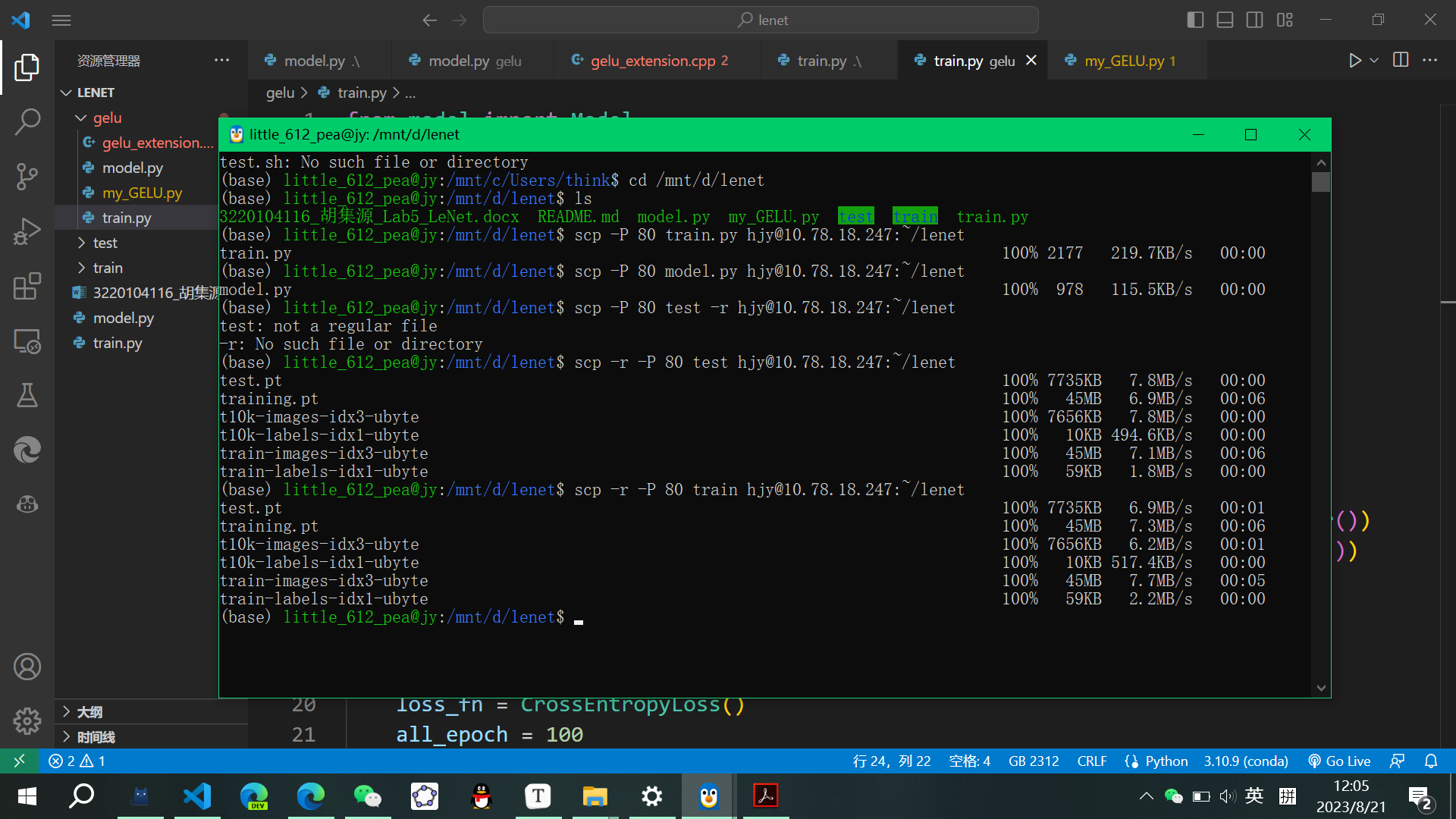
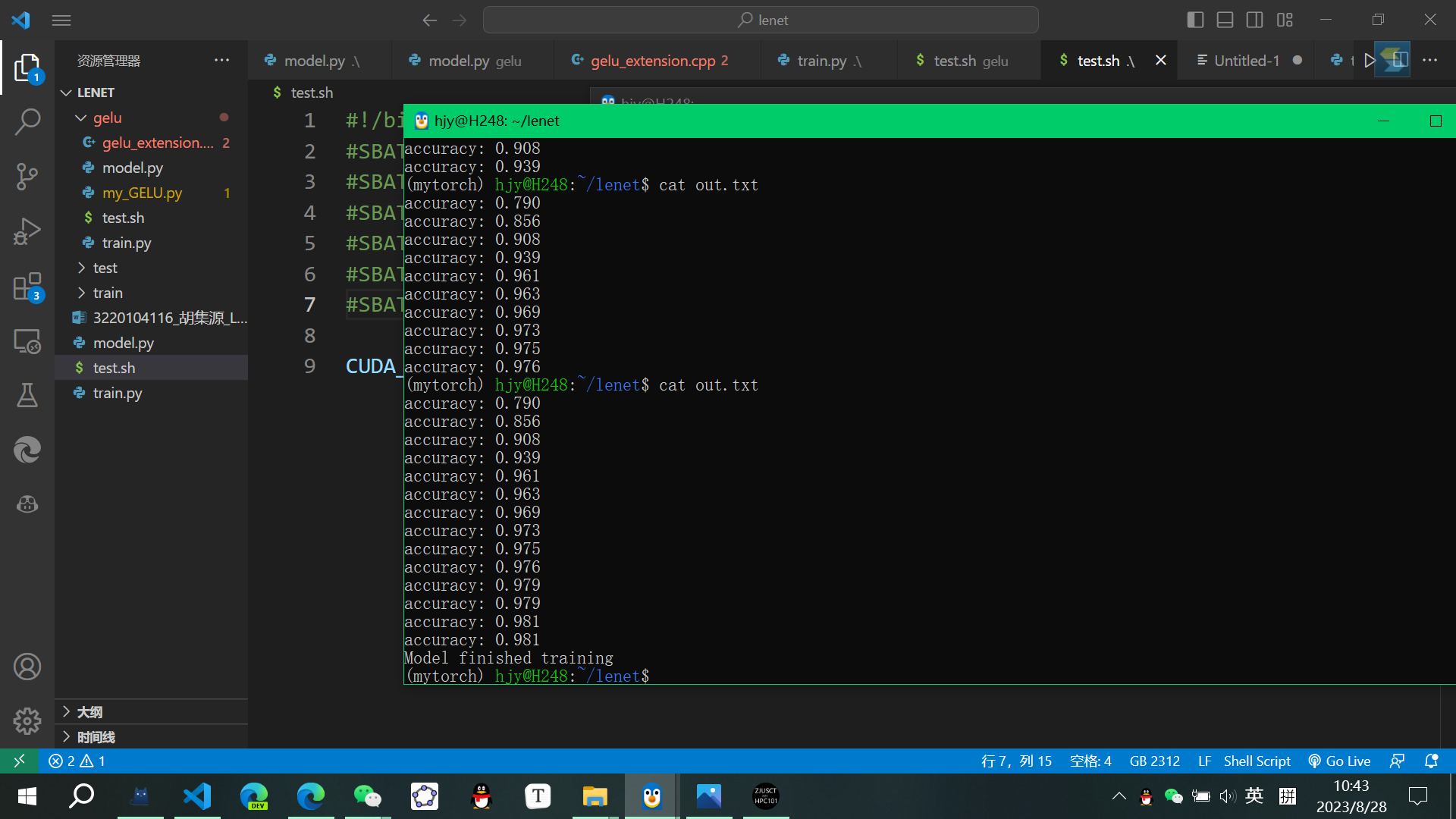
**Lab5-LeNet report**

