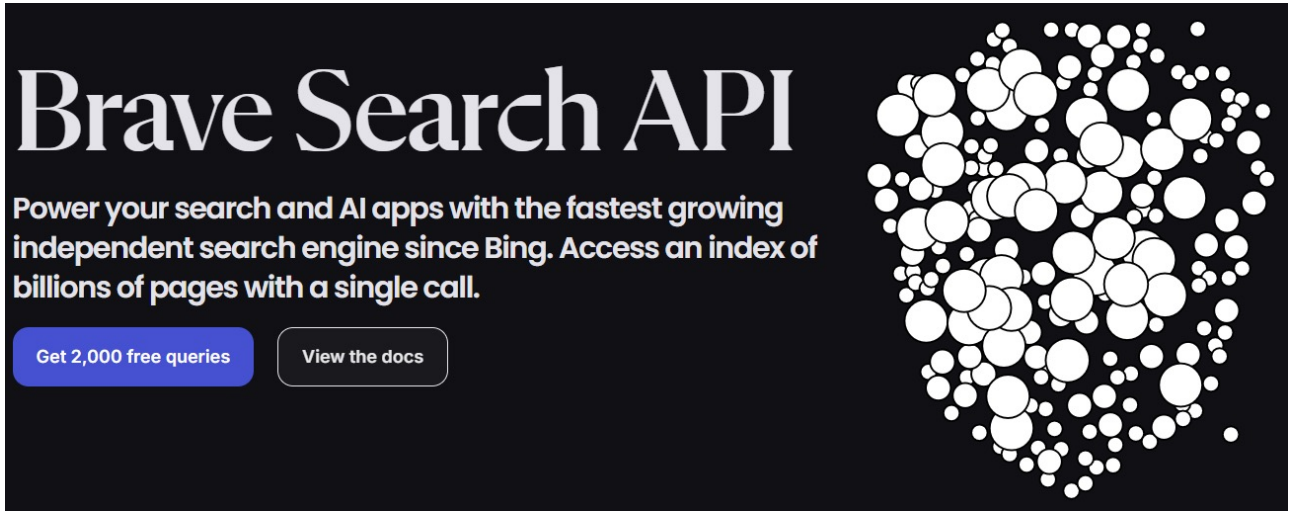
brave

Built-in

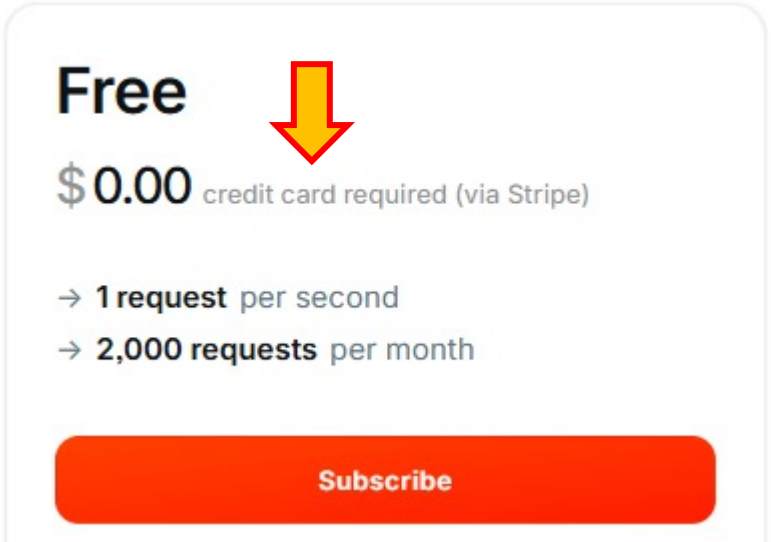
User-defined Custom

Tool Calling in LLMs

❖ Brave Tool in Llama3

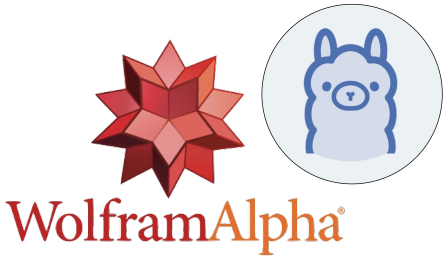


https://brave.com/search/api/

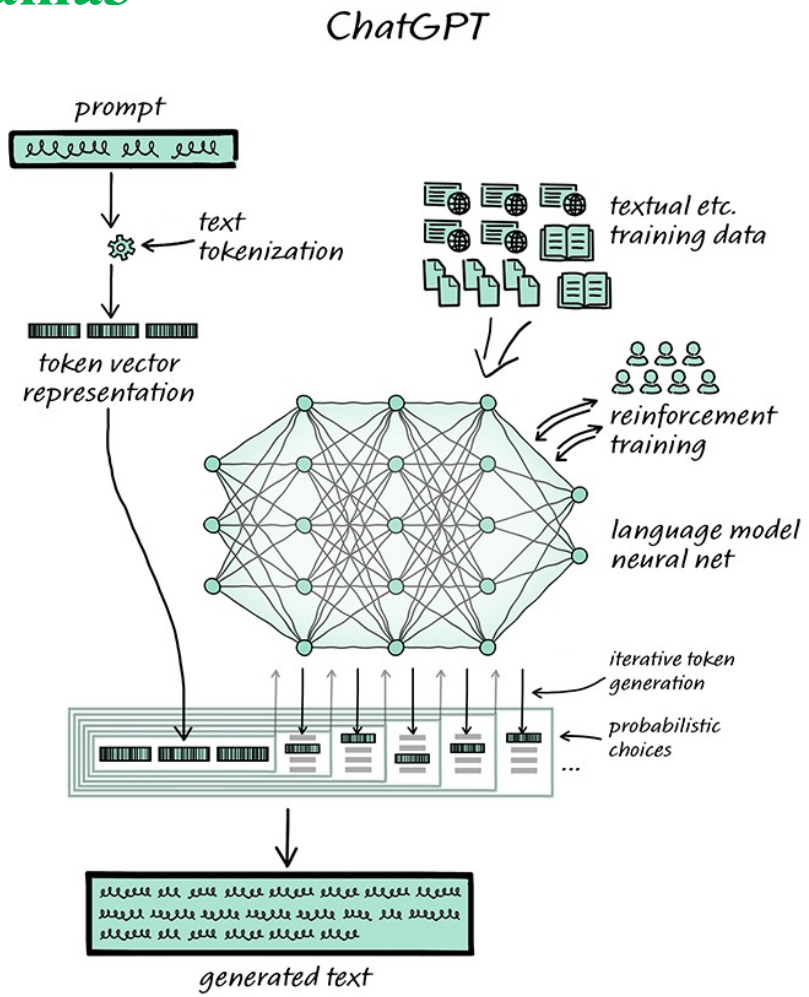


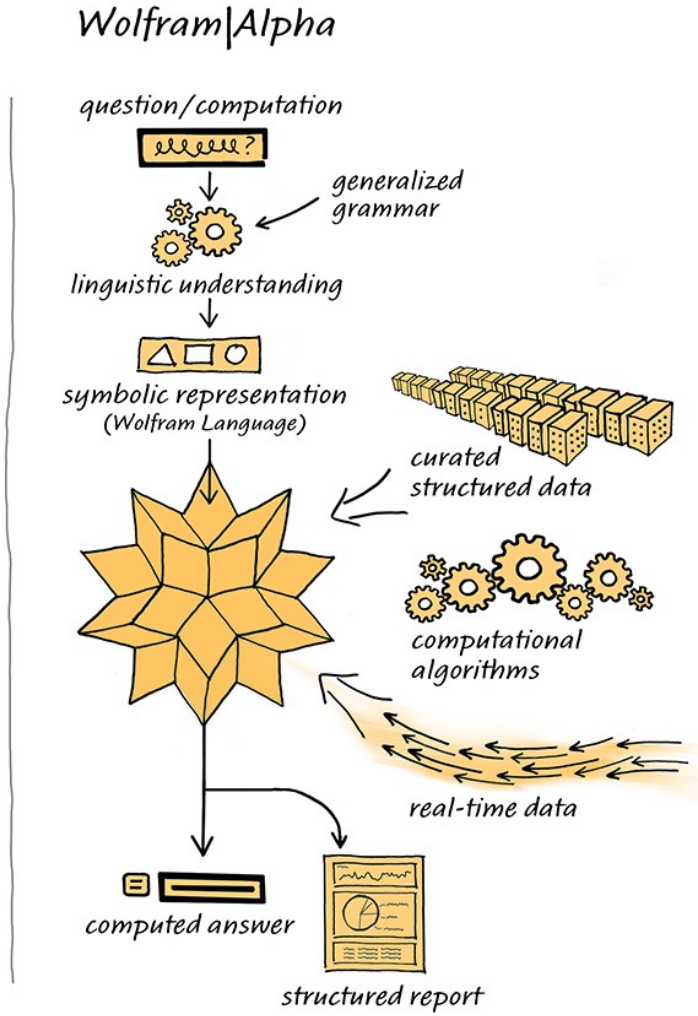
Tool Calling in LLMs

❖ WolframAlpha Tools in Llama3



https://www.wolframalpha.com/

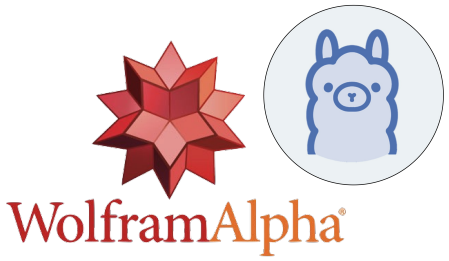




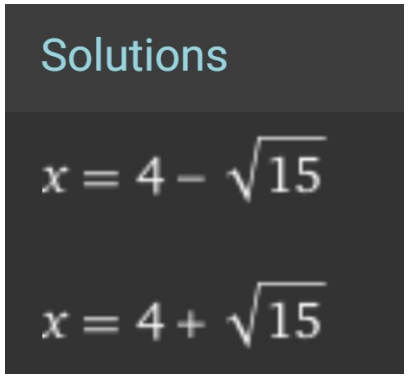
https://writings.stephenwolfram.com/2023/01/wolframalpha-as-the-way-to-bring-computational-knowledge-superpowers-to-chatgpt/

