Final Report

简介：t1 t2完全完成，t3 完成了一半

一、 T1 Llama Model

在t1中，需要实现LlamaModel类,完成load_config、load_parameters等函数，加载已经预训练好的llama_3.2_1b_instruct模型，然后在forward函数中实现前向传递过程，对于给定输入能计算出prob概率分布。其次还有num_parameters、get_kv_cache等函数，均可以直接调用a1-4已经实现好的基类。

1.load_config

调用./utils.py中实现好的load_json函数可以读取llama_3.2_1b_instruc的config.json文件，需要实现的就是将对应的参数填写进我的model中，需要注意的有两点：1.会有一些参数含义一致，但命名存在区别；2.会有一些在config.json中并未列出的参数，但在我们的实现中需要，这些参数被**extract_config传入。

为了精简代码，我定义了一个sync_para函数，用于避免大量重复操作

```
def sync_para(config_key: str,my_key: str):
    if config_key in llama_config and config_key in extra_configs:
        assert(0)
    if config_key in llama_config:
        my_config[my_key] = llama_config[config_key]
    if config_key in extra_configs:
        my_config[my_key] = extra_configs[config_key]
```

首先，如果一个参数同时出现在config.json 和extra_config中应该视为bug，然后根据config_key从config.json 和extra_config找到具体值v，my_key - v 键值对填入my_config中。

然后load_config 其实就是大量调用上面的sync_para函数来填写my_config这个字典，具体代码太长，不在此处赘述。

2.load_parameters

调用./utils.py中实现好的load_safetensors函数可以读取对应的参数保存文件，然后就是将参数矩阵一一对应赋值。以token编码层举例,:

```
pre_para = load_safetensors(params_files)
self.model.vocab_emb.embedding =
nn.Parameter(pre_para["model.embed_tokens.weight"].to(dtype=pdtype,device=pdevice))
```

其中在定义decoder_layer时，注意有的矩阵要转置。

3.forward

分为训练和推理两种模式

推理模式时，根据有没有cu_seqlens参数有不同处理,输出的是下一个token的prob概率分布

```
if cu_seqlens is not None:
    len=len(cu_seqlens)-1
    prob = torch.zeros(len,self.config.vocab_size)
    for i in range(len):
        #取出每一句最后一个token的logits进行softmax，temperature是调温参数
        prob[i] = F.softmax(logits[:,cu_seqlens[i+1]-1,:]/temperature,dim=-1)
    return prob
else:
    #没有cu_seqlens,取出最后一个即可
    return F.softmax(logits[:, -1, :].view(b,-1)/temperature,dim=-1)
```

训练模式时，根据logits和labels进行loss计算

```
if cu_seqlens is None:
    #logits : w1 w2 w3 w4 w5
    #label  : w1 w2 w3 w4 w5
    #在训练时
    #logits_:  w1 w2 w3 w4 去预测w5
    #label_:   w2 w3 w4 w5 对应上面logits每一位的下一个token
    logits_ = logits[:,:-1,:]
    labels_ = labels[:,1:]
    #改变形状，从而好进行cross_entropy计算
    logits_ = logits_.reshape(-1, self.config.vocab_size)
    labels_ = labels_.reshape(-1).to(logits_.device)
    return F.cross_entropy(logits_, labels_, reduction="mean")
else:
    logits = logits.squeeze(0)
    labels = labels.squeeze(0)
    logits_ = torch.empty((0,self.config.vocab_size),\
              device=logits.device,dtype=logits.dtype)
    labels_ = torch.empty(0,device=labels.device,dtype=labels.dtype)
    for i in range(len(cu_seqlens)-1):
    #逻辑同上一个分支
        logits_ =
torch.cat((logits_,logits[cu_seqlens[i]:cu_seqlens[i+1]-1,:]),dim=0)
        labels_ =
torch.cat((labels_,labels[cu_seqlens[i]+1:cu_seqlens[i+1]]),dim=0)
    labels_ = labels_.to(logits_.device)
    return F.cross_entropy(logits_, labels_, reduction="mean")
```

4.test_result

在docker镜像中pytest test_toy_task1.py 结果如下，测试文件中稍作修改，以便打印结果可读性更好。

打印prob张量

```
hf_prob: torch.Size([1, 128256]) torch.bfloat16 tensor([[9.0525e-07, 6.4075e-07, 1.3504e-08,  ..., 1.2824e-10, 1.2824e-10,
        1.2824e-10]], dtype=torch.bfloat16)
ref_prob: torch.Size([1, 128256]) torch.bfloat16 tensor([[9.6858e-07, 6.8918e-07, 1.4901e-08,  ..., 1.3551e-10, 1.3551e-10,
        1.3551e-10]], dtype=torch.bfloat16)
my_prob: torch.Size([1, 128256]) torch.bfloat16 tensor([[9.9093e-07, 7.2271e-07, 1.5250e-08,  ..., 1.3824e-10, 1.3824e-10,
        1.3824e-10]], dtype=torch.bfloat16)
```

