已完成：

配置思考题环境

基本函数撰写

Latex代码学习

Kv cache基本概念

即将完成

附加题Kv cache

# 附件:

## 一、函数整理

### 1.gelu 函数

#### （1）定义：

gelu（Gaussian Error Linear Unit），激活函数，其数学公式为：

其中：

x是PyTorch 张量，一般表示神经网络层的输入。

torch.tanh() 计算双曲正切，得到的值在 -1 到 1 之间。

#### （2）作用

GELU函数在深度学习模型中被广泛使用，尤其是在 Transformer 结构（如 GPT-2 和 BERT）中。作用包括：

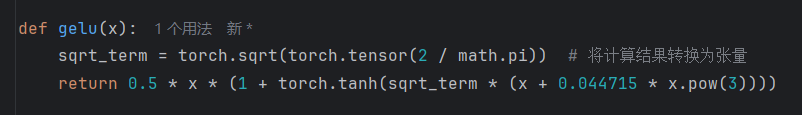
1. 非线性变换

激活函数的主要作用是引入非线性，使得神经网络能够学习复杂的模式。GELU通过保留负值的某些信息，而不是简单地将它们归零，从而在某些情况下比 ReLU 等激活函数表现更好。

2. 平滑性:

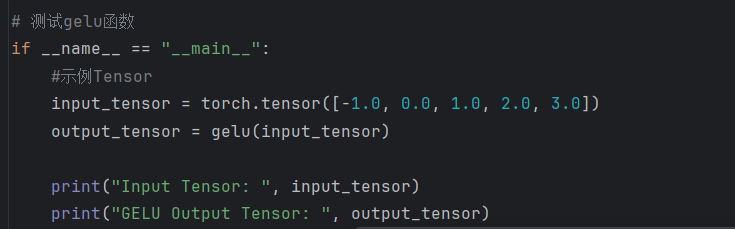
GELU 函数是一个平滑的激活函数，避免了梯度消失的问题，尤其是在较深的网络中。

#### （3）代码：

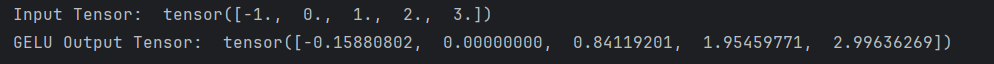


2 / math.pi 的计算结果是浮点数，通过torch.tensor()将结果转换为张量，保证torch.sqrt() 的输入是PyTorch 张量。

#### （4）测试示例及结果



gelu函数的输入：一个包含五个浮点数的 PyTorch 张量 input\_tensor，-1.0, 0.0, 1.0, 2.0和3.0。



### 2.softmax函数

#### （1）定义：

是输入向量中的第𝑖个元素。

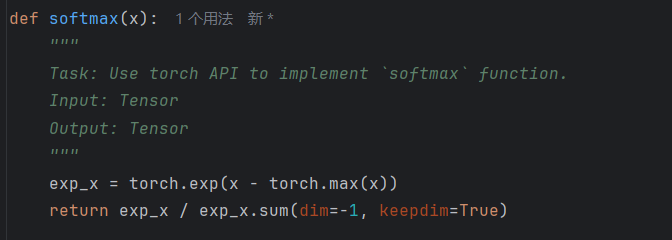
是 的指数值。

是输入向量中所有元素的指数值之和。

#### （2）作用

将输入张量 x 转换为概率分布，通常用于多分类模型的输出层，确保输出值在 [0, 1] 范围内，并且所有输出值的和为 1。

#### （3）代码：



torch.max(x)找到 x 中的最大值，避免指数运算导致的溢出；x中每个元素减去最大值，确保exp\_x的最大值为1，防止数值不稳定。exp\_x是x中每个元素的指数值。

exp\_x.sum(dim=-1, keepdim=True) 计算 exp\_x 在最后一个维度上的总和。

sum用于计算指定维度上的元素总和。dim=-1 表示在最后一个维度上求和，对于二维张量（例如形状为 [batch\_size, n]），dim=-1计算每一行的和，对于三维张量，计算每一层的和，依此类推。

Keepdim，布尔参数，指定是否保持原有维度。当keepdim=True时，输出张量的形状会保持与输入相同的维度数量，只是在求和的维度上将大小设置为1，避免后期在广播（broadcasting）过程中出现维度不匹配的问题。