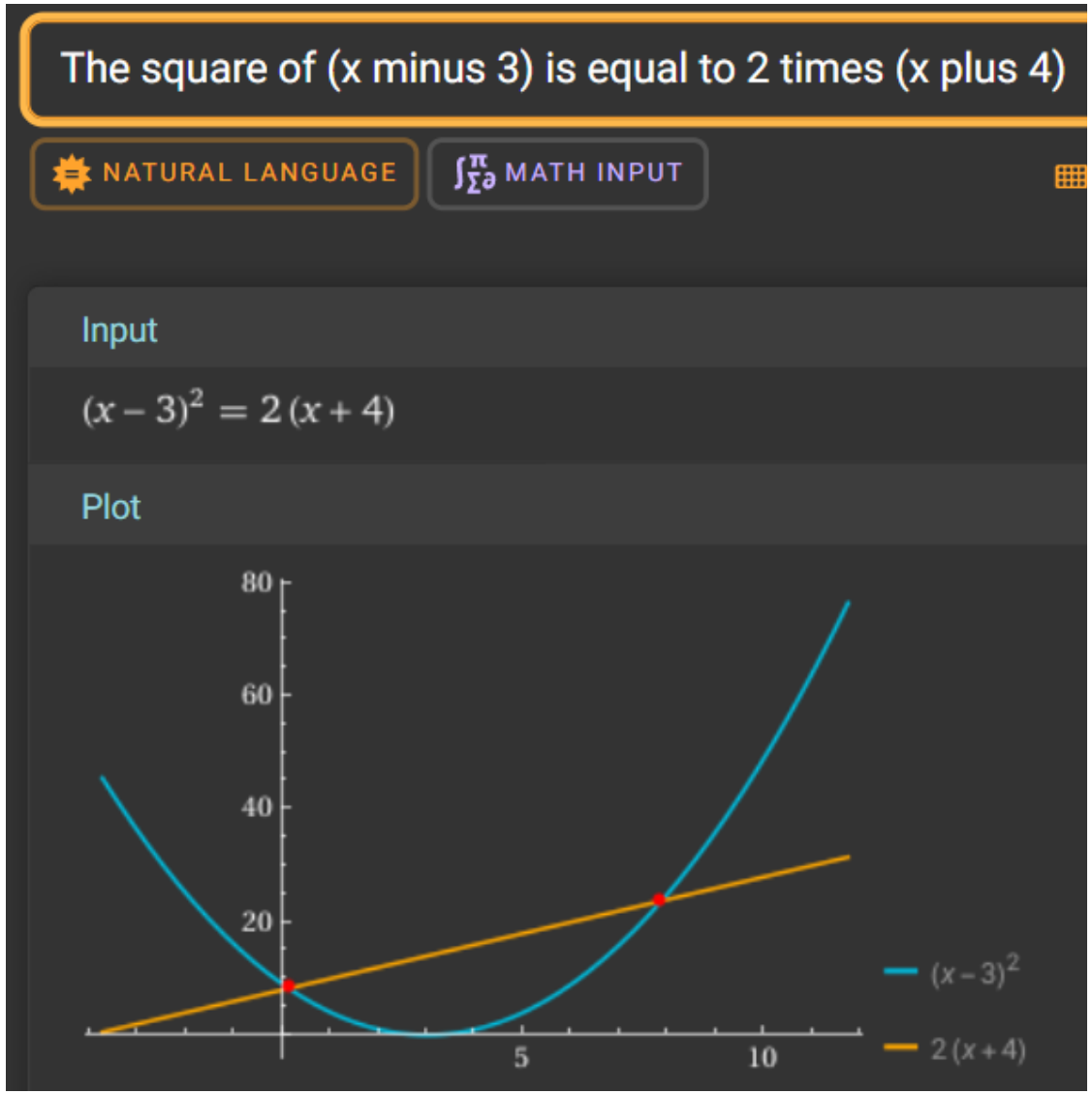
Tool Calling in LLMs

❖ WolframAlpha Tools in Llama3



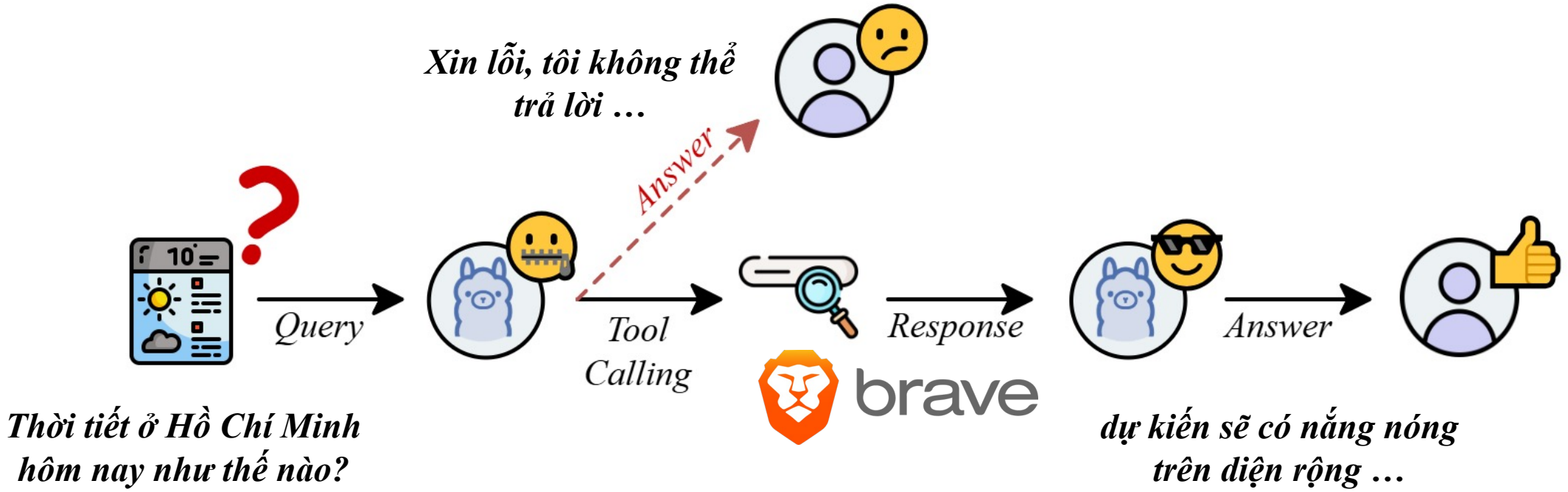
https://www.wolframalpha.com/





Tool Calling in LLMs

❖ Using Brave Tools in Llama3

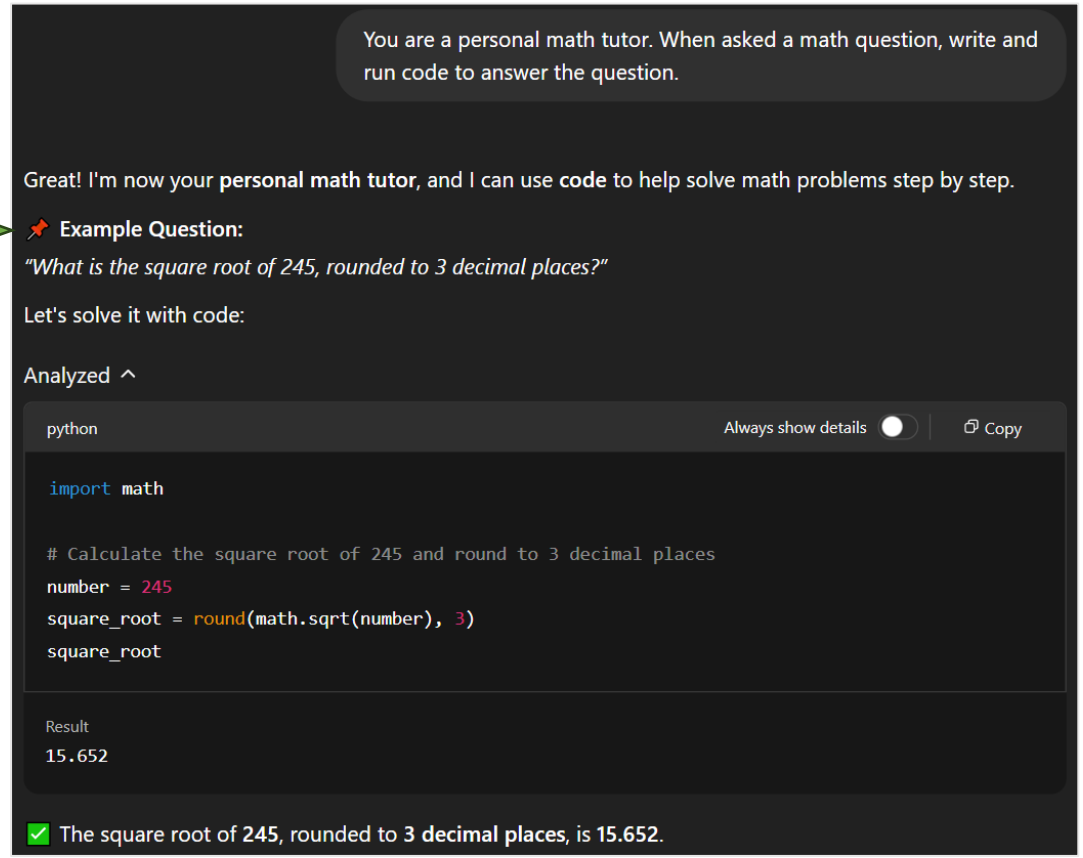


Tool Calling in LLMs

❖ Llama3 System Prompt for Tool Calling

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Environment: ipython
Tools: brave_search, wolfram_alpha
Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

You are a helpful assistant<|eot_id|>
```

WolframAlpha brave



Tool Calling in LLMs

❖ Illustration of calling tool

Input

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Environment: ipython
Tools: brave_search, wolfram_alpha
Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

You are a helpful assistant<|eot_id|>
<|start_header_id|>user<|end_header_id|>

What is the current weather in Menlo Park, California?<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

Output

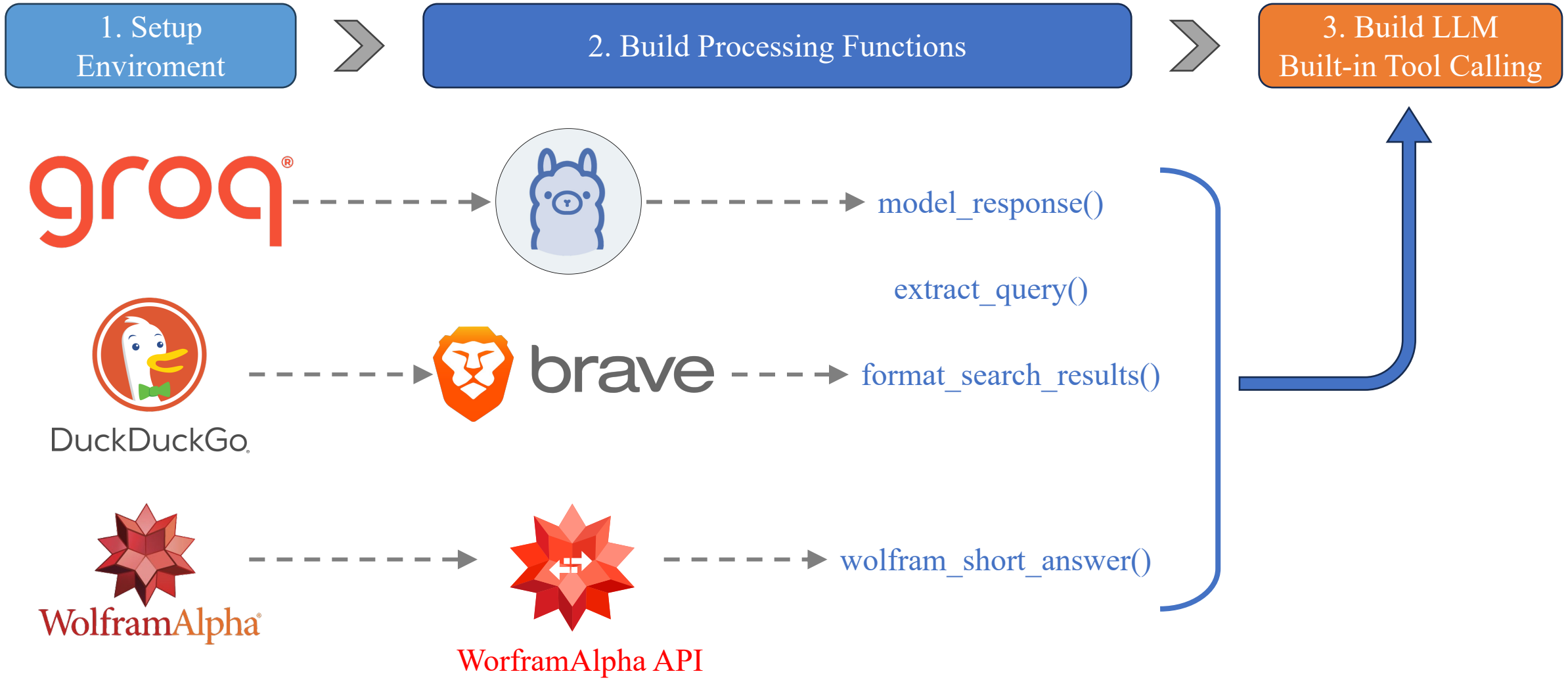
brave

```
# for Search
<|python_tag|>
brave_search.call(query="...")
<|eom_id|>

# for Wolfram
<|python_tag|>
wolfram_alpha.call(query="...")
<|eom_id|>
```

WolframAlpha

Tool Calling in LLMs

❖ Implement Built-in Tool in Llama3



Tool Calling in LLMs

❖ Install & Import libraries

1. Setup Enviroment

