QLoRA详解及代码实战

1. 背景

论文标题:QLoRA: Efficient Finetuning of Quantized LLMs

论文链接:https://arxiv.org/pdf/2305.14314.pdf

QLoRA是当前PEFT利器，将预训练模型的参数 W量化到NF4的精度（存储数据），在进行特征计算时，通过双重反量化将它还原到BF16精度**参数更新**（计算数据类型）。同LoRA一样，QLoRA也在原参数一侧添加了一个与原参数并行的**低秩适配器**，QLoRA能在<8GB显存GPU Fintune LLaMA2-7B。

2. 创新点

QLoRA将低精度存储（NF4）与高精度计算（BFloat16）结合起来，**在16位矩阵乘法过程中有效地使用量化权重**，主要有以下三个创新点：

a. **4-bit NormalFloat（NF4）**：一种新的数据类型4位NormalFloat（NF4），适合于正态分

布数据的最佳量化数据类型。

b. **双重量化(Double Quantization—DQ)**：双重量化以减少平均内存占用，使得在有限的硬件上微调更大的模型成为可能。

c. **分页优化器(Paged Optimizers)**：分页优化器来管理内存峰值。

> QLORA 有一种**低精度存储数据类型（4 bit），还有一种计算数据类型（BFloat16）**。实际上，这意味着无论何时使用 QLoRA 权重张量，我们都会将张量反量化为 BFloat16，然后执行 16 位矩阵乘法。

3. 技术方案

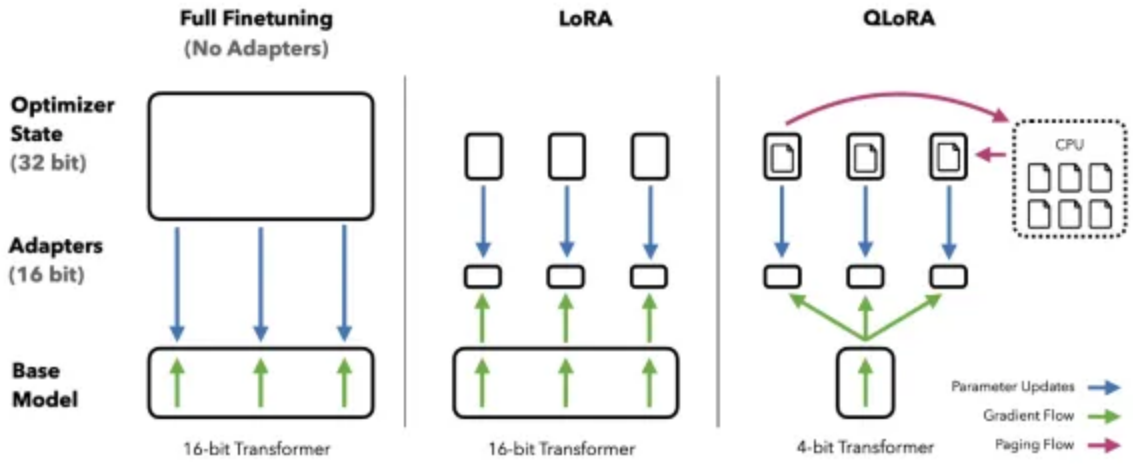
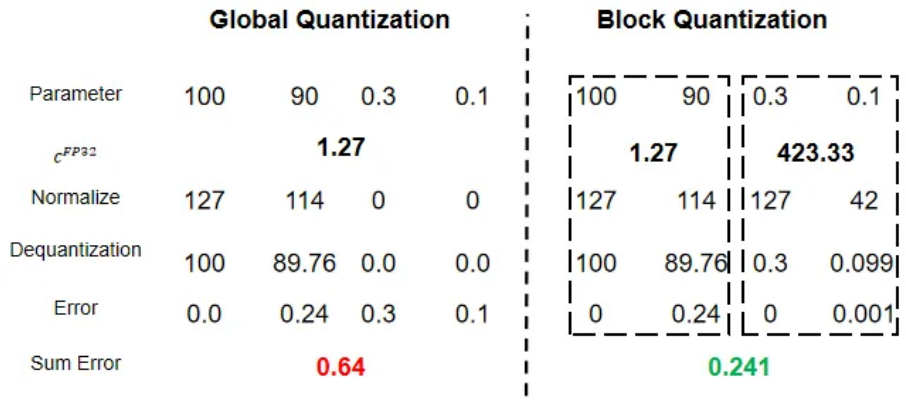