最终返回的是每个元素的指数值除以总和，从而得到概率分布。

假设 exp\_x 是一个形状为 [2, 3] 的张量：

exp\_x = torch.tensor([[0.1, 0.2, 0.3],

[0.4, 0.5, 0.6]])

调用 exp\_x.sum(dim=-1, keepdim=True) 会计算每一行的总和，输出结果为：

tensor([[0.6],

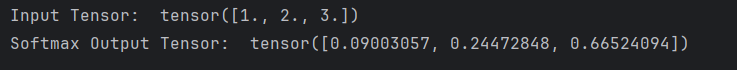
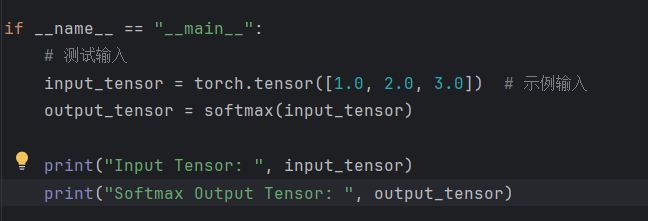
[1.5]])

第一行和0.1 + 0.2 + 0.3 = 0.6

第二行和 0.4 + 0.5 + 0.6 = 1.5

输出的形状为 [2, 1]，维持原来的二维结构，第二维的大小变为 1。

#### （4）测试示例及结果



### 3．layer\_norm函数

#### （1）定义

其中：

是输入张量。

是输入张量的均值。

是输入张量的标准差。

是缩放参数（gamma）。

是偏置参数（bias）。

是一个小的常数，避免除以零。

均值：

标准差：

 N 是数据集中的总数。

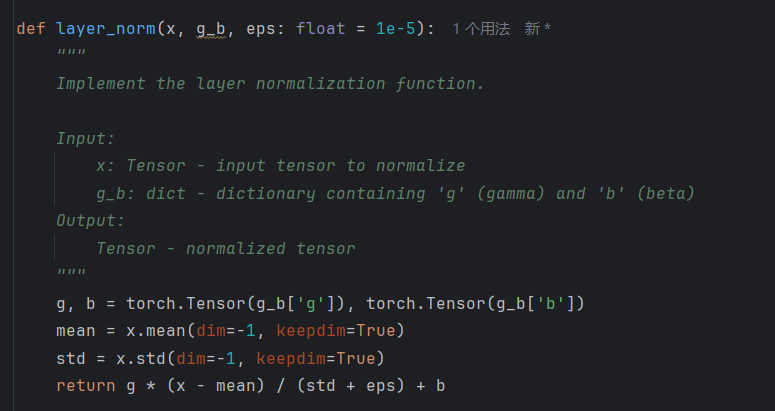
 是数据集中每个数值。

#### （2）作用

Layer Normalization（层归一化），常用于深度学习模型的技术，特别是在Transformer架构中。通过对每一层的输入进行标准化，使得神经网络的训练更稳定，加快收敛速度。目标是确保输入数据的均值为零，标准差为一，从而减少内部协变量偏移（internal covariate shift）。

与Batch Normalization不同，Layer Normalization在每个样本的所有特征上进行归一化。

#### （3）代码



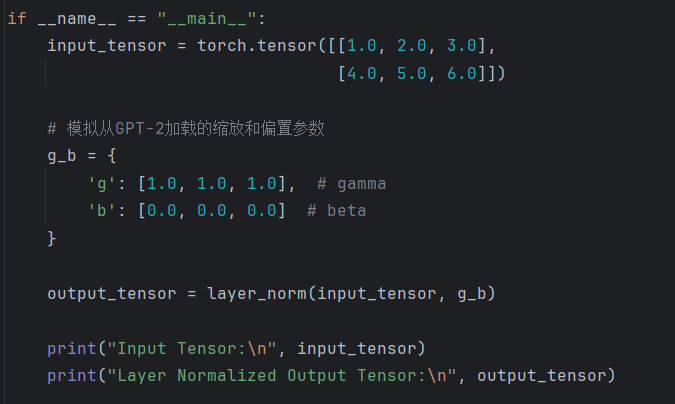
g, b = torch.Tensor(g\_b['g']), torch.Tensor(g\_b['b'])从字典g\_b中提取缩放和偏置参数，并将它们转换为PyTorch张量。

mean = x.mean(dim=-1, keepdim=True)计算输入张量x在最后一个维度上的均值。dim=-1表示在最后一个维度上计算均值；keepdim=True确保返回的均值张量保留原有的维度。

std = x.std(dim=-1, keepdim=True)计算输入张量x在最后一个维度上的标准差。keepdim=True同样是保留维度，使得标准差的形状与均值相同。

return g \* (x - mean) / (std + eps) + b进行层归一化，应用缩放和偏置参数。首先，x – mean，输入张量x减去其均值，将数据的均值归零。std + eps，在标准差std上加上一个小常数eps（如1e-5），防止除以零。最后乘缩放参数g，加偏置参数b。