```
!pip install duckduckgo-search groq wolframalpha
```

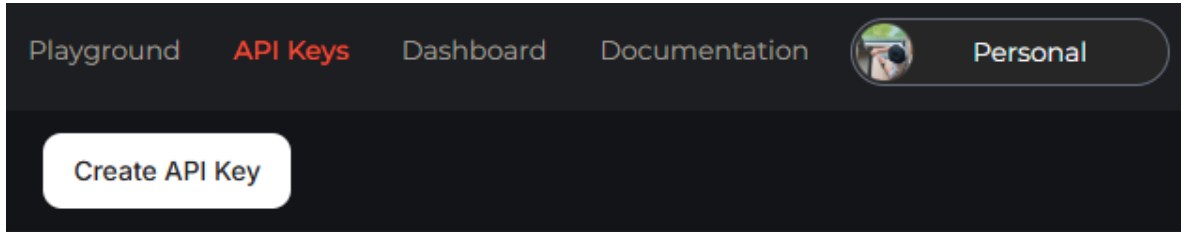
```
import os
import re
import json
import asyncio
import aiohttp
import requests
import nest_asyncio
import urllib.parse
from groq import Groq
from google.colab import userdata
from duckduckgo_search import DDGS

nest_asyncio.apply()

os.environ["GROQ_API_KEY"] = 'GROQ_API_KEY'
APP_ID = 'WF_APPID'

client = Groq()
```

Tool Calling in LLMs

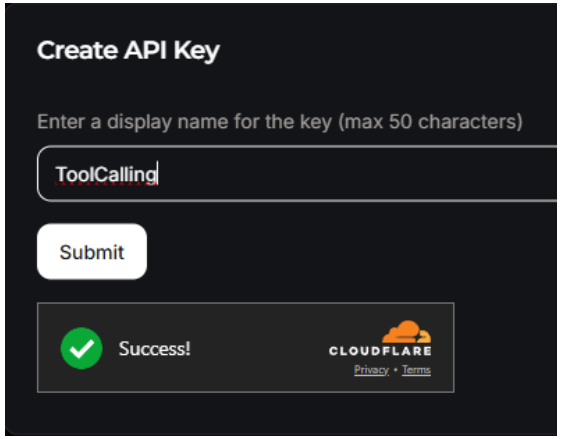
❖ How to get Groq API?

Step 1: Login
https://console.groq.com/home

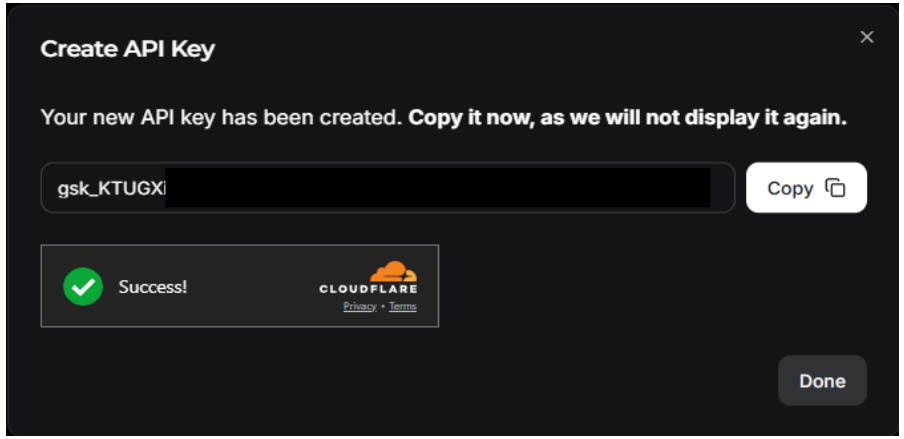
Step 2: API Keys



Step 3: Create API Keys



Step 4: Copy



Step 5: Replace

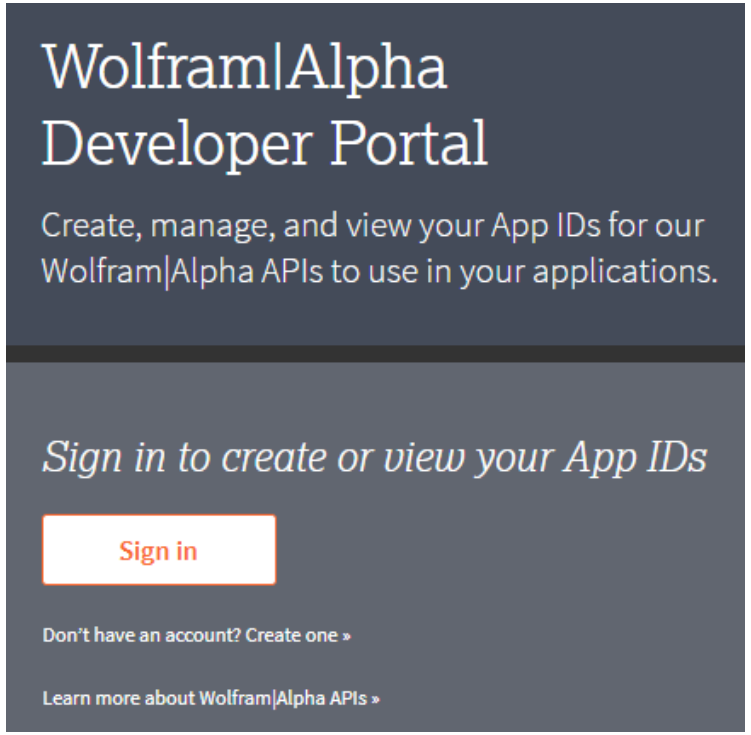
```
15 os.environ["GROQ_API_KEY"] = 'GROQ_API_KEY'
```

Tool Calling in LLMs

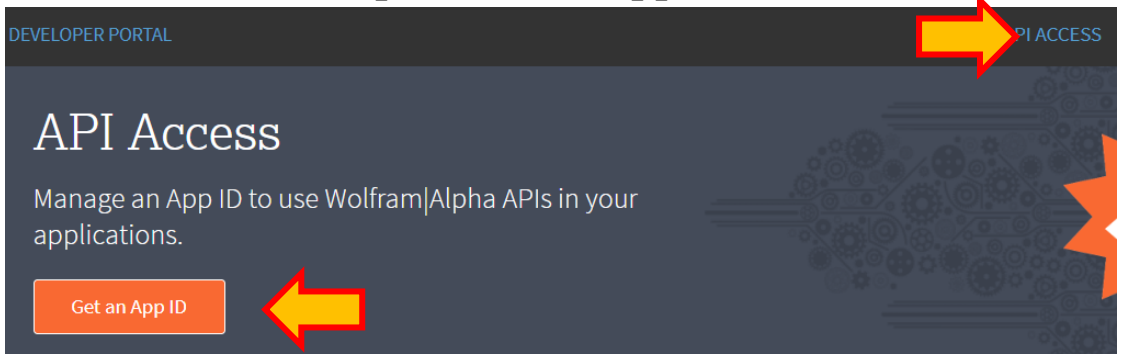
❖ How to get Wolfram App ID?

Step 1: Login

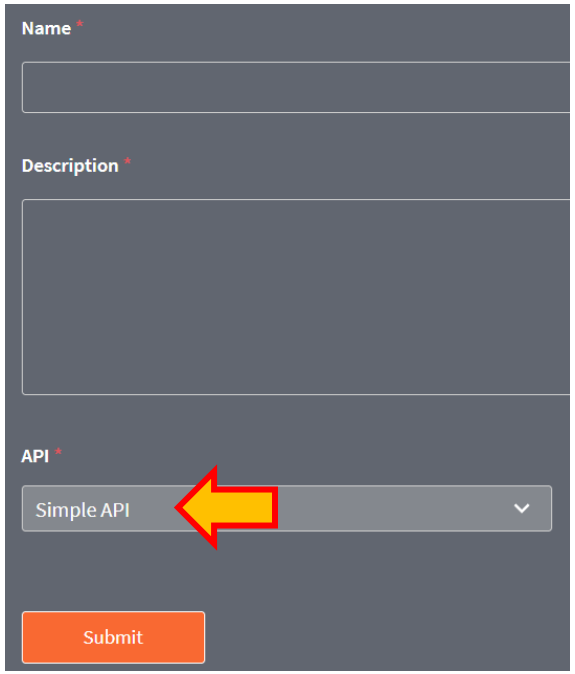
https://developer.wolframalpha.com/



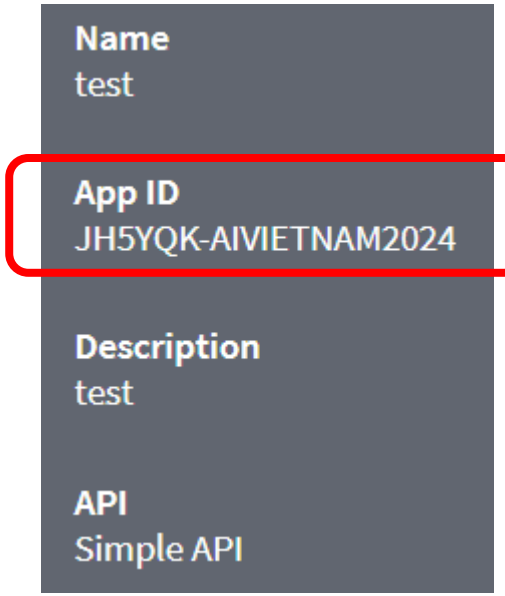
Step 2: Get an App ID



Step 3: Fill & Create App ID



Step 4: Copy App ID



Step 5: Replace

```
16 APP_ID = 'WF_APPID'
```

Tool Calling in LLMs

❖ Model

2. Build Processing Functions

```python
def model_response(prompt):
    response = client.chat.completions.create(
        messages=[
            {
                "role": "user",
                "content": prompt,
            }
        ],
        # model="llama-3.3-70b-versatile",  # for search
        model="llama-3.1-8b-instant",  # for calculation
    )

    return response
```

Can try with the following models

MODEL ID	TOOL USE SUPPORT?
meta-llama/llama-4-scout-17b-16e-instruct	Yes
meta-llama/llama-4-maverick-17b-128e-instruct	Yes
qwen-qwq-32b	Yes
deepseek-r1-distill-qwen-32b	Yes
deepseek-r1-distill-llama-70b	Yes
llama-3.3-70b-versatile	Yes
llama-3.1-8b-instant	Yes
gemma2-9b-it	Yes

https://console.groq.com/docs/tool-use

Tool Calling in LLMs

❖ Extract Query

2. Build Processing Functions

```python
def extract_query(tool_call_text):
    brave_match = re.search(r'brave_search.call\(query="(.*?)"\)', tool_call_text)
    if brave_match:
        return "brave_search", brave_match.group(1)
    wolfram_match = re.search(r'wolfram_alpha.call\(query="(.*?)"\)', tool_call_text)
    if wolfram_match:
        return "wolfram_alpha", wolfram_match.group(1)
    return None, None
```

```
<|python_tag|>brave_search.call(query="thời tiết Việt Nam ngày 8/5/2025")
```

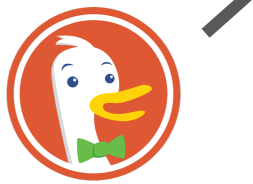
Tool Calling in LLMs

❖ Search Tool

2. Build Processing Functions

```python
def format_search_results(results):
    lines = []
    for i, item in enumerate(results, start=1):
        title = item.get("title", "No title")
        href = item.get("href", "No link")
        body = item.get("body", "No description")
        lines.append(f"{i}. {title}\n   Description: {body}\n   Link: {href}\n")
    return "\n".join(lines)
```

response



DuckDuckGo

format