可见，张量形状一致，但具体值上，三者各有差异，下面调用assert_close进行详细比较。

```
 my_prob vs hf_prob as followed:
Tensor-likes are not close!

Mismatched elements: 189 / 128256 (0.1%)
Greatest absolute difference: 0.015625 at index (0, 539) (up to 1e-05 allowed)
Greatest relative difference: 0.1708984375 at index (0, 1403) (up to 0.016 allowed)

 my_prob vs ref_prob as followed:
Tensor-likes are not close!

Mismatched elements: 89 / 128256 (0.1%)
Greatest absolute difference: 0.0078125 at index (0, 539) (up to 1e-05 allowed)
Greatest relative difference: 0.1591796875 at index (0, 14188) (up to 0.016 allowed)
```

可见，仅有0.1%的不匹配率，且相对误差和绝对误差均较小，个人觉得此误差是出于前面a1-4实现上的精度细节问题累计导致。将ref和hf的结果比较，也可见两者有微笑误差。下附图。

```
 ref_prob vs hf_prob as followed:
Tensor-likes are not close!

Mismatched elements: 175 / 128256 (0.1%)
Greatest absolute difference: 0.0234375 at index (0, 539) (up to 1e-05 allowed)
Greatest relative difference: 0.1796875 at index (0, 40075) (up to 0.016 allowed)
```

最后输出的loss也都比较接近，甚至我的model loss计算结果最低，😊。

```
hf_loss: tensor(3.3766, grad_fn=<NllLossBackward0>)
ref_loss: tensor(3.3800, grad_fn=<NllLossBackward0>)
my_loss: tensor(3.3750, dtype=torch.bfloat16, grad_fn=<NllLossBackward0>)
```

二、 T2 Inference Agent

要完成占位符、推理代理的功能。

1.prompt.py

比较简单。初始化的时候，记得用正则表达式从string中取出{}的部分即可,然后将占位符对应的value都初始化为none。

```
self.template_str=template_str
#用正则表达式匹配str中的{}
keys = re.findall(r'(?<={)(.*?)(?=})', self.template_str)
self.dict={}
for key in keys:
    self.dict[key]=None
```

keys 和 set_default函数都比较简单，一个是直接返回上面初始化的dict，一个是用新的dict参数更新上面的dict

forward时，尝试将template_str中的占位符都替换成对应的value

```
#存储prompt-value键值对的字典
prompt_dict={}
#便利每一个占位符
for  key in self.dict.keys():
    #如果参数中有该占位符对应的value，则使用
    if key in kwargs.keys():
```

```
            prompt_dict[key]=kwargs[key]
        else:
            #如果传入参数中没有，则检查该占位符是否有default默认的value，如果没有，报错
            if self.dict[key] is None:
                raise ValueError(f"KEY: '{key}' has no value")
             else:
                prompt_dict[key]=self.dict[key]
    #用format函数将占位符替换成对应的value
    return self.template_str.format(**prompt_dict)
```

2.agent

load_generation_config 同t1的load_config，没什么值得细说的。

set_prompt 和 get_prompt 就是区分一下prompt type是system还是context就行。

主要是forward,执行推理的部分。

首先是准备工作

```
query = convert_to_list(query)
system_prompt = self.system_prompt(**kwargs)
context_prompt = self.context_prompt(**kwargs)
#组织input，然后编码
input = [system_prompt + context_prompt + q for q in query]
input_ids = self.tokenizer.encode(input)
```

然后进行裁剪和padding

```
#剪裁
if self.config.truncate_length is not None:
    if self.config.truncate_side == TruncateSide.RIGHT:
        input_ids = [id[:self.config.truncate_length] for id in input_ids]
    elif self.config.truncate_side == TruncateSide.LEFT:
        input_ids = [id[-self.config.truncate_length:] for id in input_ids]
#填充
pad_len = 0
for ids in input_ids:
pad_len = max(pad_len, ids.shape[0])
#保证pad_len整除pad_to_multiple_of
if pad_len % self.config.pad_to_multiple_of != 0:
    pad_len =pad_len+self.config.pad_to_multiple_of - \
                pad_len % self.config.pad_to_multiple_of
#进行填充
for i, id in enumerate(input_ids):
    if self.config.padding_side == PaddingSide.LEFT:
        input_ids[i] = F.pad(id, (pad_len - id.shape[0], 0),
value=self.tokenizer.bos_id)
    else:
        input_ids[i] = F.pad(id, (0, pad_len - id.shape[0]),
value=self.tokenizer.eos_id)
input_ids = torch.stack(input_ids, dim=0).to(self.config.device)
```

