Open in app ↗

Search

3-ways to Set up LLaMA 2 Locally on CPU (Part 2 — Ollama)

Antoine Frd · Follow
6 min read · Jan 24, 2024

Listen Share More

Load LlaMA 2 model with `Ollama` 🚀

Install dependencies for running Ollama locally

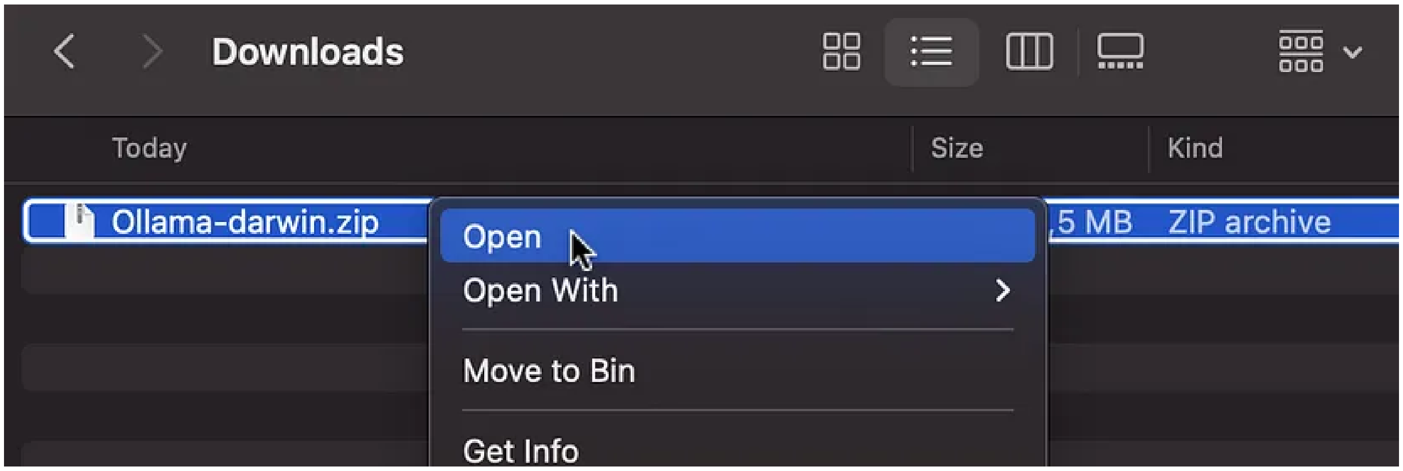
Ollama allows you to run open-source large language models, such as Llama 2, locally.

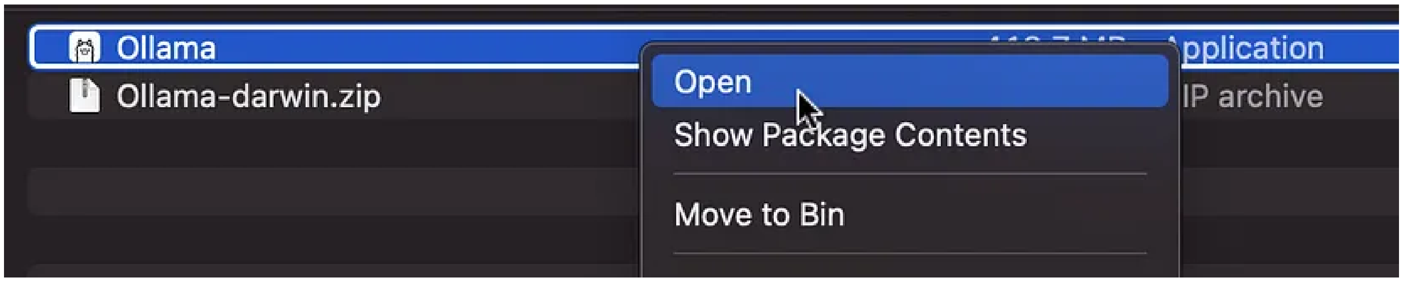
Ollama bundles model weights, configuration, and data into a single package, defined by a **ModelFile**. A ModelFile is the blueprint to create and share models with Ollama.