```
=== Tool Responses ===

1. Dự báo thời tiết 8/5/2025: Bắc và Trung ... - Báo VietNamNet
   Description: Dự báo thời tiết ngày 8/5/2025 các vùng trên cả nước:
   Link: https://vietnamnet.vn/du-bao-thoi-tiet-8-5-2025-bac-va-trung

2. Thời tiết hôm nay 8/5/2025: Cả nước nắng nóng diện rộng
   Description: Thứ năm, ngày 08/05/2025 06:07 GMT+7 . Thời tiết hôm
   Link: https://danviet.vn/thoi-tiet-hom-nay-8-5-2025-ca-nuoc-nang-n

3. Dự báo thời tiết ngày 8/5/2025: Nắng nóng cục bộ kéo dài
   Description: Dự báo thời tiết ngày Hà Nội ngày 8/5. Thời tiết Hà N
   Link: https://thoitiet24h.vn/thoi-tiet-hang-ngay/du-bao-thoi-tiet-
```

Tool Calling in LLMs

❖ Calculation Tool

2. Build Processing Functions

```
16 APP_ID = userdata.get('WF_APPID1')
```

WorframAlpha API

```
32 async def wolfram_short_answer(query):
33     encoded_query = urllib.parse.quote(query)
34     url = f"https://api.wolframalpha.com/v1/result?i={encoded_query}&appid={APP_ID}"
35     async with aiohttp.ClientSession() as session:
36         async with session.get(url) as response:
37             if response.status == 200:
38                 return await response.text()
39             else:
40                 return f"[Error {response.status}] {await response.text()}"
```

Tool Calling in LLMs

❖ All in One Function

3. Build LLM
Built-in Tool Calling

Default prompt

```
async def llama_with_tool(query):
    PROMPT = f"""
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Environment: ipython
Tools: brave_search, wolfram_alpha
Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

You are a helpful assistant specialized in performing web and computation searches. You have access to two tools:
    1. **brave_search** for general web searches
    2. **wolfram_alpha** for computation and factual queries
Always try to use these tools for informational and computational query questions.<|eot_id|><|start_header_id|>user<|end_header_id|>
{query}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
    """
```

continue …

Tool Calling in LLMs

❖ All in One Function

3. Build LLM
Built-in Tool Calling

```python
response = model_response(PROMPT)
tool_call = response.choices[0].message.content.strip()

print("=== Model Tool Call Output ===\n")
print(tool_call)

tool_name, query_content = extract_query(tool_call)
print("\n=== Extracted Query Content ===\n")
print(query_content)

if tool_name == "brave_search":
    tool_response = DDGS().text(query_content, max_results=5)
    formatted_results = format_search_results(tool_response)
elif tool_name == "wolfram_alpha":
    res = await wolfram_short_answer(query_content)
    formatted_results = "The exact value of the query is " + res
else:
    formatted_results = "No tool could be identified."
```

```
<|python_tag|>brave_search.call(query="thời tiết Việt Nam ngày 8/5/2025")
```

DuckDuckGo

WorframAlpha API

continue ...

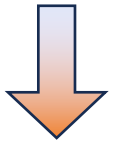
Tool Calling in LLMs

❖ All in One Function

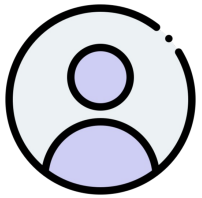
3. Build LLM
Built-in Tool Calling

WorframAlpha
API

DuckDuckGo

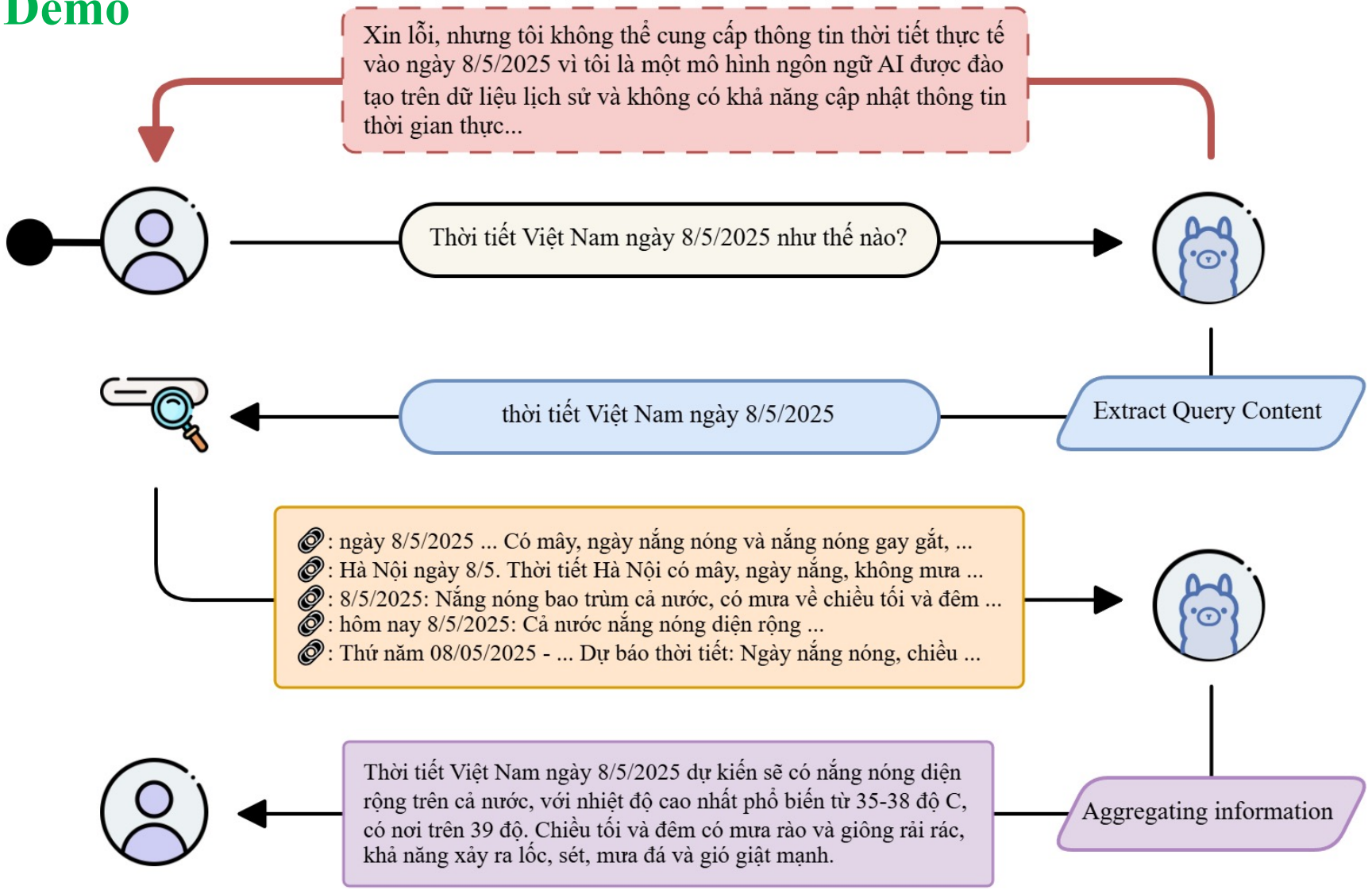