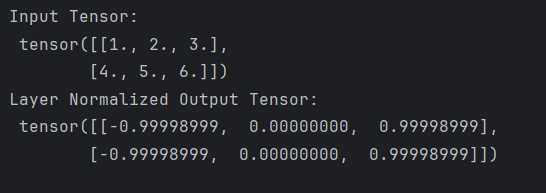
#### （4）测试示例及结果



x，张量，表示需要归一化的输入。

g\_b，字典，包含了g（缩放参数，gamma）和b（偏置参数，beta），对归一化后的输出进行变换。缩放参数，线性变换（放大或缩小）；偏置参数，进行平移（加上某个常数）。

eps: 一个小的常数，避免除以零的情况，默认为1e-5。



### 4.linear函数

#### （1）定义

其中：

y 是输出张量。

x 是输入张量，形状为[m,in]，其中m是样本数量，in是输入特征的数量。

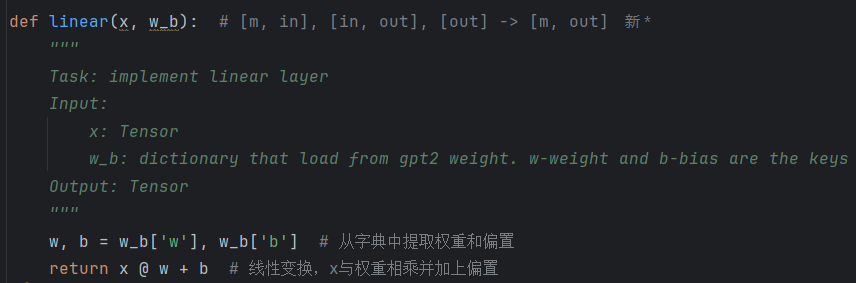
W 是权重矩阵，形状为[in,out]。

b 是偏置向量，形状为[out]。

#### （2）作用

作用是通过对输入张量进行线性变换，将输入特征映射到输出特征。它广泛用于神经网络中的隐藏层，能够帮助模型学习输入与输出之间的线性关系。

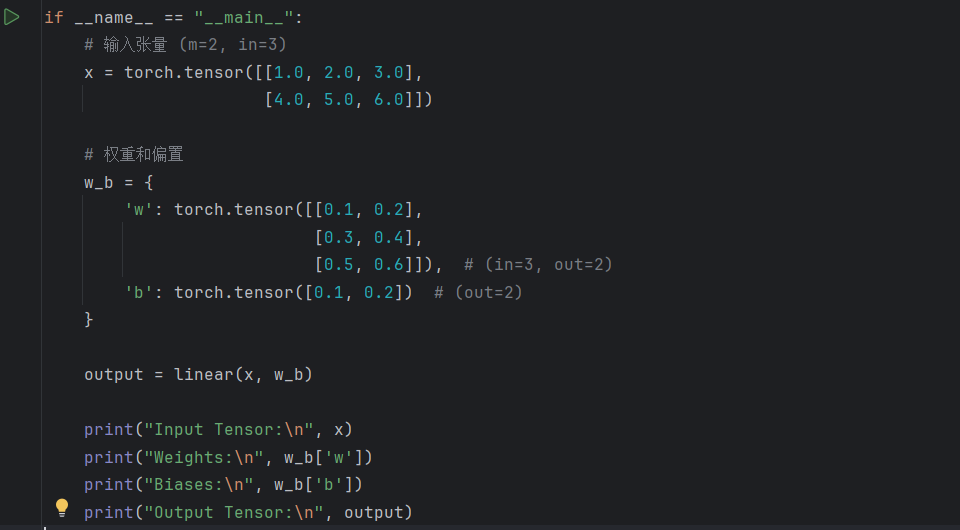
#### （3）代码



w, b = w\_b['w'], w\_b['b'] 从字典中提取权重和偏置

return x @ w + b 进行线性变换，x与权重相乘并加上偏置

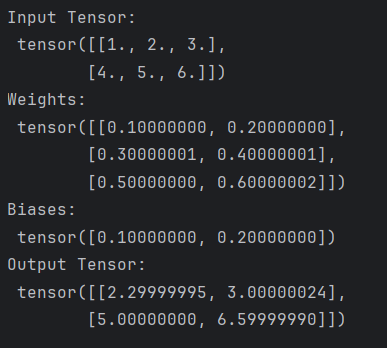
#### （4）测试示例及结果



权重 w 是一个 3×2 的矩阵，表示将三个输入特征映射到两个输出特征的权重。

偏置 b 是长度为2的向量，用于调整输出。

输出张量是2×2 的矩阵，表示经过线性变换后的结果。



### 5．ffn函数

#### （1）定义

前馈神经网络（Feed-Forward Network, FFN）

其中：

x是输入张量。

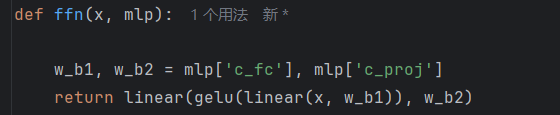
w1​和 w2是第一个和第二个线性层的权重矩阵。

b1和b2是第一个和第二个线性层的偏置向量。

#### （2）作用

将输入特征通过线性变换和非线性激活函数进行处理。在深度学习模型中，FFN 通常用于将输入特征转换为更高维的表示，增强模型的表达能力。