For a complete list of supported models and model variants, see the Ollama model library.

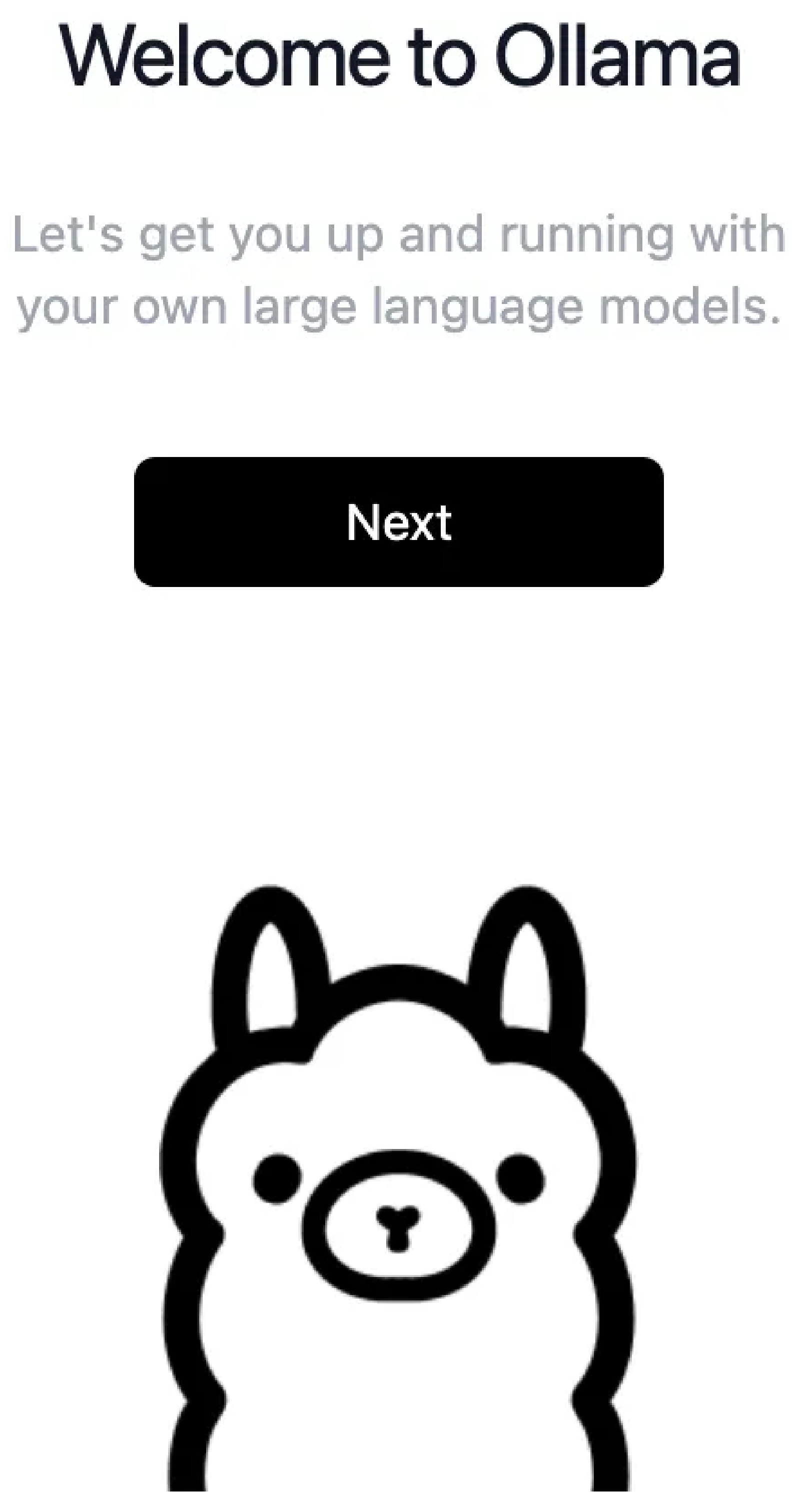
1° First, Download the app.

