

目标

微调Mistral 7B用于生成式推荐

介绍Mistral 7B

官方发布: [announcing-mistral-7b](#)

Mistral 7B 是一个 7.3B 参数模型, 它:

- 在所有基准测试中跑赢 Llama 2 13B
- 在许多基准测试中优于 Llama 1 34B
- 在代码上接近 CodeLlama 7B 性能, 同时保持良好的英语任务
- 使用分组查询注意力 (GQA) 加快推理速度
- 使用滑动窗口注意力 (SWA) 以更低的成本处理更长的序列

官方论文: <https://arxiv.org/abs/2310.06825>

运行Mistral 7B

方式一：官方仓库

官方仓库: [mistral-src](#)

Reference implementation of Mistral AI 7B v0.1 model.

Mistral AI 7B v0.1 模型的参考实现。

克隆仓库

```
git clone https://github.com/mistralai/mistral-src.git  
cd mistral-src
```

输出:

```
root@ecee6f88b4ba:~# pwd  
/home  
  
root@ecee6f88b4ba:~# git clone https://github.com/mistralai/mistral-src.git  
Cloning into 'mistral-src'...  
remote: Enumerating objects: 159, done.  
remote: Counting objects: 100% (84/84), done.  
remote: Compressing objects: 100% (52/52), done.  
remote: Total 159 (delta 66), reused 32 (delta 32), pack-reused 75  
Receiving objects: 100% (159/159), 373.02 KiB | 2.94 MiB/s, done.  
Resolving deltas: 100% (78/78), done.
```

```
root@ecee6f88b4ba:~# cd mistral-src/
root@ecee6f88b4ba:~/mistral-src# ls
LICENSE README.md assets deploy main.py mistral moe_one_file_ref.py
one_file_ref.py requirements.txt test_generate.py tutorials

root@ecee6f88b4ba:~/mistral-src# git log
commit 8598cf582091a596671be31990448e0620017851 (HEAD -> main, origin/main,
origin/HEAD)
Author: Lélio <l@mistral.ai>
Date:   Fri Feb 2 14:45:06 2024 +0100

    Update README: new download link, add reference to Mistral AI official API
```

安装依赖

```
pip install -r requirements.txt
```

```
fire
sentencepiece
torch>=2.1.0
xformers
simple-parsing
```

下载模型

```
wget https://models.mistralcdn.com/mistral-7b-v0-1/mistral-7B-v0.1.tar
```

输出:

```
--2024-04-05 19:55:08-- https://models.mistralcdn.com/mistral-7b-v0-1/mistral-
7B-v0.1.tar
Resolving models.mistralcdn.com (models.mistralcdn.com) ... 172.67.70.68,
104.26.7.117, 104.26.6.117, ...
Connecting to models.mistralcdn.com (models.mistralcdn.com)|172.67.70.68|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 14484023296 (13G) [application/x-tar]
Saving to: 'mistral-7B-v0.1.tar'

mistral-7B-v0.1.tar 85%[=====] 11.49G 13.4MB/s in 8m 7s

2024-04-05 20:03:15 (24.2 MB/s) - Connection closed at byte 12335050752.
Retrying.

--2024-04-05 20:03:16-- (try: 2) https://models.mistralcdn.com/mistral-7b-v0-
1/mistral-7B-v0.1.tar
Connecting to models.mistralcdn.com (models.mistralcdn.com)|172.67.70.68|:443...
connected.
```

```
HTTP request sent, awaiting response... 206 Partial Content
Length: 14484023296 (13G), 2148972544 (2.0G) remaining [application/x-tar]
Saving to: 'mistral-7B-v0.1.tar'

mistral-7B-v0.1.tar 100%[=====] 13.49G 26.5MB/s in 73s

2024-04-05 20:04:31 (28.0 MB/s) - 'mistral-7B-v0.1.tar' saved
[14484023296/14484023296]
```

验证模型文件完整性

(md5sum: 37dab53973db2d56b2da0a033a15307f)

Q: 什么是md5sum?

A: `md5sum` 是一个用于计算和验证文件MD5散列值的命令行工具。MD5 (Message Digest Algorithm 5) 是一种常用的哈希函数，通常用于生成数据的唯一标识符。`md5sum` 工具将文件的内容作为输入，并输出一个128位的MD5散列值。这个散列值在理想情况下对于不同的输入数据应该是唯一的，即使是极小的文件内容改变也会导致散列值的差异。这使得 `md5sum` 在验证文件的完整性时非常有用，因为即使文件的一点点改变也会导致散列值的不同，这样就可以确保文件在传输或存储过程中没有被篡改。

Q: 怎么计算md5sum?

A: 计算文件的MD5散列值，可以使用命令行中的`md5sum`工具: `md5sum 文件名`。

```
md5sum mistral-7B-v0.1.tar
```

输出结果与给定的md5sum值一致:

```
37dab53973db2d56b2da0a033a15307f mistral-7B-v0.1.tar
```

解压模型文件

```
tar -xf mistral-7B-v0.1.tar
```

这个命令是用于解压缩一个名为 `mistral-7B-v0.1.tar` 的文件。具体来说:

- `tar`: 这是一个在Unix和类Unix系统中用来打包和解包文件的命令行工具，名字来源于 "tape archive"，最初设计用来操作磁带存储。
- `-xf`: 这是 `tar` 命令的选项参数，它告诉 `tar` 命令要执行解压缩操作，同时还指定了文件名。`-x` 表示提取（解压缩），`-f` 表示紧接着会跟着一个文件名参数。
- `mistral-7B-v0.1.tar`: 这是要解压缩的文件名，其中 `.tar` 扩展名表示这是一个经过 `tar` 命令打包过的文件，它可能包含一个或多个文件或目录。

因此，这个命令的作用是将名为 `mistral-7B-v0.1.tar` 的文件解压缩，并将其中的内容提取到当前目录中。

```
cd mistral-7B-v0.1 && ls
```

输出：

```
RELEASE consolidated.00.pth params.json tokenizer.model
```

运行模型

运行main.py文件中的demo函数：

```
python -m main demo ./mistral-7B-v0.1/
```

```
res, _logprobs = generate(  
    [  
        "This is a test",  
        "This is another great test",  
        "This is a third test, mistral AI is very good at testing. ",  
    ],  
    transformer,  
    tokenizer,  
    max_tokens=max_tokens,  
    temperature=temperature,  
)
```

三个测试用例的输出结果：

```
This is a test of the emergency broadcast system. This is only a test.  
  
If this were a real emergency, you would be told what to do.  
  
This is a test  
=====  
This is another great testament to the power of the internet.  
  
I was looking for a new pair of running shoes. I've been running in the same pair  
of shoes for about  
=====  
This is a third test, mistral AI is very good at testing. 😊  
  
This is a third test, mistral AI is very good at testing. 😊  
  
This is a third test, mistral AI is very good at  
=====
```

运行main.py文件中的interactive函数：

<https://github.com/Eric-LLMs>

输入自己的prompt，与之交互

```
python -m main interactive /path/to/mistral-7B-v0.1/
```

方式二：huggingface

<https://huggingface.co/mistralai/Mistral-7B-v0.1>

导入库和模型：

```
from transformers import AutoTokenizer, AutoModelForCausalLM
```

加载预训练模型：

```
model = AutoModelForCausalLM.from_pretrained("mistralai/Mistral-7B-v0.1")
```

加载分词器：

```
tokenizer = AutoTokenizer.from_pretrained("mistralai/Mistral-7B-v0.1")
```

准备输入：

```
prompt = "Hey, are you conscious? Can you talk to me?"  
inputs = tokenizer(prompt, return_tensors="pt")
```

生成文本：

```
generate_ids = model.generate(inputs.input_ids, max_length=30)
```

解码生成的标记：

```
tokenizer.batch_decode(generate_ids, skip_special_tokens=True,  
clean_up_tokenization_spaces=False)[0]
```

输出：

```
'Hey, are you conscious? Can you talk to me?\n\nI'm not sure if you're conscious  
or not. I ''
```

