Lab: Ollama

5/12/2025

What is Ollama?

- We are all familiar with LLMs (ChatGPT, DeepSeek, Copilot)
- We can run a LLM locally with Ollama!
- Advantages running LLM locally:
  - No internet connection
  - Unlimited context and usage. (depend on the model and your computer :P)
  - Unlimited API! (Normally you have to pay...)
  - Most importantly, Privacy

Install Ollama

- To install on the host directly: https://ollama.com/download
  - For non-root installations, you may refer to: https://github.com/ollama/ollama/issues/7421 which will work but is not officially endorsed
- Docker instructions!: https://hub.docker.com/r/ollama/ollama

How to use

Run the server: ollama serve

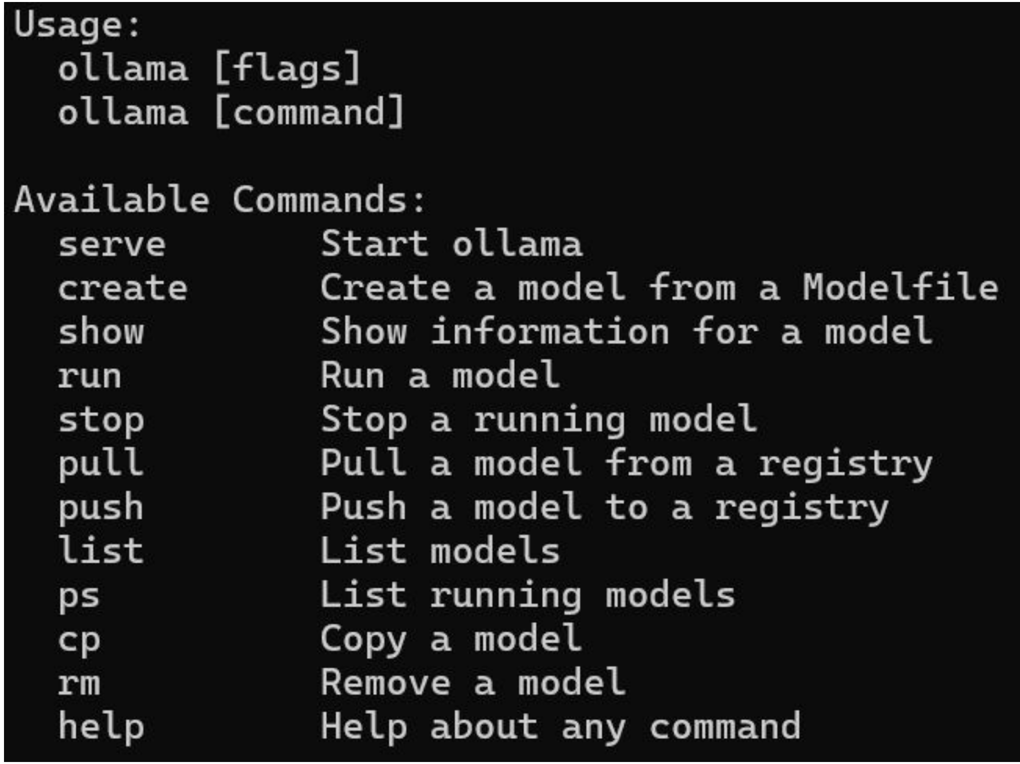
Pull a model: ollama pull [model]

Run model (CLI): ollama run [model]

List installed model: ollama list

……

Very simple!



```
Usage:
  ollama [flags]
  ollama [command]

Available Commands:
  serve       Start ollama
  create      Create a model from a Modelfile
  show        Show information for a model
  run         Run a model
  stop        Stop a running model
  pull        Pull a model from a registry
  push        Push a model to a registry
  list        List models
  ps          List running models
  cp          Copy a model
  rm          Remove a model
  help        Help about any command
```

