Fine-Tuning LLaMA-3 8B with Unsloth on a Custom Dataset

Introduction

Fine-tuning large language models (LLMs) has become an essential technique for adapting general-purpose models to specific tasks. In this notebook, we will fine-tune **LLaMA-3 8B** using **Unsloth** on a custom dataset formatted in the **Alpaca dataset style**. The fine-tuning process enables the model to learn domain-specific knowledge while retaining the general reasoning capabilities of the base model.

Why Use Unsloth?

Unsloth is an optimized library designed to accelerate LLM training and inference. It provides efficient fine-tuning capabilities, particularly for **quantized models**, which reduce memory usage and computational overhead. This makes it ideal for running large models like **llama-3-8b-bnb-4bit** in a resource-constrained environment such as Google Colab.

Fine-Tuning Overview

The fine-tuning process consists of the following steps:

1. **Loading the Pretrained Model**: We use the **Unsloth LLaMA-3 8B** model in 4-bit quantization for reduced memory consumption.
2. **Preparing the Dataset**: The dataset follows the **Alpaca format**, which contains instruction-response pairs to improve model instruction-following capabilities.
3. **Training Configuration**: We define training parameters such as batch size, learning rate, and number of epochs.
4. **Fine-Tuning Execution**: The model is trained using **FastLanguageModel** and **SFTTrainer** for efficient adaptation.
5. **Evaluation and Testing**: We evaluate the model's performance on unseen data to measure improvements.

Diagram: Fine-Tuning Workflow

```
flowchart TD;
    A[Load Pretrained Model (Unsloth LLaMA-3 8B)] --> B[Prepare Alpaca Format Dataset];
    B --> C[Configure Training Parameters];
    C --> D[Fine-Tune Model Using FastLanguageModel and SFTTrainer];
    D --> E[Evaluate and Test Model];
    E --> F[Deploy or Further Fine-Tune]
```

This structured approach ensures efficient model adaptation with minimal computational cost. The next sections will walk through each step in detail.

Here's a separate text block explaining the installation commands:

Setting Up the Environment

Before fine-tuning the **LLaMA-3 8B** model, we need to install essential dependencies. The following commands ensure that the environment is properly configured:

Installation Commands

```
!pip install "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"
!pip install --no-deps "xformers<0.0.27" "trl<0.9.0" peft accelerate bitsandbytes
```

Explanation of Dependencies

1. **Unsloth Library** (`unsloth[colab-new]`):
    - This installs the **Unsloth** library, a lightweight and optimized framework for fine-tuning large language models.
    - The `colab-new` flag ensures compatibility with the latest **Google Colab** environment.
    - Installed directly from the GitHub repository to access the most up-to-date version.
2. **Other Required Libraries**:
    - **`xformers<0.0.27`** : Optimizes memory usage when working with transformer models.
    - **`trl<0.9.0`** : A library for fine-tuning models using **Transformers Reinforcement Learning (TRL)**.
    - **`peft`** : Enables **parameter-efficient fine-tuning (PEFT)** techniques.
    - **`accelerate`** : Helps optimize model training for distributed computing.
    - **`bitsandbytes`** : Enables **4-bit and 8-bit quantization**, reducing memory usage while maintaining model performance.