```
36    print("\n=== Tool Responses ===\n")
37    print(formatted_results)
38
39    follow_up_prompt = f"Tool responses:\n{formatted_results}\nBased on these responses, provide a concise result for
      this query.: {query}"
40    follow_up_response = model_response(follow_up_prompt)
41
42    print("\n=== Final Answer ===\n")
43    return follow_up_response.choices[0].message.content
```



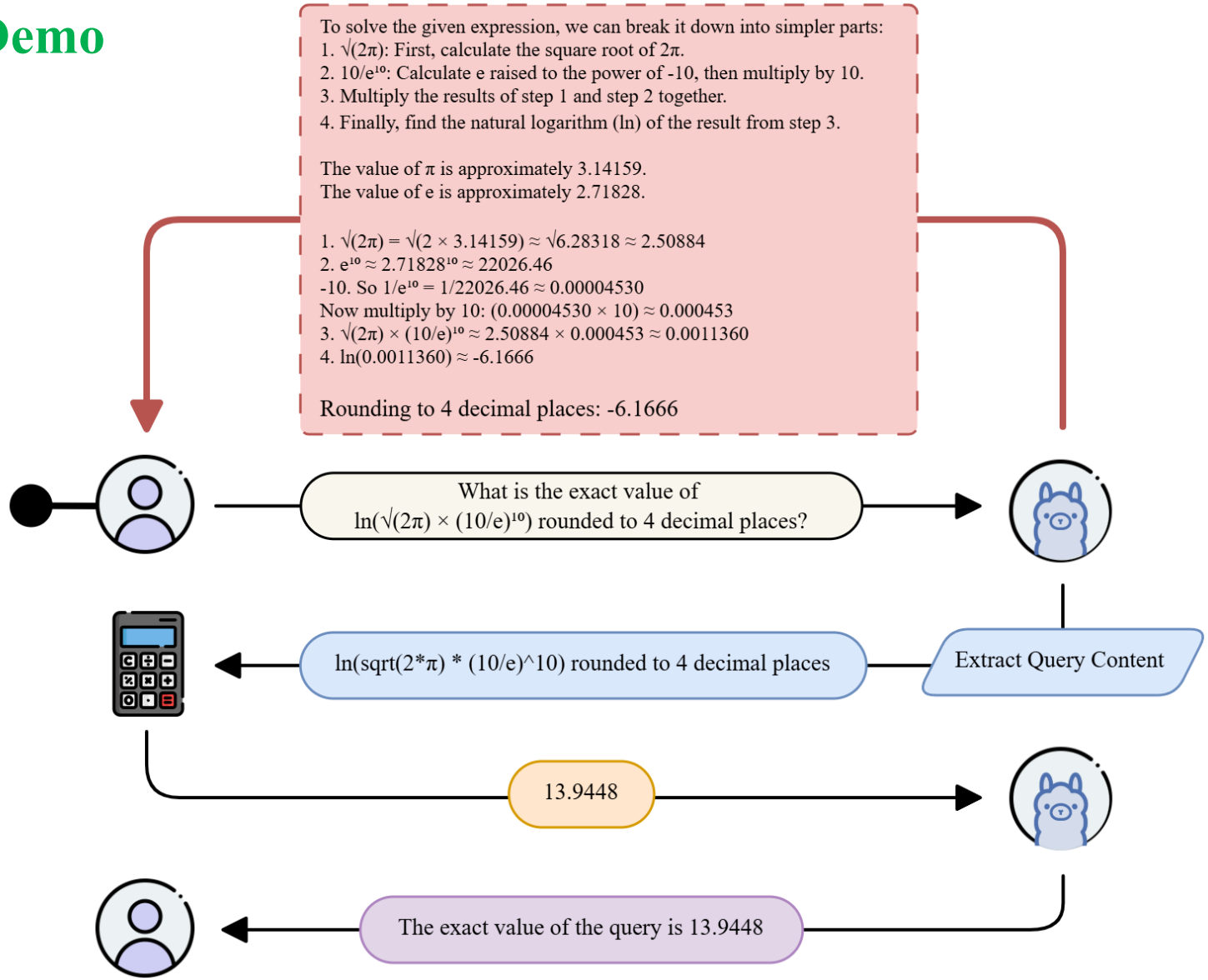
Tool Calling in LLMs

❖ Search Demo



Tool Calling in LLMs

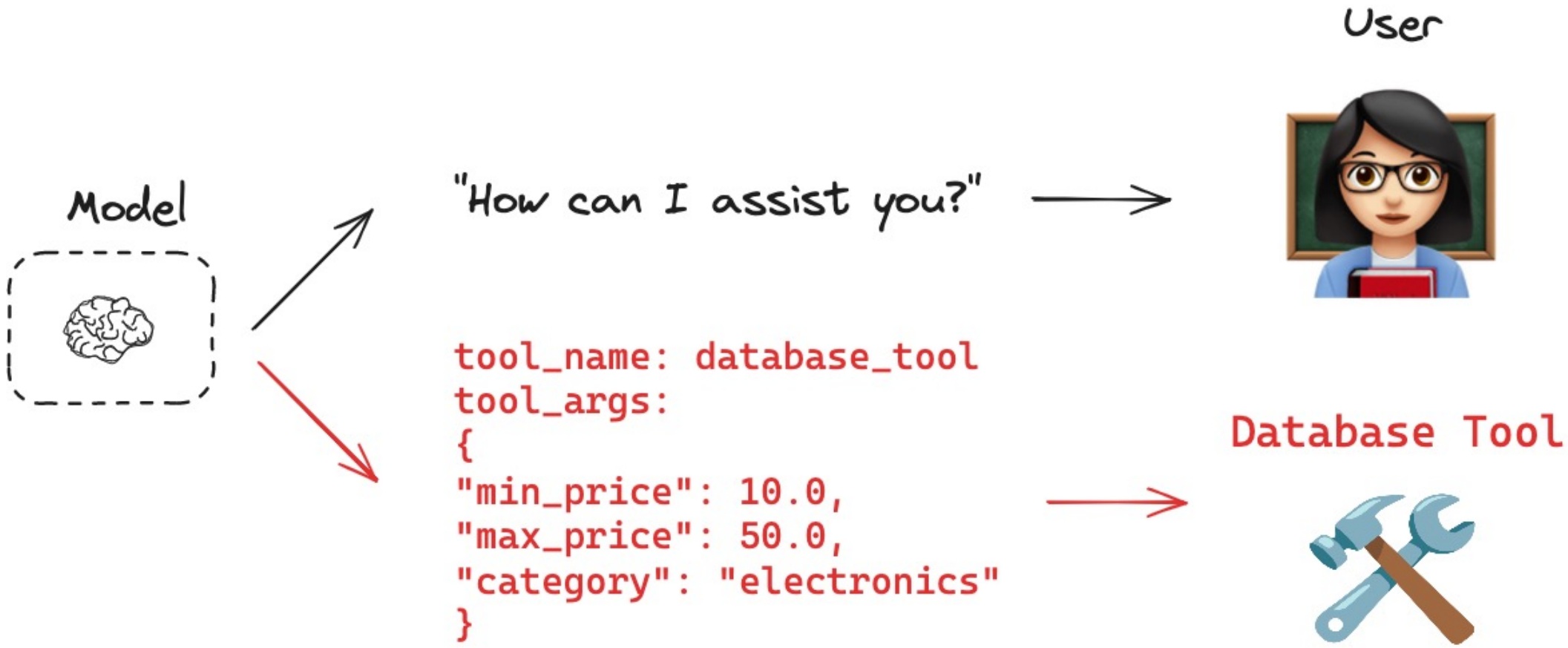
❖ Calculation Demo



Langchain Tool Calling

Tool Calling in LLMs

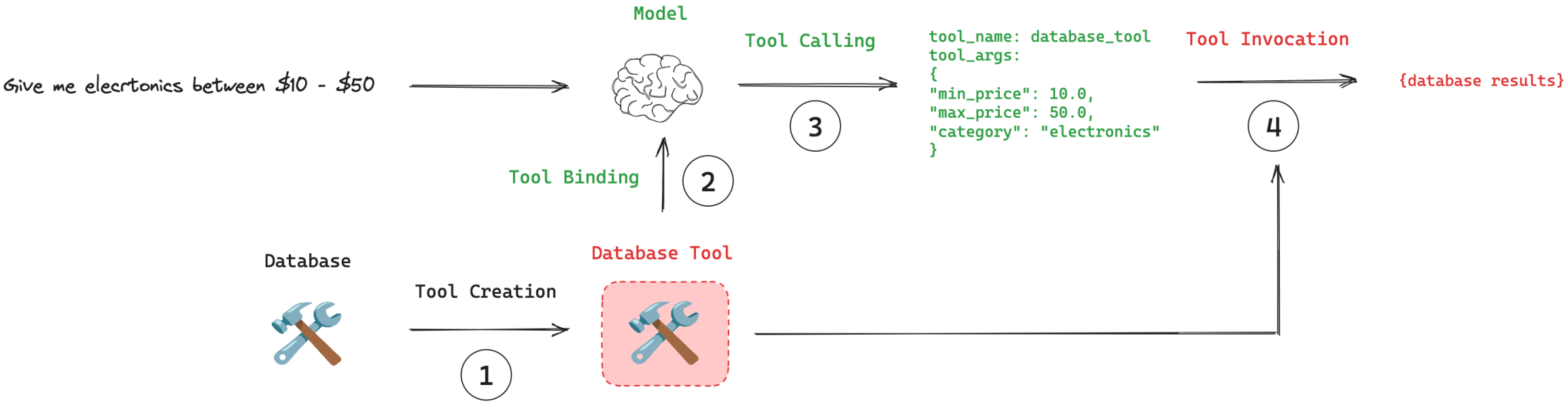
❖ Getting Started



Ref: https://python.langchain.com/docs/concepts/tool_calling/

Tool Calling in LLMs

❖ Getting Started



Ref: https://python.langchain.com/docs/concepts/tool_calling/

Tool Calling in LLMs

❖ Create tools

using the @tool decorator

```python
from langchain_core.tools import tool

@tool
def reverse_string(input_string: str) -> str:
    """Reverse a string"""
    reversed_string = input_string[::-1]

    return reversed_string

reversed_string = reverse_string.invoke("AI VIET NAM")
print(reversed_string)
```

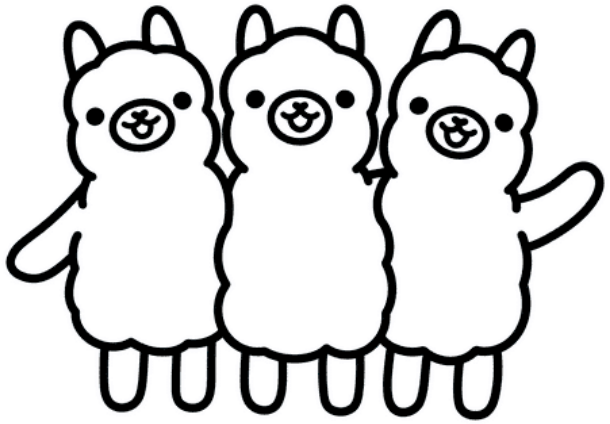
```
type(reverse_string)
[3]  ✓ 0.0s
...  langchain_core.tools.structured.StructuredTool
```

Tool Calling in LLMs

❖ Define chatmodel

```python
from langchain.chat_models import init_chat_model

model_id = "llama3.2:3b"
temperature = 0.5
max_output_tokens = 512

llm = init_chat_model(
    model=model_id,
    model_provider="ollama",
    temperature=temperature,
    max_tokens=max_output_tokens,
    format="json"
)
```

Meta



ollama run llama3

Tool Calling in LLMs

❖ Blinding tools

```python
tool
def search(query: str, top_k=3) → str:
    """Search DuckDuckGo for the given query and return the first result."""
    wrapper = DuckDuckGoSearchAPIWrapper(region="vn-vi", time="d", max_results=top_k)
    search = DuckDuckGoSearchResults(api_wrapper=wrapper, source="news")
    results = search.run(query)

    return results
```

```python
@tool
def multiply(a: int, b: int) → int:
    """Multiply a and b."""
    return a * b
```

```python
tools = [reverse_string, search, multiply]

model_with_tools = llm.bind_tools(tools)
```

Tool Calling in LLMs

❖ Tool Calling

```
response = model_with_tools.invoke("Tin tức mới nhất hôm nay")
response
```

AIMessage(content='', additional_kwargs={}, response_metadata={'model': 'llama3.2:3b', 'created_at': '2025-05-11T11:50:15.402212361Z', 'done': True, 'done_reason': 'stop', 'total_duration': 333856129, 'load_duration': 22755899, 'prompt_eval_count': 271, 'prompt_eval_duration': 104118940, 'eval_count': 21, 'eval_duration': 205630506, 'model_name': 'llama3.2:3b'}, id='run--e55a330c-fa03-481a-8705-2ddc5d939e5b-0', tool_calls=[{'name': 'search', 'args': {'query': 'tin tức mới nhất hôm nay'}, 'id': '47a2f459-bb3c-41a9-a7ce-340360eb0afd', 'type': 'tool_call'}], usage_metadata={'input_tokens': 271, 'output_tokens': 21, 'total_tokens': 292})

```
[25]  response.tool_calls

...   [{'name': 'search',
        'args': {'query': 'tin tức mới nhất hôm nay'},
        'id': '47a2f459-bb3c-41a9-a7ce-340360eb0afd',
        'type': 'tool_call'}]
```

AI VIET NAM
@aivietnam.edu.vn

Tool Calling
Instruction Fine-tuning

Tool Calling Instruction Fine-tuning

❖ 1. Setup Environment

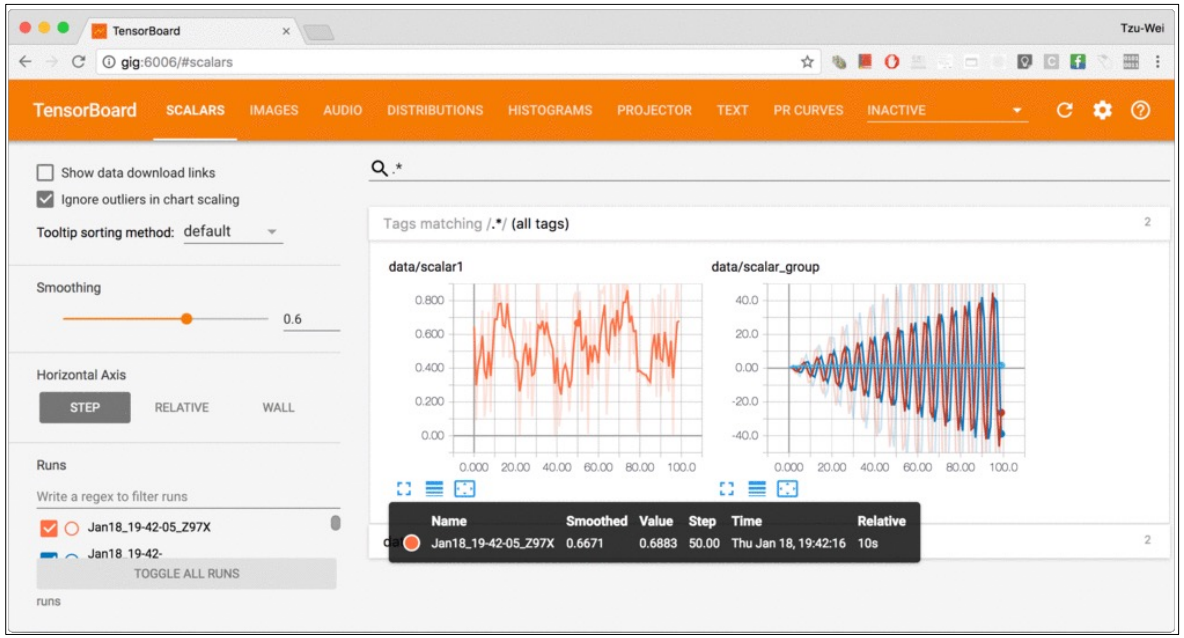
Weights & Biases

https://github.com/wandb/wandb

