MiniCPM3Attention MLA原文复现

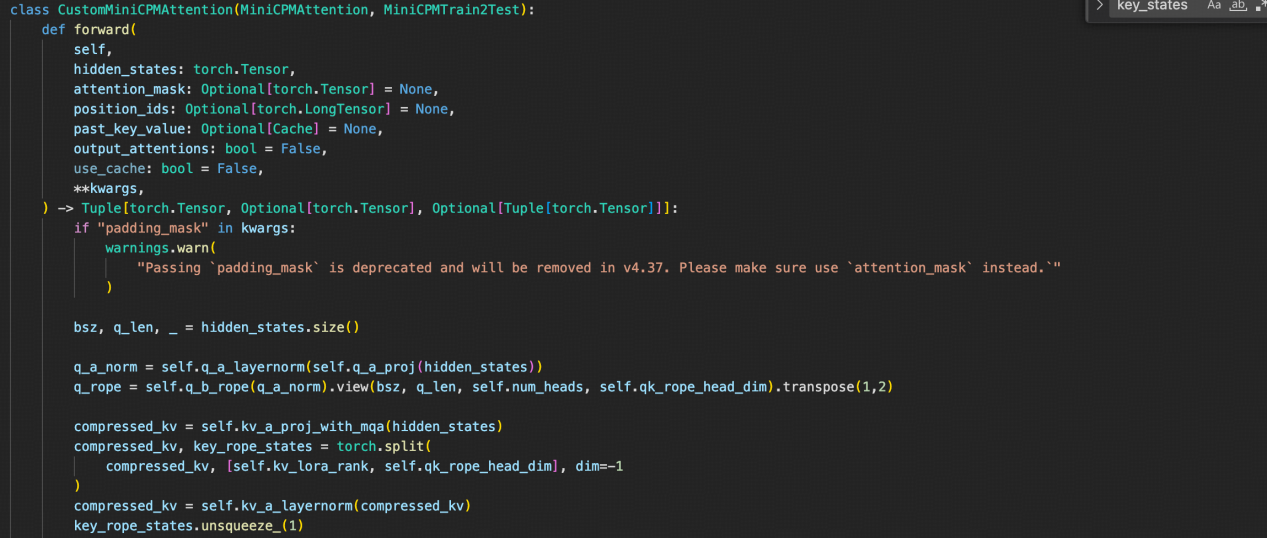
1. 实现kv-cache的真正压缩（与原始相比压缩了20倍）
2. 实现矩阵的融合(qk\_nope\_merge)