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

3.1. 分块量化(Block-wise Quantization)

全局量化方式存在一个问题，当输入中存在极大值或者离群值时，一些较小的参数无法被精确的表示，因此量化后的神经网络效果会下降很多。为了缓解这个问题，作者采用了分块量化，即将输入划分为多个block，每个block分别量化。全局量化和分块量化示意如下图所示：



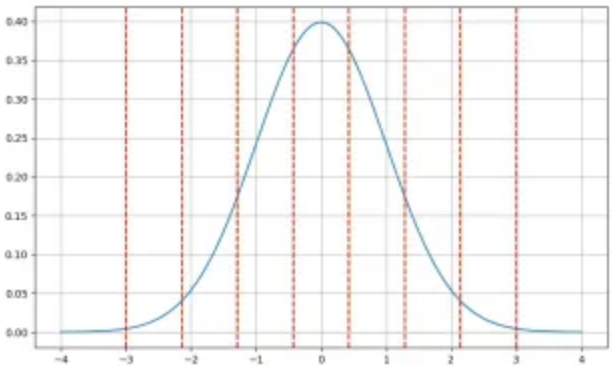
分块量化

从图中可以看到，分块量化能够明显减少量化过程中的误差(0.64 –> 0.241)。

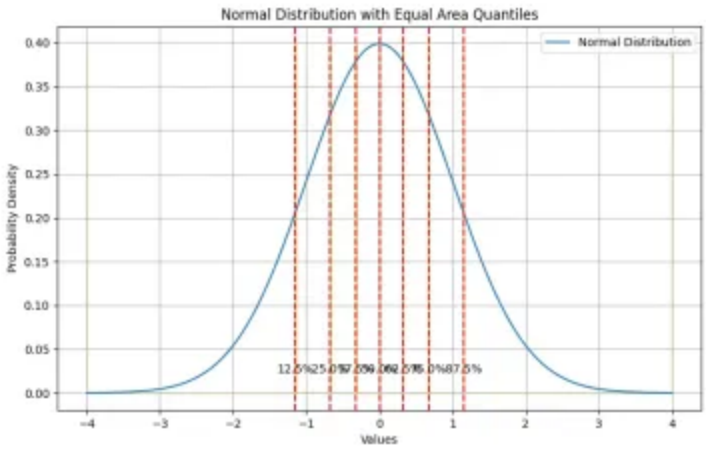
3.2. 分位量化

传统线性量化方法（对称量化）将原本不同的权重经量化后全数转化为相同的数值，导致模型出现较大误差。一般的模型参数通常**呈正态分布，而非均匀分布**。若依照线性方式进行量化，极可能导致多个不同的值被量化到相同的数值上。

> **把量化理解成装格子的过程，int8量化有256个不同的选值，相当于有256个不同的格子（即，最大值和最小值中间，等间隔地摆放256个不同的格子），然后每个参数值都跳进离它最近的格子里。**如果原始数值相当接近，它们就极有可能最终跳入同一个"格子"。**如此一来，这些数值在量化后就会归并为一个，从而引起误差。**



分位量化(Quantile Quantization)：**非对称量化方法**，以量化到4–bit为例，一共有16个数字可以选，那么可以先将输入参数从小到大进行排序再等分为16份，每一份映射一个值。这种分位量化方法量化出的参数就能保证分布尽可能与原始分布相差不大。