```
# !pip install -q -U bitsandbytes
# !pip install -q -U peft
# !pip install -q -U tensorboardX
# !pip install -q wandb
```

https://github.com/bitsandbytes-foundation/bitsandbytes

https://github.com/huggingface/peft



https://github.com/lanpa/tensorboardX

Tool Calling Instruction Fine-tuning

❖ 1. Setup Environment

https://github.com/unslothai/unsloth

Hugging Face

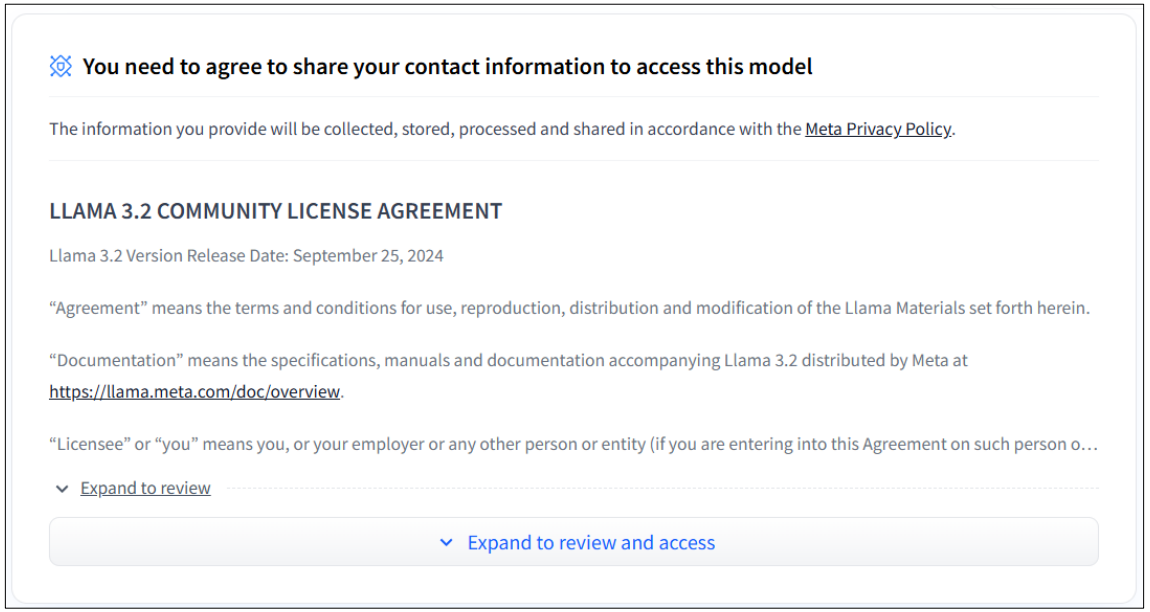
https://huggingface.co/

```python
import unsloth
import os, torch, random, json
from datasets import load_dataset
from transformers import (
    TrainingArguments,
    set_seed
)
from trl import SFTTrainer, SFTConfig
from unsloth import FastLanguageModel


set_seed(59)
os.environ["CUDA_VISIBLE_DEVICES"] = "1"
```

Tool Calling Instruction Fine-tuning

❖ 2. Config Model & PEFT

https://huggingface.co/meta-llama/Llama-3.2-1B

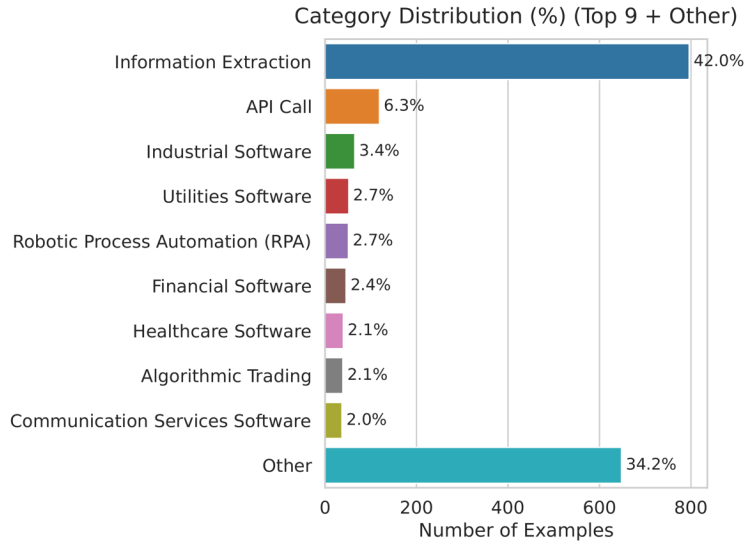


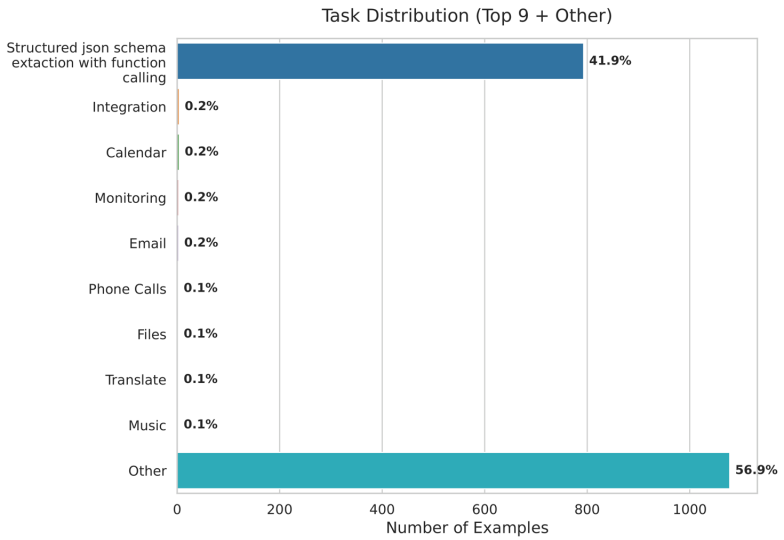

```python
model_id = "meta-llama/Llama-3.2-1B-Instruct"
max_seq_length = 2048
model, tokenizer = FastLanguageModel.from_pretrained(
    model_id,
    max_seq_length = max_seq_length,
    load_in_4bit = True,
)
```

```python
model = FastLanguageModel.get_peft_model(
    model,
    r = 16,
    lora_alpha = 32,
    lora_dropout = 0.05,
    target_modules = ["q_proj","k_proj","v_proj","o_proj",
                      "gate_proj","up_proj","down_proj"],
    bias = "none"
)
```

Tool Calling Instruction Fine-tuning

❖ 3. Build Function Calling Dataset





https://huggingface.co/datasets/NousResearch/hermes-function-calling-v1

Dataset Viewer — Auto-converted to Parquet · API · Embed · Data Studio

Subset (5): func_calling_singleturn · 1.89k rows

Split (1): train · 1.89k rows

Search this dataset

id string · lengths	conversations list · lengths	category string · classes	subcategory string · lengths	task string · lengths
36 … 36	3 … 3	63 values	3 … 67	4 … 150
85f6c398-69c7-4df2-aed1-…	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function…	IoT and Home Automation	Security Camera Management	View and Manage Security Camera…
89ef3c87-66bd-46ee-9297-…	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function…	IoT and Home Automation	Smart Home Setup	Set Up a Smart Home System
14657d01-d6d1-46df-8eb1-…	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function…	IoT and Home Automation	Thermostat Control	Adjust Smart Thermostat…
c483f963-8a29-4ff0-a684-…	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function…	IoT and Home Automation	Voice Commands for Home Tasks	Perform Home Tasks Using Voice…
81ad724a-bb74-420f-8221-…	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function…	IoT and Home Automation	Lighting Control	Control Smart Lights in a Home
7d99abac-f27f-4ed2-a9ea-…	[{ "from": "system", "value": "You are a function calling AI model. You are provided with function…	IoT and Home Automation	Garage Door Automation	Open and Close Smart Garage Doors

‹ Previous 1 2 3 ... 19 Next ›

Tool Calling Instruction Fine-tuning

❖ 3. Build Function Calling Dataset

```python
dataset_id = "NousResearch/hermes-function-calling-v1"
raw_ds = load_dataset(dataset_id, split="train")

SYS_PROMPT = (
    "Cut-off knowledge: Dec 2023\n"
    "Today: 23 July 2024\n\n"
    "You are a helpful assistant with tool-calling capabilities."
)

def to_chat(example):
    msgs = [{"role": "system", "content": SYS_PROMPT}]
    role = {"system": "system", "human": "user", "gpt": "assistant"}
    for turn in example["conversations"]:
        msgs.append({"role": role[turn["from"]], "content": turn["value"]})
    chat_str = tokenizer.apply_chat_template(
        msgs,
        tokenize=False,
        add_generation_prompt=False
    )
    return {"text": chat_str}

ds = raw_ds.map(to_chat).remove_columns(
        [c for c in raw_ds.column_names if c != "text"]
)
```


```
[
  {
    "from": "system",
    "value": "You are a function calling AI model. You are provided with function signatures within <tools> </tools> XML tags. You may call one or more functions...<tool_call>\n{'arguments': <args-dict>, 'name': <function-name>}\n</tool_call>\n"
  },
  {
    "from": "human",
    "value": "I've recently installed a new security system at my home, and I want to ensure everything is functioning...To summarize, I request the execution of the following functions..."
  },
  {
    "from": "gpt",
    "value": "<tool_call>\n{'arguments': {'camera_id': 'front_door', 'stream_quality': '1080p'}, 'name': 'get_camera_live_feed'}\n</tool_call>\n<tool_call>\n{'arguments': {'camera_id': 'front_door', 'duration': 30}, 'name': 'record_camera_feed'}\n</tool_call>\n<tool_call>\n{'arguments': {'camera_id': 'front_garden', 'start_time': '2023-04-22T15:00:00Z', 'end_time': '2023-04-22T17:00:00Z'}, 'name': 'get_recorded_feed'}\n</tool_call>\n"
  }
]
```

Tool Calling Instruction Fine-tuning

❖ 4. Train Function Calling Model

```python
training_args = TrainingArguments(
    output_dir = "./llama3-toolcall-lora",
    per_device_train_batch_size = 16,
    gradient_accumulation_steps = 2,
    num_train_epochs = 1,
    learning_rate = 2e-4,
    lr_scheduler_type = "cosine",
    bf16 = torch.cuda.is_bf16_supported(),
    logging_steps = 25,
    save_strategy = "epoch",
    push_to_hub = False,
)

sft_cfg = SFTConfig(dataset_text_field="text",
                    max_seq_length=max_seq_length)

trainer = SFTTrainer(
    model         = model,
    tokenizer     = tokenizer,
    train_dataset = ds,
    args          = training_args,
    **sft_cfg.to_dict(),
)
trainer.train()
```

230	1.558800
231	1.410900
232	1.106200
233	1.381900
234	1.385100
235	1.259700
236	1.432600
237	1.279800

```python
trainer.model.save_pretrained("./llama3-toolcall-lora")
tokenizer.save_pretrained("./llama3-toolcall-lora")
```

```python
# merged = FastLanguageModel.merge_lora(trainer.model)
# merged.push_to_hub("your-hf-name/llama3-1b-toolcall-merged", private=False)
# tokenizer.push_to_hub("your-hf-name/llama3-1b-toolcall-merged")
```

Tool Calling Instruction Fine-tuning

❖ 5. Inference

```python
import torch
from unsloth import FastLanguageModel
from peft import PeftModel

device = "cuda" if torch.cuda.is_available() else "cpu"
max_seq_length = 2048

def chat(user_msg):
    prompt = tokenizer.apply_chat_template(
        [{"role":"user","content":user_msg}],
        tokenize=False,
        add_generation_prompt=True,
    )
    inputs = tokenizer(prompt, return_tensors="pt").to("cuda")
    with torch.no_grad():
        out = model.generate(
            **inputs,
            max_new_tokens=max_seq_length,
            temperature=0.1
        )
    decoded_output = tokenizer.decode(out[0], skip_special_tokens=True)
    decoded_output = decoded_output.split("assistant")[-1].strip()

    return decoded_output


base_id = "meta-llama/Llama-3.2-1B-Instruct"
lora_dir = "./llama3-toolcall-lora"

model, tokenizer = FastLanguageModel.from_pretrained(
    base_id,
    max_seq_length = max_seq_length,
    load_in_4bit = True
)

model = PeftModel.from_pretrained(model, lora_dir, is_trainable=False)
```

chat_msg = """
You are a function calling AI model. You are provided with function signatures within <tools> </tools> XML tags.
You may call one or more functions to assist with the user query.
Don't make assumptions about what values to plug into functions.
<tools>
[
{'type': 'function',
'function': {
'name': 'get_camera_live_feed',
'description': 'Retrieves the live feed from a specified security camera.',
'parameters': {…}},
{'type': 'function',
'function': {
'name': 'list_all_cameras',
'description': 'Lists all the security cameras connected to the home network.',
'parameters': {…}},
{'type': 'function',
'function': {
'name': 'record_camera_feed',
'description': 'Starts recording the live feed from a specified security camera.',
'parameters': {…}},
{'type': 'function',
'function': {
'name': 'get_recorded_feed',
'description': 'Retrieves a previously recorded feed from a specified security camera.',
'parameters': {…}},
{'type': 'function',
'function': {
'name': 'pan_tilt_camera',
'description': 'Controls the pan and tilt functions of a PTZ (Pan-Tilt-Zoom) security camera.',
'parameters': {…}},
]
</tools>
…

continue …

Tool Calling Instruction Fine-tuning

❖ 5. Inference

For each function call, return a JSON object with function name and arguments within <tool_call> </tool_call> tags with the following schema:

<tool_call>
{'arguments': <args-dict>, 'name': <function-name>}
</tool_call>

<|eot_id|><|start_header_id|>user<|end_header_id|>

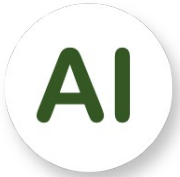
I've recently installed a new security system at my home, and I want to ensure everything is functioning as it should.

Specifically, I'd like to:
1. Check the live feed from the "front_door" camera in 1080p quality.
2. Start a 30-minute recording of the live feed from the same camera.
3. Retrieve the recorded feed from the "front_garden" camera between 15:00 and 17:00 on April 22, 2023.