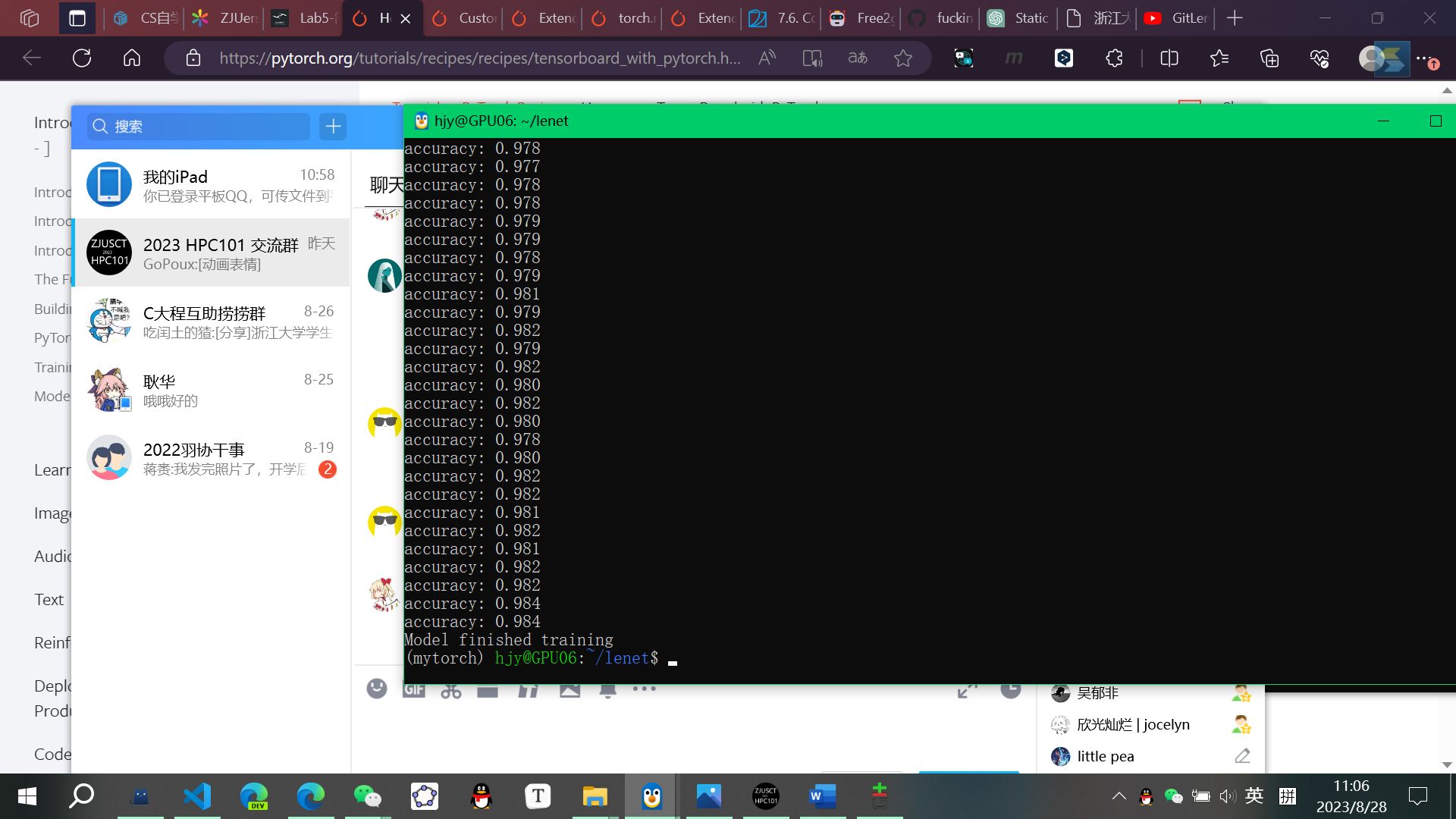
3220104116 胡集源

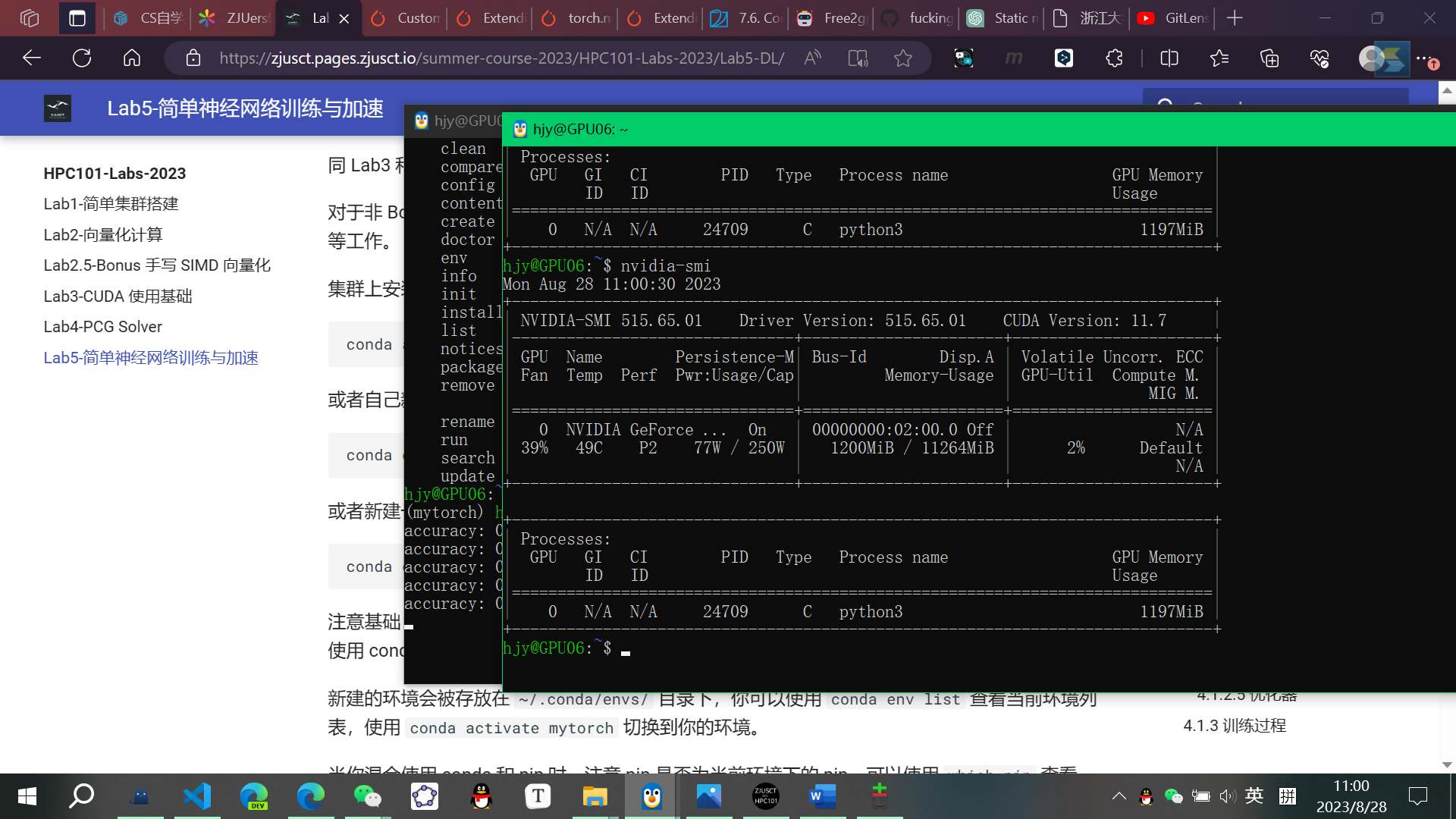
在gpu上训练 使用cuda

编写GELU算子，继承 torch.autograd.Function 并实现 forward 和 backward 函数。forward 函数接受输入并返回输出，backward 函数接受输入和梯度，并返回相对于输入的梯度