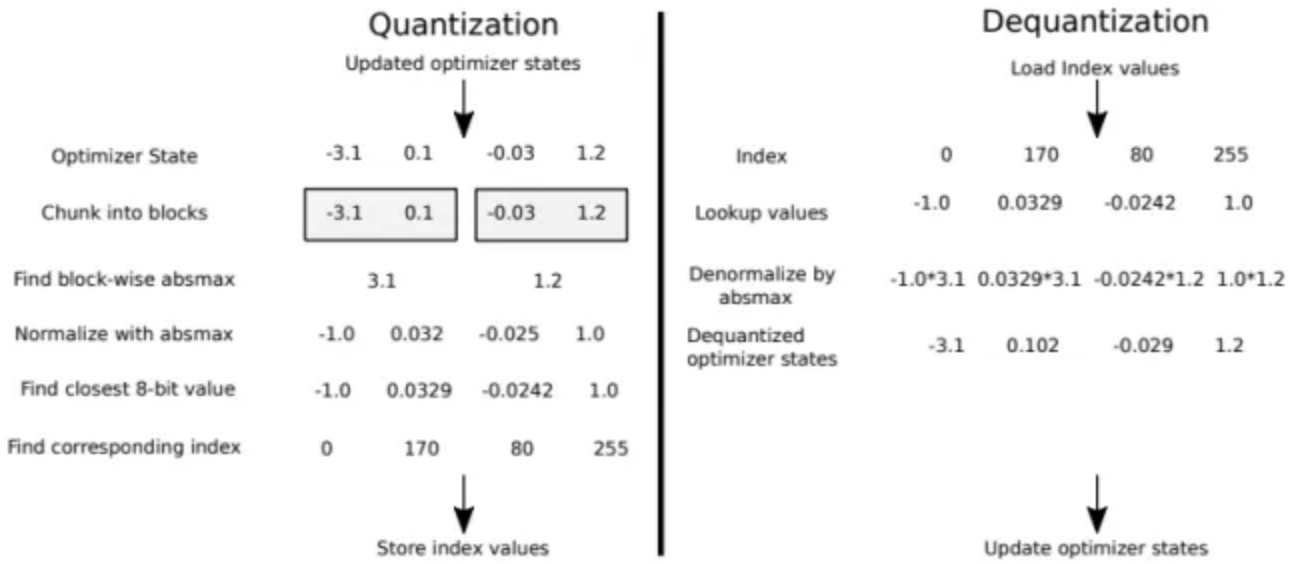


问题：分位量化会额外引入明显的计算开销（每次有参数输入进来都需要对齐进行排序并等分）。提出NF方法。

3.3. NormFloat（NF）

优化：基于参数服从正态分布假设，进行位置映射。具体地：预训练的参数基本上都服从均值为0的正态分布，可以将其缩放到[–1, 1]的范围中，将正态分布$N(0,1)$划分为2^k+1份，并缩放到[–1, 1]的范围中，直接将参数映射到对应的分位，不用每次都对参数进行排序。 此外，（**保证对称性**）作者分别**将负数和整数部分划**分为2^{k-1}份，参数0还是放在0原本的位置上，解决参数0量化后可能不在0的位置上的问题。

3.4. NF4量化过程模拟



a. 输入分块，切分到不同block中

b. 找到输入的最大值，进行归一化

c. 找到最近的NF4分位值

3.5. 双重量化

$$\mathbf{X}^{\text{Int8}} = \text{round}\left(\frac{127}{\text{absmax}(\mathbf{X}^{\text{FP32}})}\mathbf{X}^{\text{FP32}}\right) = \text{round}\left(c^{\text{FP32}} \cdot \mathbf{X}^{\text{FP32}}\right) \quad (1)$$

其中c是量化常数（quantization constant），通常是这个张量的特征的绝对值的最大值。

分块量化：每个block都会额外产生一个量化常数c。以量化32bit参数、block大小64为例，每个block会引入32bit的量化常数，对应每个参数会额外引入32/64=0.5bit的额外开销。

优化策略为：在第一次量化后，并不会直接储存量化常数c_1^{FP32}，而是按照block大小256对量化常数再量化到8bit（FP8）上去储存。

第一次量化：

假设有 64 * 256 个参数，blocksize 等于 64 的情况下，会花费256 个 32 bit 量化常量，**消耗**$32bit * 256$**的空间**。那这 256 个 32bit 数据也挺花费空间的。