By installing these dependencies, we ensure that our environment is optimized for efficient fine-tuning while making the best use of available hardware resources.

```
!pip install "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"
!pip install --no-deps "xformers<0.0.27" "trl<0.9.0" peft accelerate bitsandbytes
```

```
Requirement already satisfied: unsloth_zoo>=2025.2.2 in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+htt
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+https://github.
Requirement already satisfied: tyro in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+https://github.com/u
Requirement already satisfied: transformers!=4.47.0,>=4.46.1 in /usr/local/lib/python3.11/dist-packages (from unsloth@
Requirement already satisfied: datasets>=2.16.0 in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+https://
Requirement already satisfied: sentencepiece>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+http
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+https://github.com/u
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+https://github.com
Requirement already satisfied: wheel>=0.42.0 in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+https://git
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+https://github.com/
Requirement already satisfied: protobuf<4.0.0 in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+https://gi
Requirement already satisfied: huggingface_hub in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+https://g
Requirement already satisfied: hf_transfer in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+https://githu
Requirement already satisfied: bitsandbytes>=0.43.3 in /usr/local/lib/python3.11/dist-packages (from unsloth@ git+http
Requirement already satisfied: torch<3,>=2.0 in /usr/local/lib/python3.11/dist-packages (from bitsandbytes>=0.43.3->un
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from datasets>=2.16.0->unsloth@ gi
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (from datasets>=2.16.0->unsl
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from datasets>=2.16.0->u
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from datasets>=2.16.0->unsloth@ git+
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (from datasets>=2.16.0->uns
Requirement already satisfied: xxhash in /usr/local/lib/python3.11/dist-packages (from datasets>=2.16.0->unsloth@ git+
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.11/dist-packages (from datasets>=2.16.0-
Requirement already satisfied: fsspec<=2024.9.0,>=2023.1.0 in /usr/local/lib/python3.11/dist-packages (from fsspec[htt
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from datasets>=2.16.0->unsloth@ git
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from datasets>=2.16.0->unsloth@
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers!=4.47.0
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers!=4
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.11/dist-packages (from transformers!=4.47.
Requirement already satisfied: triton in /usr/local/lib/python3.11/dist-packages (from unsloth_zoo>=2025.2.2->unsloth@
Requirement already satisfied: accelerate>=0.34.1 in /usr/local/lib/python3.11/dist-packages (from unsloth_zoo>=2025.2
Requirement already satisfied: trl!=0.9.0,!=0.9.1,!=0.9.2,!=0.9.3,>=0.7.9 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: peft!=0.11.0,>=0.7.1 in /usr/local/lib/python3.11/dist-packages (from unsloth_zoo>=2025
Requirement already satisfied: cut_cross_entropy in /usr/local/lib/python3.11/dist-packages (from unsloth_zoo>=2025.2.
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-packages (from unsloth_zoo>=2025.2.2->unsloth@
Requirement already satisfied: docstring-parser>=0.15 in /usr/local/lib/python3.11/dist-packages (from tyro->unsloth@
Requirement already satisfied: rich>=11.1.0 in /usr/local/lib/python3.11/dist-packages (from tyro->unsloth@ git+https:
Requirement already satisfied: shtab>=1.5.6 in /usr/local/lib/python3.11/dist-packages (from tyro->unsloth@ git+https:
Requirement already satisfied: typeguard>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from tyro->unsloth@ git+ht
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datas
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets>=2.
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets>=2.16.
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets>=2
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets>
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets>=2.
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets>=2
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->dataset
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->d
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->d
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=11.1.0->ty
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=11.1.0->
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandbytes>=
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=2.0->bitsandbytes>=0.
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from tor
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch<3,>=
```

Loading the Pretrained Model

Once the dependencies are installed, the next step is to load the **LLaMA-3 8B** model using Unsloth's `FastLanguageModel`. This method ensures efficient memory usage, making it possible to fine-tune the model even in resource-limited environments like **Google Colab**.

Explanation

1. **Importing Required Libraries**
    - `FastLanguageModel` from **Unsloth**: Provides a highly optimized way to load and fine-tune large models.
    - `torch`: Used for handling tensor operations and GPU acceleration.
2. **Defining Configuration Parameters**
    - `max_seq_length = 2048`: Defines the **maximum sequence length** the model can process at once.
    - `dtype = None`: Uses the default data type for model weights.
    - `load_in_4bit = True`: Loads the model in **4-bit precision**, significantly reducing memory usage while maintaining computational efficiency.

3. **Model and Tokenizer Initialization**
    - The `from_pretrained()` method loads the LLaMA-3 8B model from **Unsloth's model hub** (`unsloth/llama-3-8b-bnb-4bit`).
    - It also loads the associated **tokenizer**, which converts raw text into tokens for model input.

Why Use 4-bit Quantization?

Loading the model in **4-bit** precision helps:
✔ **Reduce VRAM usage**, making it feasible to run on limited hardware.
✔ **Maintain efficiency**, ensuring the model retains its accuracy despite lower precision.
✔ **Enable faster training and inference**, compared to full-precision models.

