

🌐 LATEST

📈 TRENDS

🛒 SHOP

G

M

in

🐦

f

📷

🔄

📡

✉️


Join thousands of students in our [LangChain and Vector DBs in Production course](#), with over 50+ lessons and practical projects for FREE!

HOME / PUBLICATION / DATA SCIENCE / FINE-TUNING A LLAMA-2 7B MODEL FOR PYTHON CODE GENERATION

[DATA SCIENCE](#) [LATEST](#) [MACHINE LEARNING](#)

# Fine-Tuning a Llama-2 7B Model for Python Code Generation

🕒 August 10, 2023



Last Updated on August 11, 2023 by [Editorial Team](#)

**Author(s):** [Eduardo Muñoz](#)

A demo on how to fine-tune the new Llama-2 using PEFT, QLoRa, and the Huggingface utilities

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

Cookie Settings

Accept



Image by author created in [Leonardo.ai](#)

About 2 weeks ago, the world of generative AI was shocked by the company Meta's release of the new Llama-2 AI model. Its predecessor, Llama-1, was a breaking point in the LLM industry, as with the release of its weights along with new finetuning techniques, there was a massive creation of open-source LLM models that led to the emergence of high-performance models such as Vicuna, Koala, ...

In this article, we will briefly discuss some of this model's relevant points but will focus on showing how we can quickly train the model for a specific task using libraries and tools standard in this world. We will not make an exhaustive analysis of the new model, there are already numerous articles published on the subject.

## New Llama-2 model

In mid-July, Meta released its new family of pre-trained and finetuned models

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

Cookie Settings

Accept

An updated version of Llama 1, trained on a new mix of publicly available data. The pretraining corpus size was increased by 40%, the model’s context length was doubled, and grouped-query attention was adopted. Variants with 7B, 13B, and 70B parameters are released, along with 34B variants reported in the paper but not released.[1]

For pre-training, **40% more tokens were used**, reaching 2T, the context length was doubled and the grouped-query attention (GQA) technique was applied to speed up inference on the heavier 70B model. On the standard transformer architecture, RMSNorm normalization, SwiGLU activation, and rotatory positional embedding are used, the context length reaches 4096 tokens, and an Adam optimizer is applied with a cosine learning rate schedule, a weight decay of 0.1 and gradient clipping.

The **Supervised Fine-Tuning** (SFT) stage is characterized by a prioritization of quality examples over quantity, as numerous reports show that the use of high-quality data results in improved final model performance.

Finally, a **Reinforcement Learning with Human Feedback** (RLHF) step is applied to align the model with user preferences. A multitude of examples are collected where annotators select their preferred model output over a binary comparison. This data is used to train a reward model, where the focus is on helpfulness and safety.

**In short:**

- *Trained on 2T Tokens*
- *Commercial use allowed*
- *Chat models for dialogue use cases*
- *4096 default context window (can be increased)*
- *7B, 13B & 70B parameter version*

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.



## The dataset for tuning

For our tuning process, we will take a dataset containing about 18,000 examples where the model is asked to build a Python code that solves a given task. This is an extraction of the original dataset [2], where only the Python language examples are selected. Each row contains the description of the task to be solved, an example of data input to the task if applicable, and the generated code fragment that solves the task is provided [3].

```
# Load dataset from the hub
dataset = load_dataset(dataset_name, split=dataset_split)
# Show dataset size
print(f"dataset size: {len(dataset)}")
# Show an example
print(dataset[randrange(len(dataset))])
```

## Creating the prompt

To carry out an instruction fine-tuning, we must transform each one of our data examples as if it were an instruction, outlining its main sections as follows:

```
def format_instruction(sample):
    return f"""### Instruction:
Use the Task below and the Input given to write the Response,
which is a programming code that can solve the following Task:

### Task:
{sample['instruction']}

### Input:
{sample['input']}

### Response:
{sample['output']}
"""
```

Output:

```
### Instruction:
Use the Task below and the Input given to write the Response,
which is a programming code that can solve the following Task:

### Task:
Develop a Python program that prints "Hello World" whenever it
```

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

[Cookie Settings](#)[Accept](#)



Image by [Irvan Smith](#) from [Unsplash](#)

## Fine-tuning the model

To carry out this stage, we have used the [Google Colab](#) environment, where we have developed a notebook that allows us to run the training in an interactive way and also a Python script to run the training in unattended mode. For the first test runs, a *T4 instance* with a high RAM capacity is enough, but when it comes to running the whole dataset and epochs, we have opted to use an ***A100 instance*** in order to speed up the training and ensure that its execution time is reasonable.

