How to run Llama 2 locally on CPU + serving it as a Docker container

Nikolay Penkov · Follow
8 min read · Oct 29, 2023

236 3

In today's digital landscape, the large language models are becoming increasingly widespread, revolutionizing the way we interact with information and AI-driven applications. What's more, the availability of open-source alternatives to their paid counterparts is empowering enthusiasts and developers to harness the power of these models without breaking the bank.

In this post, we'll explore one of the leading open-source models in this domain: Llama 2. In this tutorial you'll understand how to run Llama 2 locally and find out how to create a Docker container, providing a fast and efficient deployment solution for Llama 2. So, let's embark on this exciting adventure and unlock the world of large language models with Llama 2!

Photo by Josiah Farrow on Unsplash

LLama 2 was created by Meta and was published with an open-source license, however you have to ready and comply with the Terms and Conditions for the model. You can find out more about the models on the official page: https://ai.meta.com/llama/

Model size

The LLama 2 model comes in multiple forms. You are going to see 3 versions of the models **7B, 13B, and 70B,** where B stands for billion parameters. As you can guess, the larger the model, the better the performance, the greater the resource needs.

Model type

Besides the model size, you will see that there are 2 model types:

- **LLama 2**, the text completion version of the model, which doesn't have a specific prompt template
- **Llama 2 Chat,** the fine-tuned version of the model, which was trained to follow instructions and act as a chat bot. This version needs a specific prompt template in order to perform the best, which we are going to discuss below.

Getting the model

All Llama 2 models are available on **HuggingFace**. In order to access them you will have to apply for an access token by accepting the terms and conditions. Let's take for example **LLama 2 7B Chat**. After opening the page you will see a form where you can apply for model access. After your request is approved, you will be able to download the model using your HuggingFace access token.

In this tutorial we are interested in the CPU version of **Llama 2**. Usually big and performant Deep Learning models require high-end GPU's to be ran.

However, thanks to the excellent work of the community, we have **llama.cpp**, which does the magic and allows running **LLama** models solely on your CPU. How this magic is done will be discussed in a future post. For now, you have to know only that the **llama.cpp** applies a custom quantization approach to compress the models in a GGUF format. This reduces their size and resource needs.

Thanks to **The Bloke**, there are already pre-made models which can be used directly with the mentioned framework. Let's get our hands dirty and download the the **Llama 2 7B Chat GGUF model**. After opening the page download the llama-2–7b-chat.Q2_K.gguf file, which is the most compressed version of the 7B chat model and requires the least resources.

Python bindings for llama.cpp

We are going to write our code in python therefore we need to run the llama.cpp in a pythonic way. This was also considered by the community and exists as a project called llama-cpp-python which can be installed with:

```
pip install llama-cpp-python
```

Docker (Optional)

If you are going to deploy your model as a docker container you will need Docker running on your system. We are going to consider Docker Desktop in this tutorial so make sure you install it.

Running the model

Let's see how we can run the model by analyzing the following python script:

```
from llama_cpp import Llama

# Put the location of to the GGUF model that you've download from HuggingFace he
model_path = "**path to your llama-2-7b-chat.Q2_K.gguf**"

# Create a llama model
model = Llama(model_path=model_path)

# Prompt creation
system_message = "You are a helpful assistant"
user_message = "Generate a list of 5 funny dog names"

prompt = f"""<s>[INST] <<SYS>>
{system_message}
<</SYS>>
{user_message} [/INST]"""

# Model parameters
max_tokens = 100

# Run the model
output = model(prompt, max_tokens=max_tokens, echo=True)

# Print the model output
print(output)
```

As you can see, the compressed model is loaded with the python bindings library by simply passing the path to the GGUF file.

The model prompt is also very important. You can see some special tokens such as <s>, [INST] and <<SYS>>. For now, remember that they have to present and the prompt has to follow the given template. More information on their role will be given soon.