#### （3）代码



x，输入的张量，通常代表特征向量，形状为 [n\_seq, n\_embd]，其中n\_seq是序列长度， n\_embd 是每个元素的特征维度。

mlp，字典，包含两个线性层的权重和偏置。mlp['c\_fc'] 是第一个线性层的参数，mlp['c\_proj'] 是第二个线性层的参数。

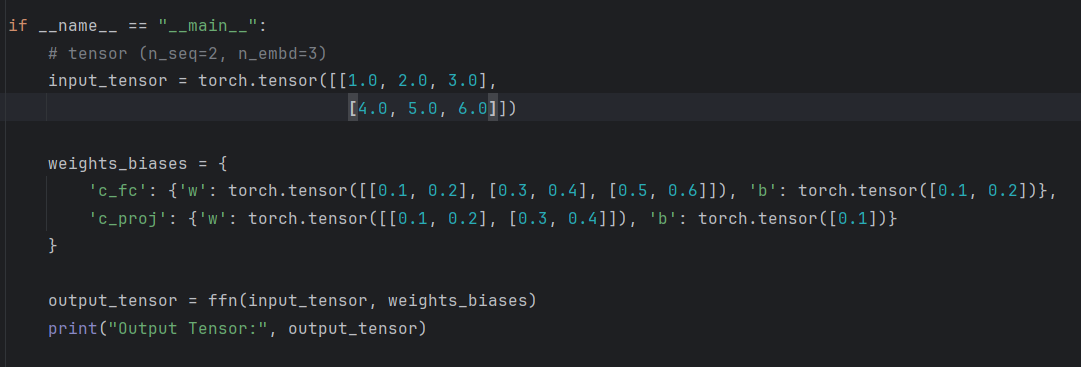
第二个表达式：

linear(x, w\_b1): 将输入 x 通过第一个线性层，计算得到新的张量。

gelu(...): 将第一个线性层的输出通过 GELU 激活函数进行非线性变换。

linear(..., w\_b2): 将经过激活的输出再通过第二个线性层，得到最终的输出。

#### （4）测试示例及结果

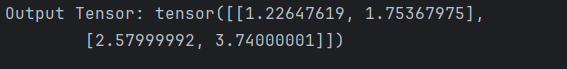


 **input\_tensor**创建一个张量，包含两个样本，每个样本有三个特征，形状为 [2, 3]。

 **weights\_biases**包含两个线性层的权重和偏置：

c\_fc: 第一个线性层的权重矩阵形状为 [3, 2]，偏置向量形状为 [2]。

c\_proj: 第二个线性层的权重矩阵形状为 [2, 1]，偏置向量形状为 [1]。



### 6．attention函数

#### （1）定义

在注意力机制中，最常用的形式是缩放点积注意力（Scaled Dot-Product Attention），其计算公式如下：

其中：

Q是查询（Query）矩阵。

K是键（Key）矩阵。

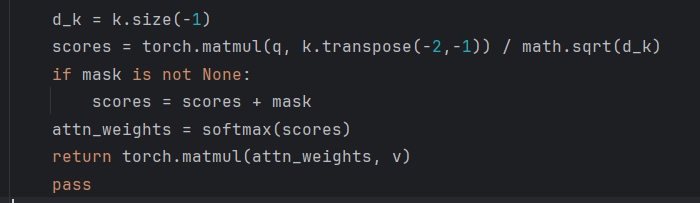
V 是值（Value）矩阵。

是键的维度，用于 缩放分数。

#### （2）作用

动态地计算不同输入之间的重要性。通过将查询与键进行比较，可以计算出每个值对给定查询的影响力，从而使模型能够更有效地关注与当前任务相关的输入部分。这在自然语言处理（NLP）和计算机视觉等领域中都非常重要，能够帮助模型更好地理解上下文。

#### （3）代码



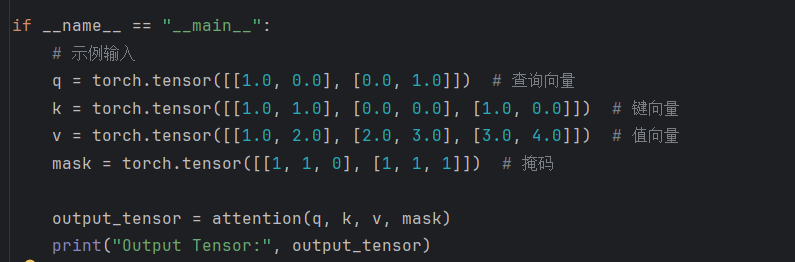
d\_k = q.size(-1) 获取查询向量的最后一维大小，表示每个查询向量的维度。