In order to be able to share the model, we will log in to the Huggingface hub using the appropriate token, so that at the end of the whole process, we will upload the model files so that they can be shared with the rest of the users.

```
from huggingface_hub import login
from dotenv import load_dotenv
import os

# Load the enviroment variables
load_dotenv()

# Login to the Hugging Face Hub
login(token=os.getenv("HF_HUB_TOKEN"))
```

## Fine-tuning techniques: PEFT, Lora, and QLora

In recent months, some papers have appeared showing how PEFT techniques can be used to train large lanquage models with a drastic reduction of RAM

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.





phase is where the PEFT technique has its purpose.

*Parameter Efficient Fine-Tuning (PEFT)* allows us to considerably reduce RAM and storage requirements by only *fine-tuning a small number of additional parameters, with virtually all model parameters remaining frozen*. PEFT has been found to produce good generalization with relatively low-volume datasets. Furthermore, it enhances the *reusability and portability* of the model, as the small checkpoints obtained can be easily added to the base model, and the base model *can be easily fine-tuned and reused in multiple scenarios* by adding the PEFT parameters. Finally, since the base model is not adjusted, all the knowledge acquired in the pre-training phase is preserved, thus *avoiding catastrophic forgetting*.

Most widely used **PEFT techniques** aim to keep the pre-trained base model untouched and add new layers or parameters on top of it. These layers are called “**Adapters**” and the technique of their adjustment “**adapter-tuning**”, we add these layers to the pre-trained base model and only train the parameters of these new layers. However, a serious problem with this approach is that these layers lead to increased latency in the inference phase, which makes the process inefficient in many scenarios.

In the **LoRa technique**, a *Low-Rank Adaptation of Large Language Models*, the idea is not to include new layers but to add values to the parameters in a way that avoids this scary problem of latency in the inference phase. LoRa trains and *stores the changes of the additional weights while freezing all the weights of the pre-trained model*. Therefore, we train a new weights matrix with the changes in the pre-trained model matrix, and this new matrix is decomposed into **2 Low-rank matrices** as explained here:

Let all the parameters of a LLM in the matrix  $W_0$  and the additional weight changes in the matrix  $\Delta W$ , the final weights become  $W_0 + \Delta W$ . The authors of LoRA [1] proposed that the change in weight change matrix  $\Delta W$  can be decomposed into two low-rank matrices  $A$  and  $B$ . LoRa does not train the parameters in  $\Delta W$  directly, but the parameters in  $A$  and  $B$ . So

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

[Cookie Settings](#)[Accept](#)



AI Community

The size of these low-rank matrices is defined by the  $r$  parameter. The smaller this value is, the fewer parameters to train, therefore, less effort and faster, but on the other hand, a potential loss of information and performance.

If you want a more detailed explanation, you can refer to the original paper, or there are plenty of articles that explain it in detail, such as [4].

Finally, **QLoRa** [6] consists of *applying quantization to the LoRa method* allowing 4-bit normal quantization, *nf4*, a type optimized for normally distributed weights; double quantization to reduce the memory footprint and the optimization of the NVIDIA unified memory. These are techniques to optimize memory usage to achieve “lighter” and less expensive training.

