先进入目录/blue/qsong1/wang.qing/benchmark\_scLLM\_API/UCE-main/UCE-main/

**UCE**

1.

先运行dataset\_making.py文件，将csv转为适合的输入。

由于每个模型的输入不太一样所以不同的csv的到输出可能不一样有些是pkl有些是ndarray。

这里主要要考虑的参数就是is\_sorted=True, seq\_length=8192

，别的就默认就行

一个是sequence是否按表达量排序，另一个是sequence的长度限制。

还有文件路径需要更改

2.

针对fixed embedding还要运行get\_ebd.py得到模型的embedding。

3.

针对fixed embedding运行benchmarking\_main\_EBD.py进行训练和预测

这里考虑的参数，别的就默认就行

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例

4.

针对fine tuning运行benchmarking\_main\_FT.py进行训练和预测

这里考虑的参数，别的就默认就行

parser.add\_argument('--ft\_list', type=list, default=['out\_proj'], help='')

控制lora微调的网络层。

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例。

注意保持目录的结构，更改可能会导致package冲突。

总共8个模型

现在UCE已经整理完成了，先按照UCE构建项目，其余的模型框架和UCE差不多的。

直接在hypegator复制会比较方便，模型数据也比较大。

代码在/blue/qsong1/wang.qing/benchmark\_scLLM\_API/UCE-main/

数据在/blue/qsong1/wang.qing/benchmark\_dataset\_API/UCE/

**tGPT**

运行步骤同UCE

代码在/blue/qsong1/wang.qing/benchmark\_scLLM\_API/tGPT-main/tGPT-main/tGPT-main

数据在/blue/qsong1/wang.qing/benchmark\_dataset\_API/tGPT/

**scGPT**

运行步骤

1. 这里没有dataset\_making.py，直接运行get\_ebd.py就行，同时这里没有参数is\_ordered，

只有max\_length=1200。来标记序列长度。

2.

针对fixed embedding运行benchmarking\_main\_EBD.py进行训练和预测

3.

针对fine tuning运行benchmarking\_main\_FT.py进行训练和预测

注意控制参数

parser.add\_argument('--max\_length', type=int, default=1200, help='')

标记序列长度。

parser.add\_argument('--ft\_list', type=list, default=['out\_proj'], help='')

控制lora微调的网络层。

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例。

代码在/blue/qsong1/wang.qing/benchmark\_scLLM\_API/scGPT-main/scGPT-main/scgpt/tasks/

数据在/blue/qsong1/wang.qing/benchmark\_dataset\_API/scGPT/

**scFoundation**

1.先运行dataset\_making.py文件，将csv转为适合的输入。

这里主参数is\_sorted和seq\_length都没有，不需要设置

2.

针对fixed embedding运行get\_ebd.py得到模型的embedding。

3.

针对fixed embedding运行benchmarking\_main\_EBD.py进行训练和预测

这里考虑的参数，别的就默认就行

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例

4.

针对fine tuning运行benchmarking\_main\_FT.py进行训练和预测

这里考虑的参数，别的就默认就行

parser.add\_argument('--ft\_list', type=list, default=['out\_proj'], help='')

控制lora微调的网络层。

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例。

代码在/blue/qsong1/wang.qing/benchmark\_scLLM\_API/scFoundation-main/scFoundation-main/model/

数据在/blue/qsong1/wang.qing/benchmark\_dataset\_API/scFoundation/

**scBERT**

1.

先运行dataset\_making.py文件，将csv转为适合的输入。

这里主要要考虑的参数就是is\_sorted=True, seq\_length=8192

，别的就默认就行

一个是sequence是否按表达量排序，另一个是sequence的长度限制。

还有文件路径需要更改

2.

针对fixed embedding还要运行get\_ebd.py得到模型的embedding。

3.

针对fixed embedding运行benchmarking\_main\_EBD.py进行训练和预测

这里考虑的参数，别的就默认就行

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例

4.

针对fine tuning运行benchmarking\_main\_FT.py进行训练和预测

这里考虑的参数，别的就默认就行

parser.add\_argument('--ft\_list', type=list, default=['to\_k','to\_v','to\_q'], help='')

控制lora微调的网络层。

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例。

代码在/blue/qsong1/wang.qing/benchmark\_scLLM\_API/scBERT-master/scBERT-master/

数据在/blue/qsong1/wang.qing/benchmark\_dataset\_API/scBert/

**Openbiomed(cellLM)**

1.

直接运行get\_ebd.py得到模型的embedding。

dataset = Zheng68k(path=dt\_path+filename,config=config["data"], seed=2023, max\_len=2000)

控制序列长度 max\_len=2000

2.

针对fixed embedding运行benchmarking\_main\_EBD.py进行训练和预测

这里考虑的参数，别的就默认就行

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例

3.

针对fine tuning运行benchmarking\_main\_FT.py进行训练和预测

这里考虑的参数，别的就默认就行

parser.add\_argument('--ft\_list', type=list, default=['to\_q','to\_k','to\_v'], help='')

控制lora微调的网络层。

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例。

代码在/blue/qsong1/wang.qing/benchmark\_scLLM\_API/OpenBioMed-main/OpenBioMed-main/

数据在/blue/qsong1/wang.qing/benchmark\_dataset\_API/cellLM/

**CellPLM**

1.

先运行dataset\_making.py文件

这没有参数可以调整，都要匹配原始模型

2.

针对fixed embedding还要运行get\_ebd.py得到模型的embedding。

3.

针对fixed embedding运行benchmarking\_main\_EBD.py进行训练和预测

这里考虑的参数，别的就默认就行

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例

4.

针对fine tuning运行benchmarking\_main\_FT.py进行训练和预测

parser.add\_argument('--max\_length', type=int, default=2048,help='seq.')

控制序列长度

这里考虑的参数，别的就默认就行

parser.add\_argument('--ft\_list', type=list, default=['query\_projection','key\_projection','value\_projection'], help='')

控制lora微调的网络层。

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例。

代码在/blue/qsong1/wang.qing/benchmark\_scLLM\_API/CellPLM-main/CellPLM-main/

数据在/blue/qsong1/wang.qing/benchmark\_dataset\_API/cellPLM/

**Geneformer**

1.

先运行dataset\_making.py文件

这里主要要考虑的参数就是dataset\_generator(directory\_path='/', is\_sorted=True):

这里seq\_length必须是2048满足模型

2.

针对fixed embedding还要运行get\_ebd.py得到模型的embedding。

3.

针对fixed embedding运行benchmarking\_main\_EBD.py进行训练和预测

这里考虑的参数，别的就默认就行

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例

4.

针对fine tuning运行benchmarking\_main\_FT.py进行训练和预测

这里考虑的参数，别的就默认就行

parser.add\_argument('--ft\_list', type=list, default=['out\_proj'], help='')

控制lora微调的网络层。

parser.add\_argument('--train\_rate', type=float, default=0.8, help='')

控制训练和测试的比例。

代码在/blue/qsong1/wang.qing/benchmark\_scLLM\_API/Geneformer-finetuing-lora-prompt\_cell\_cls/

数据在/blue/qsong1/wang.qing/benchmark\_dataset\_API/Geneformer/