Available Models

- You may find preconfigured models here: https://www.ollama.com/search
  - Those preconfigured models are tested and configured with Ollama and are confirmed to work
- You can also import models from GGUF on your own
- To change where a model is stored, edit the environment variable **OLLAMA_MODELS** to your desired directory
- Use models at your own discretion. If your computer is less powerful, use smaller models (1B/3B). Or experiment with larger ones(8B/70B/405B!) if you have the computing resources. (Don’t burn your laptop :P)

Ollama applications

Aside from what we are showing today, Ollama can be connected to many application:

- Open WebUI provide ease-to-use GUI, with a ton of customizable options
- Ollama is compatible with OpenAI API, meaning it can practically run any application that support OpenAI API, as long as you replace the endpoint base_url to '{Ollama Address}/v1', and api_key to 'ollama'

Ollama API

API Usage

When you start up ollama (ollama serve / ollama run), the API endpoint is available at **127.0.0.1:11434** (You can change by setting environment variable **OLLAMA_HOST**)

There are many endpoints defined: (https://github.com/ollama/ollama/blob/main/docs/api.md)

/api/chat: Chat

/api/create: Create a model

/api/generate: Completion

/api/embed: Generate embedding

Chat with Ollama using API

basic structure:

chat_request := {"model": model_name,"messages": messages, “stream”: True|False}

messages := [] | message + messages

message := {"role": role, "content": content}

role := "system" | "user" | "assistant" | "tool”

content := string

model_name := string

Chat with Ollama using API

- content: the content of the conversation.
- messages: must contain the **entire conversation**! (including system prompt, user, tools and assistant response)
- stream: True if you want streaming output (it will look like the LLM is talking 1 word at a time)
- Example:

```
'{ "model": "llama3.2","stream": false,"messages": [

{"role": "system","content": "reply owo to the user."},

{"role":user","content": "hello!"}

] }'
```

The “system” role is used for **system prompts** at the start of the conversation, use it to modify the agent’s behavior:)

Chat with Ollama using API

- POST the data to [API_ENDPOINT]/api/chat !
- Response:

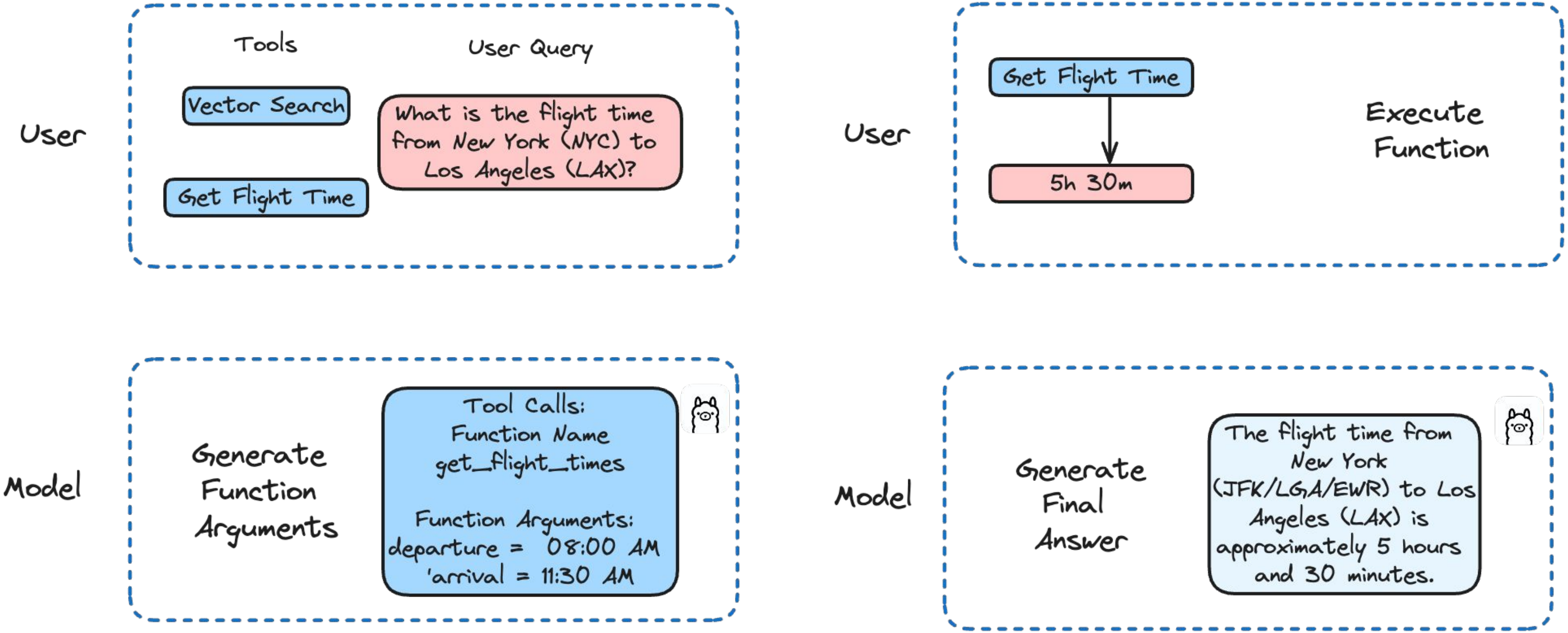
```
{"model":"llama3.2","created_at":"2025-05-06T06:58:21.5744552Z","message":{"role":"assistant","content":"owo"},"done_reason":"stop","done":true,"total_duration":290205000,"load_duration":15897800,"prompt_eval_count":32,"prompt_eval_duration":45812300,"eval_count":2,"eval_duration":227426400}
```