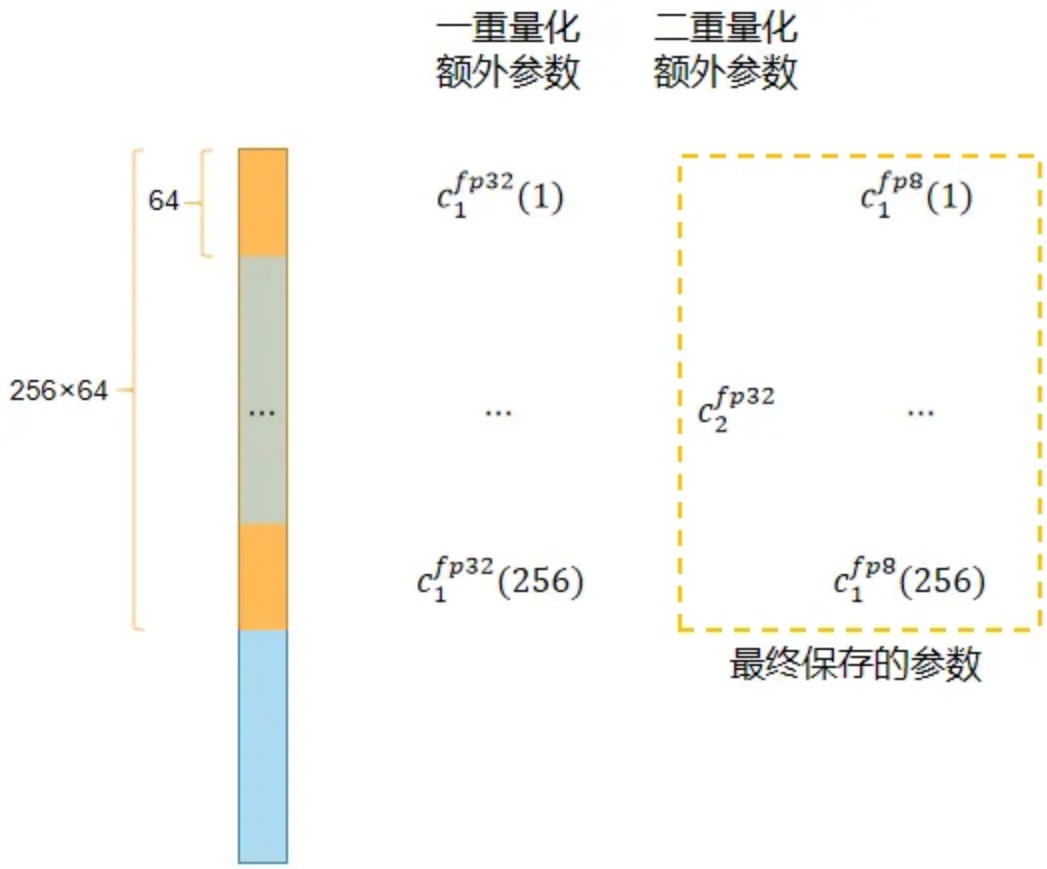
第二次量化：

对这 256 个数据进行进一步的量化，使用 blocksize 为 256，在花费掉一个32bit 量化常量（c_1^{FP32}），这 256 个 量化为 256 个 c_2^{FP8} 了，消耗掉 $8bit * 256 + 32bit * 1$ 的空间。

将 64*256 个参数，进行一层量化的话，过程中消耗的空间为 $32bit * 256$ 的空间，进行二次量化的话，过程中会消耗掉 $8bit * 256 + 32bit * 1$ 的空间。

空间节省：

对于每一个参数节约了 $0.373 = 32/64 - (8/64 + 32/64/256)$



3.6. 分页优化

分页优化：针对梯度检查点做的进一步优化，以防止在显存使用峰值时发生显存OOM的问题。QLoRA分页优化其实就是当显存不足是，将保存的部分梯度检查点转移到CPU内存上，和计算机的内存数据转移到硬盘上的常规内存分页一个道理。分页优化在GPU显存不足的时候可以把optimizer转移到内存中，在需要更新optimizer状态时再加载回来，以此来减少GPU显存的峰值占用。

3.7. QLORA与LORA对比

QLoRA将微调65B参数模型的平均内存需求从>780GB的GPU内存降低到<48GB，而其运行时间和预测性能与16位完全微调基准相比并无损失。这标志着LLM微调的方式发生了重大转变。

（一）LORA

$$\mathbf{Y}^{BF16} = \mathbf{X}^{BF16}\mathbf{W}^{BF16} + \mathbf{X}^{BF16}\mathbf{L}_1^{BF16}\mathbf{L}_2^{BF16}$$