2° Open the zip file and run the app.





Then, you should see the welcome page

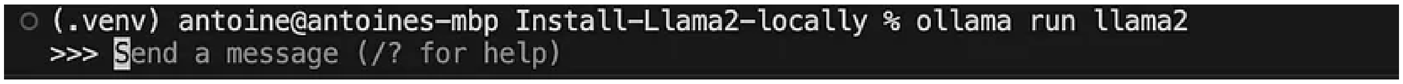

Welcome to Ollama
Let's get you up and running with your own large language models.
Next

3° Follow the instructions to install Ollama on your local machine.

Ollama CLI

You can run your first model using

```
ollama run llama2
```

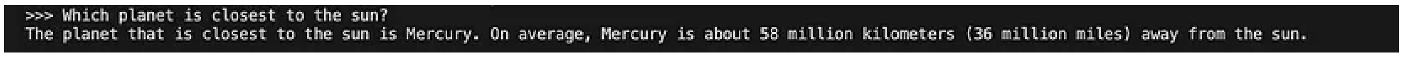


```
○ (.venv) antoine@antoines-mbp Install-Llama2-locally % ollama run llama2
  >>> Send a message (/? for help)
```

> *Note:* If `ollama run` detects that the model hasn't been downloaded yet, it will automatically trigger the `ollama pull` command. However, if you prefer to only download the model without activating it, you can use the command `ollama pull llama2`.

- When the app is running, the model is automatically served on `localhost:11434`

Then, we can write our prompt and press Enter to get a response from the model



```
>>> Which planet is closest to the sun?
The planet that is closest to the sun is Mercury. On average, Mercury is about 58 million kilometers (36 million miles) away from the sun.
```

Note: With Langchain, we don't require a PromptTemplate since Ollama integrates it:

```
% ollama show --modelfile llama2

# Modelfile generated by "ollama show"
# To build a new Modelfile based on this one, replace the FROM line with:
# FROM llama2:latest

FROM /Users/antoine/.ollama/models/blobs/sha256:8934d96d3f08982e95922b2b7a2c626a1fe873d7c3b(
TEMPLATE """[INST] <<SYS>>{{ .System }}<</SYS>>

{{ .Prompt }} [/INST]
"""
PARAMETER stop "[INST]"
PARAMETER stop "[/INST]"
PARAMETER stop "<<SYS>>"
PARAMETER stop "<</SYS>>"
```

Note: We can list all the command for Ollama using

```
ollama -h #(or --help)
```

```
Large language model runner

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
  pull        Pull a model from a registry
  push        Push a model to a registry
  list        List models
  cp          Copy a model
  rm          Remove a model
  help        Help about any command

Flags:
  -h, --help      help for ollama
  -v, --version   Show version information
```

Example:

```
ollama list
```

```
NAME            ID              SIZE    MODIFIED
llama2:latest   78e26419b446    3.8 GB  2 hours ago
```

Model names follow a `model:tag` format, where `model` can have an optional namespace such as `example/model` . Some examples are `orca-mini:3b-q4_1` and `llama2:70b` . The **tag** is optional and, if not provided, will default to `latest` . The tag is used to identify a specific version.

We only have the Llama 2 model locally because we have installed it using the command `run` . Ollama supports a list of open-source models available on ollama.ai/library.

As mentionned here, The command `ollama run llama2` run the **Llama 2 7B Chat** model.

By default, Ollama uses **4-bit quantization**. To try other quantization levels, please try the other tags. The number after the `q` represents the number of bits used for quantization (i.e. `q4` means 4-bit quantization). The higher the number, the more accurate the model is, but the slower it runs, and the more memory it requires.

