**PSGD实验报告**

1. **算法**

见文件夹中的Project\_Parallel SGD\_En1.pdf文件

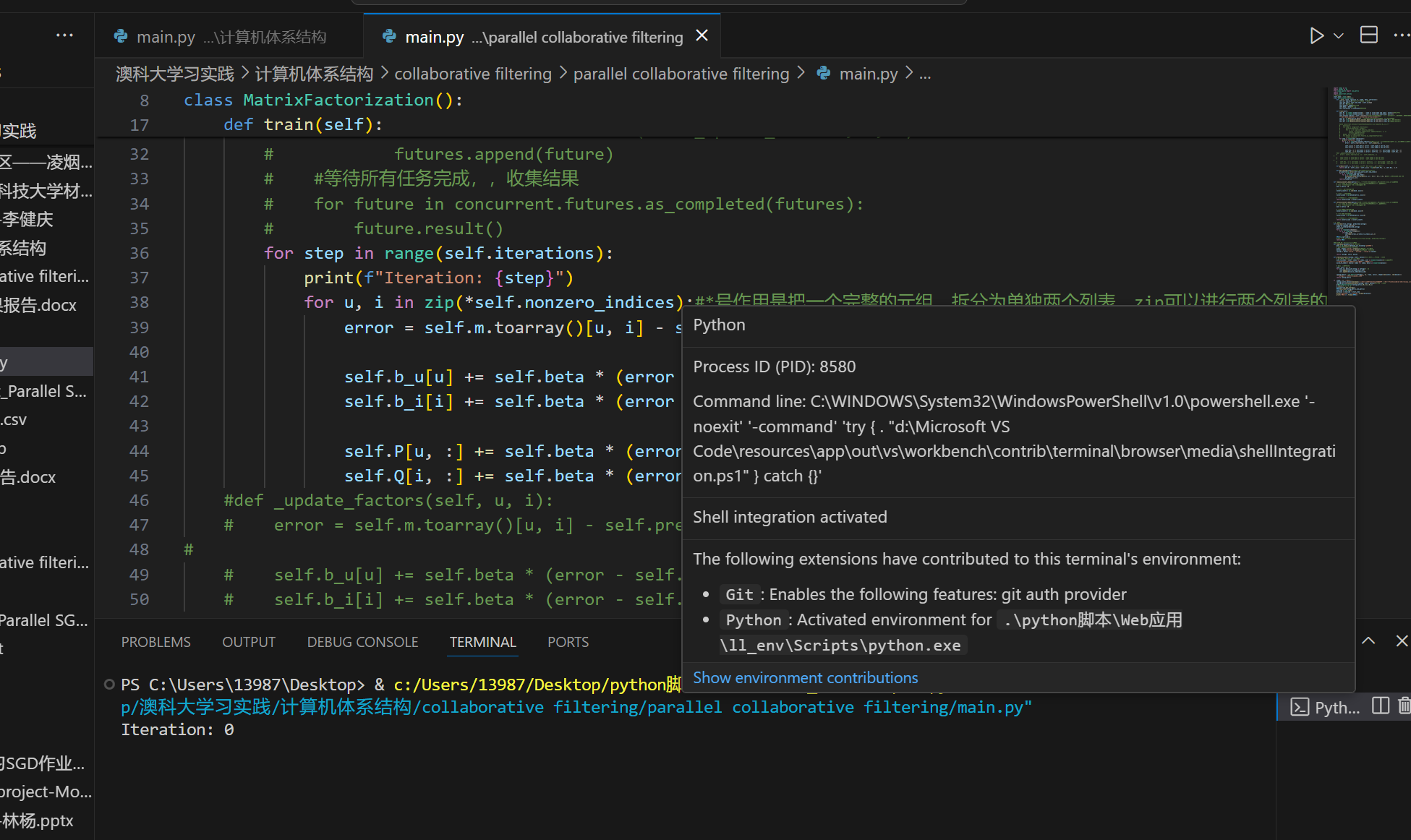
1. **代码实现**

见文件夹中的main.py文件

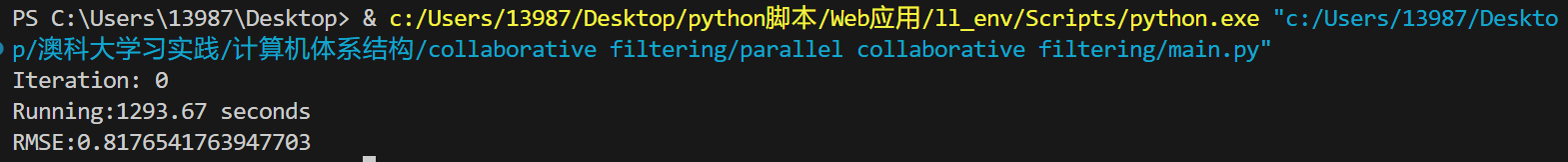
1. **串行与并行对比**

**1.正常串行**

首先对其进行串行运行，串行程序的PID进程号为8580：

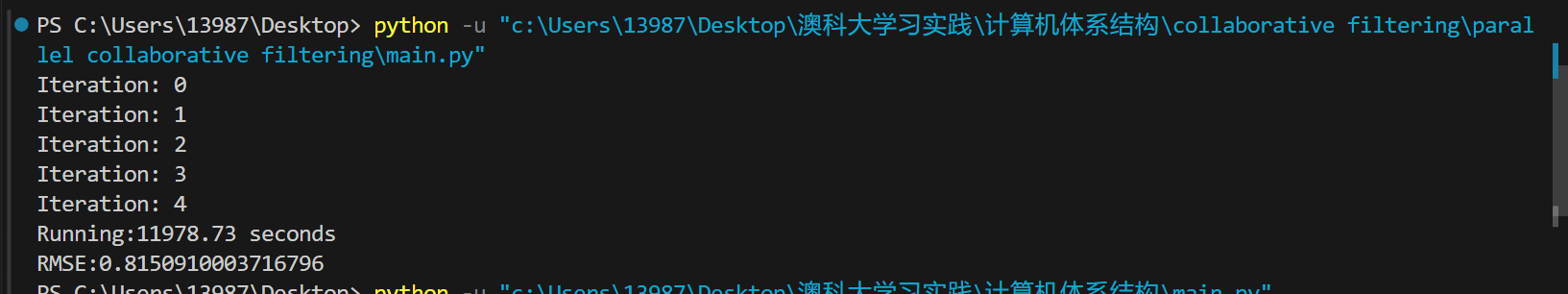


**迭代次数设置为1时运行结果：**



用时1293.67秒，RMSE约为0.818

**迭代次数设置为5时运行结果：**



用时11978.73秒，RMSE约为0.815