（二）QLoRA

$$\mathbf{Y}^{BF16} = \mathbf{X}^{BF16} doubleDequant(c_1^{FP32}, c_2^{k-bit}, \mathbf{W}^{NF4}) + \mathbf{X}^{BF16}\mathbf{L}_1^{BF16}\mathbf{L}_2^{BF16}$$

$$doubleDequant(c_1^{FP32}, c_2^{k-bit}, \mathbf{W}^{k-bit}) = dequant(dequant(c_1^{FP32}, c_2^{k-bit}), \mathbf{W}^{4bit}) = \mathbf{W}^{BF16}$$

反量化过程：

以储存8-bit为例，QLoRA提供了去量化的公式

$$doubleDequant(c_1^{FP32}, c_2^{8bit}, W^{4bit}) = dequant_{step1}(dequant_{step2}(c_1^{FP32}, c_2^{8bit}), W^{4bit})$$

> 去量化Step2， 使用 c_1^{FP32} 对256个 c_2^{8bit} ，恢复成256个FP32, 计为C_step2
>
> 去量化Step1，使用256 个C_step2 去分别恢复对应block里64个4bit数字–>64个FP32个数字，即为C_step1
>
> 这样原始参数减去C_step1得到量化误差。

4. 代码实践参考

- **bitsandbytes**：该库包含量化大型语言模型（LLM）所需的所有工具。
- **Hugging Face Transformers 和 Accelerate**：这些标准库用于 Hugging Face Hub 的高效模型训练。
- **PEFT**：该库提供了各种方法的实现，以微调少量额外的模型参数。LoRA 需要它。

4.1. NF4量化：

量化参数由 **BitsandbytesConfig** 控制，如下所示：

- 通过 **load_in_4bit** 启用 4 位加载。
- **bnb_4bit_compute_dtype** 用于线性层计算的数据类型。
- 嵌套量化通过 **bnb_4bit_use_double_quant** 启用。
- **bnb_4bit_quant_type** 指定用于量化的数据类型。支持两种量化数据类型：**fp4** （四位浮点）和 **nf4**（常规四位浮点）。我们提倡使用 **nf4** ，因为理论上它对于正态分布权重来说是最佳的。

```python
model = AutoModelForCausalLM.from_pretrained(
        model_name_or_path='/name/or/path/to/your/model',
        load_in_4bit=True,
        device_map='auto',
        max_memory=max_memory,
        torch_dtype=torch.bfloat16,
        quantization_config=BitsAndBytesConfig(
            load_in_4bit=True,
            bnb_4bit_compute_dtype=torch.bfloat16,
            bnb_4bit_use_double_quant=True,
            bnb_4bit_quant_type='nf4'
        ),
    )
```

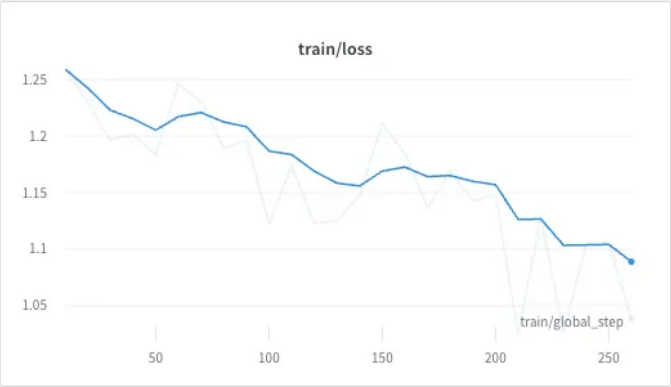
4.2. QLoRA 高效微调

Python

```python
########## 环境
pip install -q -U bitsandbytes
pip install -q -U git+https://github.com/huggingface/transformers.git
pip install -q -U git+https://github.com/huggingface/peft.git
pip install -q -U git+https://github.com/huggingface/accelerate.git

########## 数据
from datasets import load_dataset

#dataset_name = "timdettmers/openassistant-guanaco" ###Human ,.,,,,,, ### Assistant

dataset_name = 'AlexanderDoria/novel17_test' #french novels
dataset = load_dataset(dataset_name, split="train")

########## 加载模型
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig, AutoTokenizer

model_name = "TinyPixel/Llama-2-7B-bf16-sharded"

### 量化
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
)

###  模型
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
    trust_remote_code=True
)
model.config.use_cache = False

# 加载预训练模型的分词器
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token

# 创建一个 PEFT 配置对象，以便在训练和评估模型时使用。
from peft import LoraConfig, get_peft_model