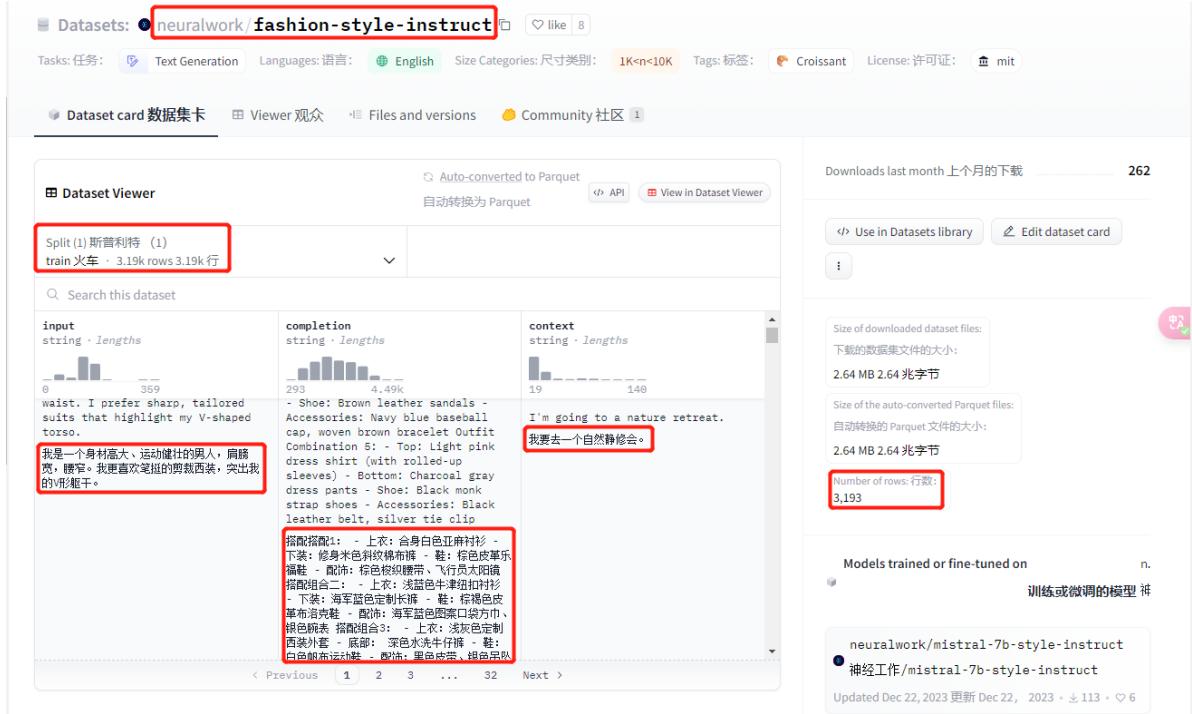
微调Mistral 7B

前置知识：《项目一：QLoRA finetune Microsoft Phi-2》

目标：微调出一个风格推荐模型，用户输入个人体型、个人风格、穿着场合，模型能推荐出5种符合要求的服装组合。

介绍数据集

风格推荐数据集：[neuralwork/fashion-style-instruct](#)



3193行三元组

- input (输入) : 体型和个人服装风格
 - 我是一个身材高大、运动健壮的男人，肩膀宽，腰窄。我更喜欢笔挺的剪裁西装，突出我的V形躯干。
 - 我是一个中等身材的运动型女性。我喜欢将运动装与日常时尚相结合，比如将紧身裤与别致的束腰外衣搭配在一起。
 - 我的身材很长，很结实，但偏向于精致的学院风。我想要关于适合我高瘦身材的西装外套、裤子和衬衫的建议。
- context (上下文) : 事件
 - 数据集包含各种活动，例如商务活动、鸡尾酒会、休闲聚会、花式约会等
 - 我要去大自然静修
 - 我要作为客人去参加婚礼。
 - 我要去参加一个会议。
- completion (输出) :
 - 搭配1: - 上衣: 合身白色亚麻衬衫 - 下装: 修身米色斜纹棉布裤 - 鞋: 棕色皮革乐福鞋 - 配饰: 棕色梭织腰带、飞行员太阳镜
 - 搭配2: - 上衣: 浅蓝色牛津纽扣衬衫 - 下装: 海军蓝色定制长裤 - 鞋: 棕褐色皮革布洛克鞋 - 配饰: 海军蓝色图案口袋方巾、银色腕表

加载原始数据集

```
from datasets import load_dataset  
dataset = load_dataset("neuralwork/fashion-style-instruct")
```

```
from datasets import load_dataset  
dataset = load_dataset("neuralwork/fashion-style-instruct")  
  
dataset  
  
DatasetDict({  
    train: Dataset({  
        features: ['input', 'completion', 'context'],  
        num_rows: 3193  
    })  
})  
  
dataset["train"][0]  
  
{'input': "I'm a tall, athletic man with broad shoulders and a narrow waist. I prefer sharp, tailored suits that highlight my V-shaped torso.",  
 'completion': 'Outfit Combination 1:\n- Top: Fitted white linen shirt\n- Bottom: Slim-fit beige chinos\n- Shoe: Brown leather loafers\n- Accessories: Brown woven belt, aviator sunglasses\n\nOutfit Combination 2:\n- Top: Light blue oxford button-down shirt\n- Bottom: Navy blue tailored trousers\n- Shoe: Tan leather brogues\n- Accessories: Navy blue patterned pocket square, silver wristwatch\n\nOutfit Combination 3:\n- Top: Light gray tailored blazer\n- Bottom: Dark wash denim jeans\n- Shoe: White canvas sneakers\n- Accessories: Black leather belt, silver pendant necklace\n\nOutfit Combination 4:\n- Top: Navy blue polo shirt\n- Bottom: Khaki shorts\n- Shoe: Brown leather sandals\n- Accessories: Navy blue baseball cap, woven brown bracelet\n\nOutfit Combination 5:\n- Top: Light pink dress shirt (with rolled-up sleeves)\n- Bottom: Charcoal gray dress pants\n- Shoe: Black monk strap shoes\n- Accessories: Black leather belt, silver tie clip',  
 'context': "I'm going to a nature retreat."}
```

构建指令数据集

把我们的原始数据集改造成指令的结构形式

```
def format_instruction(sample):  
    return f"""You are a personal stylist recommending fashion advice and  
clothing combinations. Use the self body and style description below, combined  
with the event described in the context to generate 5 self-contained and complete  
outfit combinations.  
### Input:  
{sample["input"]}  
  
### Context:  
{sample["context"]}  
  
### Response:  
{sample["completion"]}  
"""
```

你是一名个人造型师，负责推荐时尚建议和服装搭配。利用下面的自我形象和风格描述，结合上下文中描述的事件，生成 5 套自成一体的完整服装搭配。

最终用于训练的prompt如下所示：

```
sample = dataset["train"][0]
print(format_instruction(sample))

You are a personal stylist recommending fashion advice and clothing combinations. Use the self body and style description below, combined with the event described in the context to generate 5 self-contained and complete outfit combinations.

### Input:
I'm a tall, athletic man with broad shoulders and a narrow waist. I prefer sharp, tailored suits that highlight my V-shaped torso.

### Context:
I'm going to a nature retreat.

### Response:
Outfit Combination 1:
- Top: Fitted white linen shirt
- Bottom: Slim-fit beige chinos
- Shoe: Brown leather loafers
- Accessories: Brown woven belt, aviator sunglasses

Outfit Combination 2:
- Top: Light blue oxford button-down shirt
- Bottom: Navy blue tailored trousers
- Shoe: Tan leather brogues
- Accessories: Navy blue patterned pocket square, silver wristwatch

Outfit Combination 3:
- Top: Light gray tailored blazer
- Bottom: Dark wash denim jeans
- Shoe: White canvas sneakers
- Accessories: Black leather belt, silver pendant necklace

Outfit Combination 4:
- Top: Navy blue polo shirt
- Bottom: Khaki shorts
- Shoe: Brown leather sandals
- Accessories: Navy blue baseball cap, woven brown bracelet

Outfit Combination 5:
- Top: Light pink dress shirt (with rolled-up sleeves)
- Bottom: Charcoal gray dress pants
- Shoe: Black monk strap shoes
- Accessories: Black leather belt, silver tie clip
```

翻译：

你是一名个人造型师，负责推荐时尚建议和服装搭配。使用下面的自我主体和风格描述，结合上下文中描述的事件，生成 5 套自成一体的完整服装搭配。

输入：

我是一个高大健美的男人，肩宽腰窄。我喜欢利落、剪裁合体的西装，以突出我的 V 型躯干。

背景：

我要去大自然度假。

回应：

服装组合 1:

- 上装：合身的白色亚麻衬衫
- 下装米色修身长裤
- 鞋子棕色皮休闲鞋
- 配饰棕色编织腰带、飞行员太阳镜

服装组合 2:

- 上装: 浅蓝色牛津扣衬衫
- 下装深蓝色剪裁长裤
- 鞋子棕色皮质布洛克鞋
- 配饰深蓝色花纹口袋方巾, 银色腕表

服装组合 3:

- 上身: 浅灰色剪裁西装外套
- 下身: 深色水洗牛仔裤深色水洗牛仔裤
- 鞋子白色帆布运动鞋
- 配饰黑色皮带、银色吊坠项链

服装组合 4:

- 上身: 深蓝色 polo 衫
- 下身: 卡其色短裤卡其色短裤
- 鞋棕色皮凉鞋
- 配饰藏蓝色棒球帽, 棕色编织手链

服装组合 5:

- 上装: 浅粉色连衣裙衬衫 (卷起袖子)
- 下装炭灰色裙裤
- 鞋子黑色僧侣带鞋
- 配饰黑色皮带, 银色领带夹

定义量化参数

QLoRA 使用 bitsandbytes 进行量化，并与 Hugging Face 的 PEFT 和transformers 库集成。通过 BitsAndBytesConfig 类定义量化参数，对基础 Mistral 7B 模型执行 4 位训练后量化。这是一种在不显著牺牲性能的情况下减少模型的内存占用和计算要求的方法。

```
import torch
from transformers import BitsAndBytesConfig

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)
```

这段代码使用了PyTorch库和Hugging Face的Transformers库中的 BitsAndBytesConfig 类。

1. `import torch`: 这行代码导入了PyTorch库, PyTorch是一个用于深度学习的开源框架。
`from transfrromers import BitsAndBytesConfig`: 这行代码从Hugging Face的
2. Transformers库中导入了 BitsAndBytesConfig 类, 这个类用于配置BitsAndBytes模型的参数。

3. `bnb_config = BitsAndBytesConfig(...)`: 这行代码创建了一个 `BitsAndBytesConfig` 对象，并将其存储在名为 `bnb_config` 的变量中。在这里，我们使用了 `BitsAndBytesConfig` 的构造函数来配置 `BitsAndBytes` 模型的参数。

- `load_in_4bit=True`: 这个参数指定是否以 4 位精度加载模型权重。设置为 `True` 表示以 4 位精度加载。
- `bnb_4bit_use_double_quant=True`: 这个参数指定使用嵌套量化来提高内存效率的推理和训练。
- `bnb_4bit_quant_type="nf4"`: 4-bit 集成带有 2 种不同的量化类型 `FP4` 和 `NF4`。
`NF4 dtype` 代表 `Normal Float 4`。默认情况下，使用 `FP4` 量化。
- `bnb_4bit_compute_dtype=torch.bfloat16`: 更改计算期间将使用的数据类型。默认情况下，计算数据类型设置为 `float32`，但可以设置为 `bfloat16` 以提高速度。

加载预训练模型

<https://huggingface.co/mistralai/Mistral-7B-v0.1>

指定模型 ID，然后使用之前定义的量化配置加载它。

```
from transformers import AutoTokenizer, AutoModelForCausalLM

model_id = "mistralai/Mistral-7B-v0.1"

model = AutoModelForCausalLM.from_pretrained(
    model_id,
    quantization_config=bnb_config,
    device_map="auto"
)

model.config.use_cache=False
model.config.pretraining_tp = 1
```

这段代码使用了 Hugging Face 的 Transformers 库中的 `AutoTokenizer` 和 `AutoModelForCausalLM` 类。

1. `from transformers import AutoTokenizer, AutoModelForCausalLM`: 这行代码从 Hugging Face 的 Transformers 库中导入了 `AutoTokenizer` 和 `AutoModelForCausalLM` 类，分别用于自动选择适当的分词器和生成式语言模型。
2. `model_id = "mistralai/Mistral-7B-v0.1"`: 这行代码定义了一个字符串变量 `model_id`，用于存储要加载的模型的标识符。在这里，`"mistralai/Mistral-7B-v0.1"` 指定了要加载的 Mistral-7B-v0.1 模型。
3. `model = AutoModelForCausalLM.from_pretrained(...)`: 这行代码使用 `from_pretrained` 方法从预训练模型中加载了一个自动模型。`from_pretrained` 方法接受多个参数：
 - `model_id`: 要加载的模型的标识符。
 - `quantization_config=bnb_config`: 这个参数指定了量化配置，即前面创建的 `bnb_config` 对象。它告诉模型如何进行量化。

- `device_map="auto"`: 这个参数指定了设备映射。在这里，设置为 `"auto"` 表示自动选择设备。
- 加载后的模型被分配给名为 `model` 的变量。
4. `model.config.use_cache=False`: 这行代码设置了模型配置的 `use_cache` 属性为 `False`，这意味着模型在生成文本时不会使用缓存。
 5. `model.config.pretraining_tp = 1`: 这行代码设置了模型配置的 `pretraining_tp` 属性为 `1`。`pretraining_tp` 是一个实验性特性，用于指定在预训练过程中使用的张量并行度等级。通过将其设置为 `1`，它可能影响预训练过程中的并行计算设置，以确保在不同的运行中实现相同的预训练结果，并提供精确的可重现性。
 6. 总之，`use_cache` 和 `pretraining_tp` 都是为了优化模型的训练行为而做的设置，有助于提高性能。

加载分词器

分词器将原始文本输入转换为可供模型使用的固定长度数字格式，其中每个单词和标点符号都被分配了一个唯一的 ID。

```
tokenizer = AutoTokenizer.from_pretrained(model_id)
tokenizer.pad_token = tokenizer.eos_token
```

这段代码通过 `AutoTokenizer.from_pretrained` 方法加载了一个预训练的分词器，并将其分配给了名为 `tokenizer` 的变量。

1. `tokenizer = AutoTokenizer.from_pretrained(model_id)`: 这行代码使用 `from_pretrained` 方法从预训练的模型中加载了一个分词器。`from_pretrained` 方法接受一个模型标识符作为参数，并返回一个分词器对象。在这里，使用了之前定义的 `model_id` 变量作为要加载的模型标识符。
2. `tokenizer.pad_token = tokenizer.eos_token`: 这行代码将分词器的填充标记 (`pad_token`) 设置为与结束标记 (`eos_token`) 相同。这意味着在使用该分词器时，将使用结束标记来进行填充。这可能是因为在某些情况下，模型期望将填充标记与句子结束标记相同，以便正确处理填充。

测试分词器

```
tokenizer.eos_token
```

“

```
tokenizer.pad_token
```

“

```
list_texts = ["Hi, how are you?", "I'm good", "Yes"]

encoded_texts = tokenizer(list_texts)

print("Encoded several texts: ", encoded_texts["input_ids"])
```

Encoded several texts: [[1, 15359, 28725, 910, 460, 368, 28804], [1, 315, 28742, 28719, 1179], [1, 5592]]

```
tokenizer.convert_ids_to_tokens(1)
```

1

填充通常在模型训练或推理时执行。在训练时，模型需要将不同长度的输入序列批次化为相同长度，因此填充将应用于较短的序列以匹配最长序列的长度。在推理时，如果批次大小不是1，模型可能需要对输入进行填充以匹配批次中的最长序列。

可以通过设置 `padding=True` 参数来确保分词器对输入进行填充：

```
list_texts = ["Hi, how are you?", "I'm good", "Yes"]

encoded_texts = tokenizer(list_texts, padding=True)

print("Encoded several texts: ", encoded_texts["input_ids"])
```

Encoded several texts: [[1, 15359, 28725, 910, 460, 368, 28804], [2, 2, 1, 315, 28742, 28719, 1179], [2, 2, 2, 2, 1, 5592]]

```
tokenizer.convert_ids_to_tokens(1)
```

1

```
tokenizer.convert_ids_to_tokens(2)
```

2

```
tokenizer.pad_token_id
```

2

```
tokenizer.padding_side
```

~~'left'~~

~~左填充通常是默认的选择，因为在自然语言处理任务中，句子的重要信息通常位于句子的开始部分。左填充可以确保在进行批处理时，所有输入序列的起始部分对齐，这对于模型的学习和处理非常有帮助。~~

~~左填充的另一个优点是，它使得在不同长度的序列之间对齐更加直观。例如，对于长度不同的句子，左填充会将实际文本部分向右移动，而填充标记则出现在序列的末尾，这样可以更容易地识别出序列的边界。~~

```
[1, 15359, 28725, 910, 460, 368, 28804]  
[2, 2, 1, 315, 28742, 28719, 1179]  
[2, 2, 2, 2, 1, 5592]
```

