

```
<s>[INST] <<SYS>>
System prompt
<</SYS>>|

User prompt [/INST] Model answer </s>
```

Step 1: Install All the Required Packages

```
!pip install -q accelerate==0.21.0 peft==0.4.0 bitsandbytes==0.40.2 transformers==4.31.0 trl==0.4.7
```

1.peft - Parameter Efficient Transfer Learning (inside has LoRA Techniques) 2.bitsandbytes - use for (Quantization) 3.transformers - (Transformers are advanced machine learning models using self-attention mechanisms to excel in NLP tasks like translation, text generation, and question answering)

step 2: Import All the Required Libraries

```
import os
import torch
from datasets import load_dataset
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
    HfArgumentParser,
    TrainingArguments,
    pipeline,
    logging,
)
from peft import LoraConfig, PeftModel
from trl import SFTTrainer
```

In case of Llama 2, the following prompt template is used for the chat models

System Prompt (optional) to guide the model

User prompt (required) to give the instruction

Model Answer (required)

We will reformat our instruction dataset to follow Llama 2 template

- Original Dataset: <https://huggingface.co/datasets/timdettmers/openassistant-guanaco>
- Reformat Dataset following the Llama 2 template with 1k sample: <https://huggingface.co/datasets/mlabonne/guanaco-llama2-1k>
- Complete Reformat Dataset following the Llama 2 template: <https://huggingface.co/datasets/mlabonne/guanaco-llama2>

To know how this dataset was created, you can check this notebook. <https://colab.research.google.com/drive/1Ad7a9zMmkxuXT0h1Z7-rNSICA4dybpM2?usp=sharing>

Note: You don't need to follow a specific prompt template if you're using the base Llama 2 model instead of the chat version

How to fine tune Llama 2

- Free Google Colab offers a 15GB Graphics Card (Limited Resources --> Barely enough to store Llama 2–7b’s weights)
- We also need to consider the overhead due to optimizer states, gradients, and forward activations
- Full fine-tuning is not possible here: we need parameter-efficient fine-tuning (PEFT) techniques like LoRA or QLoRA.
- To drastically reduce the VRAM usage, we must fine-tune the model in 4-bit precision, which is why we’ll use QLoRA here.

Step 3

1. Load a llama-2-7b-chat-hf model(chat model)
2. Train it on the mlabonne/guanaco–llama2-1k(1,000 samples), which will produce ourfine-tuned model Llama-2-7b-chat-finetune

QLoRA will use a rank of 64 with a scaling parameter of 16. We’ll load the Llama 2 model directly in 4-bit precision using the NF4 type and train it for one epoch

```
# The model that you want to train from the Hugging Face hub
model_name = "NousResearch/Llama-2-7b-chat-hf"

# The instruction dataset to use
dataset_name = "mlabonne/guanaco-llama2-1k"

# Fine-tune model name
new_model = "Llama-2-7b-chat-finetune"

#####
# QLoRA parameters
#####

# LoRA attention dimention
lora_r = 64

# Alpha parameter for LoRA scalling
lora_alpha = 16

# Dropout probability for LoRA layers
lora_dropout = 0.1

#####
# bitsandbytes parameters
#####

# Activate 4-bit precision base model loading
use_4bit = True

# Compute dtype for 4-bit base models
bnb_4bit_compute_dtypes = "float16"

# Quantization type (fp4 or nf4)
bnb_4bit_quant_type = "nf4"

# Activate nested quantization for 4_bit base models (double quantization)
use_nested_quant = False

#####
# TrainingArguments parameters
#####

# Output directory where the model predictions and checkpoints will be stored
output_dir = "./results"

# Number of training epochs
num_train_epochs = 1

# Enable fp16/bf16 training (set bf16 to True with an A100)
fp16 = False
bf16 = False

# Batch size per GPU for training
per_device_train_batch_size = 1

# Batch size per GPU for evaluation
per_device_eval_batch_size = 1

# 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
optim = "paged_adamw_32bit"

# Learning rate schedule
lr_scheduler_type = "cosine"

# Number of training steps (overrides num_train_epochs)
max_steps = -1

# Ratio of steps for a linear warmup (from 0 to learning rate)
warmup_ratio = 0.03

# Group sequence into batches with same length
# Saves memory and speeds up training considerably
group_by_length = True

# Save checkpoint every X updates steps
save_steps = 0

# Log every X updates steps
logging_steps = 25

#####
# SFT parameters
#####

# Maximum sequence length to use
```

```
max_seq_length = None

# Pack multiple short examples in the same input sequence to increase efficiency
packing = False

# Load the entire model on the GPU 0
device_map = {"": 0}
```