```
'function': {
        'name': 'get_camera_live_feed',
        'description': 'Retrieves the live feed from a specified security camera.
'function': {
        'name': 'list_all_cameras',
        'description': 'Lists all the security cameras connected to the home network.'
'function': {
        'name': 'record_camera_feed',
        'description': 'Starts recording the live feed from a specified security camera.'
'function': {
        'name': 'get_recorded_feed',
        'description': 'Retrieves a previously recorded feed from a specified security camera.',
'function': {
        'name': 'pan_tilt_camera',
        'description': 'Controls the pan and tilt functions of a PTZ (Pan-Tilt-Zoom) security camera.'
```

```
41 response = chat(chat_msg)
42 print(response)
```

```
<tool_call>
{'arguments': {'camera_id': 'front_door','stream_quality': '1080p'}, 'name': 'get_camera_live_feed'}
</tool_call>
<tool_call>
{'arguments': {'camera_id': 'front_door', 'duration': 30}, 'name':'record_camera_feed'}
</tool_call>
<tool_call>
{'arguments': {'camera_id': 'front_garden','start_time': '2023-04-22T15:00:00', 'end_time': '2023-04-22T17:00:00'}, 'name': 'get_recorded_feed'}
</tool_call>
```



AI VIET NAM
@aivietnam.edu.vn

QUIZ

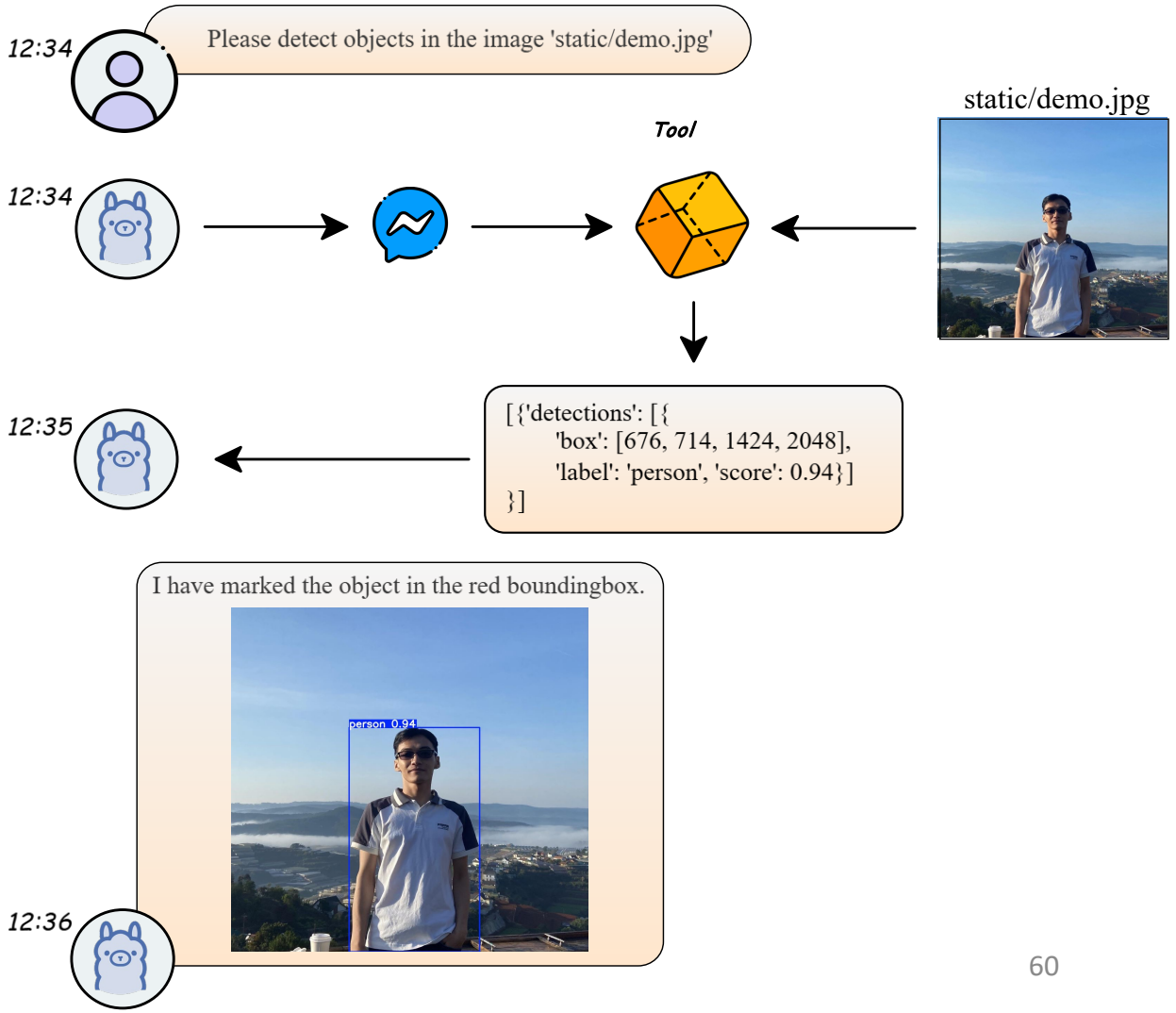
Tool Calling
Vision Task

Tool Calling in LLMs

❖ Solving Computer Vision Tasks with Tool Calling

Problem Description: Build a program capable of performing Object Detection and Object Segmentation on a given image.

- **Input:** An image file and a text instruction (e.g., "Detect objects in this image").
- **Output:** Detected objects with bounding boxes or segmentation masks, returned as structured data.



Tool Calling in LLMs

❖ Step 1: Install and import necessary libraries

```
!pip install -q --upgrade ultralytics opencv-python transformers torch vllm unsloth
```

```
import os
import torch
import json
import ast
import sys
from transformers import SamModel, SamProcessor
from transformers import AutoModelForCausalLM, AutoTokenizer
from transformers import GenerationConfig
from PIL import Image
from ultralytics import YOLO


os.environ["CUDA_VISIBLE_DEVICES"] = "1"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Tool Calling in LLMs

❖ Step 2: Define Tools

```python
detection_model_id = "yolo11n.pt"
detection_model = YOLO(detection_model_id)
def run_detection(image_path: str, is_visualize: bool = False):
    """YOLOv11: return list of {box, label, score} for a single image."""
    results = detection_model(image_path)
    r = results[0]

    detections = []
    for box in r.boxes:
        # box.xyxy is a 1×4 tensor, box.conf is a 1-element tensor, box.cls likewise
        coords  = box.xyxy.cpu().numpy().flatten().tolist()
        score   = float(box.conf.cpu().numpy().item())
        cls_id  = int(box.cls.cpu().numpy().item())
        detections.append({
            "box": coords,
            "label": r.names[cls_id],
            "score": score,
        })

    if is_visualize:
        r.save()
        r.show()

    return {"detections": detections}
```

The detection function wraps the full YOLOv11 pipeline—input, inference, JSON output, and visualization—into a single callable, letting the LLM use it without handling internal model logic.

Tool Calling in LLMs

❖ Step 2: Define Tools

```python
segmentation_model_id = "facebook/sam-vit-base"
sam_processor = SamProcessor.from_pretrained(segmentation_model_id)
sam_model = SamModel.from_pretrained(segmentation_model_id)
def run_segmentation(image_path: str):
    """SAM: return binary masks as nested lists"""
    img = Image.open(image_path).convert("RGB")
    inputs = sam_processor(images=img, return_tensors="pt")
    outputs = sam_model(**inputs)

    masks = outputs.pred_masks.squeeze(0).cpu().detach().numpy().tolist()
    return {"masks": masks}
```

Like detection, the segmentation function extracts binary masks from SAM and returns pure Python data, enabling seamless data flow between components.

To JSON 


```python
def run_detection(image_path: str, is_visualize: bool = False):
        """YOLOv11: return list of {box, label, score} for a
        single image."""

        detections = detection_model(image_path)[0]

        return {"detections": detections}
```

```
{
  "type": "function",
  "function": {
    "name": "run_detection",
    "description": "Detect objects in an image and return
bounding boxes and labels.",
    "parameters": {
      "type": "object",
      "properties": {
        "image_path": {"type": "string", "description":
"Local path to the image file."},
        "is_visualize": {"type": "bool", "description":
"Boolean flag to visualize the detection results. Default is
False."}
      },
      "required": ["image_path"]
    }
  }
}
```

Tool Calling in LLMs

❖ Step 2: Define Tools

```python
tool_functions = {
    "run_detection": run_detection,
    "run_segmentation": run_segmentation
}
```

```python
functions_lst = [
    {
        "type": "function",
        "function": {
            "name": "run_detection",
            "description": "Detect objects in an image and return bounding boxes and labels.",
            "parameters": {
                "type": "object",
                "properties": {
                    "image_path": {"type": "string", "description": "Local path to the image file."},
                    "is_visualize": {"type": "bool", "description": "Boolean flag to visualize the detection results. Default is False."}
                },
                "required": ["image_path"]
            }
        }
    },
    {
        "type": "function",
        "function": {
            "name": "run_segmentation",
            "description": "Segment objects in an image and return binary masks.",
            "parameters": {
                "type": "object",
                "properties": {
                    "image_path": {"type": "string", "description": "Local path to the image file."}
                },
                "required": ["image_path"]
            }
        }
    }
]

