GRPO

实验设计：

数据集：具有100多个品牌的12多k 的商品数据集。

规则设计：让模型根据图片在100多个选项中，选出商品的品牌信息。如果选择正确，加1分，否则不加分。

```
#提示词的内容
prompt = "<image>\nPlease identify the most appropriate brand from the following                 options based on the image content:\n"
for i, brand in enumerate(self.all_brands):
    prompt += f"{i+1}. {brand}\n"
prompt += "Please only respond with the brand name, no additional explanation needed."
```

实验内容：

a. SFT的Internvl2.5 + GRPO

b. Pretrain的Internvl2.5 + GRPO

实验设备：3090（24GB）×2

进展：

目前完成训练代码的大体内容，仍在排除bug。

问题：

1. Lora的设置，目前使用peft的loraconfig，传入GRPOTrainer，不确定后续会不会出现问题
2. GRPO设置了per device batch size后，不需要再Dataset类里设置batch size，否则数据传递时实际的batch size会相乘，同时，GRPO的batch会嵌套Dataset的batch
3. Internvl的模型重载generate的函数，在自定义的函数内容调用了Qwen原始的generate函数，而GRPOTrainer的会定位到Internvl的generate函数，二者的参数不同，所以参数对齐出现错误，比如，出现图像的shape是[batch_size, sequenze_length]的情况。
4. GRPO会自行调用tokenizer去分词提示词，从而获得inputs_ids和inputs_masks所以难以通过改变输入内容的方式去调整，比如，增加数据集的属性inputs_ids和inputs_masks。我采取的解决方法是在GRPOTrainer的类中加特判，对于Internvl的模型，使用它自己的处理输入的方法：

```
if self.model.name_or_path == "Internvl2_5":
    print("单独为Internvl2_5准备输入")

else:#GRPO通用的方法
    prompt_inputs = self.processing_class(
    prompts_text, return_tensors="pt", padding=True, padding_side="left",
add_special_tokens=False
    )
    prompt_inputs = super()._prepare_inputs(prompt_inputs)
    prompt_ids, prompt_mask = prompt_inputs["input_ids"],
prompt_inputs["attention_mask"]
```

5. 由于GRPOTrainer在真正调用generate的位置和用tokenizer获取input_ids之间还调用最大输入长度固定input_ids的长度，而此时Internvl还没有对数据进行处理，所以我将最大长度设置为None。之所以不将Internvl的函数放在前面，因为我想让它和generate并列：

```
if self.model.name_or_path == "Internvl2_5":
    print("单独为Internvl2_5准备生成")
    pixel_values = torch.cat([x['pixel_values'] for x in inputs], dim=0)
    questions = [x['prompt'] for x in inputs]
    num_patches_list = [x['pixel_values'].shape[0] for x in inputs]
    prompt_ids, prompt_mask, completion_ids, responses =
self.model.batch_chat(
        self.processing_class,
        pixel_values,
        questions,
        dict(max_new_tokens=self.max_completion_length, do_sample=True),
        num_patches_list,
        history=None,
        return_history=False,
        is_all=True
    )
```

6. 接下来的问题是，Internvl 处理后的outputs_ids是切除input_ids后的生成的内容，但是GRPO默认的是模型输出的未切除的内容，所以而这三个变量在后续都被使用到，所以继续添加了不同的处理：

```
#Internvl
prompt_completion_ids = torch.cat([prompt_ids, completion_ids], dim=1)

#GRPO通用
prompt_length = prompt_ids.size(1)
```

```
prompt_ids = prompt_completion_ids[:, :prompt_length]
completion_ids = prompt_completion_ids[:, prompt_length:]
```

7. 接下来的问题是，参考模型的对数概率计算，在没有预设的情况下，使用待训练的模型参考，但是此时传递了一个‘logits_to_keep’的新参数，而在Internvl 的forward 的过程中，没有该参数。而且它还不传递图像的参数，Internvl需要。该问题正在解决，预计在这个位置，特判重写。

```
with self.accelerator.unwrap_model(self.model).disable_adapter():
    ref_per_token_logps = self._get_per_token_logps(
        self.model, prompt_completion_ids, attention_mask, logits_to_keep
    )
```

所有问题的根源和解决：

出现上面所有问题的根源是grpo针对llm，而internvl是vlm。vlm类没有继承llm所有的属性，只是把llm当自己的成员，像logits_to_keep实际上是vlm.language_model的向前传播的参数，所以我在第7个问题上特判用self.model.language解决了问题，后续代码可以跑通了。