- You can see the message in the “message” key.
- Add the message to the messages parameter in your next request to keep the conversation going!
- To close the instance(so it wouldn’t consume your resources), send the request with **“keep_alive”: 0**

Tool Calling: Concepts

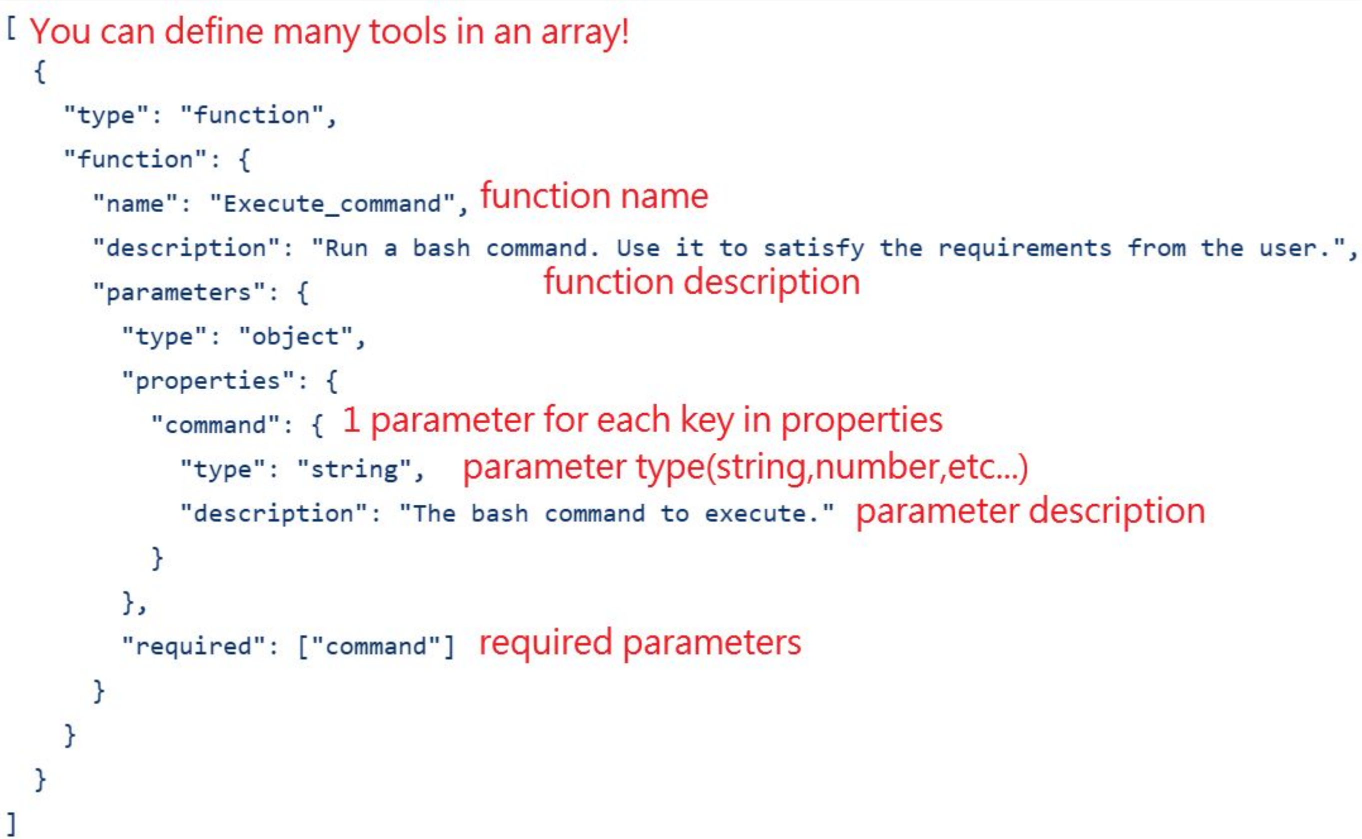
- Can LLM do more than just talking? We can give LLM some tools to use:D
- How tool calling works:
  a. First we define a tool function. (For example, let's say a function "get_weather" that looks up the weather.) We add the definition to the request so the LLM recognizes it.
  b. When the LLM needs the tool (ex. the user asks "what's the weather?"), it sends a tool call back to the program.
  c. The program processes the tool call and execute the tool with the requested parameters.
  d. The program returns the results back to the LLM.
  e. The LLM sees the results and tell the user answer.

Tool Calling: Concepts



(Source: https://zilliz.com/blog/function-calling-ollama-llama-3-milvus)

Defining a Tool



```
[ You can define many tools in an array!
  {
    "type": "function",
    "function": {
      "name": "Execute_command", function name
      "description": "Run a bash command. Use it to satisfy the requirements from the user.",
      "parameters": {                    function description
        "type": "object",
        "properties": {
          "command": { 1 parameter for each key in properties
            "type": "string",   parameter type(string,number,etc...)
            "description": "The bash command to execute." parameter description