▼ **Step 4:Load everything and start the fine-tuning process**

1. First of all, we want to load the dataset we defined. Here our dataset is already preprocessed but, usually, this is where you would reformat the prompt, filter out bad text, combine multiple datasets etc.
2. Then, we're configuring bitsandbytes for 4-bit quantization.
3. Next, we're loading the Llama 2 model in 4-bit precision on a GPU with the corresponding tokenizer
4. Finally, we're loading configurations for QLoRA, regular training parameters and passing everything to SFTTrainer. The training can finally start!

```
# Load datasets (you can process it here)
dataset = load_dataset(dataset_name, split="train")

# Load tokenizer and model with QLoRA configuration
compute_dtype = getattr(torch, bnb_4bit_compute_dtypes)

bnb_config = BitsAndBytesConfig(
    load_in_4bit=use_4bit,
    bnb_4bit_quant_type=bnb_4bit_quant_type,
    bnb_4bit_compute_dtype=compute_dtype,
    bnb_4bit_use_double_quant=use_nested_quant,
)

# Check GPU compatibility with bfloat16
if compute_dtype == torch.float16 and use_4bit:
    major, _ = torch.cuda.get_device_capability()
    if major >= 8:
        print("=" * 80)
        print("Your GPU supports bfloat16: accelerate training with bf16=True")
        print("=" * 80)

# Load base model--Llama2
model=AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
    device_map=device_map,
    # Load the model in 8-bit to reduce memory usage
    load_in_8bit=True
)
model.config.use_cache = False
model.config.pretraining_tp = 1

# Load Llama tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token #eos---End Of Statement
tokenizer.padding_side = "right" # Fix weird overflow issue with fp16 training


# Load LORA configuration
peft_config = LoraConfig(
    lora_alpha=lora_alpha,
    lora_dropout=lora_dropout,
    r=lora_r,
    bias="none",
    task_type="CAUSAL_LM",
)

# Most Important Part
# Set training parameters
training_arguments = TrainingArguments(
    output_dir=output_dir,
    num_train_epochs=num_train_epochs,
    per_device_train_batch_size=per_device_train_batch_size,
    gradient_accumulation_steps=gradient_accumulation_steps,
    optim=optim,
    save_steps=save_steps,
    logging_steps=logging_steps,
    learning_rate=learning_rate,
    weight_decay=weight_decay,
    fp16=fp16,
    bf16=bf16,
    max_grad_norm=max_grad_norm,
    max_steps=max_steps,
    warmup_ratio=warmup_ratio,
    group_by_length=group_by_length,
    lr_scheduler_type=lr_scheduler_type,
    report_to="tensorboard"
)

# Set supervised fine-tuning parameters
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    peft_config=peft_config, #LoRA Config
    dataset_text_field="text",
    max_seq_length=max_seq_length,
```

```
tokenizer=tokenizer,
args=training_arguments,
packing=packing,
)

# Train model
trainer.train()
```



Loading checkpoint shards: 100%2/2 [01:04<00:00, 29.66s/it]

/usr/local/lib/python3.10/dist-packages/peft/utils/other.py:102: FutureWarning: prepare_warnings.warn()

/usr/local/lib/python3.10/dist-packages/trl/trainer/sft_trainer.py:159: UserWarning: You warnings.warn()

Map: 100%1000/1000 [00:01<00:00, 967.63 examples/s]

You're using a LlamaTokenizerFast tokenizer. Please note that with a fast tokenizer, use

/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:464: UserWarning: torch warnings.warn()

[1000/1000 34:19, Epoch 1/1]

Step	Training Loss
25	1.620200
50	1.729900
75	1.131100
100	1.654600
125	1.164200
150	1.545600
175	1.106000
200	1.453100
225	1.057900
250	1.368700
275	1.119500
300	1.526900
325	1.344900
350	1.405800
375	1.249500
400	1.343400
425	1.189200
450	1.375100
475	1.123200
500	1.204300
525	1.149000
550	1.262000
575	1.174000
600	1.559300
625	1.218100
650	1.369000
675	1.096800
700	1.592700
725	1.182200
750	1.429100
775	1.111600
800	1.398200
825	1.099100
850	1.587500
875	1.124300
900	1.477300
925	1.095700
950	1.673900
975	1.286400
1000	1.336900