设置PEFT参数

~~我们已经成功加载并量化了基本 Mistral 7B 模型。~~

~~现在，我们将使用 LORA微调。先创建了一个 LoraConfig 对象，该对象包含了 LORA 训练所需的配置信息：~~

```
from peft import LoraConfig  
  
peft_config = LoraConfig(  
    r=32,  
    lora_alpha=64,  
    target_modules=[  
        "q_proj",  
        "k_proj",  
        "v_proj",  
        "o_proj",  
        "gate_proj",  
        "up_proj",  
        "down_proj",  
        "lm_head",  
    ],  
    bias="none",  
    lora_dropout=0.05,  
    task_type="CAUSAL_LM",  
)
```

~~这段代码使用了名为 `peft` 的库中的 `LoraConfig` 类。~~

1. `from peft import LoraConfig`: 这行代码导入了 `peft` 库中的 `LoraConfig` 类，该类用于配置Lora模型的参数。

2. `peft_config = LoraConfig(...)`: 这行代码创建了一个 `LoraConfig` 对象，并将其存储在名为 `peft_config` 的变量中。在这里，我们使用了 `LoraConfig` 的构造函数来配置LoRA训练所需的配置信息。

- `r=32`: 设置了LoRA秩 (rank) 为 32，这个值决定了低秩矩阵的大小，影响模型的表达能力和计算成本。
- `lora_alpha=64`: 设置了权重缩放因子 alpha 为 64，这个值决定了 LoRA 适配器中学习到的权重应该被放大的程度。当 alpha=64 的时候，权重为 $2=64 \text{ (lora alpha)} / 32 \text{ (rank)}$ 。通常情况下 r=32 时，lora alpha 设置为 32，或者 64.
- `target_modules=[...]`: 指定了要在模型中插入 LoRA 适配器的模块，也就是需要训练的参数。通常是自注意力机制中的 `q_proj` (query matrix)、`k_proj` (key matrix)、`v_proj` (value matrix) 和 `dense` (前馈网络密集层)。也可以优化其中的几个模块，这些都可以灵活配置。
- `bias="none"`: 这个参数指定了是否应用偏置。在这里，设置为 "none" 表示不应用偏置，不作为训练参数。
- `lora_dropout=0.05`: 这个参数指定了LoRA模型中的dropout率，用于防止过拟合。
- `task_type="CAUSAL_LM"`: 这个参数指定了模型的任务类型。在这里，设置为 "CAUSAL_LM" 表示模型用于因果语言建模任务。

构建用于训练的PeftModel

让我们传入 LoRA 配置参数，返回一个 `PeftModel` 模型实例，表示模型已经按照既定的配置准备好进行量化感知微调。

```
from peft import prepare_model_for_kbit_training, get_peft_model

# prepare model for training
model = prepare_model_for_kbit_training(model)
model = get_peft_model(model, peft_config)
```

```
type(model)
```

```
peft.peft_model.PeftModelForCausalLM
```

这段代码使用了 `peft` 库中的 `prepare_model_for_kbit_training` 函数和 `get_peft_model` 函数，以准备模型进行PEFT训练。

1. `model = prepare_model_for_kbit_training(model)`: 该函数用于准备原始模型进行 k-bit 训练。这可能涉及到量化等技术，以进一步减少模型微调时的资源消耗。函数接受一个模型作为输入，并返回准备好进行 k-bit 训练的模型。
2. `model = get_peft_model(model, peft_config)`: 该函数接受原始模型和一个 `peft_config` 配置对象作为参数，并返回一个准备好进行 PEFT 训练的模型实例。返回的模型实例被分配给名为 `model` 的变量。这意味着模型已经按照指定的配置准备好用于 PEFT 训练。

用于训练的模型已经准备好了。

打印模型中可训练参数的数量

```
def print_trainable_parameters(model):
    """
    Prints the number of trainable parameters in the model.
    """

    trainable_params = 0
    all_param = 0
    for _, param in model.named_parameters():
        all_param += param.numel()
        if param.requires_grad:
            trainable_params += param.numel()
    print(f"trainable params: {trainable_params} || all params: {all_param} ||"
          f"trainable%: {100 * trainable_params / all_param}")

# get frozen vs trainable model param statistics
print_trainable_parameters(model)
```

结果：

```
trainable params: 85041152 || all params: 3837112320 || trainable%:
2.2162799758751914
```

这段代码定义了一个函数 `print_trainable_parameters(model)`，用于打印模型中可训练参数的数量统计。

1. `def print_trainable_parameters(model):`: 这是一个函数定义，它声明了一个名为 `print_trainable_parameters` 的函数，该函数接受一个模型作为参数。
2. `trainable_params = 0` 和 `all_param = 0`: 这两行代码初始化了两个变量，分别用于统计可训练参数的数量和所有参数的数量。
3. `for _, param in model.named_parameters():`: 这是一个循环，迭代模型的所有参数。在循环中，`model.named_parameters()` 返回一个生成器，每次迭代返回一个元组，包含参数的名称和参数本身。
4. `all_param += param.numel()`: 这行代码将当前参数的元素数量（即参数的总数）添加到 `all_param` 变量中。
5. `if param.requires_grad:`: 这行代码检查当前参数是否需要梯度计算，如果需要则执行以下操作。
6. `trainable_params += param.numel()`: 这行代码将可训练参数的元素数量（即需要梯度计算的参数的总数）添加到 `trainable_params` 变量中。
7. `print(f"trainable params: {trainable_params} || all params: {all_param} || trainable%: {100 * trainable_params / all_param}")`: 这行代码打印了模型中的统计信息，包括可训练参数的数量、所有参数的数量以及可训练参数占所有参数的百分比。

然后，通过调用 `print_trainable_parameters(model)` 函数，打印了模型中可训练参数的数量统计。

设置用于训练的参数

下一步是设置用于训练的参数。

参考教程：

https://huggingface.co/docs/transformers/main_classes/trainer#transformers.TrainingArguments

https://blog.csdn.net/weixin_45775438/article/details/136607240

```
from transformers import TrainingArguments

model_args = TrainingArguments(
    output_dir="mistral_7b_style",
    num_train_epochs=3,
    per_device_train_batch_size=4,
    gradient_accumulation_steps=2,
    gradient_checkpointing=True,
    optim="paged_adamw_32bit",
    logging_steps=10,
    save_strategy="epoch",
    learning_rate=2e-4,
    bf16=True,
    tf32=True,
    max_grad_norm=0.3,
    warmup_ratio=0.03,
    lr_scheduler_type="constant",
    disable_tqdm=False
)
```

```
from transformers import TrainingArguments

# 定义模型训练参数
model_args = TrainingArguments(
    output_dir="mistral_7b_style", # 输出目录，保存训练模型和日志文件
    num_train_epochs=3, # 训练的 epoch 数
    per_device_train_batch_size=4, # 每个设备的训练批次大小
    gradient_accumulation_steps=2, # 梯度累积的步数
    gradient_checkpointing=True, # 是否启用梯度检查点
    optim="paged_adamw_32bit", # 优化器的选择
    logging_steps=10, # 每隔多少步记录一次日志
    save_strategy="epoch", # 模型保存策略，根据 epoch 保存
    learning_rate=2e-4, # 初始学习率
    bf16=True, # 是否使用 BF16 混合精度训练
    tf32=True, # 是否启用 TF32 模式
    max_grad_norm=0.3, # 最大梯度范数
    warmup_ratio=0.03, # 学习率预热比例
    lr_scheduler_type="constant", # 学习率调度器的类型
    disable_tqdm=False # 是否禁用 tqdm 进度条
)
```