This setup ensures that we can efficiently fine-tune the **LLaMA-3 8B** model with minimal resource overhead.

```
from unsloth import FastLanguageModel
import torch

max_seq_length = 2048
dtype = None
load_in_4bit = True

model,tokenizer = FastLanguageModel.from_pretrained(
    model_name="unsloth/llama-3-8b-bnb-4bit",
    max_seq_length=max_seq_length,
    dtype=dtype,
    load_in_4bit=load_in_4bit,
)
```

```
==((====))==  Unsloth 2025.2.4: Fast Llama patching. Transformers: 4.48.2.
   \\   /|    GPU: Tesla T4. Max memory: 14.741 GB. Platform: Linux.
O^O/ \_/ \    Torch: 2.5.1+cu124. CUDA: 7.5. CUDA Toolkit: 12.4. Triton: 3.1.0
\        /    Bfloat16 = FALSE. FA [Xformers = None. FA2 = False]
 "-____-"     Free Apache license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!
```

model.safetensors: 100% 5.70G/5.70G [01:08<00:00, 262MB/s]

generation_config.json: 100% 198/198 [00:00<00:00, 15.7kB/s]

tokenizer_config.json: 100% 50.6k/50.6k [00:00<00:00, 2.77MB/s]

tokenizer.json: 100% 9.09M/9.09M [00:00<00:00, 34.4MB/s]

special_tokens_map.json: 100% 350/350 [00:00<00:00, 18.4kB/s]

Applying Parameter-Efficient Fine-Tuning (PEFT)

To make fine-tuning more efficient, we apply **Parameter-Efficient Fine-Tuning (PEFT)** to the **LLaMA-3 8B** model. PEFT allows us to fine-tune only a subset of the model's parameters, significantly reducing memory usage and computation time.

Explanation

1. **Using `get_peft_model()`**
    - This method applies **LoRA (Low-Rank Adaptation)**, which fine-tunes only specific layers of the model while keeping the rest frozen.
    - LoRA is crucial for handling large-scale models efficiently in resource-limited environments.
2. **Parameter Breakdown**
    - `r=16` : Defines the **rank** for LoRA adaptation, controlling the number of trainable parameters.
    - `target_modules=["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj"]` : Specifies which layers of the model to fine-tune (attention and feedforward layers).
    - `lora_alpha=16` : A scaling factor that adjusts the contribution of LoRA-adapted weights.
    - `lora_dropout=0` : No dropout is applied, ensuring full utilization of fine-tuned parameters.
    - `bias="none"` : No additional biases are introduced, keeping the fine-tuning lightweight.
    - `use_gradient_checkpointing="unsloth"` : Saves GPU memory by recomputing activations during backpropagation.
    - `random_state=3407` : Ensures reproducibility of fine-tuning results.
    - `use_rslora=False` : Disables **Rank-Stabilized LoRA**, which is an alternative LoRA technique.
    - `loftq_config=None` : No additional quantization is applied beyond the existing 4-bit setting.

Why Use LoRA for Fine-Tuning?

✔ **Reduces computational cost** by training only a small fraction of model parameters.
✔ **Minimizes memory usage**, making it feasible to fine-tune large models on consumer GPUs.
✔ **Preserves pre-trained knowledge**, adapting the model to new tasks without catastrophic forgetting.

By applying **LoRA-based PEFT**, we optimize the fine-tuning process while ensuring efficient resource usage.

```
model = FastLanguageModel.get_peft_model(
    model,
    r=16,
    target_modules = ["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    lora_alpha=16,
    lora_dropout=0,
    bias="none",
    use_gradient_checkpointing="unsloth",
    random_state=3407,
    use_rslora=False,
    loftq_config=None,
)
```

Loading the Dataset

To fine-tune the **LLaMA-3 8B** model, we need a structured dataset. In this case, we use a dataset formatted in the **Alpaca style**, which is commonly used for instruction-tuned language models.

Explanation

1. **Importing the Dataset Library**
    - `load_dataset` from the **datasets** library (Hugging Face's `datasets` module) is used to load structured datasets efficiently.
2. **Specifying the Dataset File**
    - `url = "formatted_data_alpaca.jsonl"` : The dataset is stored in a **JSON Lines (JSONL) format** file, where each line represents a single training example.
    - `data_files={"train": url}` : We define the dataset partition, assigning `"train"` as the key for the training data.
    - `split="train"` : Loads only the training split from the dataset.

Alpaca Dataset Format

The dataset follows the **Alpaca format**, which contains **instruction-based samples** structured as follows:

```
{
    "instruction": "Explain the importance of data encryption.",
    "input": "",
    "output": "Data encryption is crucial for protecting sensitive information..."
}
```

This structured format helps the model learn to **follow instructions**, making it suitable for real-world applications like **chatbots, content generation, and AI assistants**.