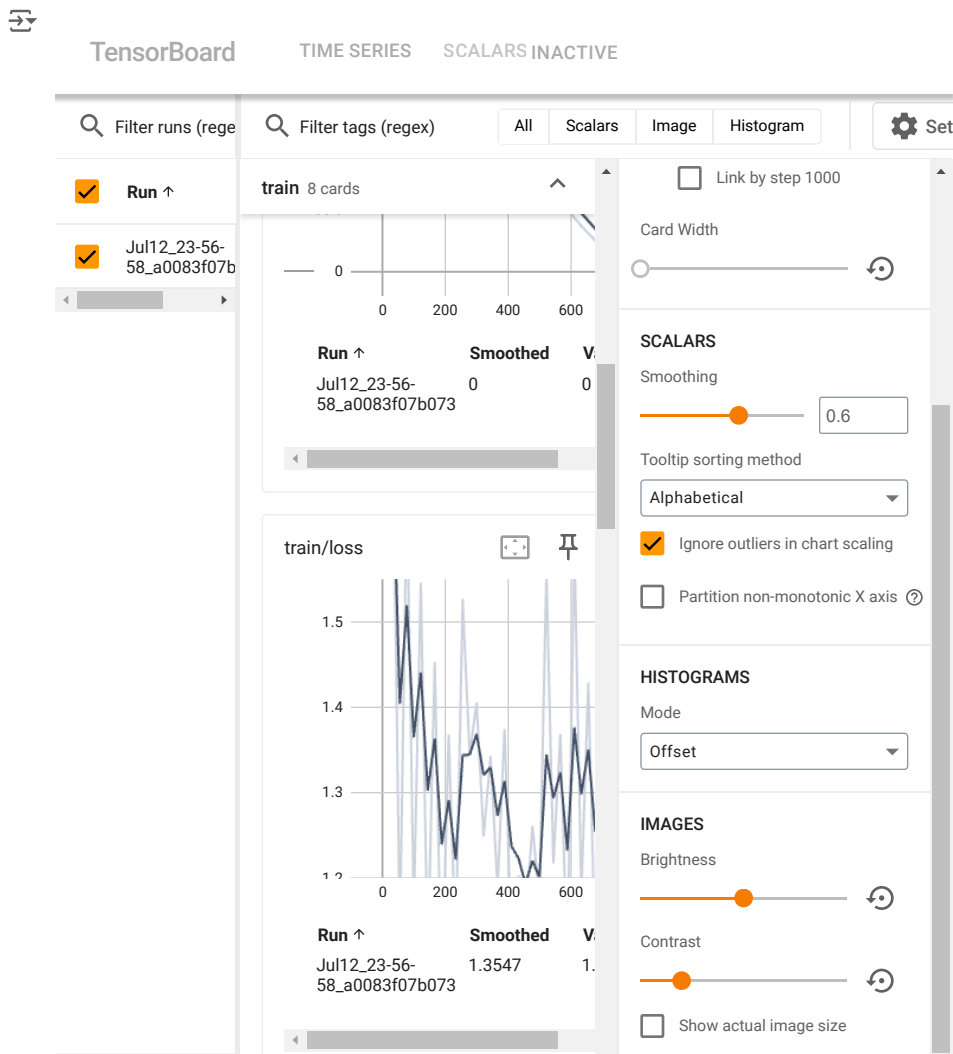
TrainOutput(global_step=1000, training_loss=1.3234088268280029, metrics={'train_runtime': 2065.7721, 'train_samples_per_second': 0.484, 'train steps per second': 0.484. 'total flos': 8606228626022400.0. 'train loss':

```
# Save trained Model
trainer.model.save_pretrained(new_model)
```

Step 5: Check the Plots on tensorboard, as follows

https://colab.research.google.com/drive/1TrOFpWz-IQMNIrpIOgVvXqg4IW-6SHNF#scrollTo=Byj1-3AJIRBP&forceEdit=true&sandboxMode=true&print...4/5

```
%load_ext tensorboard
%tensorboard --logdir results/runs
```



Step 6: Use the text generation pipeline to ask questions like "What is a large language model?" Note that I'm formatting the input to match Llama 2 prompt template.

```
# Ignore warnings
logging.set_verbosity(logging.CRITICAL)

# Run text generation pipeline with our next model
#prompt = "What is a large language model?"
prompt = "How to own a plane in the United States?"
pipe = pipeline(task="text-generation", model=model, tokenizer=tokenizer, max_length=200)
result = pipe(f"<s>[INST] {prompt} [/INST]")
print(result[0]['generated_text'])
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:464: UserWarning: torch.utils.checkpoint: the use_reentrant parameter
warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:91: UserWarning: None of the inputs have requires_grad=True. Gradients
warnings.warn(
<s>[INST] How to own a plane in the United States? [/INST] In the United States, owning a plane is a complex and expensive process that

1. Determine your budget: Owning a plane can be very expensive, so it's important to determine how much you can afford to spend before y

2. Research the different types of planes: There are many different types of planes available, including small propeller planes, large c

3. Find a plane: Once you have determined your budget and researched the different types of planes, you can start looking for a plane. Y
```

Start coding or [generate](#) with AI.