          }
        },
        "required": ["command"] required parameters
      }
    }
  }
]
```

Tool calling: User Input

- Add the “tools” key with the tool definitions on the top level to tell LLM what tools it can use.
- Example: The user wants to know the result of factorize 21. He asks LLM and also hint it he has a tool called “factor”

```
{  "messages":[{"role":"user","content":"factorize 21."}],

   "model":"llama3.2","stream":false,

   "tools":[

        {"type":"function","function":{"name":"factor","description":"input a number, factorize the number","parameters":{"type":"object","properties":{"x":{"type":"number","description":"the number to factorize."}}},"required":["x"]}}

        ]

}
```

Tool calling: Function Call

- LLM’s Response: LLM requested to run “factor” with argument x=“21”

  ```
  {"model":"llama3.2","created_at":"2025-05-08T15:54:44.9630005Z",
  "message":{"role":"assistant","content":"","tool_calls":[{"function":{"name":"factor","arguments":{"x":21}}}]},
  "done_reason":"stop","done":true,"total_duration":686282200,"load_duration":20519400,"prompt_eval_count":164,"prompt_eval_duration":210104000,"eval_count":16,"eval_duration":454617000}
  ```

- The response message now contains no content and a new array: **tool_calls**, which is what the LLM wants to run. You can extract the tool name (.function.name) and arguments (.function.arguments) for each call!
- Message with **role:“tool”** is used to send the results back to LLM.
- Let’s provide LLM the results we executed and keep the conversation going!

