Local
Large Language Model
with Python

Local LLM through Ollama

Ollama is an open-source tool

- enable an LLM model to be run directly on our local machine
  - which can be download (‘pull’) from the Ollama ready-to-use LLM library

Ollama runs as a daemon (a server program running at the background providing service upon request), which we can interact with the LLM through

- command line interface
- Python programs that uses HTTP requests to Server’s endpoint (@ localhost:11434 port) using its REST.API
- Python programs that uses Ollama Python library provide functions that allow Python applications to interface to the Ollama server (through REST.API)

Ollama Python Library

Ollama Python library provide functions such as generate(), chat(), etc

- internally issue HTTP requests based n REST API to the Ollama server
  - E.g. HTTP's POST request

Ollama server processes the request

- E.g., loads the model specified, perform inference
- sends back an HTTP response with the processed data

Ollama Python library receives the response

- parses the data and send it to your Python application

Aside: Installing Ollama

Download the OllamaSetup.exe installation program from ollama.com

- run the .exe file and follow the installation wizard

Ollama will start automatically when installation completes

```
C:\Users\aschvun>ollama run deepseek-r1:1.5b
pulling manifest
pulling aabd4debf0c8: 100%                                                1.1 GB
pulling c5ad996bda6e: 100%                                                 556 B
pulling 6e4c38e1172f: 100%                                                1.1 KB
pulling f4d24e9138dd: 100%                                                 148 B
pulling a85fe2a2e58e: 100%                                                 487 B
verifying sha256 digest
writing manifest
success
>>> Send a message (/? for help)
```

Test run the LLM model (e.g., deepseek-r1:1.5b)

```
>>> Hello
Hello! How can I assist you today? 😊

>>> Send a message (/? for help)
```

Aside: Installing Python Library for Ollama

Next install the Ollama Python library

```
C:\Users\aschvun>py -m pip install ollama
Collecting ollama
  Downloading ollama-0.6.0-py3-none-any.whl.metadata (4.3 kB)
Collecting httpx>=0.27 (from ollama)
  Downloading httpx-0.28.1-py3-none-any.whl.metadata (7.1 kB)
Collecting pydantic>=2.9 (from ollama)
```

Examples of functions available in the library

- generate(model, prompt)
- chat(model, messages)

LLM with Python Application

A basic Python program (e.g. using IDLE) to interact with the LLM

```
from ollama import generate
model = "deepseek-r1:1.5b"

response = generate(model, 'What does 30*5 equal to?')
print("DeepSeek:", response["response"])
```

```
>>>
    = RESTART: C:\Users\aschvun\AppData\Local\Programs\Python\Python311\AI6120 - DeepSeek0.py
    DeepSeek: <think>
    To calculate 30 multiplied by 5, I first recognize that 30 is a multiple of 10.

    I then break down the multiplication into two parts: multiplying 3 by 5 and then adding a

### Final Answer:

\[
30 \times 5 = \boxed{150}
\]
```

Generate()

```
from ollama import generate
model = "deepseek-r1:1.5b"

# Define a system prompt
system_prompt = "You are a helpful AI."

while True:
    user_prompt = input("You: ")
    if not user_prompt:
        print('Bye!')
        break #exit on empty input

    response = generate(
                    model=model,
                    system=system_prompt,
                    prompt=user_prompt)
    print("DeepSeek:", response["response"])
```

Generate() with Streaming

Using Streaming to have a more interactive UX

- show the chunk of responses as they are being generated

```
from ollama import generate
model = "deepseek-r1:1.5b"

while True:
    myprompt = input("You: ")
    if myprompt.lower() == 'exit':
        break

    for chunk in generate(model=model, prompt=myprompt, stream=True):
        print(chunk["response"], end='', flush = True)
```

Chat()

Using Chat with system prompts to set context and instructions for the model

```
from ollama import chat
model = "deepseek-r1:1.5b"

# Define a system prompt
system_prompt = "You speaks and sounds like a waiter in a restaurant."

# Chat with a system prompt
response = chat(model,
                messages=[
                    {'role': 'system', 'content': system_prompt},
                    {'role': 'user', 'content': 'What do you recommend for today menu?'}
                ])
print(response.message.content)
```

The REST.API interface

We can also use the REST API in the Python program

- receive structured responses in JSON format.
- but need to handle a lot of details c.f. using Ollama Python library

```python
import requests
import json
url = "http://localhost:11434/api/chat" #Chat endpoint
model = 'deepseek-r1:1.5b'

payload = {
    "model": model,
    "messages": [
        {"role": "system", "content": "You are a Python assistant."},
        {"role": "user", "content": "Write a function that reverses a string."}
    ]
}

response = requests.post(url, json=payload, stream=True)

for line in response.iter_lines():
    if line:
        data = json.loads(line)
        print(data.get("message", {}).get("content", ""), end="")
```