首先要实现权重的转换，将原始的矩阵进行拆分，构造qk\_merge\_nope, q\_b\_rope和v\_b

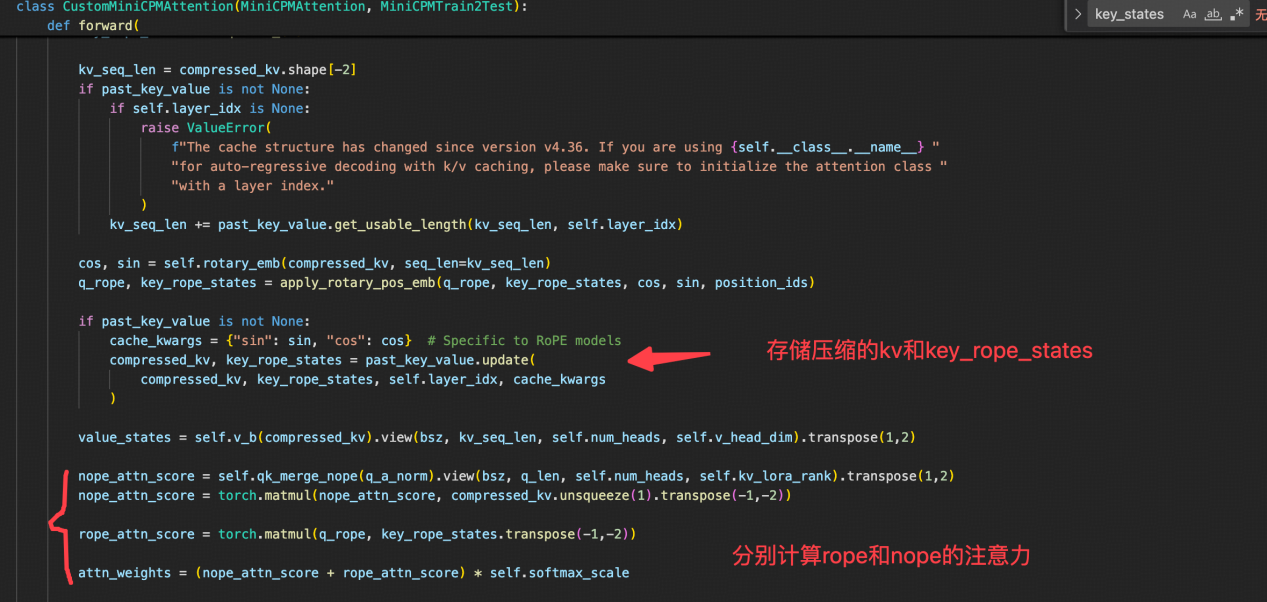


重写forward方法，主要实现

1. 存储压缩的kv以及key\_rope\_states
2. 两次attn\_score的计算，然后进行加法（类似mlp的tp）

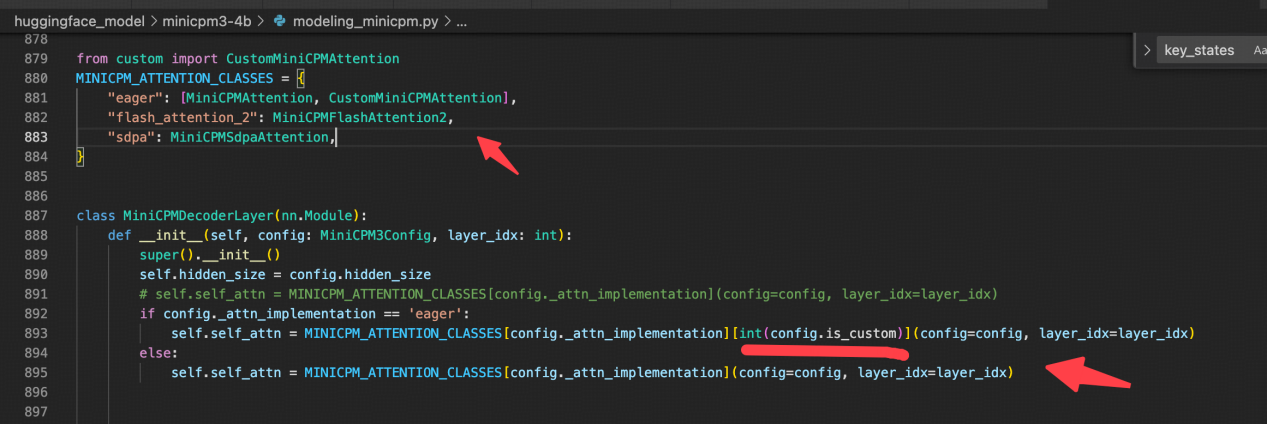


在此段之后的forward代码就一样了



原始代码更改

主要适配新增加的Attention模块



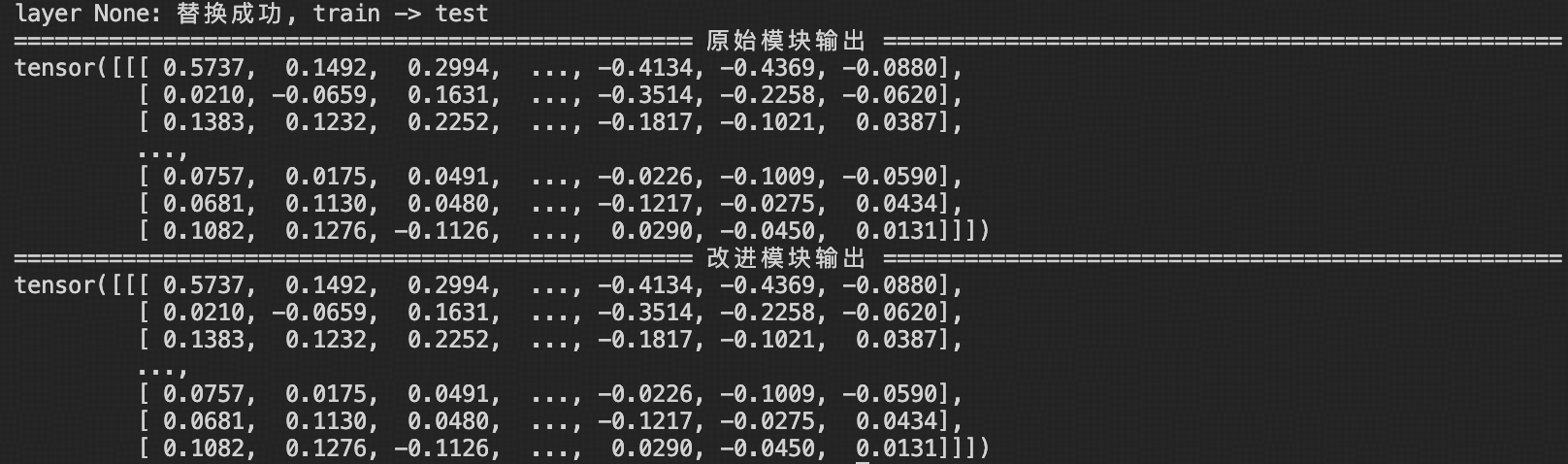
性能测试

1. 测试两种attention的输出，主要验证模型权重的转换是否正确

示例代码



结果正确，说明权重转换没问题



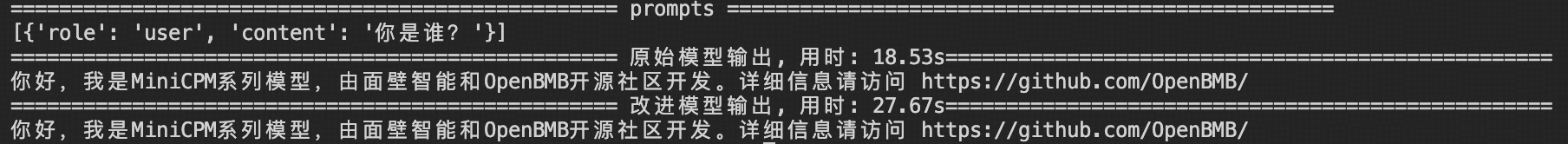
推理代码

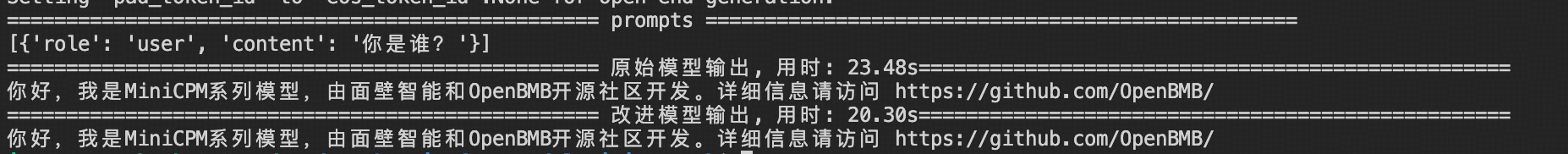


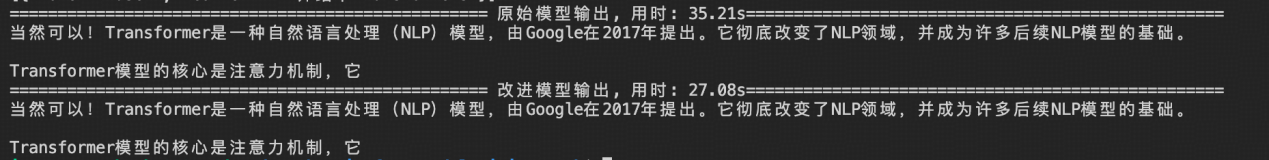