q和k的点积，计算每个查询与所有键的相似度，并进行缩放（除以 ）以提高数值稳定性。

通过 softmax 函数将注意力分数转换为权重，这样它们的和为 1。

最后，将注意力权重与值向量进行加权求和，得到最终的输出。

#### （4）测试示例及结果



### 7．mha函数

#### （1）定义

多头注意力机制（Multi-Head Attention, MHA）

给定输入的查询𝑄、𝐾和𝑉，多头注意力机制的计算过程可以表示为：

1、注意力计算

2、多头组合

其中：

是输出的权重矩阵。

#### （2）作用

Transformer模型中的一个关键组件，允许模型在不同的表示子空间中并行地学习信息。通过使用多个注意力头，模型能够捕获输入序列中不同部分之间的关系。

#### （3）代码

……

**qkv = x.chunk(3, dim=-1)**

使用 chunk 方法将 x 沿着最后一个维度（dim=-1）分成3个部分。这样，qkv 将是一个包含三个张量的元组，每个张量的形状为 [n\_seq, n\_embd / 3]。

……

**causal\_mask = torch.triu(torch.ones(x.shape[0], x.shape[0], device=x.device) \* -1e9, diagonal=1) # [n\_seq, n\_seq]**

因果掩码（Causal Mask）：当前时间的输出仅依赖于当前和之前的输入，而不依赖于未来的输入。torch.triu函数生成一个上三角矩阵，其中上三角部分（对角线以上）填充的是-1e9（非常小的值），而其他部分为0。这意味着在注意力计算中，未来的时间步将被掩盖（变成负无穷），从而使其对当前时间步的注意力无效。

……

**x = torch.cat(out\_heads, dim=-1)** # Concatenate along the last dimension to merge heads

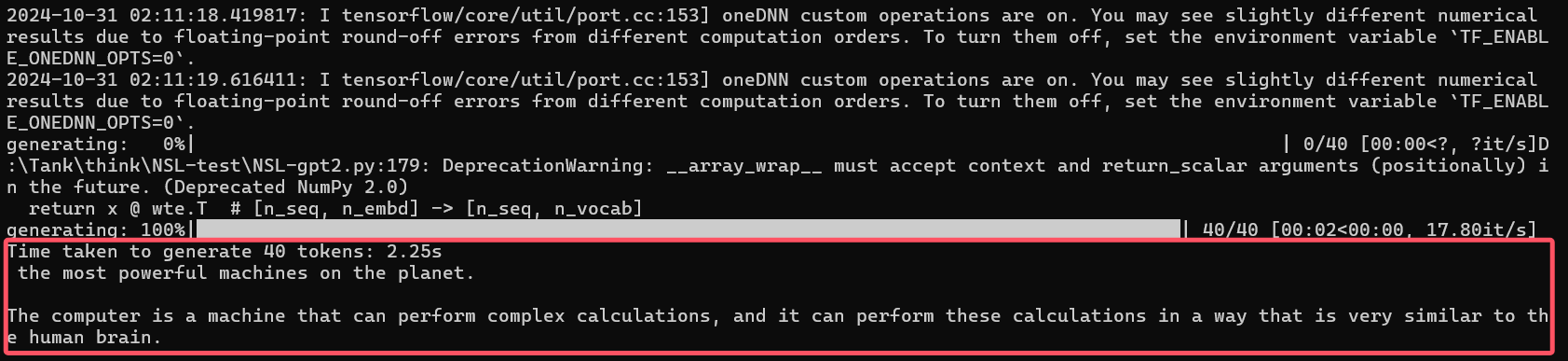
将多个注意力头的输出沿最后一个维度进行拼接。每个注意力头的输出具有相同的形状 [n\_seq, d\_k]（假设每个头的维度为d\_k），而合并后的输出形状将变为 [n\_seq, n\_head \* d\_k]。