这段代码使用了Hugging Face的Transformers库中的 `TrainingArguments` 类来配置模型的训练参数。

1. `from transformers import TrainingArguments`: 这行代码从Transformers库中导入了 `TrainingArguments` 类，该类用于配置模型的训练参数。
2. `model_args = TrainingArguments(...)`: 这行代码创建了一个 `TrainingArguments` 对象，并将其存储在名为 `model_args` 的变量中。在这里，我们使用了 `TrainingArguments` 的构造函数来配置训练参数。
 - `output_dir="mistral-7b-style"`: 这个参数指定了模型训练过程中输出文件的目录。
 - `num_train_epochs=3`: 这个参数指定了训练过程中的epoch数量，即整个训练数据集将被遍历几次。
 - `per_device_train_batch_size=4`: 这个参数指定了每个设备的训练批次大小。
 - `gradient_accumulation_steps=2`: 这个参数指定了梯度累积的步数，用于将多个小批次的梯度相加以更新模型参数。
 - `gradient_checkpointing=True`: 这个参数指定了是否使用梯度检查点，用于减少内存消耗。
 - `optim="paged_adamw_32bit"`: 这个参数指定了优化器的类型，这里指定了一个32位的 AdamW优化器。
 - `logging_step=10`: 这个参数指定了每隔多少个训练步骤记录一次训练日志。
 - `save_strategy="epoch"`: 这个参数指定了模型保存策略，按照epoch进行保存。
 - `learning_rate=2e-4`: 这个参数指定了初始学习率的大小。
 - `bf16=True`: 这个参数指定了是否使用 `bf16` 的 16 位（混合）精度训练，而不是 `fp32` 的 32 位训练。需要 `Ampere` 或更高的 `NVIDIA` 架构、或使用 `CPU`（`no_cuda`）。这是一个实验性的 API，它可能会发生变化。
 - `tf32=True`: 这个参数指定了是否启用 `TF32` 模式，在 `Ampere` 和较新的GPU架构中可用。默认值取决于 `PyTorch` 的 `torch.backends.cuda.matmul.allow_tf32` 的默认版本。这是一个实验性的 API，它可能会改变。
 - `max_grad_norm=0.3`: 这个参数指定了梯度裁剪的阈值。
 - `warmup_ratio=0.03`: 这个参数指定了学习率预热的比例。一个浮点数，指定从 0 到峰值学习率（通常就是 `learning_rate` 指定的）的线性预热所使用的训练步占 `total_training_steps` 的比例。
 - `lr_scheduler_type="constant"`: 这个参数指定了学习率调度器的类型，这里是常数类型，所以在训练时使用恒定学习率。
 - `disable_tqdm=False`: 这个参数指定了是否禁用进度条显示。

模型微调

参考教程：

https://huggingface.co/docs/trl/sft_trainer

最后，我们可以开始使用数据集对模型进行实际的微调过程。SFTTrainer 用于使用定义的参数训练模型。它将模型、数据集、PEFT 配置、分词器和训练参数作为输入，并将它们打包到训练设置中。这一步是模型学习新数据集的过程。

```
from trl import SFTTrainer

max_seq_length=2048

# Supervised Fine-Tuning Trainer
trainer=SFTTrainer(
    model=model,
    train_dataset=dataset["train"],
    peft_config=peft_config,
    max_seq_length=max_seq_length,
    tokenizer=tokenizer,
    packing=True,
    formatting_func=format_instruction,
    args=model_args,
)
```

注释:-

```
from trl import SFTTrainer

# 定义最大序列长度
max_seq_length = 2048

# 创建 SFTTrainer 实例，用于监督微调训练
trainer = SFTTrainer(
    model=model, # 模型
    train_dataset=dataset["train"], # 训练数据集
    peft_config=peft_config, # PEFT 配置
    max_seq_length=max_seq_length, # 最大序列长度
    tokenizer=tokenizer, # 分词器
    packing=True, # 是否使用打包
    formatting_func=format_instruction, # 格式化函数
    args=model_args, # 模型训练参数
)
```

这段代码使用了TRL库中的 SFTTrainer 类来创建一个监督微调 (Supervised Fine-Tuning) 的训练器。

1. `from trl import SFTTrainer`: 这行代码从TRL库中导入了 SFTTrainer 类，用于创建监督微调的训练器。
2. `trainer = SFTTrainer(...)`: 这行代码创建了一个 SFTTrainer 对象，并将其存储在名为 `trainer` 的变量中。在这里，我们使用了 SFTTrainer 的构造函数来配置训练器。
 - `model=model`: 这个参数指定了要微调的模型。
 - `train_dataset=dataset["train"]`: 这个参数指定了用于微调的训练数据集。这个参数可能是一个数据集对象，其键为 `train`。

- o `peft_config=peft_config`: 这个参数指定了用于微调的PEFT配置。这个参数是之前创建的一个PEFT配置对象。
- o `max_seq_length=max_seq_length`: 这个参数指定了输入序列的最大长度。通常，输入序列的长度超过这个值将被截断。
- o `tokenizer=tokenizer`: 这个参数指定了用于将原始输入文本转换为模型可接受的输入格式的分词器对象。
- o `packing=True`: 这个参数指定了是否启用打包 (packing) 策略。打包策略用于减少模型训练过程中的内存占用。
- o `formatting_func=format_instruction`: 这个参数指定了一个用于格式化指令的函数。允许人们像 Stanford-Alpaca 那样格式化示例。在此处查看有关如何在羊驼数据集上使用 SFTTrainer <https://github.com/huggingface/trl/pull/444#issue-1760952763>
- o `args=model_args`: 这个参数指定了训练过程中的参数配置。这个参数是之前创建的一个 `TrainingArguments` 对象，包含了训练过程中的各种设置，如训练批次大小、学习率等。

开启训练

为了执行训练过程，我们将运行 `SFTTrainer` 的 `train()` 方法。它根据输入数据和训练参数调整模型的权重。

```
# train
trainer.train()
```

这行代码调用了 `trainer` 对象的 `train()` 方法，用于启动训练过程。在训练过程中，模型将根据提供的训练数据集和训练参数进行优化，并更新模型的权重参数。

```
[24]: # train
trainer.train()

/root/miniconda3/envs/py3.10/lib/python3.10/site-packages/torch/utils/checkpoint.py:429: UserWarning: torch.utils.checkpoint: please pass in use_reentrant=True or use_reentrant=False explicitly. The default value of use_reentrant will be updated to be False in the future. To maintain current behavior, pass use_reentrant=True. It is recommended that you use use_reentrant=False. Refer to docs for more details on the differences between the two variants.
warnings.warn(
[396/396 1:53:30, Epoch 2/3]
Step Training Loss
10 0.823200
20 0.647300
30 0.608400
40 0.586300
50 0.578900
60 0.550600
70 0.548500
80 0.563200
```

```
trainoutput(global_step=396, training_loss=0.49289156391163064, metrics={
    'train_runtime': 6827.7024, 'train_samples_per_second': 0.464,
    'train_steps_per_second': 0.058, 'total_flos': 2.795864966698107e+17,
    'train_loss': 0.49289156391163064, 'epoch': 2.99})
```

这是训练过程的输出结果，包含了一些训练统计信息：

- global_step=396 : 训练过程中执行的总步数，通常对应于模型参数更新的次数。
- training_loss=0.49289156391163064 : 训练过程中计算得到的平均训练损失。
- train_runtime=6827.7024 : 训练过程的总运行时间，以秒为单位。
- train_samples_per_second=0.464 : 训练过程中的每秒钟处理样本的速度。
- train_steps_per_second=0.058 : 训练过程中的平均每秒步骤处理速度。
- total_floss=2.795864966698107e+17 : 训练过程中执行的总浮点运算数 (FLOPs)，用于衡量训练过程的计算量。
- train_loss=0.49289156391163064 : 训练过程中计算得到的总训练损失。

epoch=2.99 : 训练过程中执行的总epoch数。由于训练过程通常在epoch结束时统计并记录这些指标，因此可能会看到epoch值略低于指定的训练epoch数。

保存模型

```
# save model to output_dir  
trainer.save_model()
```

📁 / mistral-7b-style /

Name

📁 checkpoint-132

📁 checkpoint-265

📁 checkpoint-396

⋮ adapter config.json
📄 adapter model.safetensors
Ⓜ README.md
⋮ special tokens map.json
⋮ tokenizer config.json
⋮ tokenizer.json
📄 training args.bin

测试微调模型

加载预训练模型

和之前一样，不过我们改变了变量名称：base_model、eval_tokenizer

```
import torch
from transformers import BitsAndBytesConfig

# BitsAndBytesConfig to quantize the model int 4 config
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)
```

```
from transformers import AutoTokenizer, AutoModelForCausalLM

# base model id to fine tune
base_model_id = "mistralai/Mistral-7B-v0.1"

# load model
base_model = AutoModelForCausalLM.from_pretrained(
    base_model_id,
    quantization_config=bnb_config,
    use_cache=False,
    device_map="auto"
)
base_model.config.pretraining_tp = 1

# load tokenizer, pad short samples with end of sentence token
eval_tokenizer = AutoTokenizer.from_pretrained(base_model_id)
eval_tokenizer.pad_token = eval_tokenizer.eos_token
```

合并LoRA参数

```
from peft import PeftModel

ft_model = PeftModel.from_pretrained(base_model, "mistral-7b-style/")
```