然后就是生成阶段，需要注意的是streaming时，要打印每一次的第一个token

```
#生成阶段，需要判断是否开启streaming,流模式下，生成一个token打印一次，而不是最后一起输出
if self.config.streaming is True:
    print(self.tokenizer.decode(input_ids[0])[0],end='')
input_ids, next_token = self.get_next_token(input_ids,input_ids)
if self.config.streaming is True:
    print(self.tokenizer.decode(next_token)[0],end='')
cur_len = 1
# 循环生成
while cur_len < self.config.max_new_tokens:
    input_ids, next_token = self.get_next_token(input_ids,next_token.unsqueeze(1))
    if self.config.streaming is True:
        print(self.tokenizer.decode(next_token)[0],end='')
    cur_len += 1
```

最后将生成的结果，从token解码成文本，然后按格式组织再返回即可。

其中使用了两个自定义函数，get_next_token,k_p_sample

```
def get_next_token(
        self,
        input_ids: torch.LongTensor,
        now_token_ids: torch.LongTensor,
    ) -> (torch.LongTensor, torch.LongTensor):
        probs = self.model(now_token_ids,temperature=self.config.temperature)
        # 采取sample策略
        if self.config.decode_strategy == DecodeStrategy.SAMPLING:
            #对prob进行k p sample的预处理
            probs = k_p_sample(probs,k=self.config.top_k,p=self.config.top_p)
            #归一化
            probs = probs / probs.sum(dim=-1,keepdim=True)
            torch.manual_seed(self.config.sampling_seed)
            #执行采样
            next_token = torch.multinomial(probs, num_samples=1).squeeze(1)
        elif self.config.decode_strategy == DecodeStrategy.GREEDY:
            #贪心则取最大prob值对应的token
            next_token = torch.argmax(probs,dim=-1)
        #将新的token拼到最后
        input_ids = torch.cat([input_ids, next_token[:,None]], dim=-1)
        del probs
        return input_ids, next_token
```

```
def k_p_sample(probs: torch.Tensor, k: int =10, p: float = 1.0) -> torch.Tensor:
    def k_sample(probs: torch.Tensor):
        #同之前的实验
        result = torch.zeros_like(probs)
        values, indices = torch.topk(probs, k=k, dim=-1)
        result.scatter_(dim=-1,index=indices,src=values)
        return result
    def p_sample(probs: torch.Tensor):
        #copy from the web provided by references
        sorted_probs, sorted_indices = torch.sort(probs, dim=-1, descending=True)
        cumulative_probs = sorted_probs.cumsum(dim=-1)
        mask = cumulative_probs < p
        #修改mask，多往后取一位，则就是最小的index保证cumulative结果大于等于p
```

```
        mask = torch.cat([mask.new_ones(mask.shape[:-1]+
(1,)),mask[...,:-1]],dim=-1)
        sorted_probs[~mask] = 0
        res =
torch.zeros_like(sorted_probs).scatter(dim=-1,index=sorted_indices,src=sorted_probs
)
        return res
    return p_sample(k_sample(probs))
```

3.test_result

直接对比hf ref my_model三者的续写结果

hf如下：

```
**********hf result**********

========== The 0-th sample in the batch ==========
[generated_text]: You're a helpful assitant on life.
Fill the sentence below for you to make it reasonable.
The key to life is to focus on your passion and pursue your dreams with persistence and dedication.
What is the recommended dosage of

========== The 1-th sample in the batch ==========
[generated_text]: You're a helpful assitant on life.
Fill the sentence below for you to make it reasonable.
The only thing we have to fear is the noise of our own ignorance.
I think I can complete that sentence to make it suitable to fit

========== The 2-th sample in the batch ==========
[generated_text]: You're a helpful assitant on life.
Fill the sentence below for you to make it reasonable.
The cat jumped on the keyboard and accidentally wrote a letter to my boss, telling him that the company was under attack.

Can you explain
```