In the prompt there are two user inputs:

- The **system message** which can be used to instill specific knowledge or constraint to the LLM. Alternatively, it can be omitted and the model will follow the system message it was trained on.
- The **user message** which is the actual user prompt. Here you can define the concrete task that you want the model to do (e.g. code generation, or generating funny dog names in our case)

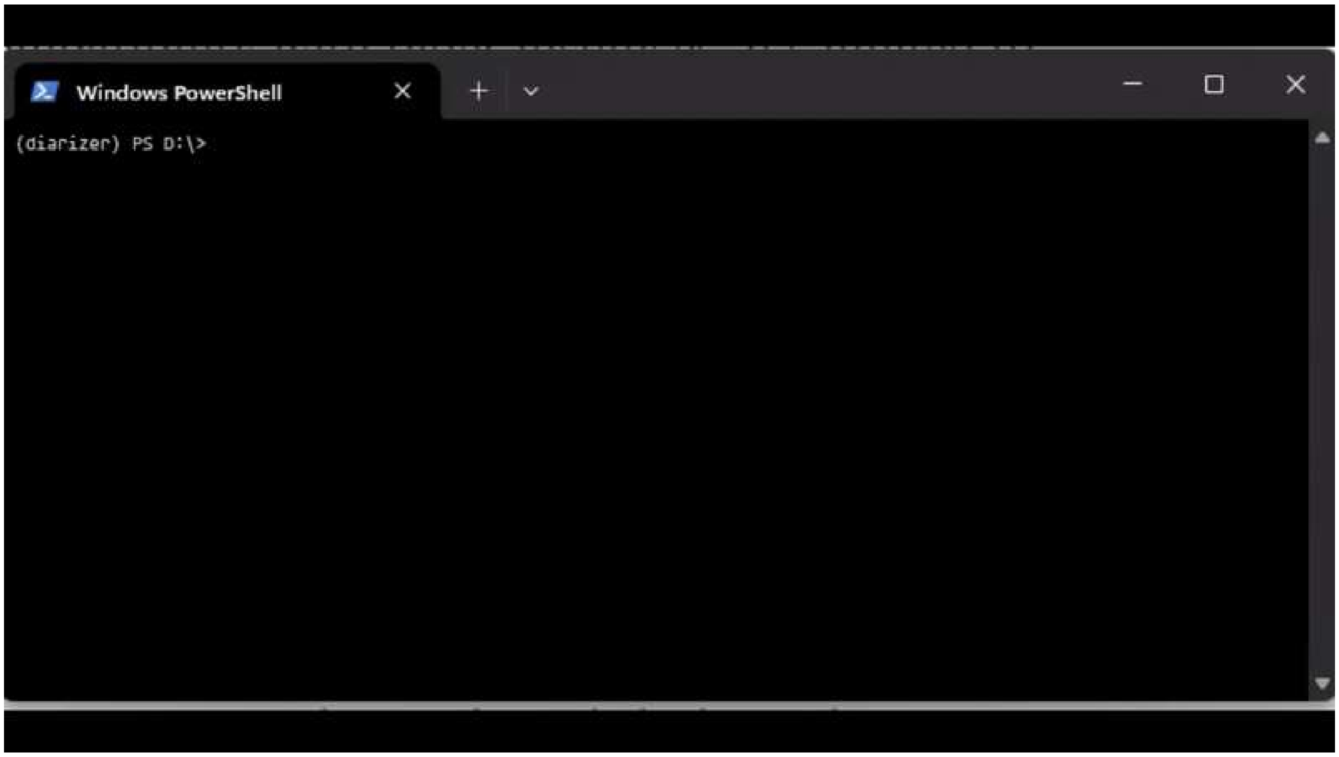
Lastly, the parameter **max_tokens** determines how many tokens the model will generate. For now think of tokens as number of words, but bare in mind that this is not always the case as some words can be presented as multiple tokens.

There are some other additional parameters which can be configured for more advanced cases, but they will be discussed in separate posts.

Now let’s save the code as **llama_cpu.py** and run it with:

```
python llama_cpu.py
```

Here is an example:



```
(diarizer) PS D:\>
```

As you can see from the experiment, the model output was:

```
{
   "id":"cmpl-078b3c61-8ced-4d7e-8fb1-688083f97a89",
   "object":"text_completion",
   "created":1698571747,
   "model":"D:\\models\\llama-2-7b-chat.Q2_K.gguf",
   "choices":[
      {
         "text":"<s>[INST] <<SYS>>\nYou are a helpful assistant\n<</SYS>>\nGener
         "index":0,
         "logprobs":"None",
         "finish_reason":"length"
      }
   ],
   "usage":{
      "prompt_tokens":38,
      "completion_tokens":100,
      "total_tokens":138
   }
}
```

The maximum number of tokens that we selected was not enough to complete the generation so the model cut it short. You have to consider the expected length of your prompt output and select the number of tokens more precisely.