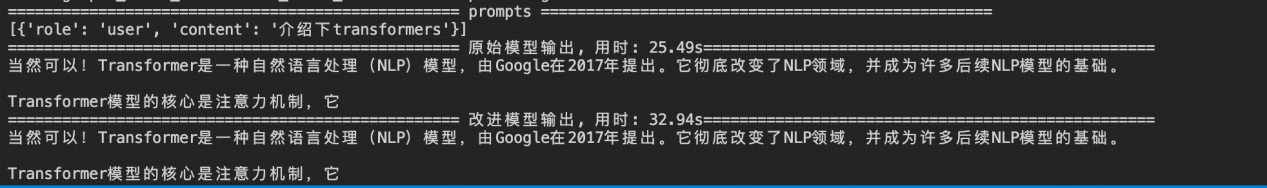
CPU&FP32实验结果

输出一致，速度有快有慢，QAQ。





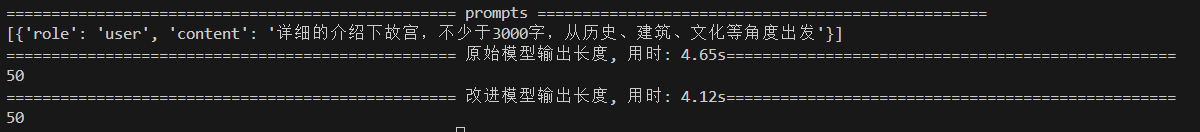


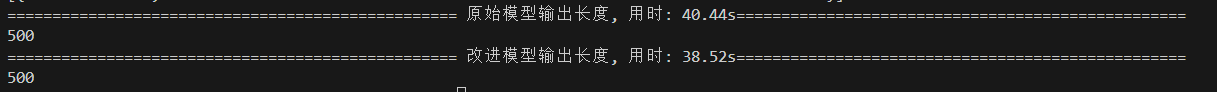
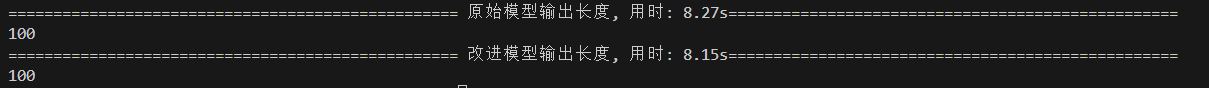


笔记本4080&BF16 实验结果

输出速度对比

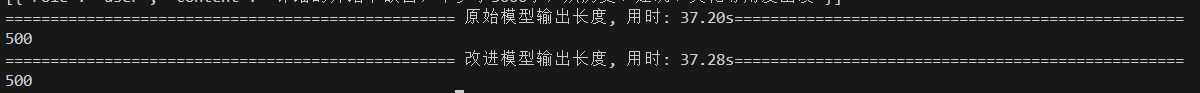
raw vs custom



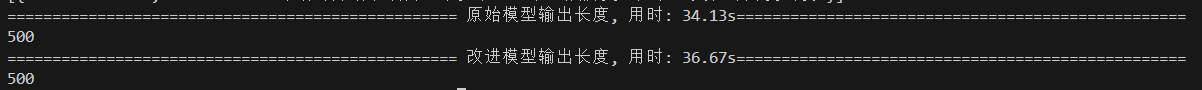


# 下面的例子可能不够充分，flash-attn2可能对更长的序列更占优势，由于4080显卡只有12G显存，无法充分实验。500token的情况下，速度基本一致

Flash-attn2 vs custom



Sdpa vs custom



细节分析

1. kv cache节省分析，minicpm3\_config如下

num\_head = 40

num\_kv\_head = 40

v\_head\_dim = 64

k\_head\_dim = qk\_nope\_dim + qk\_rope\_dim = 96

kv\_lora\_rank = 256

单个token单层的内存消耗量(bf16 or fp16)

raw\_mem = (40 \* 64 + 40 \* 96) \* 2

custom\_mem = (256 + 64) \* 2

save\_rate = raw\_mem / custom\_mem = 20

因此改进后，kv-cache的开销减少20倍

1. 计算换内存

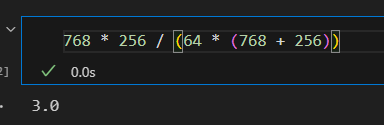
细心的coder们应该发现了，难道使用推理模式下的mla，可以大大节省显存，就没有缺点吗？其实解码时，每次前向都需要将历史压缩的kv取出来，然后将全部的kv全部进行一次升维的映射（是不是感觉这样等于又没有使用kv\_cache啊， 叫它“半”cache吧，毕竟只有attn的部分操作需要取出全部，embeding和mlp不需要）。