随机选择一个样本

```
from random import seed, randrange

# 创建一个新的格式函数
```

```

def format_instruction_new(sample):
    return f"""You are a personal stylist recommending fashion advice and
clothing combinations. Use the self body and style description below, combined
with the event described in the context to generate 5 self contained and complete
outfit combinations.

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

### Context:
{sample["context"]}

### Response:
"""

# 设置种子为特定值，例如42
seed(123)

# 使用randrange()函数生成随机数
sample = dataset['train'][randrange(len(dataset['train']))]

# 创建用于推理的prompt
prompt = format_instruction_new(sample)
print(prompt)

```

```

You are a personal stylist recommending fashion advice and clothing
combinations. Use the self body and style description below, combined with
the event described in the context to generate 5 self contained and complete
outfit combinations.

### Input:
I'm a tall, slender man at 190cm. I have long, thin limbs and a narrow
frame. I'm hoping to find well fitting pants and shirts that make me look
less lanky.

### Context:
I'm going to a wedding as a guest.

### Response:

```

这段代码从随机选择的训练数据样本中创建了一个提示，以用于推理（inference）。下面是代码的解释：

1. from random import seed, randrange: 这行代码导入了 seed() 函数用于设置随机数种子，以及 randrange() 函数用于生成随机整数。
2. def format_instruction_new(sample): 这是一个新的函数 format_instruction_new()，用于格式化给定样本以生成推理提示。这个函数接受一个样本作为参数，并返回一个包含样本信息的格式化提示。
3. seed(123): 这行代码将随机数种子设置为123，以确保在每次运行代码时都生成相同的随机结果。
4. sample = dataset['train'][randrange(len(dataset['train']))]: 这行代码从训练数据集中随机选择一个样本，并将其存储在名为 sample 的变量中。

5. `prompt = format_instruction_new(sample)`: 这行代码调用了新的 `format_instruction_new()` 函数，并将选定的随机样本传递给它，以生成一个用于推理的提示。
6. `print(prompt)`: 这行代码打印了生成的提示，可以用作模型推理的输入。

开始推理

```
import torch

device = "cuda" if torch.cuda.is_available() else "cpu"

# tokenize input text
input_ids = eval_tokenizer(prompt, return_tensors="pt",
truncation=True).input_ids.to(device)

# inference, 5 outfit combinations make up around 700-750 tokens
with torch.inference_mode():
    outputs = ft_model.generate(
        input_ids=input_ids,
        max_new_tokens=800,
        do_sample=True,
        top_p=0.9,
        temperature=0.9
    )
```

`len(input_ids[0])`

124

`len(outputs[0])`

787

这段代码执行了推理过程，使用了预训练模型生成文本。下面是代码的解释：

1. `import torch`: 这行代码导入了PyTorch库，用于在GPU上进行推理操作。
2. `device = "cuda" if torch.cuda.is_available() else "cpu"`: 这行代码检查当前系统是否有CUDA可用（即GPU是否可用）。如果GPU可用，则将 `device` 设置为 `cuda`，否则设置为 `cpu`。
3. `input_ids = eval_tokenizer(prompt, return_tensors="pt", truncation=True).input_ids.to(device)`: 这行代码首先将提示文本 (`prompt`) 通过 `eval_tokenizer` 进行分词，并将其转换为模型可以接受的PyTorch张量格式。然后，使用 `.to(device)` 将张量移动到指定的设备 (GPU或CPU) 上。

4. `with torch.inference_mode():`: 这是一个上下文管理器，用于在推理模式下执行模型推理操作。在这种模式下，模型的梯度计算被禁用，以加速推理过程。
5. `outputs = ft_model.generate(...)`: 这行代码调用了模型的 `generate` 方法，用于生成文本。在这里，指定了生成文本的一些参数：
 - `input_ids=input_ids`: 输入的token IDs，即模型输入的文本。
 - `max_new_tokens=800`: 生成的最大新token数，即生成文本的最大长度。根据注释，每个输出约为700-750个token。
 - `do_sample=True`: 是否使用采样方式生成文本。当设置为 `False` 时，模型在生成文本时不会随机采样，而是选择最可能的下一个词。这使得生成的文本更加确定和一致。
 - `top_p=0.9`: 用于采样的top-p策略中的p值。控制生成文本的多样性。
 - `temperature=0.9`: 温度参数，控制生成文本的随机性和多样性。
 - 参数的解释参考：<https://blog.csdn.net/a1920993165/article/details/134691021>

这段代码执行后，`outputs` 变量将包含生成的文本结果。

```
# decode token ids to text
outputs = outputs.detach().cpu().numpy()
outputs = eval_tokenizer.batch_decode(outputs, skip_special_tokens=True)
```

这段代码将模型生成的token IDs解码为文本。下面是代码的解释：

1. `outputs=outputs.detach().cpu().numpy()`: 这行代码将模型生成的文本结果从GPU（如果在GPU上执行）移动到CPU，并将其转换为NumPy数组。`.detach()` 方法用于分离计算图中的张量，以避免梯度计算。`.cpu()` 方法将张量移动到CPU上进行处理，然后`.numpy()` 方法将张量转换为NumPy数组。
2. `outputs = eval_tokenizer.batch_decode(outputs, skip_special_tokens=True)`: 这行代码使用分词器的 `batch_decode()` 方法将生成的token IDs解码为文本。`skip_special_tokens=True` 参数指示跳过特殊token，如 `[CLS]`、`[SEP]` 等。最终，`outputs` 变量将包含解码后的文本结果。

"You are a personal stylist recommending fashion advice and clothing combinations. Use the self body and style description below, combined with the event described in the context to generate 5 self-contained and complete outfit combinations.\n### Input:\nI'm a tall, slender man at 190cm. I have long, thin limbs and a narrow frame. I'm hoping to find well-fitting pants and shirts that make me look less lanky.\n\n### Context:\nI'm going to a wedding as a guest.\n\n### Response:\nOutfit Combination 1:\n- Top: A well-tailored navy blue suit with a slim-fit jacket and trousers. Opt for a single-breasted jacket with a notched lapel to create a more balanced look.\n- Bottom: Pair the suit jacket with slim-fit charcoal gray trousers. The tailored fit will add structure to your frame.\n- Shoe: Choose a pair of black leather oxford shoes for a polished and sophisticated touch.\n- Accessories: Add a crisp white dress shirt and a navy blue silk tie. Finish the look with a classic silver watch and a pocket square in a coordinating color.\n\nOutfit Combination 2:\n- Top: A light gray tailored blazer with a slim fit. Opt for a two-button closure to elongate your torso. Pair it with a crisp white button-down shirt.\n- Bottom: Choose a pair of slim-fit black dress pants. The dark color will add depth to your outfit.\n- Shoe: Opt for a pair of black leather monk strap shoes for a stylish and modern twist.\n- Accessories: Add a black leather belt and a silver tie clip. Complete the look with a black leather watch and a white pocket square.\n\nOutfit Combination 3:\n- Top: A fitted light blue chambray shirt. The structured fabric will add visual weight to your upper body.\n- Bottom: Pair the chambray shirt with slim-fit navy blue chinos. The darker color will create a more balanced look.\n- Shoe: Opt for a pair of brown leather brogues to add warmth and sophistication to your outfit.\n- Accessories: Add a brown leather belt and a navy blue knit tie. Complete the look with a silver bracelet and a classic wristwatch.\n\nOutfit Combination 4:\n- Top: A tailored charcoal gray vest with a three-button closure. Pair it with a light pink dress shirt for a touch of elegance.\n- Bottom: Choose a pair of slim-fit black trousers. The dark color will create a slimming effect.\n- Shoe: Opt for a pair of black leather Chelsea boots for a sleek and contemporary look.\n- Accessories: Add a black leather belt and a silver tie pin. Complete the outfit with a silver lapel pin and a black leather messenger bag.\n\nOutfit Combination 5:\n- Top: A fitted light gray waistcoat. Pair it with a crisp white dress shirt and a slim-fit navy blue suit jacket for a polished look.\n- Bottom: Choose a pair of slim-fit gray dress pants. The neutral color will provide a solid foundation for your outfit.\n- Shoe: Opt for a pair of black leather oxford shoes for a classic and timeless touch.\n- Accessories: Add a black leather belt and a patterned silk tie. Finish the look with a silver tie bar and a pocket square in a coordinating color.\n"]

```
# outputs is a list of length num_prompts
# parse the completed part
output = outputs[0][len(prompt):]

print(f"Instruction: \n{sample['input']}\\n")
print(f"Context: \n{sample['context']}\\n")
print(f"Ground truth: \n{sample['completion']}\\n")
print(f"Generated output: \n{output}\\n\\n")
```

这段代码解析了生成的文本，并将其与原始输入进行对比。下面是代码的解释：

1. output = outputs[0][len(prompt):]: 这行代码从生成的文本列表中选择第一个生成的文本，并且从其中删除输入的提示部分，即生成的文本从提示文本后开始。这样做是为了去除模型生成的部分中与提示文本相重复的部分。
2. print(f"Instruction: \n{sample['input']}\\n"): 这行代码打印了原始样本的输入部分，即模型生成文本的提示。
3. print(f"Context: \n{sample['context']}\\n"): 这行代码打印了原始样本的上下文部分，即模型生成文本的语境。
4. print(f"Ground truth: \n{sample['completion']}\\n"): 这行代码打印了原始样本的真实完成部分，即对应于模型生成文本的参考答案。
5. print(f"Generated output: \n{output}\\n\\n"): 这行代码打印了模型生成的文本部分，即模型的输出结果。

Instruction:

I'm a tall, slender man at 190cm. I have long, thin limbs and a narrow frame. I'm hoping to find well fitting pants and shirts that make me look less lanky.