(Optional) Serving the model as Docker container

As you can see, the model output provided by llama-cpp-python is very neat and can help us create fast a local model endpoint using Flask. Our goal is to invoke the model in a parametrized way so that we can dynamically select the number of maximum tokens, and system and user prompt. For this we are going to create an HTTP server that hosts our model and deploy the server as a Docker container.

Creating a serving endpoint

As a first step we need to install Flask to create a server:

```
pip install Flask
```

Now we can create a serving endpoint, which will accept **POST** requests on http://localhost:5000/llama and expect an input JSON with **max_tokens**, **system_message**, and **user_message** specified. Here is how this can be done:

```
from flask import Flask, request, jsonify
from llama_cpp import Llama

# Create a Flask object
app = Flask("Llama server")
```

```
    model = None

    @app.route('/llama', methods=['POST'])
    def generate_response():
        global model

        try:
            data = request.get_json()

            # Check if the required fields are present in the JSON data
            if 'system_message' in data and 'user_message' in data and 'max_tokens'
                system_message = data['system_message']
                user_message = data['user_message']
                max_tokens = int(data['max_tokens'])

                # Prompt creation
                prompt = f"""<s>[INST] <<SYS>>
                {system_message}
                <</SYS>>
                {user_message} [/INST]"""

                # Create the model if it was not previously created
                if model is None:
                    # Put the location of to the GGUF model that you've download fro
                    model_path = "**path to your llama-2-7b-chat.Q2_K.gguf**"

                    # Create the model
                    model = Llama(model_path=model_path)

                # Run the model
                output = model(prompt, max_tokens=max_tokens, echo=True)

                return jsonify(output)

            else:
                return jsonify({"error": "Missing required parameters"}), 400

        except Exception as e:
            return jsonify({"Error": str(e)}), 500

    if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5000, debug=True)
```

Save the code as **llama_cpu_server.py** and run it with:

```
python llama_cpu_server.py
```

This will start a local server on port 5000. You can make a POST request with cURL as follows (or use Postman instead 😳) :

```
curl -X POST -H "Content-Type: application/json" -d '{
  "system_message": "You are a helpful assistant",
  "user_message": "Generate a list of 5 funny dog names",
  "max_tokens": 100
}' http://127.0.0.1:5000/llama
```

You can play around with different input parameters to see how the model works.

Dockerizing the server

Now that we have a working Flask server that hosts our model, we can create a Docker container.

For this we have to build a Docker image that contains the model and the server logic:

```
# Use python as base image
FROM python

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY ./llama_cpu_server.py /app/llama_cpu_server.py
COPY ./llama-2-7b-chat.Q2_K.gguf /app/llama-2-7b-chat.Q2_K.gguf
```

```
# Install the needed packages
RUN pip install llama-cpp-python
RUN pip install Flask

# Expose port 5000 outside of the container
EXPOSE 5000

# Run llama_cpu_server.py when the container launches
CMD ["python", "llama_cpu_server.py"]
```

Save the specified configuration as “Dockerfile ”in the same folder of llama_cpu_server.py

NOTE: Make sure that the model file llama-2–7b-chat.Q2_K.gguf and the server file llama_cpu_server.py are in the same directory as the Dockerfile. Also make sure that the model path specified in llama_cpu_server.py is relative to the llama_cpu_server.py file or the container build and run will fail!

Afterwards you can build and run the Docker container with:

```
docker build -t llama-cpu-server .
docker run -p 5000:5000 llama-cpu-server
```

The Dockerfile will creates a Docker image that starts a container with port 5000 exposed to the outside world (i.e. locally in your network). You can now make POST requests to the same endpoint as previously:

```
curl -X POST -H "Content-Type: application/json" -d '{
  "system_message": "You are a helpful assistant",
  "user_message": "Generate a list of 5 funny dog names",
  "max_tokens": 100
}' http://127.0.0.1:5000/llama
```

Congrats, you have your own locally hosted Llama 2 Chat model now, which you can use for any of your needs 🙌.