获取Tensorboard 模型的损失曲线、LeNet-5 的准确率曲线等截图

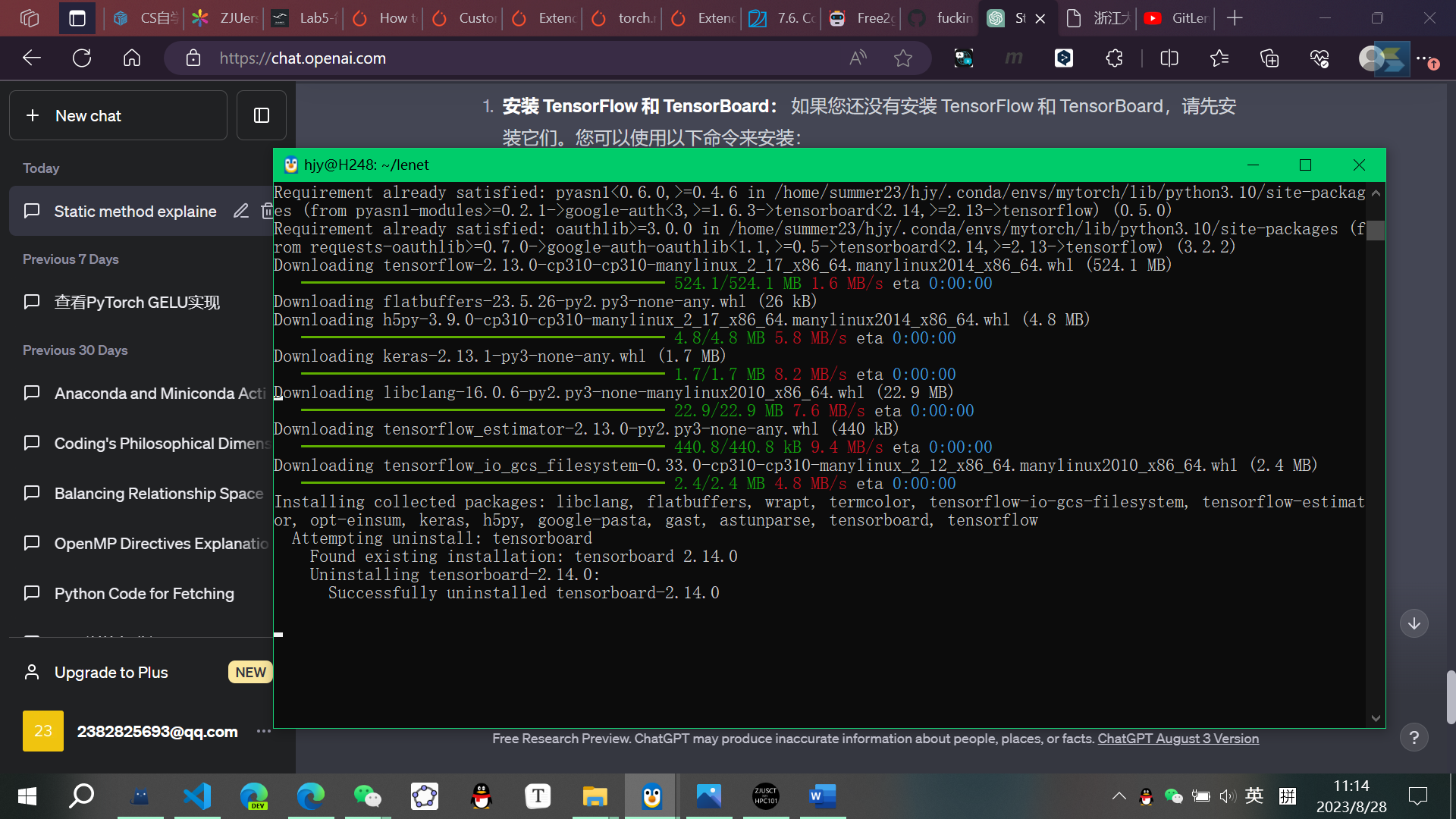
ssh hjy@10.78.18.247 -p 80

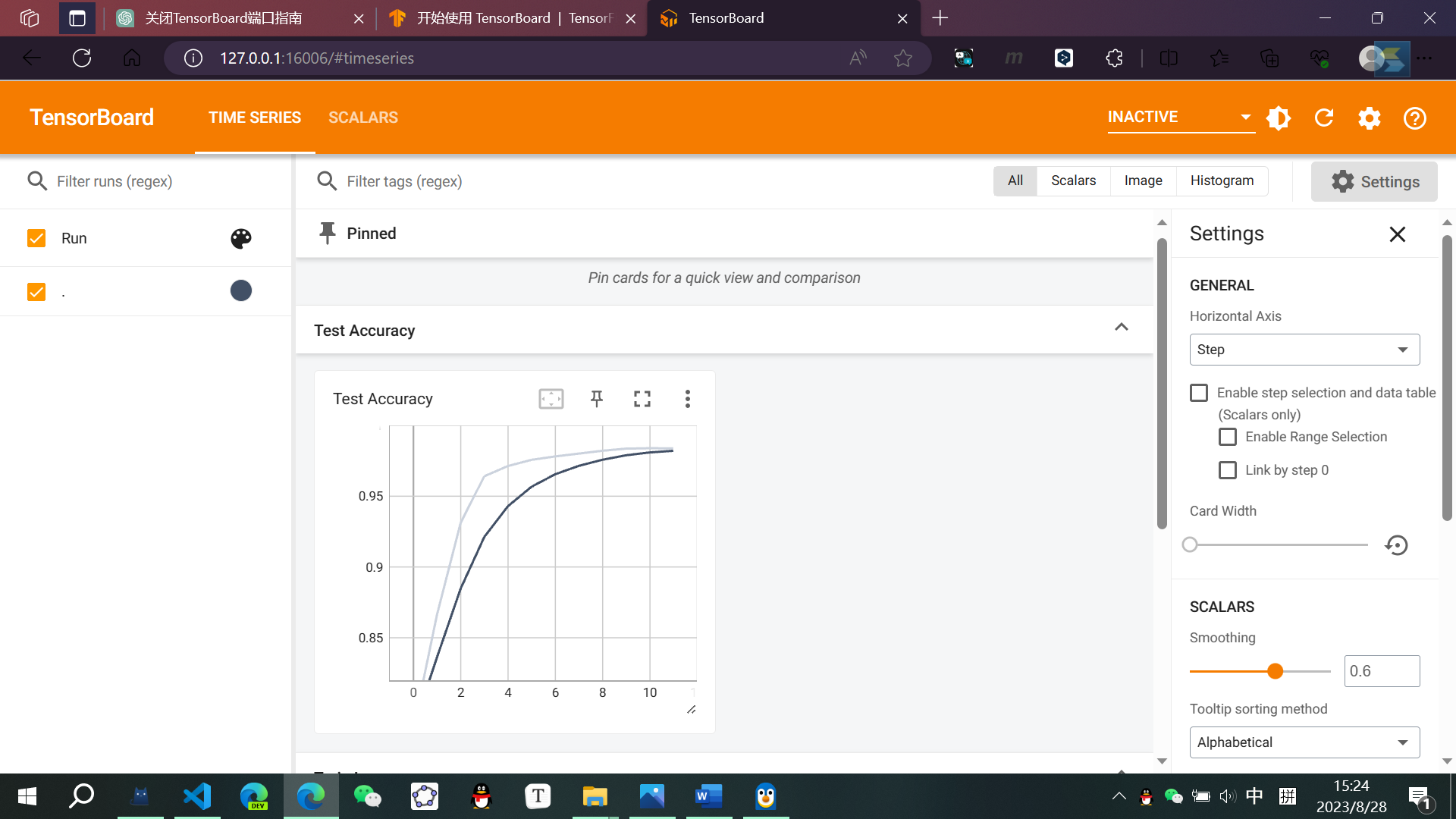
salloc -p 2080Ti -N 1 -n 1 --cpus-per-task=8 --gpus=1