1. 实际上模型的参数增加了一些

这个merge操作增加了参数







总结

主要是在原始attention的基础上进行修改，复现了论文中MLA的操作，而flash-attn和sdpa，得到输出结果是经过softmax处理后，是非线性变化，无法进行reduce操作得到真实值，可能在推理速度上不如它们快，但是绝对是省显存的。由于本人能力有限，不会写C++和CUDA，事实上，nope和rope的操作是可以同时进行的，充分发挥GPU的并行，可以进一步提升效果。

附件

#custom.py  
from typing import Any, Dict, Tuple

from torch import Tensor

from modeling\_minicpm import \*

class MiniCPMTrain2Test(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def train2test(self):

# 不可逆的操作,因为qk\_merge是两个矩阵的乘法的结果，无法从结果倒推回两个矩阵。

# 或者不删除原始的 self.q\_b\_proj， self.kv\_b\_proj，但是会增大显存开销

device = next(self.parameters()).device

dtype = next(self.parameters()).dtype

q\_b\_weight = self.q\_b\_proj.weight.data.reshape(self.num\_heads, self.qk\_nope\_head\_dim + self.qk\_rope\_head\_dim, self.q\_lora\_rank)

q\_b\_nope\_weight, q\_b\_rope\_weight = q\_b\_weight.split([self.qk\_nope\_head\_dim, self.qk\_rope\_head\_dim], dim=1)

q\_b\_rope\_weight = q\_b\_rope\_weight.reshape(self.num\_heads \* self.qk\_rope\_head\_dim, self.q\_lora\_rank)

kv\_b\_weight = self.kv\_b\_proj.weight.data.reshape(self.num\_heads, self.qk\_nope\_head\_dim + self.v\_head\_dim, self.kv\_lora\_rank)

k\_b\_nope\_weight, v\_b\_weight = kv\_b\_weight.split([self.qk\_nope\_head\_dim, self.v\_head\_dim], dim=1)

v\_b\_weight = v\_b\_weight.reshape(self.num\_heads \* self.v\_head\_dim, self.kv\_lora\_rank)

qk\_megre\_nope\_weight = torch.einsum('hdq,hdk->hkq', q\_b\_nope\_weight, k\_b\_nope\_weight).reshape(self.num\_heads \* self.kv\_lora\_rank, self.q\_lora\_rank)

self.qk\_merge\_nope = nn.Linear(self.q\_lora\_rank, self.num\_heads \* self.kv\_lora\_rank, bias=False).to(device).to(dtype)

self.qk\_merge\_nope.weight.data.copy\_(qk\_megre\_nope\_weight)

self.q\_b\_rope = nn.Linear(self.q\_lora\_rank, self.num\_heads \* self.qk\_rope\_head\_dim, bias=False).to(device).to(dtype)

self.q\_b\_rope.weight.data.copy\_(q\_b\_rope\_weight)

self.v\_b = nn.Linear(self.kv\_lora\_rank, self.num\_heads \* self.v\_head\_dim, bias=False).to(device).to(dtype)

self.v\_b.weight.data.copy\_(v\_b\_weight)

del self.q\_b\_proj

del self.kv\_b\_proj

print(f'layer {self.layer\_idx}: 替换成功, train -> test')

if torch.cuda.is\_available():

torch.cuda.empty\_cache()

class CustomMiniCPMAttention(MiniCPMAttention, MiniCPMTrain2Test):

def forward(

self,

hidden\_states: torch.Tensor,

attention\_mask: Optional[torch.Tensor] = None,

position\_ids: Optional[torch.LongTensor] = None,

past\_key\_value: Optional[Cache] = None,

output\_attentions: bool = False,

use\_cache: bool = False,

\*\*kwargs,

) -> Tuple[torch.Tensor, Optional[torch.Tensor], Optional[Tuple[torch.Tensor]]]:

if "padding\_mask" in kwargs:

warnings.warn(

"Passing `padding\_mask` is deprecated and will be removed in v4.37. Please make sure use `attention\_mask` instead.`"