更好的解决思路是在外部重新设计类，以适应grpo的trainer，不修改库函数很难实现，因为它在处理输入时只有prompt一个输入，并且它将其默认为文本，后面有对该文本的获取模板的操作（后续还有对它的奖励函数的操作），然后给它放入分词器中，获取inpupts_ids等信息，在整个过程中，如果我们把对图像的处理放在分词器中，是最简单的，只用在外部重构分词器，但是，库函数中根本没给图像输入留参数。所以我们得继续修改获取模板的函数，和奖励函数，以及后续设计到对于文本的操作。

```
prompts = [x["prompt"] for x in inputs]
prompts_text = [maybe_apply_chat_template(example, self.processing_class)["prompt"] for example in inputs]
#特判
if self.model.name_or_path == "Internvl2_5":
    print("单独为Internvl2_5准备输入")
else:
    prompt_inputs = self.processing_class(
        prompts_text, return_tensors="pt", padding=True, padding_side="left", add_special_tokens=False
    )#分词的处理
    prompt_inputs = super()._prepare_inputs(prompt_inputs)
    prompt_ids, prompt_mask = prompt_inputs["input_ids"], prompt_inputs["attention_mask"]
```

暂时的解决方法，在第7个问题的特判后，就可以进行实验了。

新的问题：

首先是显存的问题，auto map后模型会成为DataParallel，然后又会丧失一大部分属性，导致代码无法运行，如果手动重建继承类，在过掉重建类后，还是会出现同样的问题，有点套娃的感觉。

所以在单卡的情况下，就将lora的训练参数量下调：

```
def get_lora_config():
    return LoraConfig(
        task_type="CAUSAL_LM",
        r=2,
        lora_alpha=8,
        lora_dropout=0.1,
        target_modules=["q_proj", "k_proj"],
        bias="none"
    )
```

2025/3/31更新

我们在Intern-VL-2.5-7B, Qwen2.5-VL-7B, Gemma3-4B三个模型上进行了训练测试。测试结果并未看到有准确率的提升。

InternVL:

重建类后，我们使用accelerate进行训练的管理，详见vlm_grpo_lora.py。我们在accelerate config时设置了gpu使用后，就不必再用环境变量控制。问题设置的基本思路是让模型从一个包含所有品牌的list中找到正确的答案，并且不让模型做解释。奖励函数的设置只看答案的对错赋分。这样的做法不符合grpo的原则，但是Input的长度过长同时输出很短，这样的请况下跑通了代码。为后续的工作准备了一个基本的训练模板。

work > chat_grpo_output > checkpoint-150 > {} trainer_state.json > ...


```
"log_history": [
  {
    "grad_norm": 0.0,
    "kl": 0.0024821037222864106,
    "learning_rate": 9.933333333333334e-06,
    "loss": 0.0001,
    "reward": 0.0,
    "reward_std": 0.0,
    "rewards/reward_func": 0.0,
    "step": 1
  },
  {
    "completion_length": 5.625,
    "epoch": 0.0012531328320802004,
    "grad_norm": 0.0,
    "kl": 0.0015911260670691263,
    "learning_rate": 9.866666666666668e-06,
    "loss": 0.0001,
    "reward": 0.03125,
    "reward_std": 0.0625,
    "rewards/reward_func": 0.03125,
    "step": 2
  },
```


这里是151、152step的训练情况。

Qwen2.5:

Qwen2/2.5-VL系列和InternVL面临同样的问题，这次我从开源项目open_r1的源码找到可用的内容，稍加修改（添加量化的代码，和某些超参数的赋值），就可以解决问题。源码位于./open_r1/下，和Qwen的训练脚本保持一致。

这次使用了qlora，依旧只微调线性层，然后问题模板换成如下内容：

```
prompt_text = "Please tell me the brand of the product in the picture between labels <answer/> and </answer> and explain the reason between labels <thinking/> and </thinking>"
```

量化配置：

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

Lora配置：

```python
peft_config = LoraConfig(
    task_type="CAUSAL_LM",  # 因为是Causal Language Model
    inference_mode=False,
    r=8,                     # LoRA 秩
    lora_alpha=32,           # LoRA alpha参数
    lora_dropout=0.1,        # Dropout概率
    target_modules=[         # 需要训练的模型层
        "q_proj",
        "k_proj",
        "v_proj",
        "o_proj",
        ],
    bias="none",
)