Memory requirements

- 7b models require at least 8GB of RAM
- 13b models require at least 16GB of RAM
- 70b models require at least 64GB of RAM

Model variants

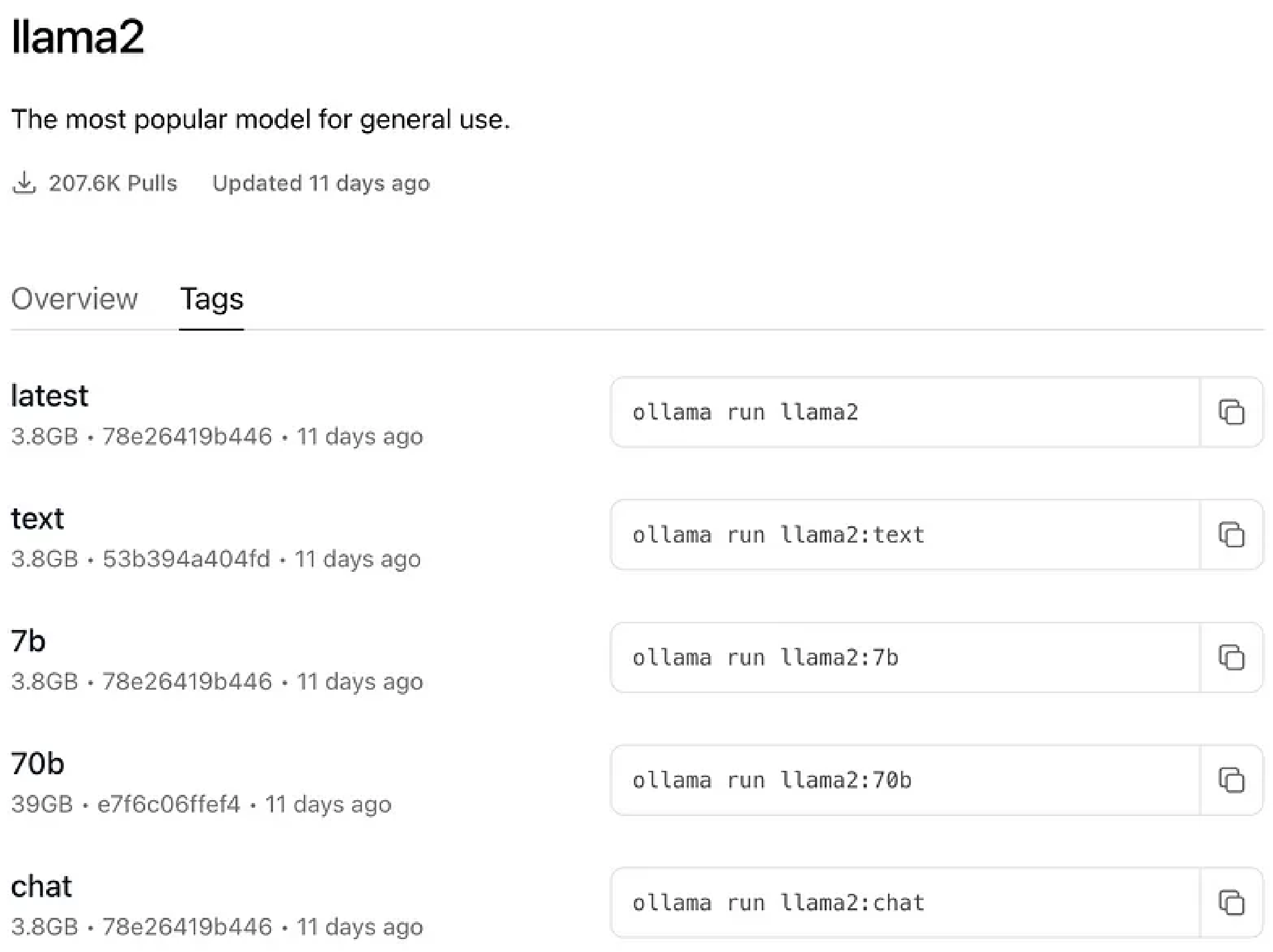
- **Chat** is fine-tuned for chat/dialogue use cases. By **default** in Ollama. (Tagged as -*chat* in the tags tab).

