DALL-E

https://github.com/kuprel/min-dalle

https://blog.csdn.net/qq_36332660/article/details/134273737

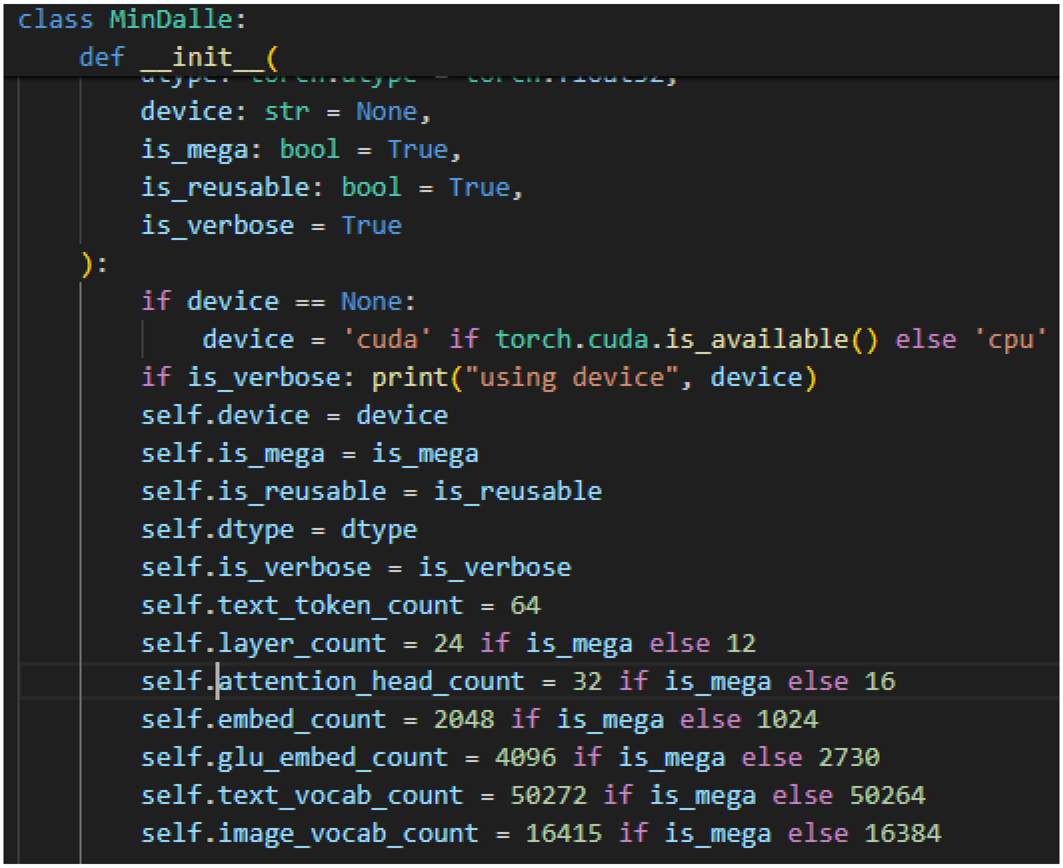
https://blog.csdn.net/Friedrichor/article/details/128086733

使用AutoDL时下载模型时的问题

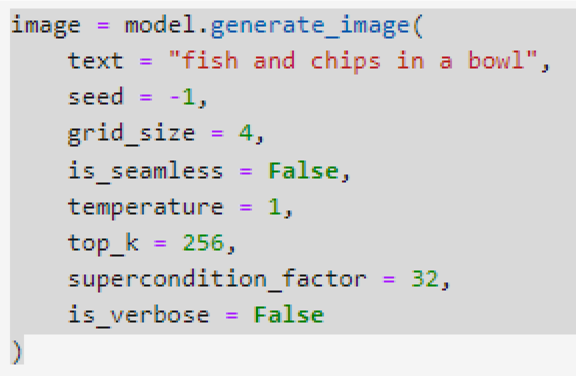
https://blog.csdn.net/weixin_46141492/article/details/135206086?spm=1001.2101.3001.6650.3&utm_medium=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-3-135206086-blog-131850846.235^v43^pc_blog_bottom_relevance_base2&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~CTRLIST~Rate-3-135206086-blog-131850846.235^v43^pc_blog_bottom_relevance_base2