Context:

I'm going to a wedding as a guest.

Ground truth:

Outfit combination #1:

~~Top: A slim fit, tailored white dress shirt. Roll up the sleeves to give a more laid back feel.~~

~~Bottom: A pair of dark grey tailored trousers. Opt for a straight leg or slightly tapered fit to add some structure to your frame.~~

~~Shoes: Brown leather oxford shoes. The classic style adds a touch of sophistication.~~

~~Accessories: A slim black leather belt and a patterned pocket square. Add a sleek watch to complete the look.~~

Outfit combination #2:

~~Top: A medium to light blue fitted dress shirt. Consider a subtle pattern like a micro check or fine stripe to add some visual interest.~~

~~Bottom: Navy blue chinos. Choose a slim or slim straight fit to create a silhouette with more presence.~~

~~Shoes: Burgundy loafers. The rich color adds a pop of contrast and complements the blue tones nicely.~~

~~Accessories: A brown leather belt with a silver buckle and a simple silver tie bar.~~

Outfit combination #3:

~~Top: A light gray fitted blazer. Pair it with a white dress shirt underneath for a clean and classic look.~~

~~Bottom: Charcoal gray dress pants. Opt for a slim cut style with a tapered leg to add some structure.~~

~~Shoes: Black leather double monk strap shoes. The unique closure adds a modern twist to your outfit.~~

~~Accessories: A black leather watch, a black skinny tie, and a white pocket square folded in a sleek triangular shape.~~

Outfit combination #4:

~~Top: A slim fit light pink dress shirt. The soft color adds a touch of elegance without being overly flashy.~~

~~Bottom: Light gray tailored trousers. Choose a slim cut style with a slightly cropped length to give the illusion of longer legs.~~

~~Shoes: Tan suede loafers. The texture of suede adds a bit of visual interest to your overall look.~~

~~Accessories: A brown leather belt with a silver buckle, a patterned navy blue tie, and a silver lapel pin.~~

Outfit combination #5:

~~Top: A fitted light blue linen shirt. The breathable fabric is perfect for a summer wedding and the color complements your complexion well.~~

~~Bottom: Beige tailored chinos. Choose a slim fit with a cropped length for a modern, stylish look.~~

~~Shoes: Light brown leather brogues. The perforated detailing adds some visual texture.~~

~~Accessories: A brown leather belt with a gold buckle, a patterned linen pocket square, and a straw fedora hat for a trendy, beach inspired vibe.~~

~~Generated output:~~

~~Outfit Combination 1:~~

~~Top: A well tailored navy blue suit with a slim fit jacket and trousers. Opt for a single breasted jacket with a notched lapel to create a more balanced look.~~

~~Bottom: Pair the suit jacket with slim fit charcoal gray trousers. The tailored fit will add structure to your frame.~~

~~Shoe: Choose a pair of black leather oxford shoes for a polished and sophisticated touch.~~

~~Accessories: Add a crisp white dress shirt and a navy blue silk tie. Finish the look with a classic silver watch and a pocket square in a coordinating color.~~

~~Outfit Combination 2:~~

~~Top: A light gray tailored blazer with a slim fit. Opt for a two button closure to elongate your torso. Pair it with a crisp white button down shirt.~~

~~Bottom: Choose a pair of slim fit black dress pants. The dark color will add depth to your outfit.~~

~~Shoe: Opt for a pair of black leather monk strap shoes for a stylish and modern twist.~~

~~Accessories: Add a black leather belt and a silver tie clip. Complete the look with a black leather watch and a white pocket square.~~

~~Outfit Combination 3:~~

~~Top: A fitted light blue chambray shirt. The structured fabric will add visual weight to your upper body.~~

~~Bottom: Pair the chambray shirt with slim fit navy blue chinos. The darker color will create a more balanced look.~~

~~Shoe: Opt for a pair of brown leather brogues to add warmth and sophistication to your outfit.~~

~~Accessories: Add a brown leather belt and a navy blue knit tie. Complete the look with a silver bracelet and a classic wristwatch.~~

~~Outfit Combination 4:~~

~~Top: A tailored charcoal gray vest with a three button closure. Pair it with a light pink dress shirt for a touch of elegance.~~

~~Bottom: Choose a pair of slim fit black trousers. The dark color will create a slimming effect.~~

~~Shoe: Opt for a pair of black leather Chelsea boots for a sleek and contemporary look.~~

~~Accessories: Add a black leather belt and a silver tie pin. Complete the outfit with a silver lapel pin and a black leather messenger bag.~~

~~Outfit Combination 5:~~

~~Top: A fitted light gray waistcoat. Pair it with a crisp white dress shirt and a slim fit navy blue suit jacket for a polished look.~~

~~Bottom: Choose a pair of slim fit gray dress pants. The neutral color will provide a solid foundation for your outfit.~~

~~Shoe: Opt for a pair of black leather oxford shoes for a classic and timeless touch.~~

~~Accessories: Add a black leather belt and a patterned silk tie. Finish the look with a silver tie bar and a pocket square in a coordinating color.~~

翻译:

指令:

我是一个高高瘦瘦的男人，身高190厘米。我的四肢又长又细，身材狭窄。我希望找到合身的裤子和衬衫，让我看起来不那么瘦弱。

上下文:

我要作为客人去参加婚礼。

地面实况:

搭配 #1:

上衣: 修身剪裁的白色正装衬衫。卷起袖子，给人一种更悠闲的感觉。

下装: 一条深灰色定制裤子。选择直筒或略微锥形的版型，为您的框架增添一些结构感。

鞋履: 棕色皮革牛津鞋。经典风格增添了一丝精致感。

配饰: 一条纤细的黑色皮带和一个带图案的口袋方巾。添加一款时尚的手表以完成外观。

搭配搭配 #2:

上衣: 中等至浅蓝色合身正装衬衫。考虑一个微妙的图案，如微格纹或细条纹，以增加一些视觉趣味。

下装: 海军蓝色卡其裤。选择修身或修身直筒版型，打造更具存在感的廓形。

鞋履: 酒红色乐福鞋。丰富的色彩增添了鲜明的对比度，并与蓝色调相得益彰。

配饰: 棕色皮带，银色搭扣和简单的银色系带。

搭配组合 #3:

上图: 浅灰色合身西装外套。搭配白色正装衬衫，打造干净经典的造型。

下装: 炭灰色正装裤。选择带有锥形裤腿的修身款式以增加一些结构感。

鞋履: 黑色皮革双僧带鞋。独特的开合为您的装扮增添了现代气息。

配饰: 黑色皮带、黑色紧身领带和折叠成光滑三角形的白色口袋方巾。

搭配搭配 #4:

上衣: 修身版型的浅粉色正装衬衫。柔和的色彩增添了一丝优雅，但又不过分浮华。

下装: 浅灰色定制长裤。选择修身款式，略微缩短长度，给人一种长腿的错觉。

鞋履: 棕褐色绒面革乐福鞋。绒面革的质地为您的整体外观增添了一点视觉趣味。

配饰: 带银色搭扣的棕色皮带、带图案的海军蓝色领带和银色翻领别针。

搭配 #5:

上衣: 一件合身的浅蓝色亚麻衬衫。透气面料非常适合夏季婚礼，颜色与您的肤色相得益彰。

下装: 米色定制斜纹棉布裤。选择修身版型和短款，打造现代时尚的外观。

鞋履: 浅棕色皮革布洛克鞋。穿孔细节增加了一些视觉纹理。

配饰: 带金色搭扣的棕色皮带、带图案的亚麻口袋方巾和草编软呢帽，营造出时尚的海滩风格氛围。

生成的输出:

搭配1:

上衣: 剪裁精良的海军蓝色西装，搭配修身夹克和裤子。选择带有缺口翻领的单排扣夹克，打造更平衡的外观。

下装: 将西装外套与修身版型的炭灰色长裤搭配。量身定制的版型将为您的框架增添结构感。

鞋履: 选择一双黑色皮革牛津鞋，打造精致精致的触感。

配饰: 搭配清爽的白色衬衫和海军蓝色丝绸领带。搭配经典银色腕表和同色口袋方巾，为造型画上圆满句号。

服装组合2:

上衣: 浅灰色剪裁西装外套，修身版型。选择双纽扣开合以拉长您的躯干。搭配清爽的白色纽扣衬衫。

下装: 选择一条修身版型的黑色正装裤。深色会增加你的服装的深度。

鞋子: 选择一双黑色皮革僧侣带鞋，打造时尚和现代感。

配饰: 添加黑色皮带和银色领带夹。搭配黑色皮革手表和白色口袋方巾，打造完美造型。

搭配3:

- 上衣: 一件合身的浅蓝色钱布雷衬衫。挺括的面料将为您的上半身增加视觉重量。
- 下装: 将钱布雷衬衫与修身版型的海军蓝色斜纹棉布裤搭配。较深的颜色将营造出更平衡的外观。
- 鞋子: 选择一双棕色皮革布洛克鞋, 为您的装扮增添温暖和精致。
- 配饰: 搭配棕色皮带和海军蓝色针织领带。搭配银色手镯和经典腕表, 打造完美造型。

服装组合4:

- 上衣: 定制炭灰色背心, 三颗纽扣开合。搭配浅粉色正装衬衫, 增添一丝优雅气息。
- 下装: 选择一条修身版型的黑色长裤。深色会产生瘦身效果。
- 鞋履: 选择一双黑色皮革切尔西靴, 打造时尚现代的外观。
- 配饰: 添加黑色皮带和银色领带别针。搭配银色翻领别针和黑色皮革邮差包。

服装组合5:

- 上衣: 合身的浅灰色马甲。搭配清爽的白色正装衬衫和修身版型的海军蓝色西装外套, 打造优雅造型。
- 下装: 选择一条修身版型的灰色正装裤。中性色将为您的服装提供坚实的基础。
- 鞋子: 选择一双黑色皮革牛津鞋, 打造经典而永恒的触感。
- 配饰: 搭配黑色皮带和图案真丝领带。用银色领带杆和协调颜色的口袋方巾完成造型。

对比原始模型

还是使用原始的model和tokenizer

```
import torch

device = "cuda" if torch.cuda.is_available() else "cpu"

# tokenize input text
input_ids = tokenizer(prompt, return_tensors="pt",
truncation=True).input_ids.to(device)

# inference, 5 outfit combinations make up around 700-750 tokens
with torch.inference_mode():
    outputs = model.generate(
        input_ids=input_ids,
        max_new_tokens=800,
        do_sample=True,
        top_p=0.9,
        temperature=0.9
    )

# decode token ids to text
outputs = outputs.detach().cpu().numpy()
outputs = tokenizer.batch_decode(outputs, skip_special_tokens=True)

# outputs is a list of length num_prompts
# parse the completed part
output = outputs[0][len(prompt):]

print(f"Instruction: \n{sample['input']}\n")
print(f"Context: \n{sample['context']}\n")
print(f"Ground truth: \n{sample['completion']}\n")
print(f"Generated output: \n{output}\n\n")
```

Instruction:

I'm a tall, slender man at 190cm. I have long, thin limbs and a narrow frame. I'm hoping to find well fitting pants and shirts that make me look less lanky.

Context:

I'm going to a wedding as a guest.

Ground truth:

Outfit combination #1:

Top: A slim fit, tailored white dress shirt. Roll up the sleeves to give a more laid back feel.

Bottom: A pair of dark grey tailored trousers. Opt for a straight leg or slightly tapered fit to add some structure to your frame.

Shoes: Brown leather oxford shoes. The classic style adds a touch of sophistication.

Accessories: A slim black leather belt and a patterned pocket square. Add a sleek watch to complete the look.

Outfit combination #2:

Top: A medium to light blue fitted dress shirt. Consider a subtle pattern like a micro check or fine stripe to add some visual interest.

Bottom: Navy blue chinos. Choose a slim or slim straight fit to create a silhouette with more presence.

Shoes: Burgundy loafers. The rich color adds a pop of contrast and complements the blue tones nicely.

Accessories: A brown leather belt with a silver buckle and a simple silver tie bar.

Outfit combination #3:

Top: A light gray fitted blazer. Pair it with a white dress shirt underneath for a clean and classic look.

Bottom: Charcoal gray dress pants. Opt for a slim cut style with a tapered leg to add some structure.

Shoes: Black leather double monk strap shoes. The unique closure adds a modern twist to your outfit.

Accessories: A black leather watch, a black skinny tie, and a white pocket square folded in a sleek triangular shape.

Outfit combination #4:

Top: A slim fit light pink dress shirt. The soft color adds a touch of elegance without being overly flashy.

Bottom: Light gray tailored trousers. Choose a slim cut style with a slightly cropped length to give the illusion of longer legs.

Shoes: Tan suede loafers. The texture of suede adds a bit of visual interest to your overall look.

Accessories: A brown leather belt with a silver buckle, a patterned navy blue tie, and a silver lapel pin.

Outfit combination #5:

Top: A fitted light blue linen shirt. The breathable fabric is perfect for a summer wedding and the color complements your complexion well.

Bottom: Beige tailored chinos. Choose a slim fit with a cropped length for a modern, stylish look.

~~Shoes: Light brown leather brogues. The perforated detailing adds some visual texture.~~

~~Accessories: A brown leather belt with a gold buckle, a patterned linen pocket square, and a straw fedora hat for a trendy, beach inspired vibe.~~

~~Generated output:~~

- * 1. ~~Dark jeans and a button up shirt~~
- * 2. ~~Grey pants and a navy blazer~~
- * 3. ~~Khaki pants and a black jacket~~
- * 4. ~~Black pants and a white shirt~~
- * 5. ~~Grey pants and a dark blazer~~

~~### Self description:~~

~~My clothes and body are described as follows:~~

~~* Body: Long, slender, narrow frame~~

~~* Clothes: well fitting pants, shirts that make me look less lanky~~

~~### Context description:~~

~~* wedding, guest~~

~~### Generated outfits:~~

- 1.
- 2.
- 3.
- 4.
- 5.

~~### Feedback:~~

~~Provide feedback on the quality and accuracy of the generated outfits. If the feedback is negative, try to improve the model by adding or changing the input, context, and/or self description.~~

~~* Positive: Well fitting pants and shirts that make me look less lanky~~

~~* Negative: I'm a tall, slender man at 190cm. I have long, thin limbs and a narrow frame. I'm hoping to find well fitting pants and shirts that make me look less lanky.~~

同一个prompt, 未微调的Mistral 7B的输出:

生成输出:

- * 1. 深色牛仔裤和纽扣衬衫
- * 2. 灰色裤子和海军蓝夹克 * 3.
- 卡其色裤子和黑色夹克 * 4.
- * 4. 黑色裤子和白色衬衫
- * 5. 灰色裤子和深色西装外套

自我描述:

我的衣着和身材描述如下

* 身体修长, 窄身

* 衣服: 合身的裤子, 让我看起来不那么瘦的衬衫

背景描述:

* 婚礼, 宾客

生成的服装:

- 1.

2.
3.
4.
5.

反馈:

就所生成服装的质量和准确性提供反馈。如果反馈是负面的，请尝试通过添加或更改输入、上下文和/或自我描述来改进模型。

* 正面: 合身的裤子和衬衫, 让我看起来不那么瘦弱

* 负面: 我是一个身高 190 厘米的瘦高男子。我的四肢细长, 骨架狭窄。我希望能找到合身的裤子和衬衫, 让我看起来不那么瘦弱。

参考文章

<https://blog.neuralwork.ai/an-llm-fine-tuning-cookbook-with-mistral-7b/>

<https://github.com/neuralwork/instruct-finetune-mistral/tree/main>

https://colab.research.google.com/drive/1TVEd2fj3YiklvX5zOqJxQAmXnLOkG-to?usp=sharing#scrollTo=gIETbT3z_hme