

!!!!!!!!!!!!!!!!!!!!!!!!!!!!

- !!!!!!!!!!!!!!!!!!!!!!!HG!!!!

```
import os
os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com'
```

- !!!!!!! export USE_MODELSCOPE_HUB=1 !! export HF_ENDPOINT=https://hf-mirror.com

peft!!!!!!!!!!!!!! peft !!! LoraConfig [] get_peft_model !!!!!!!!!!!!!

- LoraConfig

```
self.loraconfig = LoraConfig(
    r=config["lora"]["r"],
    target_modules=config["lora"]["target_modules"],
    bias=config["lora"]["bias"],
    lora_alpha=config["lora"]["lora_alpha"],
    lora_dropout=config["lora"]["lora_dropout"],
)
```

- r !!!!!!!!!!!!!!!
 - lora_alpha !!!!!2r!!!!
 - lora_dropout !!!!!!!!!!!!!!!0
 - target_modules !!!!!!!!!!!!!!!
- get_peft_model
 - !!!!!!!!!!!!!!!lora!!!!!!!!!!!!!!!!!!!!
 - .print_trainable_parameters() !!!!!!!!!!!!!!!

```
self.peft_model = get_peft_model(self.model,self.loraconfig)
self.peft_model.print_trainable_parameters()
```

!! transformers !!!!!!!!!!!!!!!!!!!!!!!

- AutoModelForCausalLM
 - !!!!!!!!!!!!!!!
 - device_map !!!!!!!!!!!!!!!
 - quantization_config !!!!!!!
 - trust_remote_code !!!!!!!!!!!!!!!

```
self.model = AutoModelForCausalLM.from_pretrained(
    config["model"]["model_path"],
    device_map=config["model"]["device_map"],
    trust_remote_code=config["model"]["trust_remote_code"],
```

```
quantization_config=quan_configs,
)
```

- AutoTokenizer

- 初始化AutoTokenizer

```
self.tokenizer = AutoTokenizer.from_pretrained(
    config["model"]["model_path"],
    trust_remote_code=config["tokenizer"]["trust_remote_code"],
)
```

- 使用tokenizer

- max_length 最大长度
- return_tensors 返回 pt (PyTorch)
- ``truncation`` 截断

```
tokenized = self.tokenizer(
    combined_texts,
    truncation=True,
    max_length=config["tokenizer"]["max_length"],
    padding=False,
    return_tensors=None # Python
)
```

- .tokenizer.encode 将文本转换为 Token ID

- text 文本
- add_special_tokens 添加特殊token
- max_length 最大长度
- truncation 截断
- return_tensors 返回 pt

```
turn_tokens = self.tokenizer.encode(
    self.tokenizer.apply_chat_template([turn],
    tokenize=False),
    add_special_tokens=False
)
```

- .apply_chat_template 应用聊天模板

- tokenize 是否token IDs true
- return_tensors 返回 "pt" (PyTorch), "np" (NumPy)
- max_length 最大token

- `truncation` `bool`
- `padding` `str` "longest" `bool` `True` `bool` `False`
- `return_dict` `bool`

```
tokenized = self.tokenizer.apply_chat_template(
    conversation,
    tokenize=True,
    max_length=config["tokenizer"]["max_length"],
    truncation=config["tokenizer"]["truncation"],
    return_tensors=None,
    return_dict=True
)
```

- `BitsAndBytesConfig`

- `load_model`
 - `load_in_8bit` `bool`
 - `load_in_4bit` `bool`
 - `torch_dtype` `torch.dtype`

```
quan_configs = BitsAndBytesConfig(
    load_in_8bit=config["model"]["load_in_8bit"],
    torch_dtype=config["model"]["torch_dtype"]
)
```

- `Trainer`

- `Trainer`
 - `model` `torch.nn.Module`
 - `tokenizer` `PreTrainedTokenizer`
 - `args` `TrainingArguments`
 - `train_dataset` `Dataset`
 - `eval_dataset` `Dataset`
 - `data_collator` `DataCollator`
 - `callbacks` `TrainerCallback`

```
trainer = Trainer(
    model=model,
    tokenizer=self.tokenizer,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    data_collator=data_collator,
```

```
callbacks=[EarlyStoppingCallback(config["training"]
["patience"])]
)
```

- TrainingArguments

- 参数

- output_dir 输出目录
- learning_rate 学习率
- num_train_epochs 训练轮数
- weight_decay 权重衰减L2正则
- per_device_train_batch_size 每个设备训练批次大小
- per_device_eval_batch_size 每个设备评估批次大小
- eval_strategy 评估策略
- save_strategy 保存策略
- eval_steps 每N步评估
- save_steps 每N步保存
- logging_steps 每N步记录
- logging_dir 记录TensorBoard目录
- report_to 报告
- fp16 混合精度训练
- gradient_accumulation_steps 梯度累积步数
- gradient_checkpointing 梯度检查点ing 节省显存
- max_grad_norm 梯度范数
- load_best_model_at_end 是否在训练结束后加载最佳模型
 - 是否保存最佳模型
- lr_scheduler_type 学习率调度器
 - cosine 余弦
 - linear 线性
- metric_for_best_model 最佳模型指标

```
training_args = TrainingArguments(
    output_dir=config["model"]["output_path"],
    learning_rate=config["training"]["learning_rate"],
    num_train_epochs=config["training"]["num_epochs"],
```

```

        weight_decay=config["training"]["weight_decay"],
        per_device_train_batch_size=config["training"]
["per_device_train_batch_size"],
        per_device_eval_batch_size=config["training"]
["per_device_eval_batch_size"],
        eval_strategy=config["training"]["eval_strategy"],
        save_strategy=config["training"]["save_strategy"],
        eval_steps=config["training"]["eval_steps"],
        save_steps=config["training"]["save_steps"],
        logging_steps=config["training"]["logging_steps"],
        logging_dir=config["training"]["logging_dir"],
        report_to=config["training"]["report_to"],
        load_best_model_at_end=config["training"]
["load_best_model_at_end"],
        fp16=config["training"]["fp16"],
        gradient_accumulation_steps=config["training"]
["gradient_accumulation_steps"],
        dataloader_num_workers=4, # 默认值
    )

```

- `datasets.Dataset` 来自 Hugging Face `datasets` 库

- `.from_dict()` 从字典中加载数据集
- `.map()` 对数据集进行映射
 - `function` 函数
 - `batched` 是否按批次
 - `batch_size` 批次大小
 - `num_proc` 进程数
- `.save_to_disk` 保存数据集到磁盘

- `load_dataset` 来自 `datasets` 库

- `split` 数据集分割
- `streaming` 流式加载，避免 OOM

```

self.dataset = load_dataset("BelleGroup/multiturn_chat_0.8M",
split="train[:70%]",streaming=true)

```

- `DataCollatorForSeq2Seq` 用于序列到序列的数据收集器

- `tokenizer` 分词器
- `model` 模型
- `padding` 填充
- `max_length` 最大长度
- `pad_to_multiple_of` 填充到指定的倍数
- `label_pad_token_id` 标签填充 token ID
- `return_tensors` 返回的张量类型

```
data_collator = DataCollatorForSeq2Seq(  
    tokenizer=self.tokenizer,  
    pad_to_multiple_of=config["dataloader"]  
["pad_to_multiple_of"],  
    return_tensors=config["dataloader"]["return_tensors"],  
    padding=config["dataloader"]["padding"],  
    label_pad_token_id=config["dataloader"]  
["label_pad_token_id"]  
)
```