min-dalle代码解读

最外层参数



```python
class MinDalle:
    def __init__(
        dtype: torch.dtype = torch.float32,
        device: str = None,
        is_mega: bool = True,
        is_reusable: bool = True,
        is_verbose = True
    ):
        if device == None:
            device = 'cuda' if torch.cuda.is_available() else 'cpu'
        if is_verbose: print("using device", device)
        self.device = device
        self.is_mega = is_mega
        self.is_reusable = is_reusable
        self.dtype = dtype
        self.is_verbose = is_verbose
        self.text_token_count = 64
        self.layer_count = 24 if is_mega else 12
        self.attention_head_count = 32 if is_mega else 16
        self.embed_count = 2048 if is_mega else 1024
        self.glu_embed_count = 4096 if is_mega else 2730
        self.text_vocab_count = 50272 if is_mega else 50264
        self.image_vocab_count = 16415 if is_mega else 16384
```

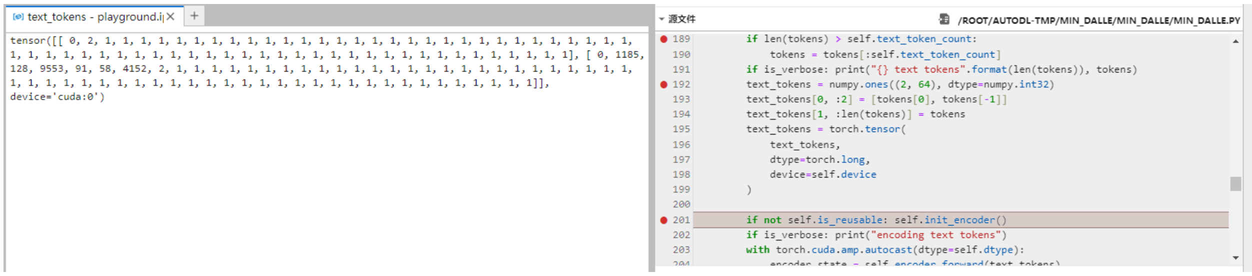


```python
image = model.generate_image(
    text = "fish and chips in a bowl",
    seed = -1,
    grid_size = 4,
    is_seamless = False,
    temperature = 1,
    top_k = 256,
    supercondition_factor = 32,
    is_verbose = False
)
```

generate_raw_image_stream():

image_count = grid_size ** 2

text_tokens 形(2，64)



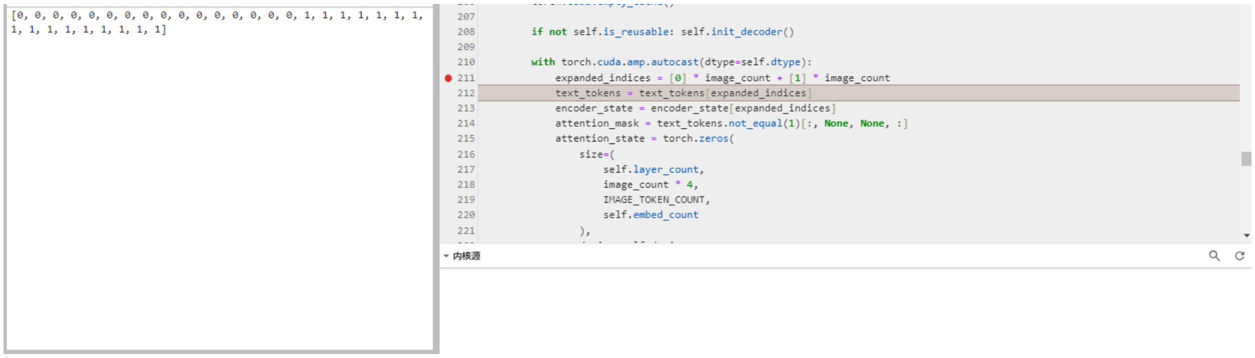
text_tokens - playground.i|
+
源文件
/ROOT/AUTODL-TMP/MIN_DALLE/MIN_DALLE/MIN_DALLE.PY

```
tensor([[ 0, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [ 0, 1185,
128, 9553, 91, 58, 4152, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]],
device='cuda:0')
```

```python
        if len(tokens) > self.text_token_count:
            tokens = tokens[:self.text_token_count]
        if is_verbose: print("{} text tokens".format(len(tokens)), tokens)
        text_tokens = numpy.ones((2, 64), dtype=numpy.int32)
        text_tokens[0, :2] = [tokens[0], tokens[-1]]
        text_tokens[1, :len(tokens)] = tokens
        text_tokens = torch.tensor(
            text_tokens,
            dtype=torch.long,
            device=self.device
        )

        if not self.is_reusable: self.init_encoder()
        if is_verbose: print("encoding text tokens")
        with torch.cuda.amp.autocast(dtype=self.dtype):
            encoder_state = self.encoder.forward(text_tokens)
```

