**内容概述：**

1. 拓扑神经网络代码
   1. 完成加速（训练速度提升了36倍，GPU利用率达到了100%）
2. RL失败原因排查
   1. Ulimit配置的值太小，循环体中创建句柄，导致溢出。
   2. 使用进程池却没有错误处理，导致主程序一直运行，而进程池中的进程全部宕不work了。

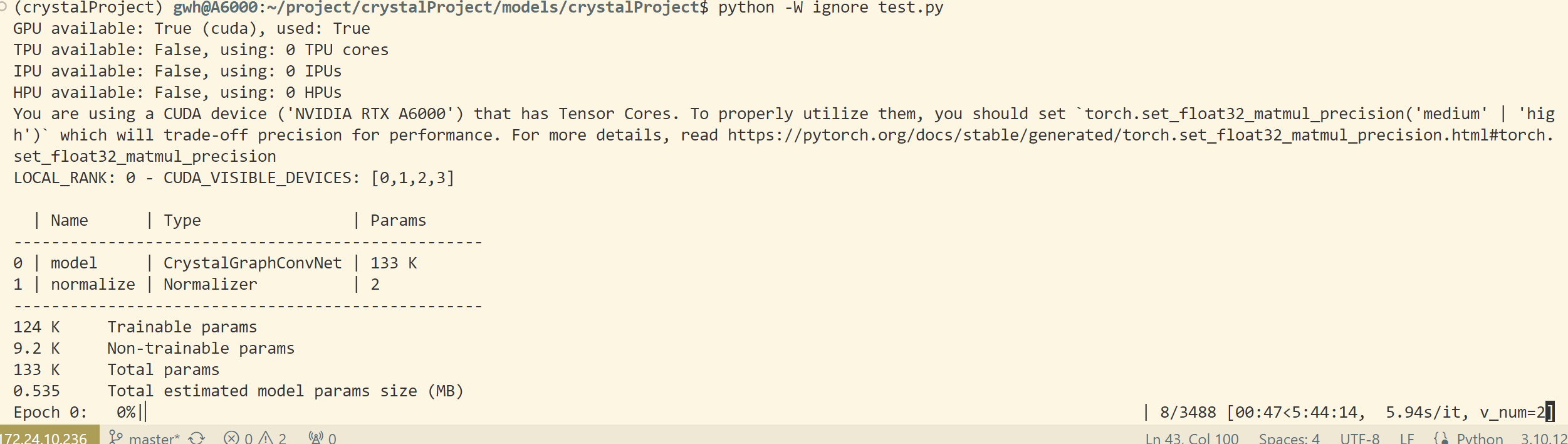
**具体内容：**

1. **拓扑神经网络加速trick介绍（留个mark）**
   1. 数据预处理，（解决IO瓶颈和CPU瓶颈）

参考的代码中dataset是实时处理数据的，当任务中晶体的原子数较少时，计算任务并不大，所以可行。但是程序中**对边的计算复杂度是O(n2)**，导致在处理多孔材料时，预处理成为了最大的性能瓶颈，对于甲烷存储数据训练一轮epoch需要花费半天，不利于实验。

为了解决这个问题，首先考虑到，预处理不需要每次epoch都处理一遍，可以提前处理好，于是写了一个单独的函数用于对数据进行预处理。整个过程分为三个阶段，最后成功解决了问题：

* + 1. 预处理函数处理全部流程，最后生成的数据直接可以被模型接受，点和边的特征全部被保留，利用torch.save进行存储。
    2. 预处理函数仍然处理全部流程，最后生成的数据利用numpy保存，减少文件大小。
    3. 预处理只保留简单的信息，原子的信息只保存原子序数，边只保留距离值，特征化留给后续进行操作。
  1. 内存异步加载数据导入GPU（指令级并行）
     1. 修改了dataloader，原先的流程是先从内存将数据转移到显存上，在进行GPU运算，现在采用异步执行，减少GPU等待时间，。
  2. 利用GPU完成transform（利用GPU代替CPU处理矩阵运算）
     1. 数据预处理模块中原子的特征只有一个原子序数，边也只有距离，都是标量，目的是为了减少文件大小，降低IO时间，所以将特征化的操作留给CPU进行处理，由于特征化是一些矩阵运算，CPU处理起来耗时比较大，GPU需要等待CPU特征化，并没有充分发挥GPU的性能。
     2. 利用GPU实现特征化操作，方法是将特征化操作实现为一个没有训练参数的网络，集成到模型中去，这一处理是十分有效的，原因有两点：1. GPU处理速度快，2. 不使用CPU处理也减少了从内存拷贝到显存上的数据量，进一步降低了IO。
  3. 效果展示，**速度从6s/it加速到6it/s**。





1. RL失败原因排查
   1. 失败原因分析
      1. Linux中ulimit默认设置是1024，即每个进程至多可以同时打开1024个file，预测器中每次运行都是创建两个日志logger，logger中会打开一个句柄（fileHaddler）并且不会释放，即每次运行都会多两个句柄，当运行了五百多次循环后，进程中打开的句柄会达到可以打开的文件极限，程序报错不在work。
      2. 但是还有一个问题，程序已经达到了可以打开的文件极限，应当报错不在运行了，但是程序已经再运行，只是什么输出都没有了。这个原因是因为RL程序的主框架是利用进程池完成的，进程池中的程序报错后，主进程不会受到影响一般会继续执行，继续调用进程池时当再次需要打开一个file的时候仍会继续报错并退出，这就导致了程序仍然在运行，但是没有任何输出。
   2. 解决方案
      1. 临时解决方案

可以通过修改.bashrc，添加ulimit -n 1048576，重启一下终端，后续就可以了。

* + 1. 根本解决方案

第三版预测器中会修复重复打开日志的问题，不在使用预处理模块。将处理流程并入到dataset中。

**本周安排：**

1. 复现论文结果，复现三个数据集中论文展示的效果。
   1. 晶体数据集
   2. qmof数据集
   3. COF甲烷存储数据集
2. 进行对比实验
   1. 不同层对于结果的影响3，5，7（之前做过这个实验，结果是5层效果最好）
   2. 验证截断距离的选择对结果的影响，对比截断距离8和2
   3. 验证注意力机制对结果的影响，对比5层网络，截断距离设置为8，加入注意力机制和一般图网络对结果的影响。