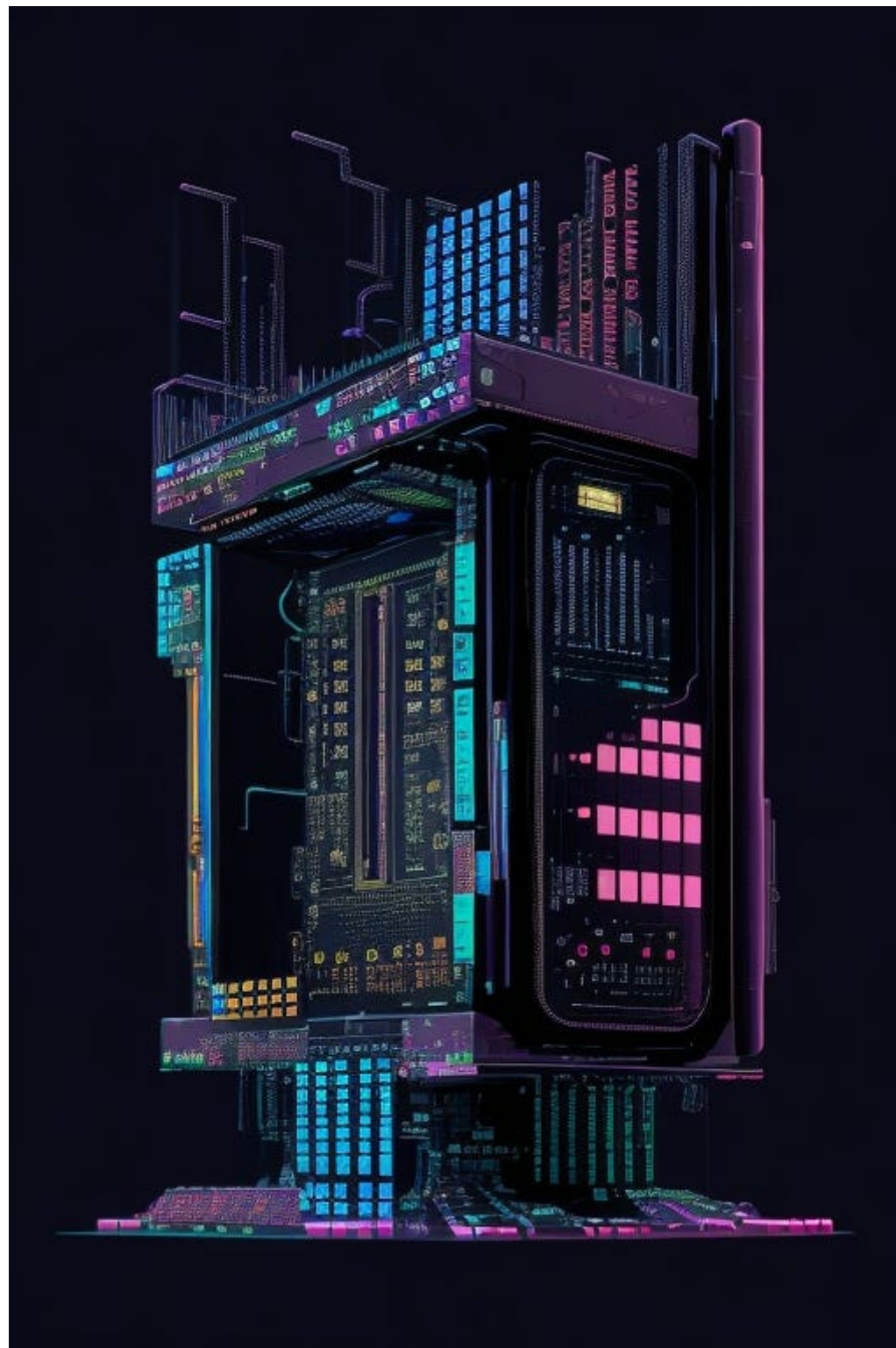


Image by the author from Leonardo.ai

Implementing QLoRa in our experiment requires specifying the *BitsAndBytes*

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

[Cookie Settings](#)[Accept](#)



AI Community

```
# BitsAndBytesConfig int-4 config
bnb_config = BitsAndBytesConfig(
    load_in_4bit=use_4bit,
    bnb_4bit_use_double_quant=use_double_nested_quant,
    bnb_4bit_quant_type=bnb_4bit_quant_type,
    bnb_4bit_compute_dtype=compute_dtype
)
# Load model and tokenizer
model = AutoModelForCausalLM.from_pretrained(model_id,
    quantization_config=bnb_config, use_cache = False,
    device_map=device_map)
model.config.pretraining_tp = 1
# Load the tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_id,
    trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
```