expanded_indices创建时



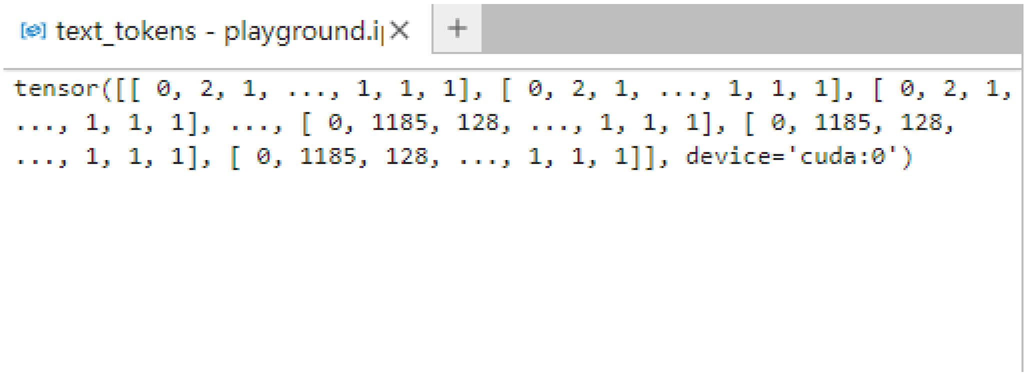
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```python

        if not self.is_reusable: self.init_decoder()

        with torch.cuda.amp.autocast(dtype=self.dtype):
            expanded_indices = [0] * image_count + [1] * image_count
            text_tokens = text_tokens[expanded_indices]
            encoder_state = encoder_state[expanded_indices]
            attention_mask = text_tokens.not_equal(1)[:, None, None, :]
            attention_state = torch.zeros(
                size=(
                    self.layer_count,
                    image_count * 4,
                    IMAGE_TOKEN_COUNT,
                    self.embed_count
                ),
```

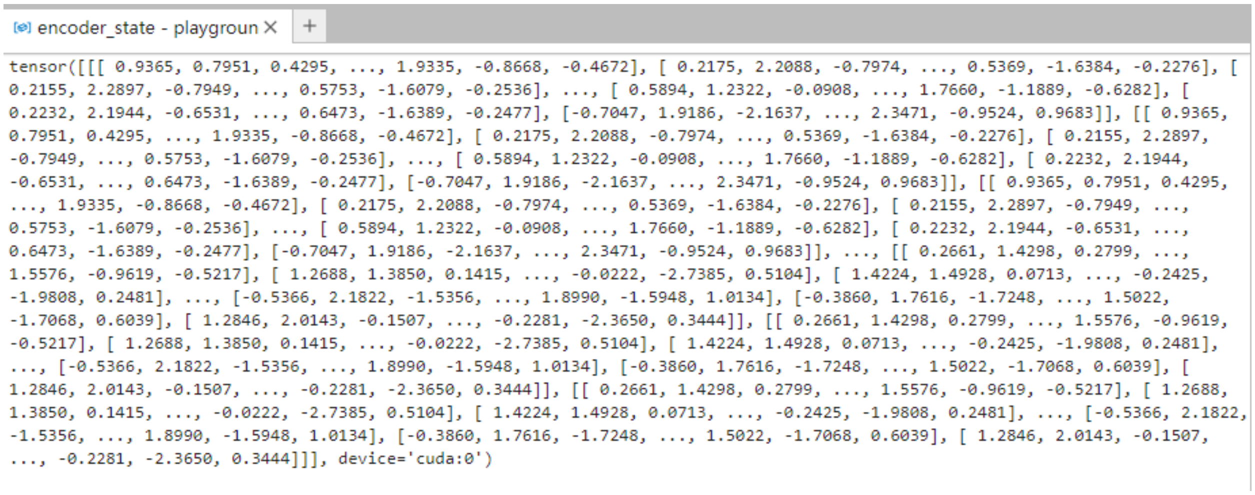
text_tokens经过expanded_indices索引后 形如(32, 64)