)

bsz, q\_len, \_ = hidden\_states.size()

q\_a\_norm = self.q\_a\_layernorm(self.q\_a\_proj(hidden\_states))

q\_rope = self.q\_b\_rope(q\_a\_norm).view(bsz, q\_len, self.num\_heads, self.qk\_rope\_head\_dim).transpose(1,2)

compressed\_kv = self.kv\_a\_proj\_with\_mqa(hidden\_states)

compressed\_kv, key\_rope\_states = torch.split(

compressed\_kv, [self.kv\_lora\_rank, self.qk\_rope\_head\_dim], dim=-1

)

compressed\_kv = self.kv\_a\_layernorm(compressed\_kv)

key\_rope\_states.unsqueeze\_(1)

kv\_seq\_len = compressed\_kv.shape[-2]

if past\_key\_value is not None:

if self.layer\_idx is None:

raise ValueError(

f"The cache structure has changed since version v4.36. If you are using {self.\_\_class\_\_.\_\_name\_\_} "

"for auto-regressive decoding with k/v caching, please make sure to initialize the attention class "

"with a layer index."

)

kv\_seq\_len += past\_key\_value.get\_usable\_length(kv\_seq\_len, self.layer\_idx)

cos, sin = self.rotary\_emb(compressed\_kv, seq\_len=kv\_seq\_len)

q\_rope, key\_rope\_states = apply\_rotary\_pos\_emb(q\_rope, key\_rope\_states, cos, sin, position\_ids)

if past\_key\_value is not None:

cache\_kwargs = {"sin": sin, "cos": cos} # Specific to RoPE models

compressed\_kv, key\_rope\_states = past\_key\_value.update(

compressed\_kv, key\_rope\_states, self.layer\_idx, cache\_kwargs

)

value\_states = self.v\_b(compressed\_kv).view(bsz, kv\_seq\_len, self.num\_heads, self.v\_head\_dim).transpose(1,2)

nope\_attn\_score = self.qk\_merge\_nope(q\_a\_norm).view(bsz, q\_len, self.num\_heads, self.kv\_lora\_rank).transpose(1,2)

nope\_attn\_score = torch.matmul(nope\_attn\_score, compressed\_kv.unsqueeze(1).transpose(-1,-2))

rope\_attn\_score = torch.matmul(q\_rope, key\_rope\_states.transpose(-1,-2))

attn\_weights = (nope\_attn\_score + rope\_attn\_score) \* self.softmax\_scale

if attn\_weights.size() != (bsz, self.num\_heads, q\_len, kv\_seq\_len):

raise ValueError(

f"Attention weights should be of size {(bsz, self.num\_heads, q\_len, kv\_seq\_len)}, but is"

f" {attn\_weights.size()}"

)

assert attention\_mask is not None

if attention\_mask is not None:

if attention\_mask.size() != (bsz, 1, q\_len, kv\_seq\_len):

raise ValueError(

f"Attention mask should be of size {(bsz, 1, q\_len, kv\_seq\_len)}, but is {attention\_mask.size()}"

)

attn\_weights = attn\_weights + attention\_mask

# upcast attention to fp32

attn\_weights = nn.functional.softmax(

attn\_weights, dim=-1, dtype=torch.float32

).to(value\_states.dtype)

attn\_weights = nn.functional.dropout(

attn\_weights, p=self.attention\_dropout, training=self.training

)

attn\_output = torch.matmul(attn\_weights, value\_states)

if attn\_output.size() != (bsz, self.num\_heads, q\_len, self.v\_head\_dim):

raise ValueError(

f"`attn\_output` should be of size {(bsz, self.num\_heads, q\_len, self.v\_head\_dim)}, but is"

f" {attn\_output.size()}"