Example: `ollama run llama2`

- **Pre-trained** is without the chat fine-tuning. (Tagged as -*text* in the tags tab).

Example: `ollama run llama2:text`

Note: You can select the model you want with the appropriate tag:


llama2

The most popular model for general use.

207.6K Pulls Updated 11 days ago

Overview Tags

latest
3.8GB • 78e26419b446 • 11 days ago

```
ollama run llama2
```

text
3.8GB • 53b394a404fd • 11 days ago

```
ollama run llama2:text
```

7b
3.8GB • 78e26419b446 • 11 days ago

```
ollama run llama2:7b
```

70b
39GB • e7f6c06ffef4 • 11 days ago

```
ollama run llama2:70b
```

chat
3.8GB • 78e26419b446 • 11 days ago

```
ollama run llama2:chat
```

Find the Llama 2's tags tab here.

Ollama API

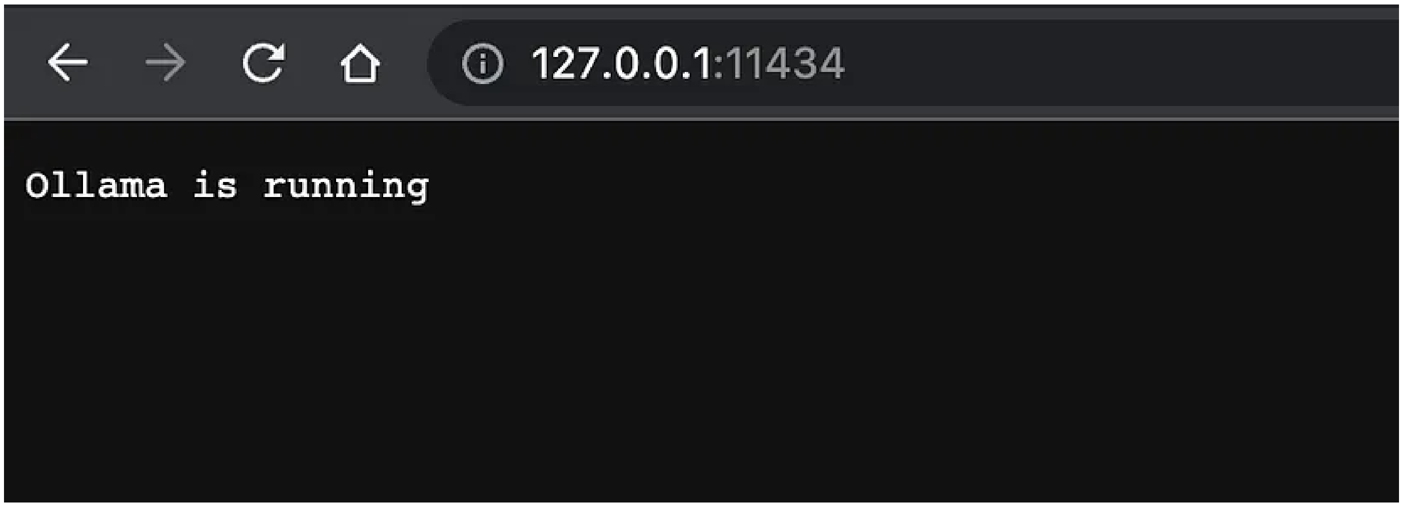
The Ollama API provides a simple and consistent interface for interacting with the models:

- Easy to integrate — The installation process is streamlined and user-friendly.
- Flexibility — Adjusting the model's parameters or using custom model by bringing a GGUF or GGML file.
- Data Privacy — Keeps your data in-house, enhancing security.