Parameters defined,

```
# Activate 4-bit precision base model loading
use_4bit = True
# Compute dtype for 4-bit base models
bnb_4bit_compute_dtype = "float16"
# Quantization type (fp4 or nf4)
bnb_4bit_quant_type = "nf4"
# Activate nested quantization for 4-bit base models (double
quantization)
use_double_nested_quant = False
# LoRA attention dimension
lora_r = 64
# Alpha parameter for LoRA scaling
lora_alpha = 16
# Dropout probability for LoRA layers
lora_dropout = 0.1
```

And the next steps are well-known for all Hugging Face users, setting up the training arguments, and creating a Trainer. As we are executing an instruction fine-tuning we call to the **SFTTrainer** method that encapsulates the *PEFT model definition* and other steps.

```
# Define the training arguments
args = TrainingArguments(
    output_dir=output_dir,
    num_train_epochs=num_train_epochs,
```

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

[Cookie Settings](#)
[Accept](#)





AI Community

```

fp16=fp16,
bf16=bf16,
max_grad_norm=max_grad_norm,
warmup_ratio=warmup_ratio,
group_by_length=group_by_length,
lr_scheduler_type=lr_scheduler_type,
disable_tqdm=disable_tqdm,
report_to="tensorboard",
seed=42
)
# Create the trainer
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    peft_config=peft_config,
    max_seq_length=max_seq_length,
    tokenizer=tokenizer,
    packing=packing,
    formatting_func=format_instruction,
    args=args,
)
# train the model
trainer.train() # there will not be a progress bar since tqdm is
disabled

# save model in local
trainer.save_model()

```

The parameters can be found on my [GitHub repository](#), most of them are commonly used in other fine-tuning scripts on LLMs and are the following ones:

```

# Number of training epochs
num_train_epochs = 1
# Enable fp16/bf16 training (set bf16 to True with an A100)
fp16 = False
bf16 = True
# Batch size per GPU for training
per_device_train_batch_size = 4
# Number of update steps to accumulate the gradients for
gradient_accumulation_steps = 1
# Enable gradient checkpointing
gradient_checkpointing = True
# Maximum gradient normal (gradient clipping)
max_grad_norm = 0.3
# Initial learning rate (AdamW optimizer)
learning_rate = 2e-4
# Weight decay to apply to all layers except bias/LayerNorm
weights
weight_decay = 0.001
# Optimizer to use

```

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

Cookie Settings

Accept



AI Community

```

save_steps = 0
# Log every X updates steps
logging_steps = 25
# Disable tqdm
disable_tqdm= True

```

## Merge the base model and the adapter weights

As we mention, we have trained “modification weights” on the base model, our final model requires merging the pretrained model and the adapters in a single model.

```

from peft import AutoPeftModelForCausalLM

model = AutoPeftModelForCausalLM.from_pretrained(
    args.output_dir,
    low_cpu_mem_usage=True,
    return_dict=True,
    torch_dtype=torch.float16,
    device_map=device_map,
)

# Merge LoRA and base model
merged_model = model.merge_and_unload()

# Save the merged model
merged_model.save_pretrained("merged_model", safe_serialization=True)
tokenizer.save_pretrained("merged_model")
# push merged model to the hub
merged_model.push_to_hub(hf_model_repo)
tokenizer.push_to_hub(hf_model_repo)

```

You can find and download the model in my Hugging Face account [edumunozsala/llama-2-7b-int4-python-code-20k](https://huggingface.co/edumunozsala/llama-2-7b-int4-python-code-20k). **Give it a try!**

## Inferencing or generating Python code

And finally, we will show you how you can download the model from the Hugging Face Hub and call the model to generate an accurate result:

```

import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

# Get the tokenizer
tokenizer = AutoTokenizer.from_pretrained(hf_model_repo)

```

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

[Cookie Settings](#)
[Accept](#)