对代码进行并行化改造，改造方式如下：

1.1引入concurrent.futures模块，使用ProcessPoolExecutor创建一个进程池，以便在多个进程中并行执行任务。

1.2在train方法中，将原来的for循环拆分为两层：外层循环遍历迭代次数，内层循环将每个用户-物品对的更新操作封装成一个异步任务，通过executor.submit(self.\_update\_factors, u, i)提交到进程池中执行。

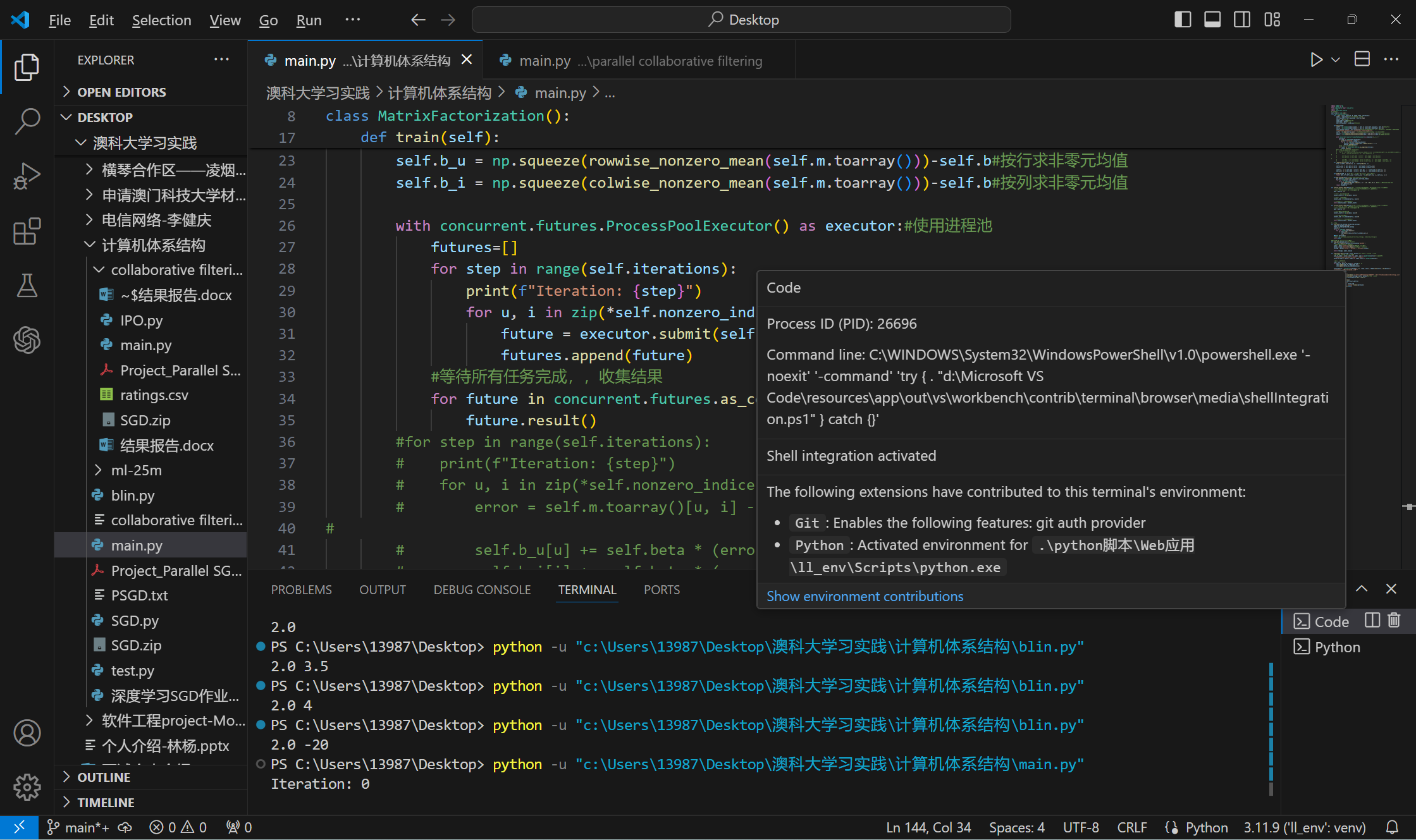
1.3 定义一个新的私有方法\_update\_factors(u, i)，该方法接受一个用户索引u和一个物品索引i，执行单个用户-物品对的更新逻辑。这是为了将更新操作与主训练流程解耦，便于并行化。