ref如下：

```
**********ref_model result**********

========== The 0-th sample in the batch ==========

[PromptType.ALL]: You're a helpful assitant on life.
Fill the sentence below for you to make it reasonable.
The key to life is to be a good friend to others.
The best way to do this is to be a good

========== The 1-th sample in the batch ==========

[PromptType.ALL]: You're a helpful assitant on life.
Fill the sentence below for you to make it reasonable.
The only thing we have to fear is fear itself. (Franklin D. Roosevelt)

What is the only thing we have to fear is

========== The 2-th sample in the batch ==========

[PromptType.ALL]: You're a helpful assitant on life.
Fill the sentence below for you to make it reasonable.
The cat jumped on the keyboard and accidentally typed out a message that said "I'm purr-fectly happy to be stuck in this
```

my_model如下：

```
************************my_model result************************

======================== The 0-th sample in the batch ========================

[PromptType.ALL]: You're a helpful assitant on life.
Fill the sentence below for you to make it reasonable.
The key to life is to be a good friend to others.
The best way to do this is to be a good

======================== The 1-th sample in the batch ========================

[PromptType.ALL]: You're a helpful assitant on life.
Fill the sentence below for you to make it reasonable.
The only thing we have to fear is fear itself. (Franklin D. Roosevelt)

What is the only thing we have to fear is

======================== The 2-th sample in the batch ========================

[PromptType.ALL]: You're a helpful assitant on life.
Fill the sentence below for you to make it reasonable.
The cat jumped on the keyboard and accidentally typed out a novel.

What was the cat's motivation for doing so?

A) To get attention
```

by the way,解释一句，上面我的模型 case3 的输出 —— A) = ANSWER)

可见，我的实现和ref在case 1 2上续写结果一致，在case3上不同，hf则和其他两个三种case的结果都不同。但细读每一个续写结果，都是符合语言逻辑的，可能会有一些思想跳脱，但并没有低级错误。

个人认为，上面的续写差异是由于t1的prob计算三者就存在细微差异导致的。

在和别的同学沟通后，也佐证了我上面的猜想。

故，t2至此不再细究续写结果是否和ref完全一致。

三、T3 LoRA Trainer

出于时间原因，并未完成t3，令人遗憾。但从内容来看，t3分为两个任务，一，准备数据集，从json文件中读取数据，然后进行编码、裁剪、填充等多次处理。二、编写训练器trainer。我只完成了第一部分。

1.qa.py

几个函数都很基本，除了batch。

一开始我以为只是单纯的load_jsonl，然后读取出的dict就是要用的数据。

但，通过test_toy 和打印内容可知。batch 的返回值不是单纯的将load_jsonl的内容取一个batch返回。需要将load_jsonl加载的数据，组织整理成由不同key-value组成的dict,以qa为例，其中包含sample、input_ids、labels等。

遂，定义了reformat_data函数，对load的dict_list重新进行组织。

```python
question_prefix = self.config.prefix_template(prefix=self.config.question_prefix)
answer_prefix = self.config.prefix_template(prefix=self.config.answer_prefix)
#将问题和答案拆分开，并分别加上前缀
```

```
question = [question_prefix + i[self.config.question_key] + self.config.sep_str for
i in data]
labels = [i[self.config.answer_key] for i in data]
#将问题和答案转换为token
token_question = self.tokenizer.encode(question)
answer_prefix = self.tokenizer.encode(answer_prefix)[0]
answer_pre_len=answer_prefix.shape[0]
labels = self.tokenizer.encode(labels)
#将答案开头的无效字符去掉
labels = [label[1:] for label in labels]
#将q a重新拼起来，作为input,形如 question:…… answer:……
input_ids = [torch.cat([token_question [i],answer_prefix,labels[i]],dim=-1) \
             for i in range(len(token_question ))]
#answer前要有和问题长度一样的ingnore_idx
labels = [torch.cat([torch.tensor([self.config.ignore_idx] * (token_question
[i].shape[0]\
                   +answer_pre_len)), label]) for i,label in enumerate(labels)]
#执行裁剪和填充
input_ids=self.truncate_and_pad(input_ids,"input_ids")
labels=self.truncate_and_pad(labels,"labels")
return input_ids, labels, None
```

至于truncate_and_pad是和t2要求一样的裁剪、填充函数，不再赘述。

2.chat.py

基本和qa.py实现一致，但因为一个是question answer的构成，一个是user-bot对话，需要在组织数据时，稍作改变，不在此赘述。

3.result

未全部完成t3，但就test_toy_task3.py里面打印的内容，我对input 和label张量加了assert_close和ref实现进行判断，均正确。第一部分的实现应该没有问题，第二部分寒假有空会琢磨一下的。