Tool calling: Function Response

- New request: The user/system tell LLM the executing result for “factor”

```
{"messages":[
	{"role":"user","content":"factorize 21."},
	{"role":"assistant","content":"","tool_calls":[{"function":{"name":"factor","arguments":{"x":21}}}]},
	{"role":"tool","content":"21=2*14"}
	],"model":"llama3.2","stream":false,"tools":[{"type":"function","function":{"name":"factor","description":"input a number, factorize the number","parameters":{"type":"object","properties":{"x":{"type":"number","description":"the number to factorize."}}},"required":["x"]}}]
}
```

- Remember to include the message from the assistant as well!

Tool calling: Model Output

- Response:

```
{"model":"llama3.2","created_at":"2025-05-08T16:09:03.7319809Z",

"message":{"role":"assistant","content":"The result of factoring 21 is that it can be
expressed as the product of two numbers: 2 and 14, or in other words, 21 = 2 * 11."},

"done_reason":"stop","done":true,"total_duration":1182150800,"load_duration":18661800,"pro
mpt_eval_count":92,"prompt_eval_duration":9384000,"eval_count":41,"eval_duration":11523984
00}
```

- The LLM recognizes and uses the tool correctly!
- Let’s use the LLM to conquer the challenging tasks in NASA :D

Prompting

Some prompting strategies you might need to improve your agent :D

1. Chain Of Thought: "...... Let's think step by step."
2. Role playing: "You are an Expert in PowerShelll, ..."
3. Few-Shot Prompting: Provide some examples!

   "Query: List my files in my directory. Response: ls -al

   Query: Free up my disk space. Response: rm -rf / ......"

4. Formatted prompt: To not confuse instructions with inputs!

   "Translate the following bash script to powershell: <INPUT>[input]</INPUT>

   OUTPUT:"

Objectives

Lab

- You are required to implement a simple *natural language shell*™
- Given any natural language input, the program should be able to
  - come up with a clip of code executable by your OS’ shell (i.e. Bash, PowerShell, etc.)
  - Execute it on your behalf
  - Provide a short clip of explanation interpreting the output of the generated code
- Some similar existing example includes Claude Code, OpenAI Codex, and AIChat

Lab

- Don’t need to be general! You can come up with any LLM applications as long as it actually interacts with the shell.
- Some examples:
  - Docker Helper
  - KVM Installer
  - LDAP Searcher
  - Security Investigator
  - Network Wizard
  - Process Manager
  - etc…

Requirement - 1

- Use your own environment
  - Most computers should be able to run this smoothly
- For models, you can use any LLM model suitable for your environment
  - For this very simple task, llama3.2:3b should be enough, but you are encouraged to try other ones!
  - Be careful not to burn your computer! We feel like 8B models should be the maximum for computers with 8GB RAM, for example.
- If you don't have any capable computer, use 204 Computers
- For security reasons, **DO NOT USE NASAWS and CSIE WORKSTATION FOR THIS LAB**