```

Tool Calling in LLMs

❖ Step 3: Define system prompt

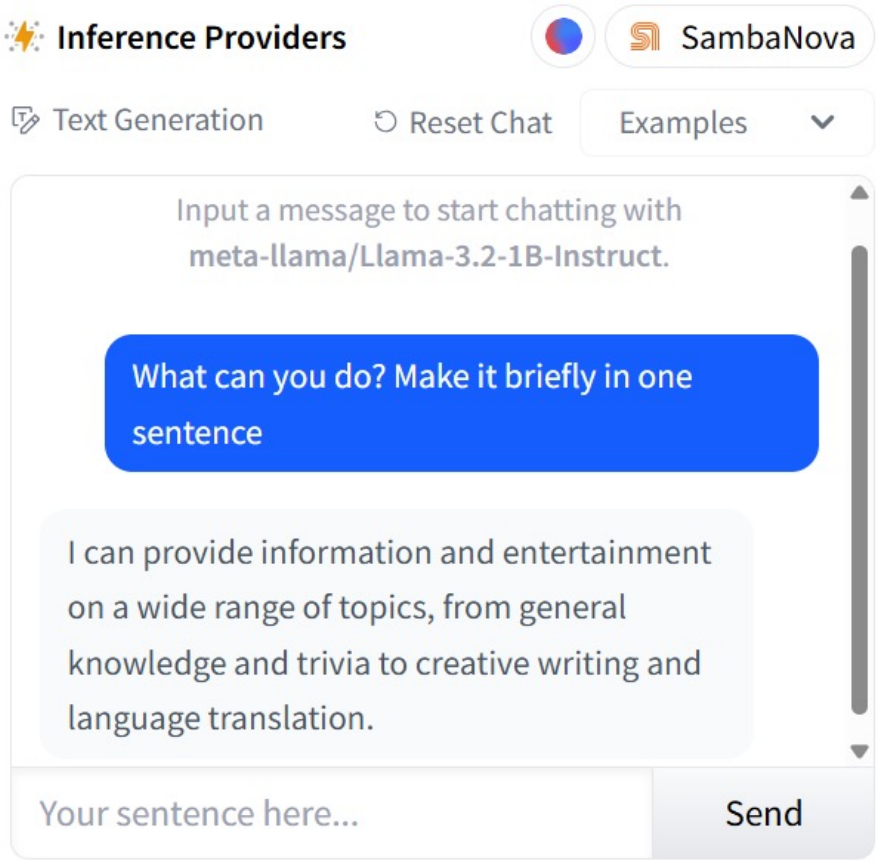
```
SYSTEM_PROMPT = """
You are an expert in composing functions. You are given a question and a set of possible functions.
Based on the question, you will need to make one or more function/tool calls to achieve the purpose.
If none of the function can be used, point it out. If the given question lacks the parameters required by the function,
also point it out. You should only return the function call in tools call sections.

If you decide to invoke any of the function(s), you MUST put it in the format of [func_name1(params_name1=params_value1, params_name2=params_value2...), func_name2(params)]\n
You SHOULD NOT include any other text in the response.

Here is a list of functions in JSON format that you can invoke.\n\n{functions}\n""".format(functions=functions_lst)
```

Once having the function list in JSON format, we define a system prompt to help the model understand the task and available tools.

Tool Calling in LLMs

❖ Step 4: Load Pre-trained model

```python
model_id = "meta-llama/Llama-3.2-1B-Instruct"
llm = AutoModelForCausalLM.from_pretrained(model_id,
                                           device_map=device,
                                           torch_dtype=torch.bfloat16)
tokenizer = AutoTokenizer.from_pretrained(model_id)
```



Tool Calling in LLMs

❖ Step 5: Initialize generation config & conversation

```python
messages = [
    {
      "role": "system",
      "content": SYSTEM_PROMPT
    },
    {
      "role": "user",
      "content": "Please segment objects in the image 'static/cat.jpg'."
    }
]
```

```python
generation_config = GenerationConfig(
    max_new_tokens=512,
    temperature=0.5,
    top_p=0.98,
    top_k=50,
    repetition_penalty=1.2,
    do_sample=True,
    num_return_sequences=1,
    bos_token_id=tokenizer.bos_token_id,
    eos_token_id=tokenizer.eos_token_id,
    pad_token_id=tokenizer.eos_token_id,
)

```

Tool Calling in LLMs

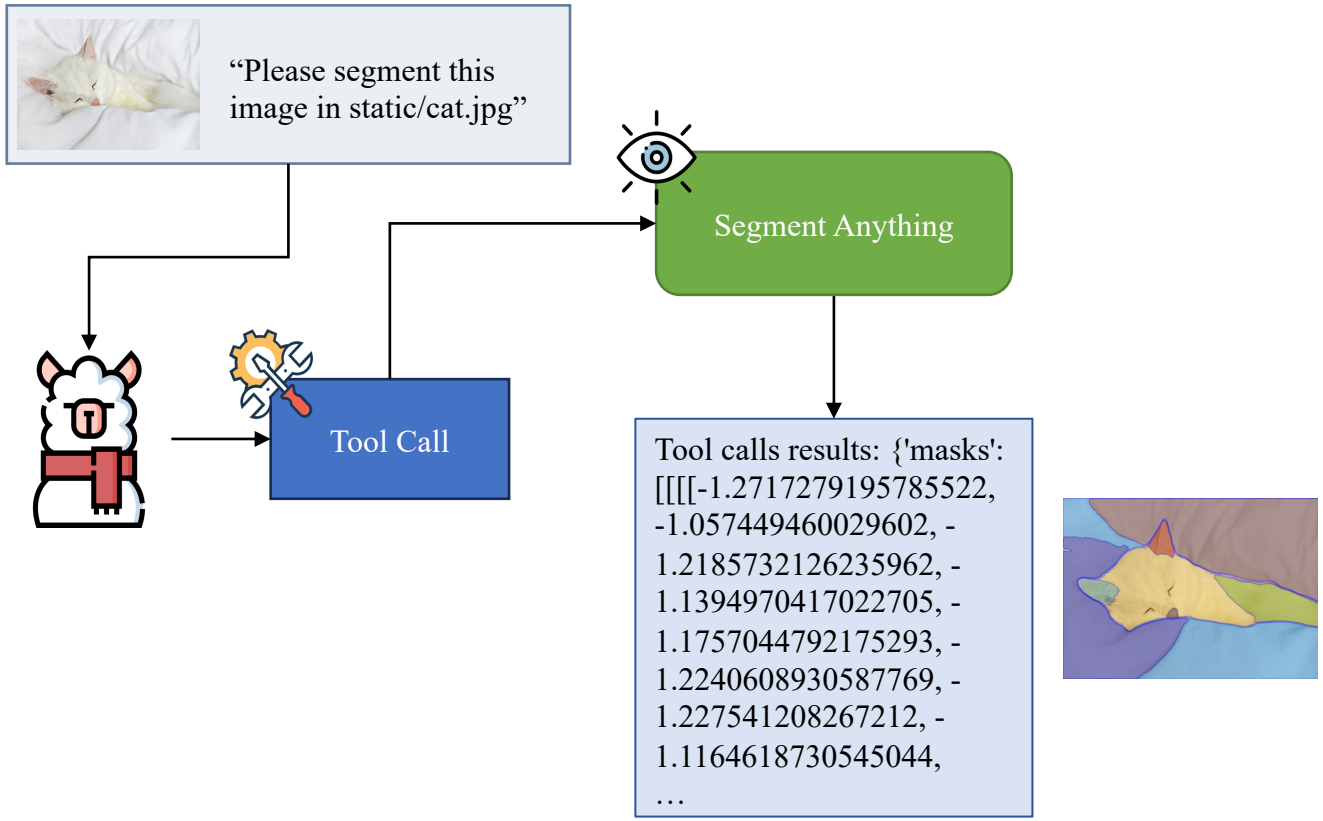
❖ Step 6: Tokenization

```
tokenized_chat = tokenizer.apply_chat_template(messages,
                                               tokenize=True,
                                               add_generation_prompt=True,
                                               return_attention_mask=True,
                                               return_tensors="pt").to(device)
outputs = llm.generate(
    tokenized_chat,
    generation_config=generation_config)
raw = tokenizer.decode(outputs[0], skip_special_tokens=True)
tool_calls_lst_str = raw.split("assistant")[-1]
print(tool_calls_lst_str)
try:
    tree = ast.parse(tool_calls_lst_str, mode='eval')
    call_nodes = tree.body.elts
except SyntaxError as e:
    print("Cannot parse the function call. Try again.")
    sys.exit(1)
```

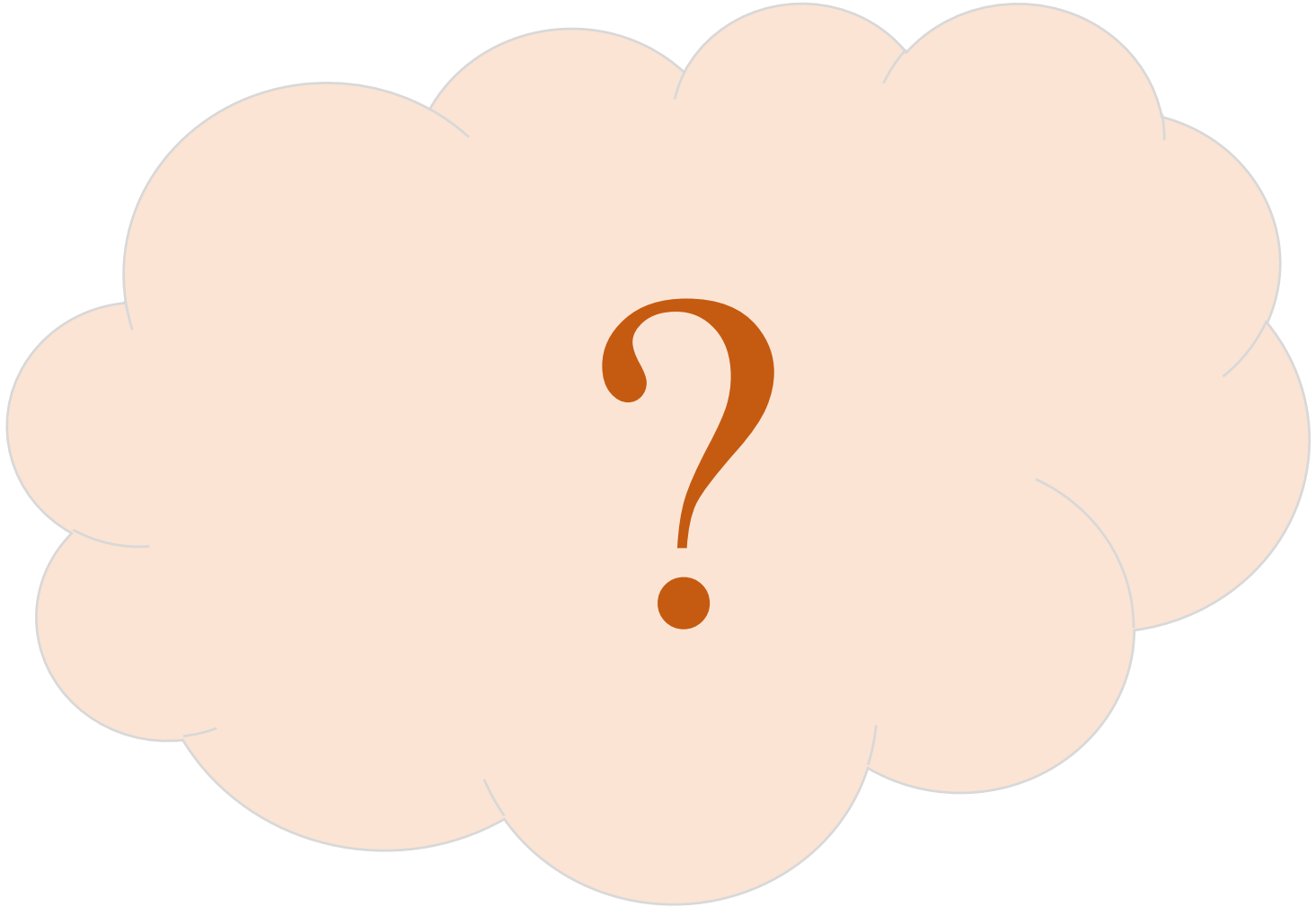
Tool Calling in LLMs

❖ Step 7: Run Tool Call

```python
tool_calls_result = []
for call in call_nodes:
    function_name = call.func.id
    parameters = {
        kw.arg: ast.literal_eval(kw.value)
        for kw in call.keywords
    }

    result = tool_functions[function_name](**parameters)
    tool_calls_result.append(result)

print("Tool calls results:", tool_calls_result)
```



Summarization and Q&A




Thank you!