Get the code from Github repo for this tutorial:
https://github.com/penkow/llama-docker

Running in production

This tutorial showed how to deploy **Llama 2 locally as Docker container.** However, **this is not a production ready code.** As you saw, we are running **Flask in debug mode**, we are **not exposing** all model parameters such as **top_p**, **top_k**, **temperature** and **n_ctx.** The model is not utilizing the full chat capabilities because there is no user session implemented, and previous context will be lost at every new request. Another thing to mentions is that this solution is **not scalable** in its current form, and consequent requests will break the server.

All this is important if you want to host the Llama 2 models in a production environment, but can't be covered in a single post.

For any questions or feedback use my website to contact me,

or find me on LinkedIn: https://www.linkedin.com/in/penkow/

Happy Coding :) !

Photo by Raspopova Marina on Unsplash

Large Language Models · AI · Llm · Machine Learning · Llama 2

Written by Nikolay Penkov

Follow

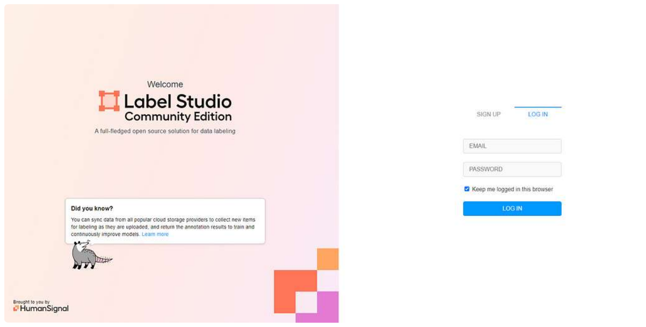
63 Followers

https://penkow-dev.web.app

More from Nikolay Penkov

Nikolay Penkov

How to deploy LLama 2 as an AWS Lambda function for scalable...


Welcome
Label Studio
Community Edition
A full-fledged open source solution for data labeling
Brought to you by HumanSignal
SIGN UP
LOG IN
EMAIL
PASSWORD
Keep me logged in this browser
LOG IN

Did you know?

You can sync data from all popular cloud storage providers to collect new items for labeling as they are uploaded, and return the annotation results to train and continuously improve models. Learn more

Nikolay Penkov

Creating your own dataset for LLM training using Label Studio

AWS Lambda is a powerful serverless computing service, offering a myriad of...

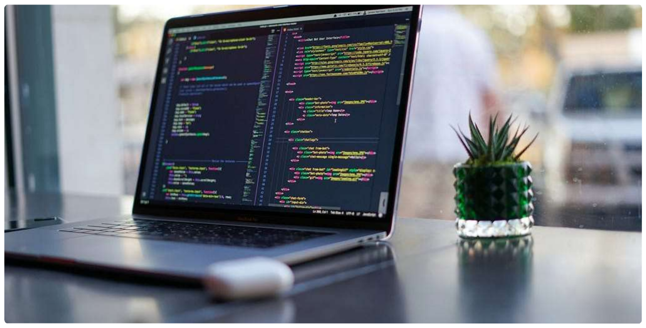
6 min read · Oct 31, 2023

29 3

In the realm of Language Model Training, the critical task of data labeling takes center...

4 min read · Nov 10, 2023

14



Nikolay Penkov

RAG basics using a self hosted OpenAI compatible LLM server

Advanced AI language models , such as OpenAI's ChatGPT and Google's Gemini hav...

9 min read · Feb 8, 2024

5

Nikolay Penkov

[Part 1] How models like ChatGPT work?—Depicting the Transform...

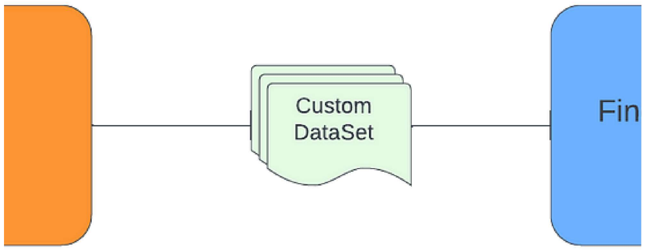
If you've stumbled upon this page, chances are you've already caught wind of the buzz...