In this article, we'll discover the simplicity of using Llama 2 with the Ollama API, avoiding intricate configurations.

1° First, initiate the Ollama server:

```
ollama serve
```



```
Ollama is running
```

Note: If you encounter an error message like `"Error: listen tcp 127.0.0.1:11434: bind: address already in use"`, it indicates the Ollama instance is already running by default, and you can proceed to the next step.

2° Use the Llama 2 model

Request

```
curl -X POST <http://localhost:11434/api/generate> -d '{
 "model": "llama2",
 "prompt":"Which planet is closest to the sun?"
}'

#We pass the body of the POST message to Curl with the -d (or --data command-line) option
```

Response

Streaming is enabled by default (`"stream": true`). A stream list of JSON objects is returned:

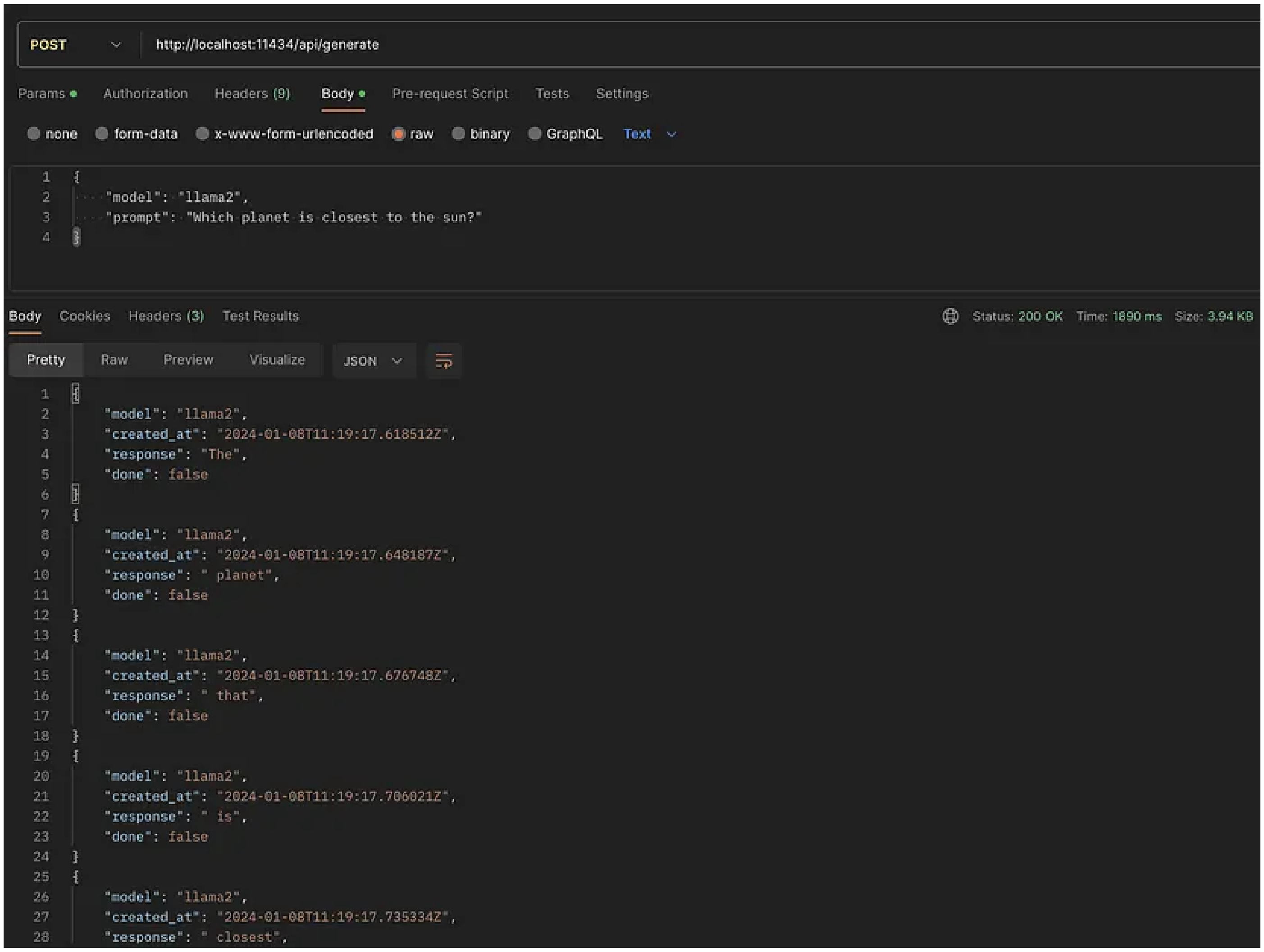
```
{
    "model": "llama2",
    "created_at": "2024-01-08T11:19:17.618512Z",
    "response": "The",
    "done": false
}
```

