Alternative Ollama models

The `llama3` in the `ollama run llama3` command refers to the instruction-fine-tuned 8-billion-parameter Llama 3 model. Using Ollama with the `llama3` model requires approximately 16 GB of RAM. If your machine does not have sufficient RAM, you can try using a smaller model, such as the 3.8-billion-parameter `phi3` model via `ollama run llama3`, which only requires around 8 GB of RAM.

For more powerful computers, you can also use the larger 70-billion-parameter Llama 3 model by replacing `llama3` with `llama3:70b`. However, this model requires significantly more computational resources.

Once the model download is complete, we are presented with a command-line interface that allows us to interact with the model. For example, try asking the model, "What do llamas eat?"

```
>>> What do llamas eat?
Llamas are ruminant animals, which means they have a four-chambered
stomach and eat plants that are high in fiber. In the wild,
llamas typically feed on:

1. Grasses: They love to graze on various types of grasses, including tall
grasses, wheat, oats, and barley.
```

Note that the response you see might differ since Ollama is not deterministic as of this writing.

You can end this `ollama run llama3` session using the input `/bye`. However, make sure to keep the `ollama serve` command or the Ollama application running for the remainder of this chapter.

The following code verifies that the Ollama session is running properly before we use Ollama to evaluate the test set responses:

```
import psutil

def check_if_running(process_name):
    running = False
    for proc in psutil.process_iter(["name"]):
        if process_name in proc.info["name"]:
            running = True
            break
    return running

ollama_running = check_if_running("ollama")

if not ollama_running:
    raise RuntimeError(
        "Ollama not running. Launch ollama before proceeding."
)
print("Ollama running:", check_if_running("ollama"))
```

Ensure that the output from executing the previous code displays `Ollama running: True`. If it shows `False`, verify that the `ollama serve` command or the Ollama application is actively running.

Running the code in a new Python session

If you already closed your Python session or if you prefer to execute the remaining code in a different Python session, use the following code, which loads the instruction and response data file we previously created and redefines the `format_input` function we used earlier (the `tqdm` progress bar utility is used later):

```
import json
from tqdm import tqdm

file_path = "instruction-data-with-response.json"
with open(file_path, "r") as file:
    test_data = json.load(file)

def format_input(entry):
    instruction_text = (
        f"Below is an instruction that describes a task. "
        f"Write a response that appropriately completes the request."
        f"\n\n### Instruction:\n{entry['instruction']}"
    )


    input_text = (
        f"\n\n### Input:\n{entry['input']}" if entry["input"] else ""
    )
    return instruction_text + input_text
```

An alternative to the `ollama run` command for interacting with the model is through its REST API using Python. The `query_model` function shown in the following listing demonstrates how to use the API.

Listing 7.10 Querying a local Ollama model

```
import urllib.request

def query_model(
    prompt,
    model="llama3",
    url="http://localhost:11434/api/chat"
):
    data = {                    <-- Creates the data payload as a dictionary
        "model": model,
        "messages": [
            {"role": "user", "content": prompt}
        ],
        "options": {            <-- Settings for deterministic responses
            "seed": 123,
```

```
            "temperature": 0,
            "num_ctx": 2048
        }
    }

    payload = json.dumps(data).encode("utf-8")
    request = urllib.request.Request(
        url,
        data=payload,
        method="POST"
    )

    request.add_header("Content-Type", "application/json")

    response_data = ""
    with urllib.request.urlopen(request) as response:
        while True:
            line = response.readline().decode("utf-8")
            if not line:
                break
            response_json = json.loads(line)
            response_data += response_json["message"]["content"]

    return response_data
```

Converts the dictionary to a JSON-formatted string and encodes it to bytes

Creates a request object, setting the method to POST and adding necessary headers

Sends the request and captures the response

Before running the subsequent code cells in this notebook, ensure that Ollama is still running. The previous code cells should print `"Ollama running: True"` to confirm that the model is active and ready to receive requests.

The following is an example of how to use the `query_model` function we just implemented:

```
model = "llama3"
result = query_model("What do Llamas eat?", model)
print(result)
```

The resulting response is as follows:

```
Llamas are ruminant animals, which means they have a four-chambered
stomach that allows them to digest plant-based foods. Their diet
typically consists of:

1. Grasses: Llamas love to graze on grasses, including tall grasses,
short grasses, and even weeds.
...
```

Using the `query_model` function defined earlier, we can evaluate the responses generated by our fine-tuned model that prompts the Llama 3 model to rate our fine-tuned model's responses on a scale from 0 to 100 based on the given test set response as reference.