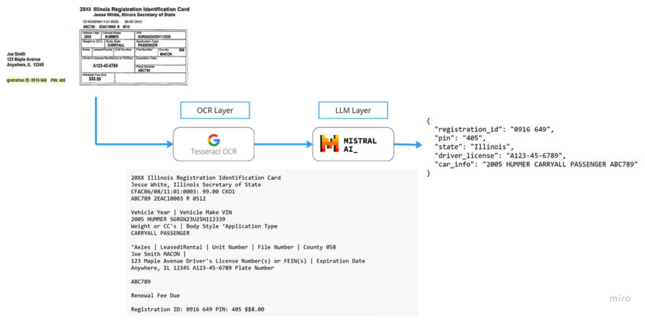
10 min read · Jan 26, 2024

2

See all from Nikolay Penkov

Recommended from Medium





```
20XX Illinois Registration Identification Card
Jesse White, Illinois Secretary of State
CFAC06/08/11:01:0003: 99.00 CKO1
ABC789 2EAC10003 R 0512

Vehicle Year | Vehicle Make VIN
2005 HUMMER SGRGN23U25H112339
Weight or CC's | Body Style 'Application Type
CARRYALL PASSENGER

'Axles | LeasediRental | Unit Number | File Number | County 058
Joe Smith MACON |
123 Maple Avenue Driver's License Number(s) or FEIN(s) | Expiration Date
Anywhere, IL 12345 A123-45-6789 Plate Number

ABC789

Renewal Fee Due

Registration ID: 0916 649 PIN: 405 $$8.00
```

```
{
  "registration_id": "0916 649",
  "pin": "405",
  "state": "Illinois",
  "driver_license": "A123-45-6789",
  "car_info": "2005 HUMMER CARRYALL PASSENGER ABC789"
}
```

Suman Das

Fine Tune Large Language Model (LLM) on a Custom Dataset with...

The field of natural language processing has been revolutionized by large language...

15 min read · Jan 25, 2024

601 7

Júlio Almeida in GoPenAI

Open-Source LLM Document Extraction Using Mistral 7B

Introduction

6 min read · Feb 2, 2024

274 2

Lists

Natural Language Processing

1202 stories · 682 saves

Predictive Modeling w/ Python

20 stories · 915 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 307 saves

Practical Guides to Machine Learning

10 stories · 1073 saves

Kasper Groes Albin Ludvi... in Towards Data Scie...

Set up a local LLM on CPU with chat UI in 15 minutes

This blog post shows how to easily run an LLM locally and how to set up a ChatGPT-lik...

5 min read · Feb 6, 2024

408 6

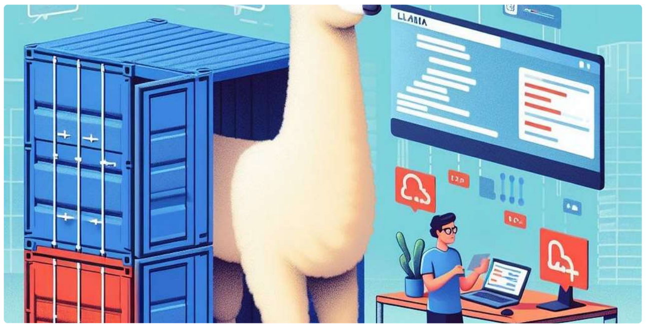
Geronimo

Finetuning Llama 2 and Mistral

A beginner's guide to finetuning LLMs with QLoRA

17 min read · Nov 5, 2023

583 15



Ahmed Tariq

A Simple Guide to Run the LLama Model in a Docker Container

Before we start, I'm assuming that you guys already have the concepts of containerizatio...

9 min read · Dec 28, 2023

30

Duy Huynh

Build your own RAG and run it locally: Langchain + Ollama +...

With the rise of Large Language Models and its impressive capabilities, many fancy...

6 min read · Dec 5, 2023

403 2

See more recommendations