* 1. 使用concurrent.futures.as\_completed(futures)等待所有提交的任务完成，并确保结果已正确处理。在并行计算场景下，无需关心任务完成的顺序。

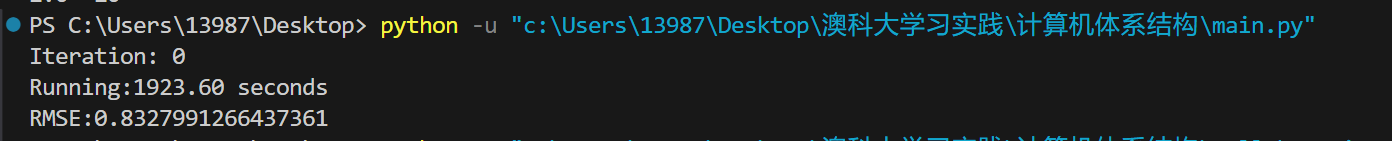
通过上述改动，MatrixFactorization类的训练过程得以并行化，理论上可以显著减少训练所需的时间，尤其是在大规模数据集上。请注意，实际性能提升会受到硬件资源（如CPU核心数、内存大小、I/O速度等）以及数据集特性的限制。此外，由于进程间通信和数据同步的开销，对于非常小的数据集或极少量的迭代次数，可能并行化带来的收益并不明显

**2. 并行化运行**

改造后运行此并行化程序，并行化程序的PID是26696：

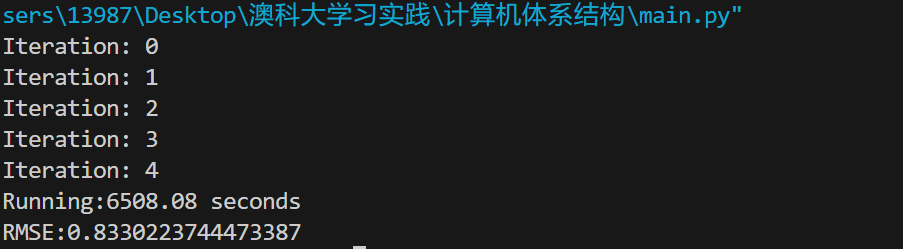


**迭代次数设置为1时运行结果：**



用时1923秒，RMSE约为0.832

**迭代次数设置为5时运行结果：**

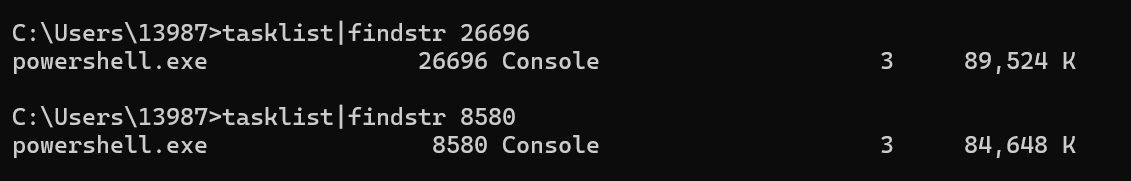


用时6508.08，RMSE约为0.833

3.串行程序和并行程序的比较：

3.1运行时：

不难看出，并行化程序的内存使用更高，比串行化程序约高了5000KB，可见代码的并行化可能会导致内存占用提高。



3.2 运行结果：

从输出结果上来看，设置迭代次数为1时，串行程序用时1293.67秒，RMSE约为0.818；并行程序用时1923秒，RMSE约为0.832。

设置迭代次数为5时，串行程序用时11978.73秒，RMSE约为0.815；并行程序运行时间6508.08秒，RMSE约为0.833

由此看出，并行化程序在设置迭代次数提高时，运行时间大大优于串行程序，但是以RMSE为精度考量时，并行程序的精度仍然有待提高。