```
tensor([[ 0, 2, 1, ..., 1, 1, 1], [ 0, 2, 1, ..., 1, 1, 1], [ 0, 2, 1,
..., 1, 1, 1], ..., [ 0, 1185, 128, ..., 1, 1, 1], [ 0, 1185, 128,
..., 1, 1, 1], [ 0, 1185, 128, ..., 1, 1, 1]], device='cuda:0')
```

encoder_state由DalleBartEncoder的forward()创建时形如(2, 64, 2048)

经expanded_indices索引后 形如(32, 64, 2048)



```
tensor([[[ 0.9365, 0.7951, 0.4295, ..., 1.9335, -0.8668, -0.4672], [ 0.2175, 2.2088, -0.7974, ..., 0.5369, -1.6384, -0.2276], [
0.2155, 2.2897, -0.7949, ..., 0.5753, -1.6079, -0.2536], ..., [ 0.5894, 1.2322, -0.0908, ..., 1.7660, -1.1889, -0.6282], [
0.2232, 2.1944, -0.6531, ..., 0.6473, -1.6389, -0.2477], [-0.7047, 1.9186, -2.1637, ..., 2.3471, -0.9524, 0.9683]], [[ 0.9365,
0.7951, 0.4295, ..., 1.9335, -0.8668, -0.4672], [ 0.2175, 2.2088, -0.7974, ..., 0.5369, -1.6384, -0.2276], [ 0.2155, 2.2897,
-0.7949, ..., 0.5753, -1.6079, -0.2536], ..., [ 0.5894, 1.2322, -0.0908, ..., 1.7660, -1.1889, -0.6282], [ 0.2232, 2.1944,
-0.6531, ..., 0.6473, -1.6389, -0.2477], [-0.7047, 1.9186, -2.1637, ..., 2.3471, -0.9524, 0.9683]], [[ 0.9365, 0.7951, 0.4295,
..., 1.9335, -0.8668, -0.4672], [ 0.2175, 2.2088, -0.7974, ..., 0.5369, -1.6384, -0.2276], [ 0.2155, 2.2897, -0.7949, ...,
0.5753, -1.6079, -0.2536], ..., [ 0.5894, 1.2322, -0.0908, ..., 1.7660, -1.1889, -0.6282], [ 0.2232, 2.1944, -0.6531, ...,
0.6473, -1.6389, -0.2477], [-0.7047, 1.9186, -2.1637, ..., 2.3471, -0.9524, 0.9683]], ..., [[ 0.2661, 1.4298, 0.2799, ...,
1.5576, -0.9619, -0.5217], [ 1.2688, 1.3850, 0.1415, ..., -0.0222, -2.7385, 0.5104], [ 1.4224, 1.4928, 0.0713, ..., -0.2425,
-1.9808, 0.2481], ..., [-0.5366, 2.1822, -1.5356, ..., 1.8990, -1.5948, 1.0134], [-0.3860, 1.7616, -1.7248, ..., 1.5022,