TOWARDS AI

LATESTSPONSORSTUTORIALSNEWSLETTERCOMPANY

AI Community

```
input=""

prompt = f"""### Instruction:
Use the Task below and the Input given to write the Response,
which is a programming code that can solve the Task.

### Task:
{instruction}

### Input:
{input}

### Response:
"""

# Tokenize the input
input_ids = tokenizer(prompt, return_tensors="pt",
truncation=True).input_ids.cuda()
# Run the model to infer an output
outputs = model.generate(input_ids=input_ids, max_new_tokens=100,
do_sample=True, top_p=0.9,temperature=0.5)

# Print the result
print(f"Prompt:\n{prompt}\n")
print(f"Generated
instruction:\n{tokenizer.batch_decode(outputs.detach().cpu().numpy
(), skip_special_tokens=True)[0][len(prompt):]}")
```

```
Prompt:
### Instruction:
Use the Task below and the Input given to write the Response,
which is a programming code that can solve the Task.

### Task:
Optimize a code snippet written in Python. The code snippet should
create a list of numbers from 0 to 10 that are divisible by 2.

### Input:
arr = []
for i in range(10):
    if i % 2 == 0:
        arr.append(i)

### Response:

Generated instruction:
arr = [i for i in range(10) if i % 2 == 0]

Ground truth:
arr = [i for i in range(11) if i % 2 == 0]
```

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

me via [Linkedin](#). The code is available in my [Github Repository](#).

## References

[1] [Llama-2 paper](#)

[2] [Link to the original dataset in the Huggingface hub](#)

[3] [Link to the used dataset in the Huggingface hub](#)

[4] [Fine-tuning a GPT – LoRA by Chris Kuo/Dr. Dataman](#)

[5] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuezhi Li, Shean Wang, Lu Wang, & Weizhu Chen. (2021). [LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685](#)

[6]. [QLoRa: Efficient Finetuning of QuantizedLLMs](#)

[7] [Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning](#)

[8] [Extended Guide: Instruction-tune Llama 2 by Philipp Schmid](#).

[9] [Fine-Tune Your Own Llama 2 Model in a Colab Notebook by Maxime Labonne](#)

[10]. My [Github Repository](#)

Published via [Towards AI](#)



### Towards AI Newsletter

The AI newsletter. Read by over 90,000 AI students and practitioners.

Subscribe



**Note:** Content contains the views of the contributing authors and not Towards AI

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

Best Workstations for Deep Learning

Best Laptops for Deep Learning

Best Machine Learning Books

Machine Learning Algorithms

Neural Networks Tutorial

Best Public Datasets for Machine Learning

Neural Network Types

NLP Tutorial

Best Data Science Books

Monte Carlo Simulation Tutorial

Recommender System Tutorial

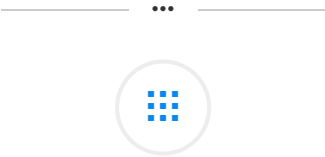
Linear Algebra for Deep Learning Tutorial

Google Colab Introduction

Decision Trees in Machine Learning

Principal Component Analysis (PCA) Tutorial

Linear Regression from Zero to Hero



## Recent Posts



Vector Database: What Is It and Why All the Hype?  
September 12, 2023



Will we soon have our own personal AI Movie Buddy?  
September 12, 2023



3 Best (Often Better) Alternatives To Histograms  
September 12, 2023

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.





# Top Important Computer Vision Papers for the Week from 4/9 to 10/9

September 11, 2023

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

TOWARDS AI

LATEST

SPONSORS

TUTORIALS

NEWSLETTER

COMPANY

AI Community

FOR READERS

Editorial

News

AI Newsletter

Subscribe

AI Community

Shop

CONTACT US

228 Park Avenue South,  
PMB 99625,  
New York, NY, 10003

Editorial ↓  
editor@towardsai.net

Press ↓  
pub@towardsai.net

Advertising/Sponsors ↓  
? **Check out our offerings ?**  
sponsors@towardsai.net

+1 (650) 246-9381

© 2019 – 2023 Towards AI Inc. | All Rights Reserved.

Terms of Service | Privacy Policy

G

M

f

in

We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies.

Cookie Settings

Accept

https://towardsai.net/p/machine-learning/fine-tuning-a-llama-2-7b-model-for-python-code-generation

15/15