```

GRPO配置：

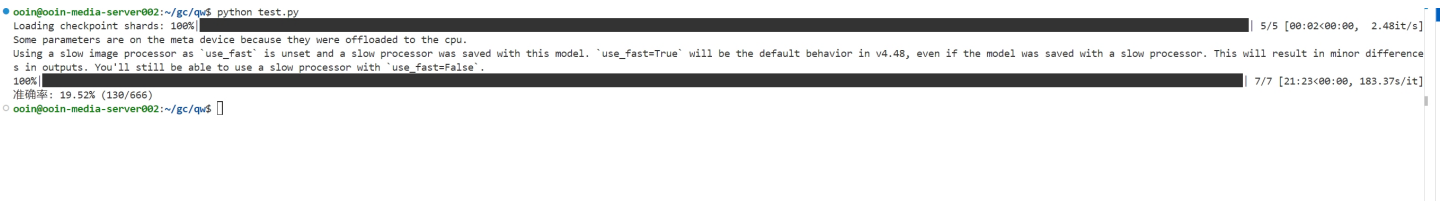
```python
def get_training_args():
    args = GRPOConfig(
        output_dir="chat_grpo_output",
        num_generations=6,
        learning_rate=1e-5,
        logging_steps=100,
        max_prompt_length=None,
        gradient_accumulation_steps=1,
        max_completion_length=200,
        per_device_train_batch_size=3,
        max_steps=1000,
        dataloader_pin_memory=False,
        model_init_kwargs={
            "quantization_config": bnb_config,
            "torch_dtype": torch.bfloat16,
            "use_cache": False
        }
    )
    args.epsilon = 0.2
    args.num_iterations = 1
    return args
```

这里额外添加的两个参数实际上是GRPOConfig就有的，可惜在open_r1的源码中没有传递进去，所以额外添加。

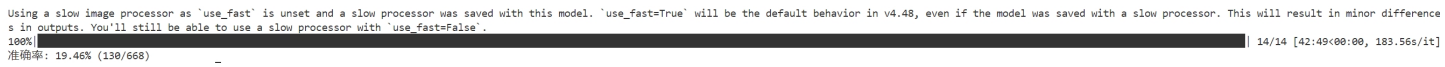
奖励函数除了提取答案的内容进行对比，还要查看思考的内容是否存在，如果答案正确且思考内容存在会额外加分。

最后的测试结果：

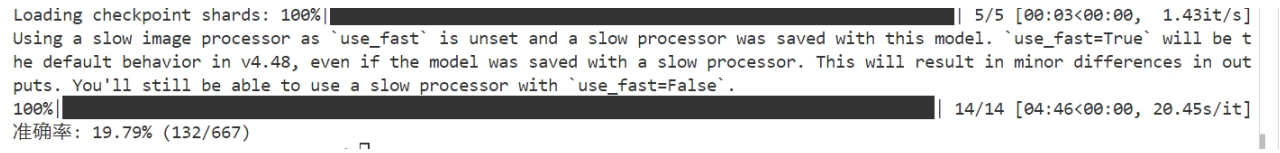
0:



500steps:



1000steps:



根据博客

https://kalomaze.bearblog.dev/grpo-judge-experiments-findings-and-empirical-observations/后面针对奖励函数的改进：

将每个batch的准确率的立方赋给正确的样本，同时根据样本的think的长度赋分。

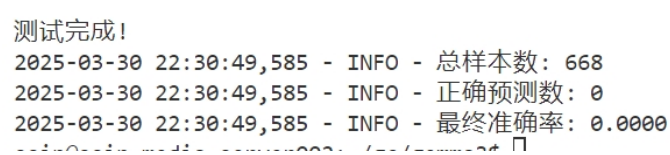
同时，修改了lora的配置，只在语言模型部分lora，其它的全参数训练。最后的结果同样是没有提升。

Gemma3：

Gemma3对比前两个模型有很大的优点，就是完全适配GRPOTrainer的参数配置。即模板化后可以把它当LLM使用。但是使用时会出现layer没有分配到device的错误，使用unsloth加载模型会避免这个问题，但它只支持单卡训练，设置上又从多卡训练来计算。比如，有4张卡，当设置num_generation为2时，会自动调整batch_size到2×4=8，但是它只用一张卡来加载batch数据。但模型在训练时auto map到4张卡上没有问题。

我们使用了4B的模型，用了unsloth的lora设置，没有进行量化。

优点明显的Gemma3最大的缺点就是准确率为0（训练前后）。



```
测试完成！
2025-03-30 22:47:37,180 - INFO - 总样本数: 668
2025-03-30 22:47:37,180 - INFO - 正确预测数: 0
2025-03-30 22:47:37,180 - INFO - 最终准确率: 0.0000
ooin@ooin-media-server002:~/gc/gemma3$
```

即使是Gemma3-12B的准确率（未训练，无法训练）也是0：

```
2025-03-30 23:25:31,700 - INFO - 总样本数: 668
2025-03-30 23:25:31,700 - INFO - 正确预测数: 0
2025-03-30 23:25:31,700 - INFO - 最终准确率: 0.0000
ooin@ooin-media-server002:~/gc/gemma3$ python test.py
```

结论：GRPO不满足商品品牌和图像的匹配任务，效果不如sft。