……

#### （4）测试示例及结果

输入**python NSL-gpt2.py "Alan Turing theorized that computers would one day become" --n\_tokens\_to\_generate 40**

**输出**



## 二、困难及解决方式

### （1）Attention函数中Mask 相关代码错误

目的：应用掩码以排除无效的注意力位置。

错误形式：

if mask is not None:  
 scores = scores.masked\_fill(mask == 0, float('-inf'))

导致输出：！！！！！！！！！！！！！！！！！！

我的理解：

masked\_fill函数在scores中将mask为0的位置填充为-inf。这意味着在softmax计算中，这些位置的值将非常小，从而导致对应的注意力权重接近于0。

潜在问题：

这种方式要求mask的形状必须与scores完全匹配。由于填充后再进行softmax，如果mask没有被正确构造（例如未将无效位置填充为0），或者其值不符合预期（如不全是0或1），则可能导致不必要的结果。

正确形式：

if mask is not None:  
 scores = scores + mask

这里的mask通常包含0（表示有效位置）和-inf（表示无效位置）。将mask直接加到scores上会使得无效位置的值变为-inf，有效位置保持原值。这确保在后续softmax计算中，无效位置将不会对注意力权重产生影响。

### （2）Mha函数的错误

正确形式：

qkv = x.chunk(3, dim=-1)

错误形式：

qkv = x.view(x.size(0), -1, 3, x.size(1) // 3)

qkv = qkv.permute(0, 2, 1, 3)

通过查询chatgpt后得知：

View和permute的使用存在问题。

view函数用于重塑张量的形状。我试图将x重塑为[n\_seq, -1, 3, n\_embd // 3]，想把x分成3个部分（q, k, v）。但是，view要求张量的总元素数量不变，这意味着x.size(0) \* x.size(1)必须等于n\_seq \* 3 \* (n\_embd // 3)。如果x的形状不符合此条件，view将会引发错误。

使用permute将张量的维度重新排列为[n\_seq, 3, n\_head, d\_k]。这实际上是一个不必要的步骤，因为在注意力机制中，查询、键和值（q, k, v）通常直接从分开的张量中提取。在这里，qkv的顺序变得不再符合预期，使得后续操作（例如在每个头部上计算注意力）变得复杂且不符合标准。

### （3）依赖添加失败

无法直接通过命令pip install -r requirements.txt安装依赖

尝试python -m ensurepip –upgrade，python -m pip install --upgrade pip，pip install setuptools，pip install --upgrade setuptools wheel后都无法解决

通过下载模型，逐个找到缺少的依赖，通过cmd命令下载，pip install regex，pip install tqdm，pip install tensorflow，pip install requests，pip install fire，pip install torch，pip install numpy

### （4）gelu函数错误

正确形式：

sqrt\_term = torch.sqrt(torch.tensor(2 / math.pi)) # 将计算结果转换为张量  
return 0.5 \* x \* (1 + torch.tanh(sqrt\_term \* (x + 0.044715 \* x.pow(3))))

错误形式：

return 0.5 \* x \* (1 + torch.tanh(torch.sqrt(2 / math.pi) \* (x + 0.044715 \* x.pow(3))))

原因：

原因是 torch.sqrt() 函数需要一个 PyTorch 张量作为输入，而我一开始传入的是一个浮点数（2 / math.pi）。要把这个计算结果转换为一个张量。通过 torch.tensor() 来实现。