By loading this dataset, we ensure that **LLaMA-3 8B** is trained on well-structured instruction-based data, improving its ability to handle various queries effectively.

```
from datasets import load_dataset

url = "formatted_data_alpaca.jsonl"
dataset = load_dataset("json", data_files = {"train" : url}, split = "train")
```

Setting Up the Fine-Tuning Trainer

Once the dataset is loaded and the model is configured, we initialize the fine-tuning process using **SFTTrainer** from the `trl` library. This trainer is optimized for **supervised fine-tuning (SFT)**, making it easy to adapt large language models for instruction-based learning.

Explanation

1. **Initializing `SFTTrainer`**
    - `SFTTrainer` is used for supervised fine-tuning, designed to work efficiently with large language models.
    - It automatically handles training loops, gradient updates, and logging.
2. **Configuring Training Parameters**
    - `model = model` : Uses the **LLaMA-3 8B** model initialized earlier.
    - `train_dataset = dataset` : Assigns the **Alpaca-formatted dataset** for training.
    - `dataset_text_field = "text"` : Defines the field containing the actual text data.
    - `max_seq_length = max_seq_length` : Limits the sequence length to 2048 tokens.

- `dataset_num_proc = 2` : Uses **two parallel processes** to speed up dataset processing.
- `packing = False` : Ensures that sequences are not artificially packed together.
- `tokenizer = tokenizer` : Uses the corresponding tokenizer for text processing.

3. **Training Arguments Configuration (`TrainingArguments`)**
   - `per_device_train_batch_size = 2` : Trains with a batch size of **2 per GPU**.
   - `gradient_accumulation_steps = 4` : Accumulates gradients over **4 steps** before performing an update, simulating a larger batch size.
   - `warmup_steps = 10` : Gradually increases the learning rate for the first **10 steps** to stabilize training.
   - `max_steps = 60` : Limits training to **60 total steps**, useful for testing and quick iterations.
   - `learning_rate = 2e-4` : Sets the **initial learning rate** to **0.0002**, balancing stability and convergence speed.
   - `fp16 = not is_bfloat16_supported()` : Uses **FP16 (16-bit floating point)** if BF16 is not supported.
   - `bf16 = is_bfloat16_supported()` : Uses **BF16 (BFloat16 precision)** if the GPU supports it, reducing memory usage.
   - `logging_steps = 1` : Logs training metrics **after every step** for better monitoring.
   - `lr_scheduler_type = "linear"` : Uses a **linear learning rate schedule**, where the learning rate decreases gradually.
   - `weight_decay = 0.01` : Applies **L2 regularization** to prevent overfitting.
   - `output_dir = "outputs"` : Saves model checkpoints and logs to the `"outputs"` directory.
   - `optim = "adamw_8bit"` : Uses **8-bit AdamW optimizer**, reducing memory consumption while maintaining performance.
   - `seed = 3407` : Ensures **reproducibility** of results.

Why Use `SFTTrainer`?

✔ **Handles dataset processing, model training, and logging seamlessly.**
✔ **Supports memory-efficient optimization with 4-bit and 8-bit training.**
✔ **Uses gradient accumulation to effectively increase batch size without exceeding memory limits.**
✔ **Incorporates mixed-precision training (FP16/BF16) for faster computation on modern GPUs.**

By configuring **`SFTTrainer`**, we efficiently fine-tune **LLaMA-3 8B** using **Unsloth**, making the process optimized for both performance and resource usage.

```
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported
from trl import SFTTrainer

trainer = SFTTrainer(
    model = model,
    train_dataset = dataset,
    dataset_text_field = "text",
    max_seq_length = max_seq_length,
    dataset_num_proc = 2,
    packing = False,
    tokenizer = tokenizer,
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 10,
        max_steps = 60,
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        lr_scheduler_type = "linear",
        weight_decay = 0.01,
        output_dir = "outputs",
        optim = "adamw_8bit",
        seed = 3407,
    ),
)
```

Map (num_proc=2): 100% 148/148 [00:01<00:00, 95.04 examples/s]

Executing the Fine-Tuning Process

Once the model, dataset, and training configuration are set up, we begin the fine-tuning process by calling the `train()` method of the **SFTTrainer**.

Explanation

1. **Starting the Training Process**
   - `trainer.train()` initiates the fine-tuning process using the configuration set in `SFTTrainer`.

   - The model will iterate over the dataset, updating its weights to improve performance on instruction-following tasks.