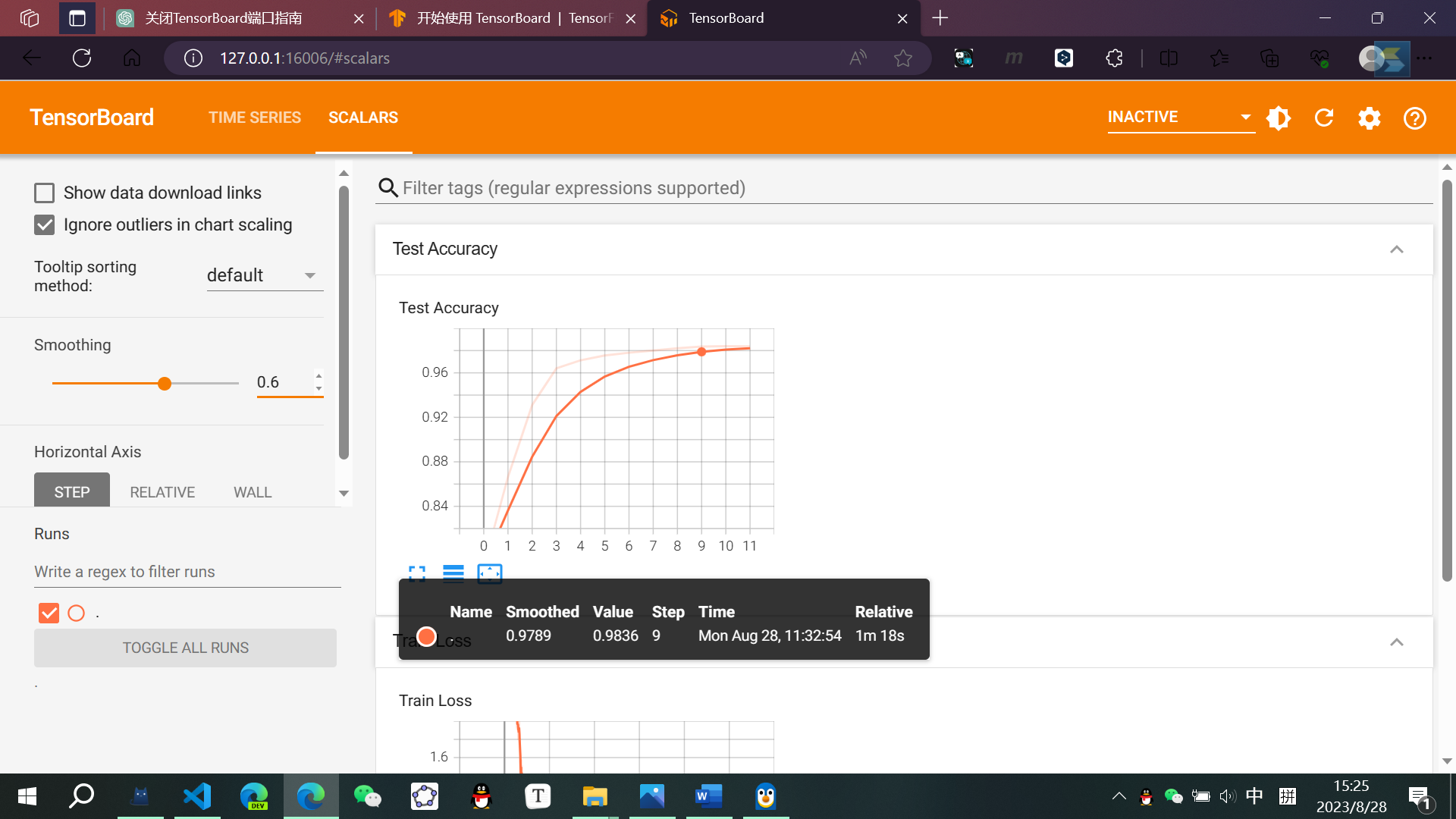
使用ssh -L 16006:127.0.0.1:6006 hjy@10.78.18.247 -p 80，将本地端口16006与远程服务器（IP地址为10.78.18.247）上的端口80进行连接