-1.7068, 0.6039], [ 1.2846, 2.0143, -0.1507, ..., -0.2281, -2.3650, 0.3444]], [[ 0.2661, 1.4298, 0.2799, ..., 1.5576, -0.9619,
-0.5217], [ 1.2688, 1.3850, 0.1415, ..., -0.0222, -2.7385, 0.5104], [ 1.4224, 1.4928, 0.0713, ..., -0.2425, -1.9808, 0.2481],
..., [-0.5366, 2.1822, -1.5356, ..., 1.8990, -1.5948, 1.0134], [-0.3860, 1.7616, -1.7248, ..., 1.5022, -1.7068, 0.6039], [
1.2846, 2.0143, -0.1507, ..., -0.2281, -2.3650, 0.3444]], [[ 0.2661, 1.4298, 0.2799, ..., 1.5576, -0.9619, -0.5217], [ 1.2688,
1.3850, 0.1415, ..., -0.0222, -2.7385, 0.5104], [ 1.4224, 1.4928, 0.0713, ..., -0.2425, -1.9808, 0.2481], ..., [-0.5366, 2.1822,
-1.5356, ..., 1.8990, -1.5948, 1.0134], [-0.3860, 1.7616, -1.7248, ..., 1.5022, -1.7068, 0.6039], [ 1.2846, 2.0143, -0.1507,
..., -0.2281, -2.3650, 0.3444]]], device='cuda:0')
```

attention_mask由text_tokens创建时形如(32, 1, 1, 64)

attention_state由torch.zero创建时形如(24, 64, 256, 2048)（self.layer_count, image_count * 4, IMAGE_TOKEN_COUNT, self.embed_count)

image_tokens由torch.full创建时形如(16, 257) （image_count, IMAGE_TOKEN_COUNT + 1)

token_indices经由torch.arange创建时形如(256）IMAGE_TOKEN_COUNT

settings 创建时内涵三个元素，分别是 [temperature, top_k, supercondition_factor] 这三个参数都在调用generate_images的API接口时指定

最后是一个for i in range(IMAGE_TOKEN_COUNT)的循环：

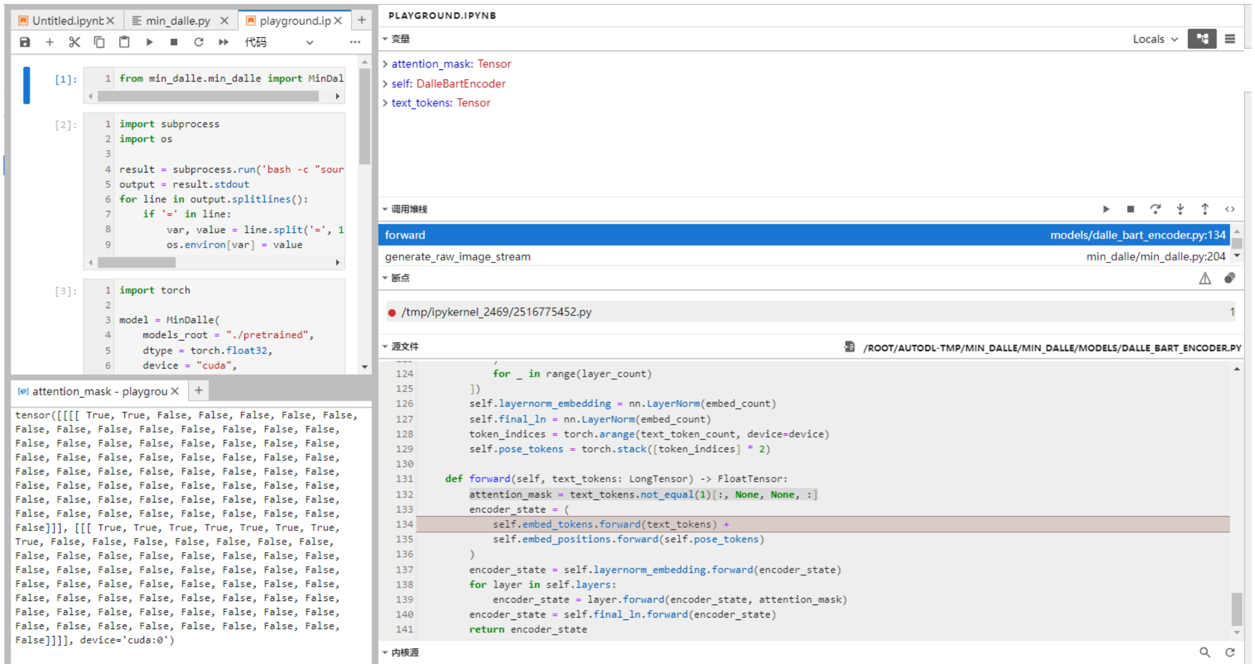
调用self.decoder.sample_tokens方法赋值给 image_tokens[:, i + 1], attention_state

再在一定条件下调用 self.image_grid_from_tokens 通过yield迭代输出

DalleBartEncoder类

forward()

attention_mask



```
[1]:
from min_dalle.min_dalle import MinDal
```

```
[2]:
import subprocess
import os

result = subprocess.run('bash -c "sour
output = result.stdout
for line in output.splitlines():
    if '=' in line:
        var, value = line.split('=', 1
        os.environ[var] = value
```