The final response in the stream also includes additional data about the generation:

```
{
        "model":"llama2",
        "created_at":"2024-01-08T11:22:13.951744Z",
        "response":"",
        "done":true,
        "context":[518,25580,29962,3532,14816,29903,29958,5299,829,14816,29903,6778,13,13,8
        "total_duration":1433785958,
        "load_duration":5523791,
        "prompt_eval_duration":488739000,
        "eval_count":36,
        "eval_duration":937119000
}
```

- `total_duration` : time spent generating the response
- `load_duration` : time spent in nanoseconds loading the model
- `prompt_eval_count` : number of tokens in the prompt
- `prompt_eval_duration` : time spent in nanoseconds evaluating the prompt
- `eval_count` : number of tokens the response
- `eval_duration` : time in nanoseconds spent generating the response
- `context` : an encoding of the conversation used in this response, this can be sent in the next request to keep a conversational memory
- `response` : empty if the response was streamed, if not streamed, this will contain the full response

Example with Postman


POST http://localhost:11434/api/generate

Body: raw, Text

```
{
    "model": "llama2",
    "prompt": "Which planet is closest to the sun?"
}
```

Status: 200 OK Time: 1890 ms Size: 3.94 KB

```
{
    "model": "llama2",
    "created_at": "2024-01-08T11:19:17.618512Z",
    "response": "The",
    "done": false
}
{
    "model": "llama2",
    "created_at": "2024-01-08T11:19:17.648187Z",
    "response": " planet",
    "done": false
}
{
    "model": "llama2",
    "created_at": "2024-01-08T11:19:17.676748Z",
    "response": " that",
    "done": false
}
{
    "model": "llama2",
    "created_at": "2024-01-08T11:19:17.706021Z",
    "response": " is",
    "done": false
}
{
    "model": "llama2",
    "created_at": "2024-01-08T11:19:17.735334Z",
    "response": " closest",
```

In a future article, we'll take a closer look at the Ollama API and explore its features and capabilities in detail. You can find more informations in the API documentation.

Ollama with Langchain

There exists a Ollama LLM wrapper in Langchain,

```
from langchain_community.llms import Ollama
from langchain.callbacks.manager import CallbackManager
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler
```

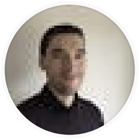
We enable Streaming with the `StreamingStdOutCallbackHandler` handler

```
llm = Ollama(
    model="llama2",
    num_gpu = 0, # To use CPU only
    callback_manager=CallbackManager([StreamingStdOutCallbackHandler()])
)
llm.invoke("Which planet is the closest to the sun?")
```

```
'The closest planet to the sun is Mercury. On average, Mercury is about 58 million kilometer
```

Congratulations! In this second article we've successfully installed Ollama and Langchain locally and use it with CPU. In the next article, we'll see how to install Llama 2 with Hugging Face. Stay tuned!

Llm Genrative Ai Llama 2 Ollama Langchain



Follow

Written by Antoine Frd

16 Followers

AI Engineer

More from Antoine Frd