Requirement - 2

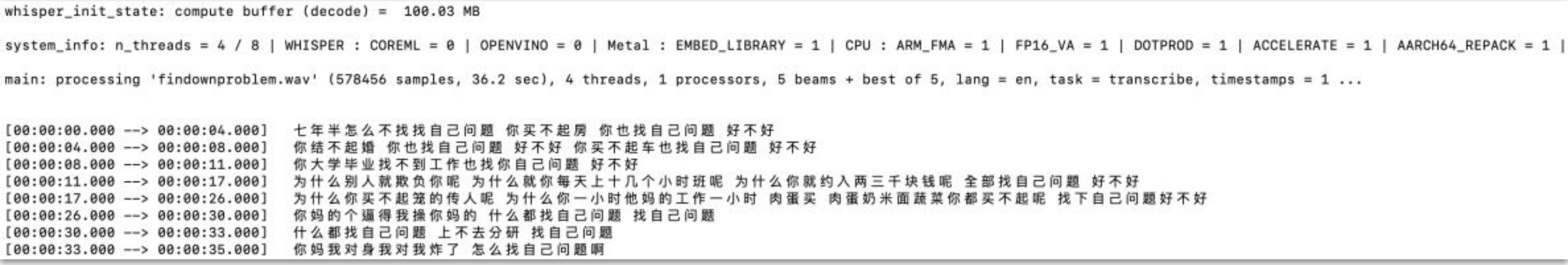
- Use the provided sample skeleton code to integrate with the API, or you may also use any programming language that you prefer
- For prompt, feel free to come up with anything as long as the prompt does not contain the explicit answers ;)
    - Examples are fine! Just make sure the LLM can generalize.

Tips

- Look closely to the JSON structure :)
- Look out on what you say as your computer is at stakes :P
- For bash users: **`jq`** is a great tool parsing JSON. As for Powershell Users: we can use the standard .NET object(Dict **`@{}`** / Array **`@()`**) and **`ConvertTo-Json/ConvertFrom-Json`** :D
- To send HTTP request, use **`curl`** or **`Invoke-WebRequest`**
- You might need to return shell errors to the model in case it got wrong. For bash a simple **`2>&1`** works, for PowerShell you might need a try-catch block and **`$_.Exception.Message`** to catch the error message.

Extras

- Some nice things to add (No bonus points btw :P)
  - Security: Prevent the LLM from executing weird stuff = =(Perhaps use an isolated context?)
  - Interactive input: Can the LLM interact with another interactive shell? (ex. LLM operates an entire VM by itself :O)
  - ASR: You can use whisper AI to perform speech recognition! (We recommend its CPP port)



```
whisper_init_state: compute buffer (decode) =  100.03 MB

system_info: n_threads = 4 / 8 | WHISPER : COREML = 0 | OPENVINO = 0 | Metal : EMBED_LIBRARY = 1 | CPU : ARM_FMA = 1 | FP16_VA = 1 | DOTPROD = 1 | ACCELERATE = 1 | AARCH64_REPACK = 1 |

main: processing 'findownproblem.wav' (578456 samples, 36.2 sec), 4 threads, 1 processors, 5 beams + best of 5, lang = en, task = transcribe, timestamps = 1 ...


[00:00:00.000 --> 00:00:04.000]   七年半怎么不找找自己问题 你买不起房 你也找自己问题 好不好
[00:00:04.000 --> 00:00:08.000]   你结不起婚 你也找自己问题 好不好 你买不起车也找自己问题 好不好
[00:00:08.000 --> 00:00:11.000]   你大学毕业找不到工作也找你自己问题 好不好
[00:00:11.000 --> 00:00:17.000]   为什么别人就欺负你呢 为什么就你每天上十几个小时班呢 为什么你就约入两三千块钱呢 全部找自己问题 好不好
[00:00:17.000 --> 00:00:26.000]   为什么你买不起篮的传人呢 为什么你一小时他妈的工作一小时 肉蛋买 肉蛋奶米面蔬菜你都买不起呢 找下自己问题好不好
[00:00:26.000 --> 00:00:30.000]   你妈的个逼得我操你妈的 什么都找自己问题 找自己问题
[00:00:30.000 --> 00:00:33.000]   什么都找自己问题 上不去分研 找自己问题
[00:00:33.000 --> 00:00:35.000]   你妈我对身我对我炸了 怎么找自己问题啊
```

Deliverable & Grading

Please Submit the following to NTU COOL:

- Short video recording of your script working
  - You can be creative on what to tell your LLM to do, as long as it is capable of running command and interpreting it (as outlined in P.20)
- The script itself
  - May be of any programming language

Deadline: 2025/5/18 23:59

Demo