```
[3]:
import torch

model = MinDalle(
    models_root = "./pretrained",
    dtype = torch.float32,
    device = "cuda",
```

```
tensor([[[[ True, True, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False]]], [[[ True, True, True, True, True, True, True,
True, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False,
False]]]], device='cuda:0')
```

> attention_mask: Tensor
> self: DalleBartEncoder
> text_tokens: Tensor

forward models/dalle_bart_encoder.py:134
generate_raw_image_stream min_dalle/min_dalle.py:204

● /tmp/ipykernel_2469/2516775452.py 1

/ROOT/AUTODL-TMP/MIN_DALLE/MIN_DALLE/MODELS/DALLE_BART_ENCODER.PY

```
            for _ in range(layer_count)
        ])
        self.layernorm_embedding = nn.LayerNorm(embed_count)
        self.final_ln = nn.LayerNorm(embed_count)
        token_indices = torch.arange(text_token_count, device=device)
        self.pose_tokens = torch.stack([token_indices] * 2)

    def forward(self, text_tokens: LongTensor) -> FloatTensor:
        attention_mask = text_tokens.not_equal(1)[:, None, None, :]
        encoder_state = (
            self.embed_tokens.forward(text_tokens) +
            self.embed_positions.forward(self.pose_tokens)
        )
        encoder_state = self.layernorm_embedding.forward(encoder_state)
        for layer in self.layers:
            encoder_state = layer.forward(encoder_state, attention_mask)
        encoder_state = self.final_ln.forward(encoder_state)
        return encoder_state
```

encoder_state 创建后形如(2, 64, 2048)

经self.layernorm_embedding后shape不变

经self.layers后shape不变

经self.final_ln.forward后shape不变

返回encoder_state

DalleBartDecoder类

sample_tokens()

传入的参数中, prev_tokens形如（16, 1), token_index形如 (1,)

经由forward()方法返回 logits, attention_state

logits经过 logits[:, -1, : 2 ** 14] 索引后形如（32，16384）

在经过下一行形如（16， 16384）

利用supercondition_factor对logits的值进行了调整，这里追根溯源维度数32出现的原因是一开始的expanded_indices的复制扩张，expanded_indices = [0] * image_count + [1] * image_count 但具体为什么有待探究

supercondition_factor的作用，注意到text_token的维度1上的数是2，而且根据上面text_token的会发现，只有后半截（也就是text_token[1]）中包含真正的描述文本的字典编码，text_token前半截(text_token[0])中开头token索引0和结尾token索引2是相连的，其他都是1，1这里很有可能就是填充token的索引。而最后对logits的处理，如果supercondition_factor越大，text_token中后半截所对应的这部分logits的比重越大，生成的图像越遵循文本。

logits_sorted形如（16，16384）

logits在接下来的操作中shape不变

通过torch.multinomial(logits, 1)[:, 0] 获得image_tokens形如（16，）

最后抽样出的image_tokens 形如（16，1）

几个疑问：

- 为什么统一减去最大值？GPT回答：为了数值稳定性，防止溢出。
- temperature的作用？GPT回答：调整 logits 的尖锐程度，使得概率分布更加平滑。

forward()

传入的参数：attention_mask　attention_state　encoder_state　prev_tokens　token_index

token_index经 .unsqueeze(0).repeat(image_count * 2, 1) 后形如（32，1）

prev_tokens经 .repeat(2, 1)后形如（32，1）

decoder_state 由self.embed_tokens的forward创建时形如（32，1，2048）

经过self.embed_position, self.layernorm_embedding后形如（32，1，2048）

最后是通过多个相同的DecoderLayer类模块的forward

传入的参数有decoder_state　encoder_state　attention_state[i]　attention_mask token_index

最后decoder_state经过self.final_ln后形状依然是（32，1，2048）

decoder_state经过self.lm_head后得到logits

self.lm_head的定义是

nn.Linear(embed_count, image_vocab_count + 1, bias=False)

形如

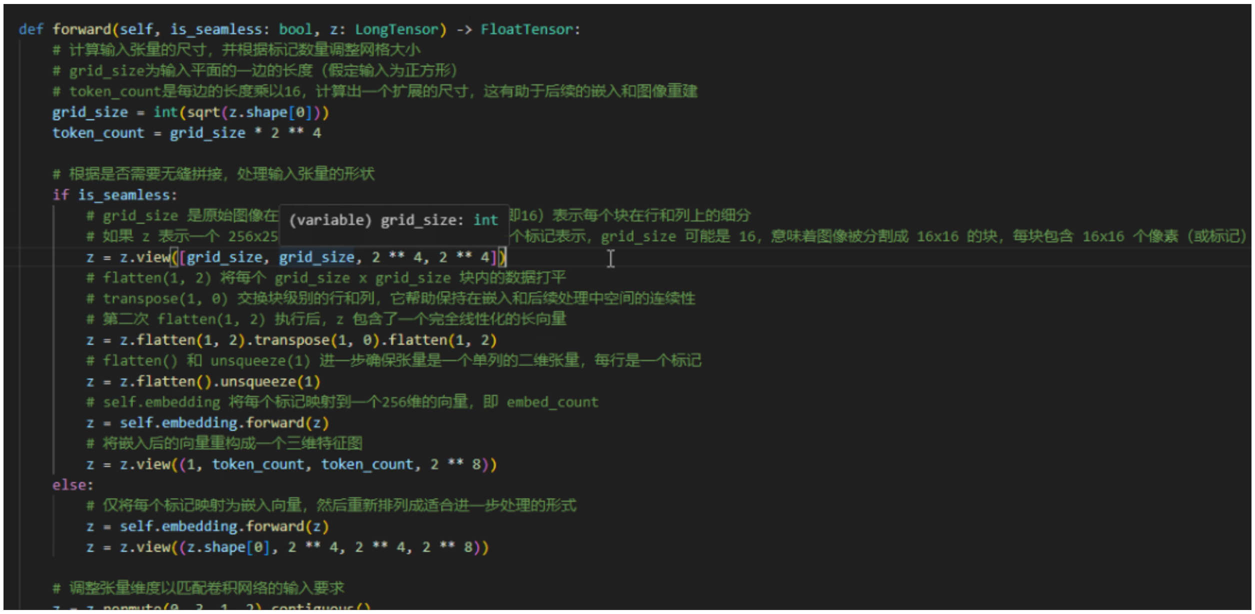
nn.Linear(2048, 16415 + 1, bias=False)

logits形如（32，1，16416）

这个方法返回logits　attention_state

其中attention_state形如（24，64，256，2048）跟定义之初shape相同

参考