)

attn\_output = attn\_output.transpose(1, 2).contiguous()

attn\_output = attn\_output.reshape(bsz, q\_len, self.num\_heads \* self.v\_head\_dim)

attn\_output = self.o\_proj(attn\_output)

if not output\_attentions:

attn\_weights = None

return attn\_output, attn\_weights, past\_key\_value

#test.py

import modeling\_minicpm

from transformers import AutoConfig, AutoTokenizer

from custom import \*

import torch

import time

path = '/data/models/MiniCPM3-4B'

def attn():

config = AutoConfig.from\_pretrained(path, trust\_remote\_code=True)

config.hidden\_size = 256

config.max\_position\_embeddings = 64

config.num\_attention\_heads = 4

config.num\_key\_value\_heads = 4

config.q\_lora\_rank = 32

config.kv\_lora\_rank = 32

inputs = torch.randn(1,8, 256)

attn\_mask = torch.ones(8,8).tril()

attn\_mask = torch.where(~attn\_mask.bool(), torch.finfo(torch.float32).min, 0)

attn\_mask = attn\_mask[None, None, :, :]

attn = modeling\_minicpm.MiniCPMAttention(config)

custom\_attn = CustomMiniCPMAttention(config)

custom\_attn.load\_state\_dict(attn.state\_dict())

custom\_attn.train2test()

with torch.inference\_mode():

out1 = attn(inputs, attn\_mask)[0]

out2 = custom\_attn(inputs, attn\_mask)[0]

print('='\*50 + ' 原始模块输出 ' +'='\*50)

print(out1)

print('='\*50 + ' 改进模块输出 ' +'='\*50)

print(out2)

def generate():

dtype = torch.bfloat16

device = 'cuda'

max\_new\_tokens = 500

config = AutoConfig.from\_pretrained(path, trust\_remote\_code=True)

tokenizer = AutoTokenizer.from\_pretrained(path, trust\_remote\_code=True)

prompts = [{'role': 'user', 'content': '详细的介绍下故宫，不少于3000字，从历史、建筑、文化等角度出发'}]

inputs = tokenizer.apply\_chat\_template(prompts, add\_generation\_prompt=True, tokenize=True, return\_tensors='pt')

inputs = inputs.to(device)

#flash\_attention\_2

config.\_attn\_implementation = 'flash\_attention\_2'

config.is\_custom = False

raw\_model = modeling\_minicpm.MiniCPM3ForCausalLM.from\_pretrained(path, device\_map=device, torch\_dtype=dtype, config=config)

raw\_model.eval()

p1 = raw\_model.num\_parameters()

with torch.inference\_mode():

torch.manual\_seed(42)

start = time.time()

model\_outputs = raw\_model.generate(

inputs, do\_sample=False, max\_new\_tokens=max\_new\_tokens

)

time1 = time.time() - start

output\_token\_ids1 = [

model\_outputs[i][len(inputs[i]):] for i in range(len(inputs))

]

responses1 = tokenizer.batch\_decode(output\_token\_ids1, skip\_special\_tokens=True)[0]

del raw\_model

torch.cuda.empty\_cache()

time.sleep(3)

config.\_attn\_implementation = 'eager'

config.is\_custom = True

custom\_model = modeling\_minicpm.MiniCPM3ForCausalLM.from\_pretrained(path, device\_map=device, torch\_dtype=dtype, config=config)

for m in custom\_model.modules():

if isinstance(m, MiniCPMTrain2Test):

m.train2test()

p2 = custom\_model.num\_parameters()

custom\_model.eval()

with torch.inference\_mode():

torch.manual\_seed(42)

start = time.time()

model\_outputs = custom\_model.generate(

inputs, do\_sample=False, max\_new\_tokens=max\_new\_tokens

)

time2 = time.time() - start

output\_token\_ids2 = [

model\_outputs[i][len(inputs[i]):] for i in range(len(inputs))

]

response2 = tokenizer.batch\_decode(output\_token\_ids2, skip\_special\_tokens=True)[0]

print(f'原始模型参数: {p1/1e9}B, 改进模型参数: {p2/1e9}B')

print('='\*50 + ' prompts ' +'='\*50)

print(prompts)

print('='\*50 + ' 原始模型输出长度, 用时: %.2fs'%time1 +'='\*50)

print(len(output\_token\_ids1[0]))

# print(responses1)

print('='\*50 + ' 改进模型输出长度, 用时: %.2fs'%time2 +'='\*50)

print(len(output\_token\_ids2[0]))

# print(response2)

if \_\_name\_\_ == '\_\_main\_\_':

generate()