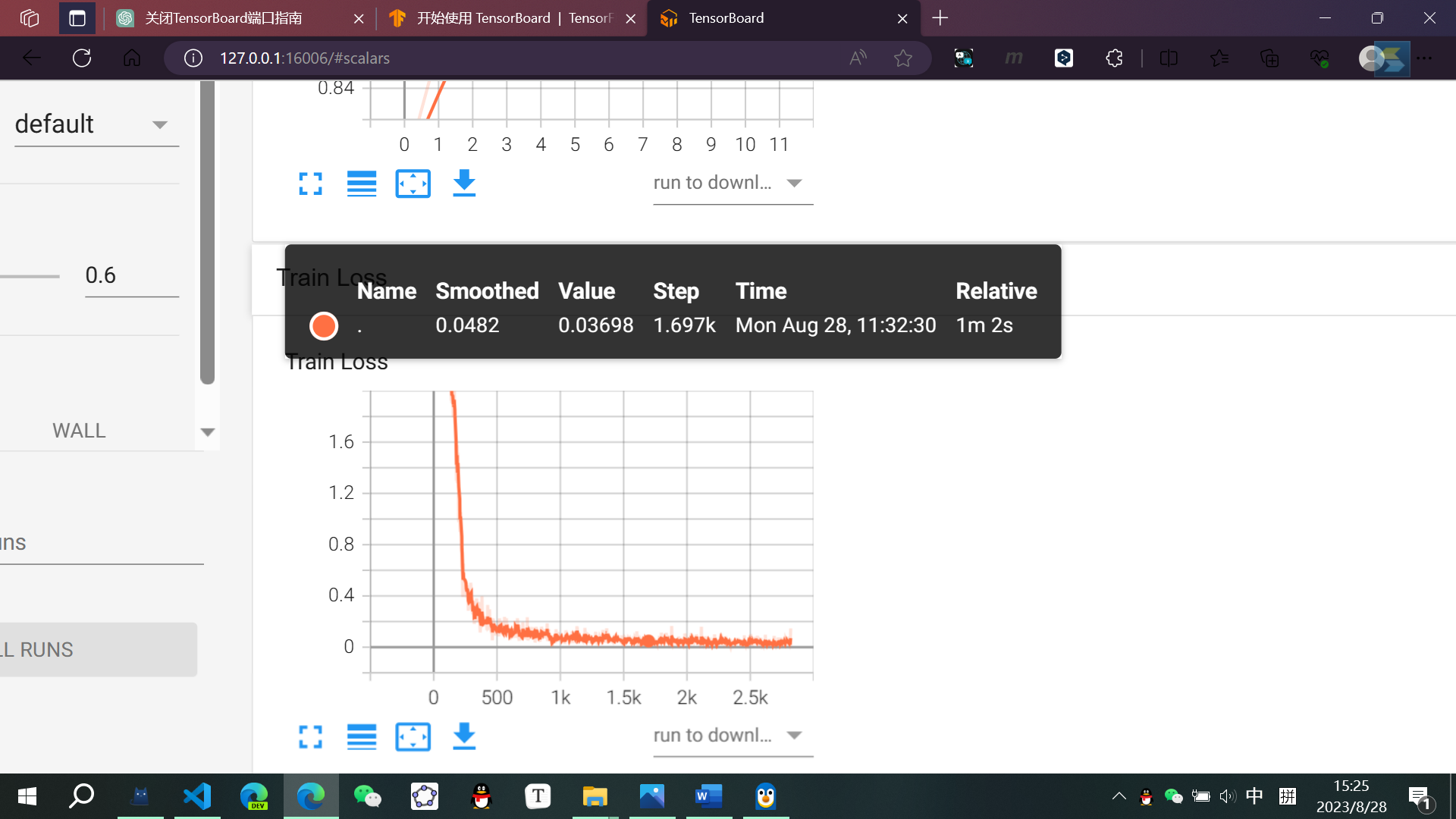
在H248节点上使用tensorboard ：

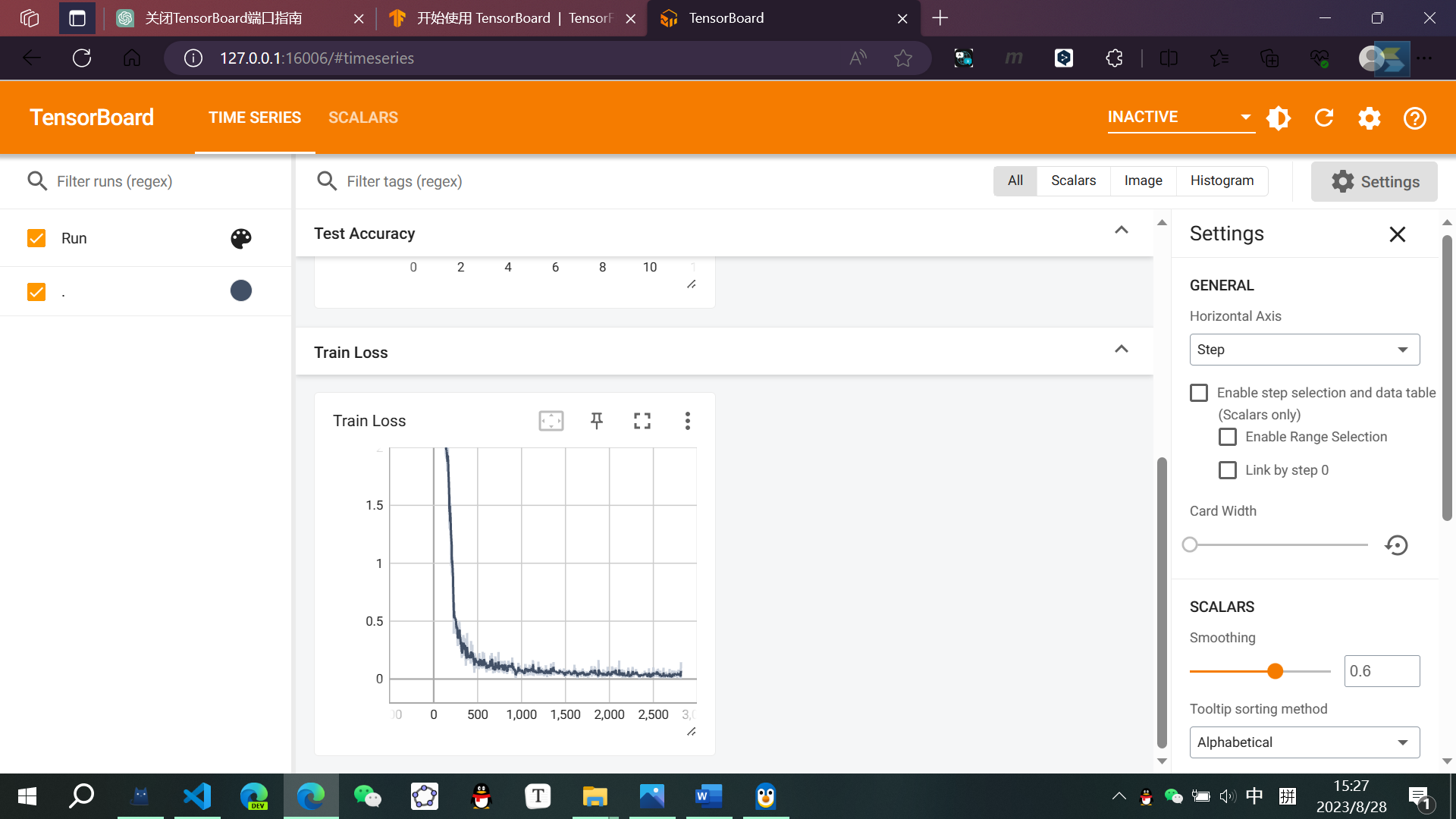
tensorboard --logdir=runs --bind\_all

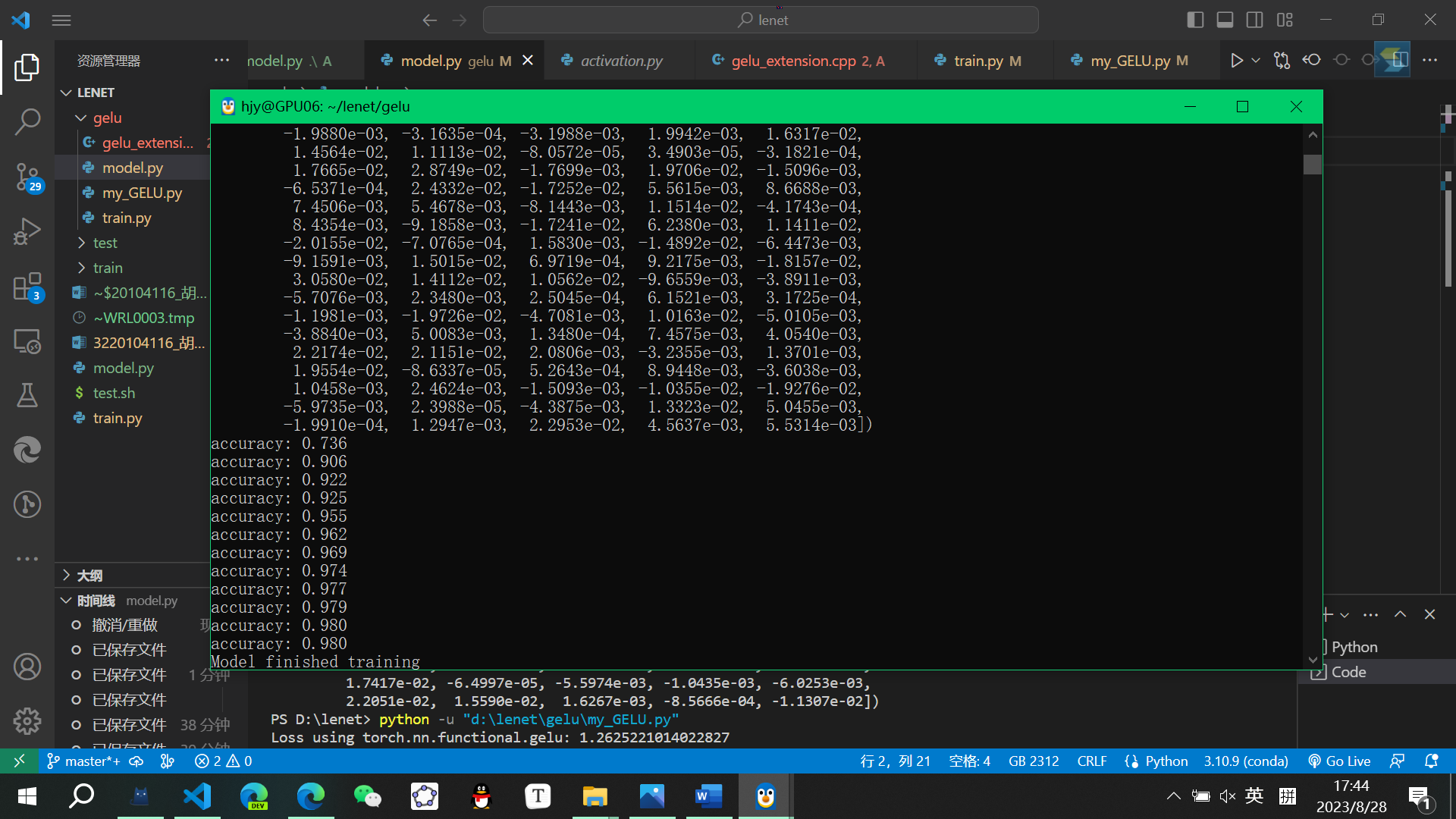


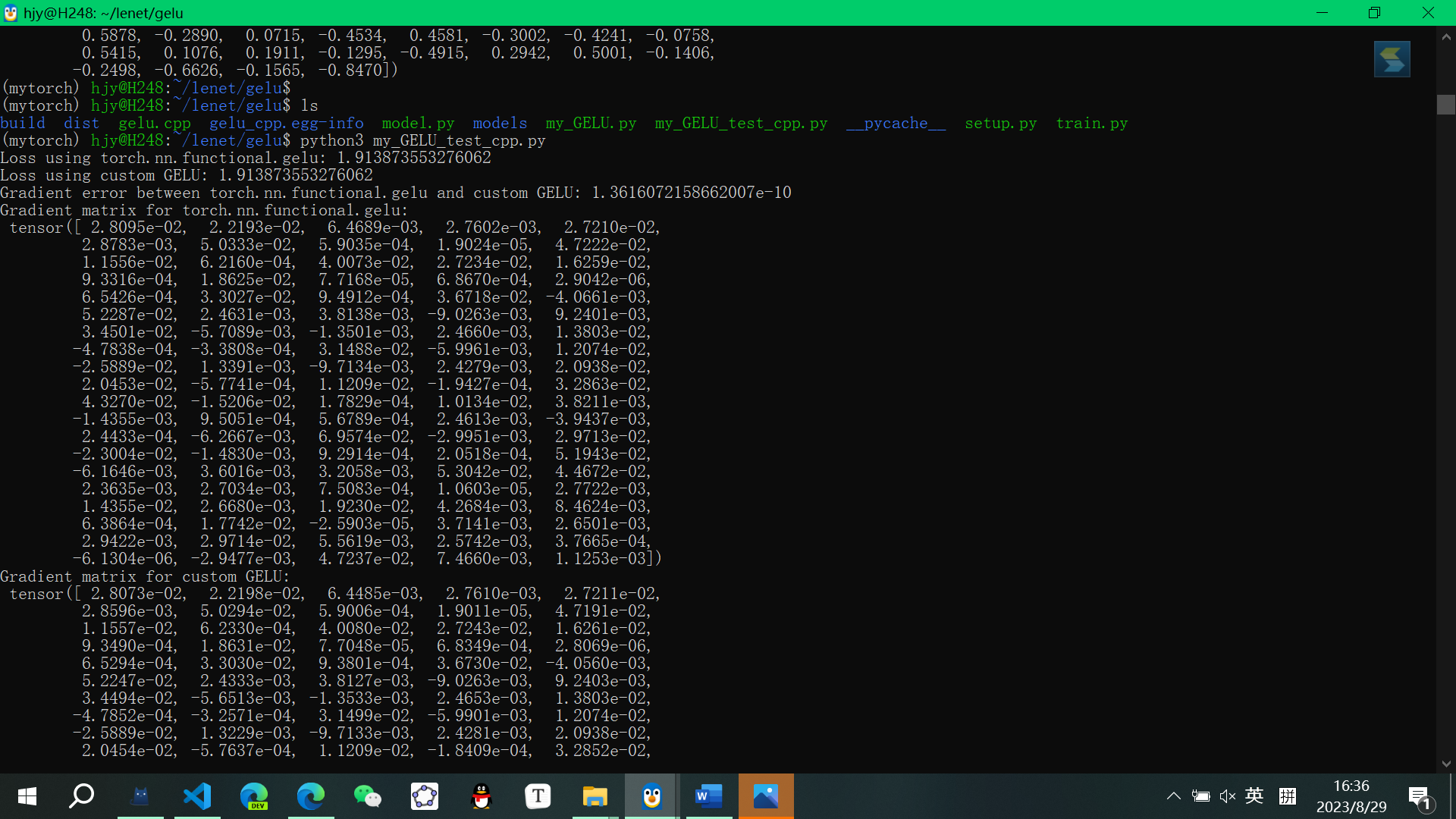












每次算一个词出来，基于概率选择下一个单词

统计每个词在所有文本中出现的概率🡪 每个词出现在一个特定句子之后的频率

Word embedding positional encoding（先后顺序位置信息）输出的结果为key

Word2vec 通过查表的方式完成映射 词表

Attention

Encoder

Decoder（解码器）

按照下列表格中给定的模型结构参数实现模型。尝试使用

分布式训练策略（DP/ZeRO，PP，TP）

混合精度训练

Gradient Accumulation

Gradient Checkpointing

CPU/NVMe Offload

等技术对你的模型进行加速，并将其与单卡训练进行对比（包括训练速度、显存占用、模型计算量等）。 你可以自行选择合适的 batch size 和训练数据量，并不需要关注 loss ，只需要保证不同的加速策略下训练的总数据量相同即可。