```
def forward(self, is_seamless: bool, z: LongTensor) -> FloatTensor:
    # 计算输入张量的尺寸，并根据标记数量调整网格大小
    # grid_size为输入平面的一边的长度（假定输入为正方形）
    # token_count是每边的长度乘以16，计算出一个扩展的尺寸，这有助于后续的嵌入和图像重建
    grid_size = int(sqrt(z.shape[0]))
    token_count = grid_size * 2 ** 4

    # 根据是否需要无缝拼接，处理输入张量的形状
    if is_seamless:
        # grid_size 是原始图像在                          即16）表示每个块在行和列上的细分
        # 如果 z 表示一个 256x25                          个标记表示，grid_size 可能是 16，意味着图像被分割成 16x16 的块，每块包含 16x16 个像素（或标记）
        z = z.view([grid_size, grid_size, 2 ** 4, 2 ** 4])
        # flatten(1, 2) 将每个 grid_size x grid_size 块内的数据打平
        # transpose(1, 0) 交换块级别的行和列，它帮助保持在嵌入和后续处理中空间的连续性
        # 第二次 flatten(1, 2) 执行后，z 包含了一个完全线性化的长向量
        z = z.flatten(1, 2).transpose(1, 0).flatten(1, 2)
        # flatten() 和 unsqueeze(1) 进一步确保张量是一个单列的二维张量，每行是一个标记
        z = z.flatten().unsqueeze(1)
        # self.embedding 将每个标记映射到一个256维的向量，即 embed_count
        z = self.embedding.forward(z)
        # 将嵌入后的向量重构成一个三维特征图
        z = z.view((1, token_count, token_count, 2 ** 8))
    else:
        # 仅将每个标记映射为嵌入向量，然后重新排列成适合进一步处理的形式
        z = self.embedding.forward(z)
        z = z.view((z.shape[0], 2 ** 4, 2 ** 4, 2 ** 8))

    # 调整张量维度以匹配卷积网络的输入要求
    z = z.permute(0, 3, 1, 2).contiguous()
```