```

```
lora_alpha = 16
lora_dropout = 0.1
lora_r = 64

peft_config = LoraConfig(
    lora_alpha=lora_alpha,
    lora_dropout=lora_dropout,
    r=lora_r,
    bias="none",
    task_type="CAUSAL_LM"
)

# 加载训练器
from transformers import TrainingArguments

output_dir = "./results"
per_device_train_batch_size = 4
gradient_accumulation_steps = 4
optim = "paged_adamw_32bit"
save_steps = 100
logging_steps = 10
learning_rate = 2e-4
max_grad_norm = 0.3
max_steps = 100
warmup_ratio = 0.03
lr_scheduler_type = "constant"

training_arguments = TrainingArguments(
    output_dir=output_dir,
    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,
    fp16=True,
    max_grad_norm=max_grad_norm,
    max_steps=max_steps,
    warmup_ratio=warmup_ratio,
    group_by_length=True,
    lr_scheduler_type=lr_scheduler_type,
)

# 然后最后将所有内容传递给训练器，创建一个训练器对象，以便对指定的语言模型进行训练。

# https://github.com/huggingface/trl
from trl import SFTTrainer
```

```python
max_seq_length = 512
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    peft_config=peft_config,
    dataset_text_field="text",
    max_seq_length=max_seq_length,
    tokenizer=tokenizer,
    args=training_arguments,
)
trainer.train()
model_to_save = trainer.model.module if hasattr(trainer.model, 'module') else trainer.model  # Take care of distributed/parallel training
model_to_save.save_pretrained("outputs")
```



推理 Python

```python
from peft import LoraConfig, get_peft_model
lora_config = LoraConfig.from_pretrained('outputs')
model = get_peft_model(model, lora_config)

dataset['text']
['''### Human: Écrire un texte dans un style baroque,
utilisant le langage et la syntaxe du 17ème siècle,
mettant en scène un échange entre un prêtre et un jeune homme
confus au sujet de ses péchés.### Assistant: Si j'en luis éton.
né ou empêché ce n'eſt pas ſans cauſe vů que ſouvent les liommes ne
ſçaventque dire non plus que celui de tantôt qui ne ſçavoit rien faire
que des civiéresVALDEN: Jefusbien einpêché confeſſant un jour un jeune
Breton Vallonqui enfin de confeſſion me dit qu'il avoit beſongné une civie
re .
Quoilui dis je mon amice peché n'eſt point écrit au livre Angeli que
d'enfernommé la ſommedes pechez ,qui eſt le livre le plus déteſtable qui f
ut
jamais fait& le plus blafphematoire d'autant qu'il eſt dédié à la plus fem
me
de bien je ne ſçai quelle penitence te donner ; mais non mon amiquel goût
y prenois-tu
? Mon fieur bon & delectable. Quoi''']

text = "Écrire un texte dans un style baroque sur la glace et le feu ### A
ssistant: Si j'en luis éton"
device = "cuda:0"

inputs = tokenizer(text, return_tensors="pt").to(device)
outputs = model.generate(**inputs, max_new_tokens=50)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))

# /usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:31: Us
erWarning: None of the inputs have requires_grad=True. Gradients will be N
one
  # warnings.warn("None of the inputs have requires_grad=True. Gradients w
ill be None")
Écrire un texte dans un style baroque sur la glace et le feu
### Assistant: Si j'en luis étonné, si j'en brûle, si j'en frissonne, si
j'en tremble, si j'en frémis, si j'en frissonne, si j'en frissonne, si
```

文档内容概述：QLoRA详解及代码实战

本文档详细介绍了QLoRA（Quantized Low-Rank Adaptation）技术，这是一种高效的大型语言模型（LLM）微调方法。QLoRA的核心在于将预训练模型的参数量化到4位NormalFloat（NF4）精度进行存储，并在特征计算时通过双重量化（Double Quantization）将其还原到BFloat16精度进行参数更新。这种方法结合了低精度存储和高精度计算，使得在资源受限的硬件上微调更大的模型成为可能。