2. **Tracking Training Statistics**
   - `trainer_stats = trainer.train()` captures the training statistics, including **loss values, learning rate updates, and training time**.
   - This information is useful for **monitoring progress and debugging** if needed.

Expected Training Behavior

✔ **Mini-batch updates**: Since we use **gradient accumulation** with batch size 2 and accumulation steps 4, the model effectively trains with a larger virtual batch size.
✔ **Gradual learning rate warmup**: For the first **10 steps**, the learning rate increases to stabilize training.
✔ **Logging at each step**: Since `logging_steps = 1`, training metrics such as loss and learning rate will be printed after every step.
✔ **Memory-efficient execution**: The use of **4-bit quantization, LoRA, and mixed-precision training (FP16/BF16)** ensures that the model fine-tunes efficiently without exceeding GPU memory limits.

Monitoring Training Progress

To track training behavior, we can print the collected statistics:

```
print(trainer_stats)
```

This will output details such as:

- **Training loss** over time.
- **Total training time** and steps completed.
- **Final model performance metrics** (if validation is included).

By executing `trainer.train()`, we ensure that **LLaMA-3 8B** is adapted to our custom dataset, improving its ability to generate instruction-based responses.

```
trainer_stats = trainer.train()
```

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 148 | Num Epochs = 4
O^O/ \_/ \    Batch size per device = 2 | Gradient Accumulation steps = 4
\        /    Total batch size = 8 | Total steps = 60
 "-____-"     Number of trainable parameters = 41,943,040
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter: ··········
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: syedaliakhtar660 (syedaliakhtar660-none) to https://api.wandb.ai. Use `wandb login --re
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
```

Tracking run with wandb version 0.19.6
Run data is saved locally in `/content/wandb/run-20250208_153935-fizsb6nu`
Syncing run **outputs** to Weights & Biases (docs)
View project at https://wandb.ai/syedaliakhtar660-none/huggingface
View run at https://wandb.ai/syedaliakhtar660-none/huggingface/runs/fizsb6nu
[60/60 07:39, Epoch 3/4]

Step	Training Loss
1	1.931200
2	0.967300
3	2.030300
4	0.989500
5	0.942400
6	0.976200
7	0.664000
8	2.390400
9	0.555700
10	0.412600
11	0.284100
12	0.151200
13	0.135400
14	2.076600
15	0.055600
16	0.059100
17	0.053800
18	0.185900
19	0.009800
20	0.003600
21	0.003000
22	0.000700
23	0.000800
24	0.000500
25	0.001000
26	1.341700
27	0.000400
28	0.000100
29	0.050800
30	0.000200
31	0.000100
32	0.000100
33	0.000100
34	0.000100
35	0.065500
36	0.000600
37	0.000100
38	0.000100

39	0.000100
40	0.000100
41	0.000200
42	0.004700
43	0.000000
44	0.000100
45	0.000100
46	0.021000
47	0.000100
48	0.000100
49	0.000100
50	0.000100
51	0.001100
52	0.000100
53	0.000100
54	0.019000
55	0.000100
56	0.000200
57	0.000100
58	0.000100
59	0.011200
60	0.000100

Saving the Fine-Tuned Model

Once fine-tuning is complete, we save the trained model in **GGUF format**, which is optimized for efficient inference, particularly on **CPU-based** environments.

Explanation

1. **Saving the Model in GGUF Format**
    - `.save_pretrained_gguf("model", tokenizer, quantization_method="f16")` stores the fine-tuned **LLaMA-3 8B** model in **GGUF format** inside the `"model"` directory.
    - **GGUF (GPTQ-compatible Unified Format)** is a more efficient format for running models in **CPU-based environments** like local machines and mobile devices.
2. **Including the Tokenizer**
    - The tokenizer is saved alongside the model to ensure that future inference tasks correctly process text input.
3. **Applying `f16` Quantization**
    - `quantization_method="f16"` : Saves the model in **16-bit floating point precision (FP16)** to balance memory efficiency and inference speed.
    - FP16 offers lower memory usage than full precision (`fp32`) while maintaining good performance.

Why Use GGUF?

✔ **Optimized for CPU Inference**: GGUF models run efficiently on devices without powerful GPUs.
✔ **Smaller File Size**: Quantization reduces the storage requirements.
✔ **Compatible with llama.cpp**: GGUF models can be used with **llama.cpp**, a lightweight and fast inference framework.

```
model.save_pretrained_gguf("model", tokenizer, quantization_method="f16")
```