Model size Hidden size Attention-heads Layers Sequence length Learning rate

1.6B 1600 32 48 1024 5e-4

import torch

import math

# Test the derivative function

x = torch.tensor([1.0, 2.0, 3.0], requires\_grad=True)

y = gelu\_derivative(x)

print(y)

@staticmethod

def backward(ctx, grad\_output):

input, = ctx.saved\_tensors

exp\_term = torch.exp(-0.5 \* input\*\*2)

cdf\_term = 0.5 \* (1 + torch.erf(input / math.sqrt(2.0)))

gradient = 0.5 \* (1.0 + torch.tanh((math.sqrt(2.0 / math.pi) \* (input + 0.044715 \* input\*\*3)))) + \

0.5 \* input \* (1 - torch.tanh((math.sqrt(2.0 / math.pi) \* (input + 0.044715 \* input\*\*3)))\*\*2) \* \

(math.sqrt(2.0 / math.pi) \* (1 + 0.044715 \* input\*\*2)) \* exp\_term - \

0.5 \* input \* (1.0 + torch.tanh((math.sqrt(2.0 / math.pi) \* (input + 0.044715 \* input\*\*3)))) \* exp\_term \* cdf\_term

return grad\_output \* gradient

#这两个损失值应该相等，因为两个GELU实现的功能是一样的

#但是，由于torch.nn.functional.gelu是基于C++实现的，而custom GELU是基于Python实现的，

#所以custom GELU的运行速度会比torch.nn.functional.gelu慢很多

#但是，custom GELU可以通过torch.autograd.Function实现反向传播，

#而torch.nn.functional.gelu不可以，所以custom GELU可以用于模型的训练

#而torch.nn.functional.gelu只能用于模型的推理

GELU 的 PyTorch C++ 实现源码位于 PyTorch 库的aten/src/ATen/native/Activation.cpp 文件中

