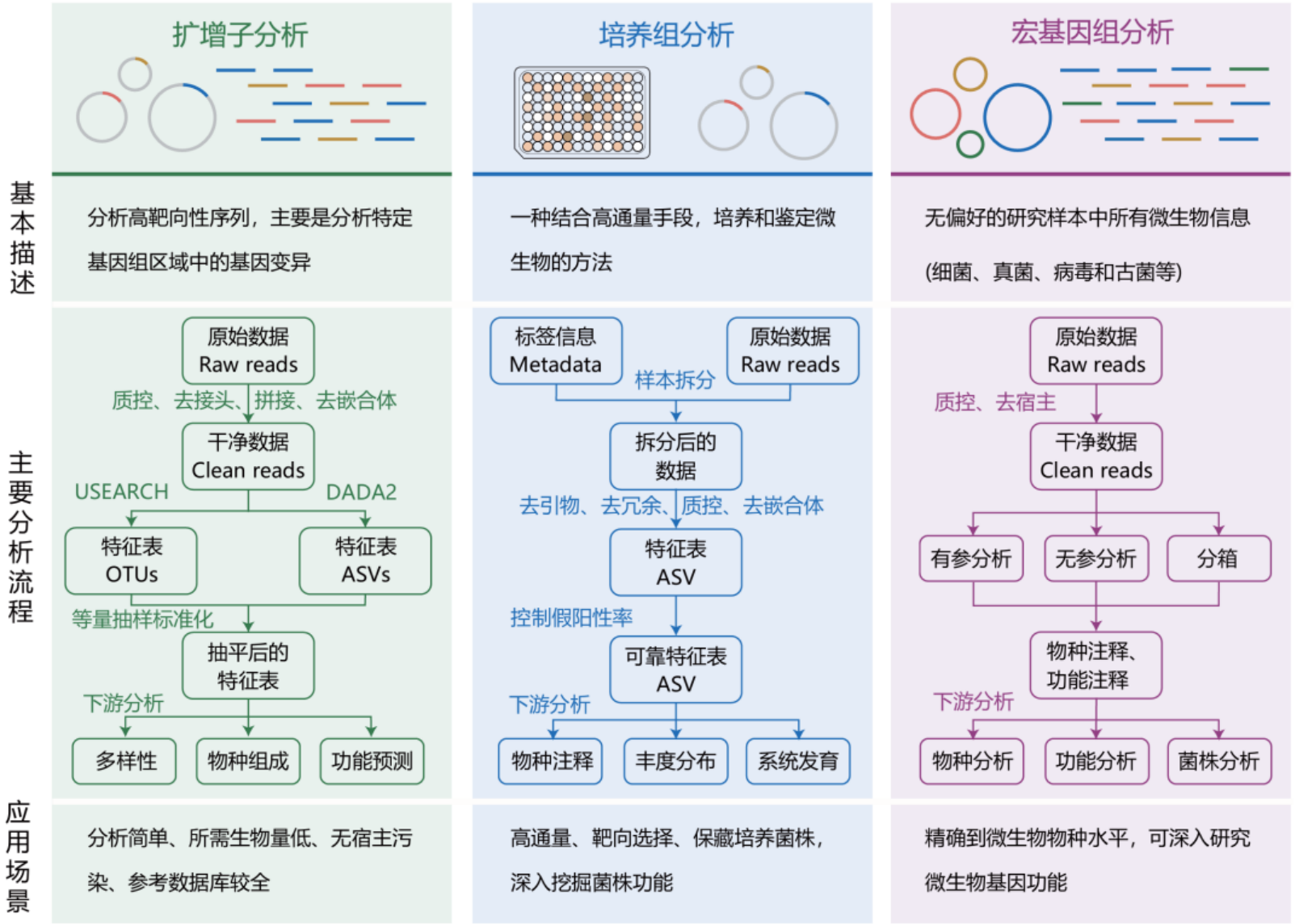
第六章 组学大模型

微生物组概况

微生物组二代高通量测序分析方法和应用



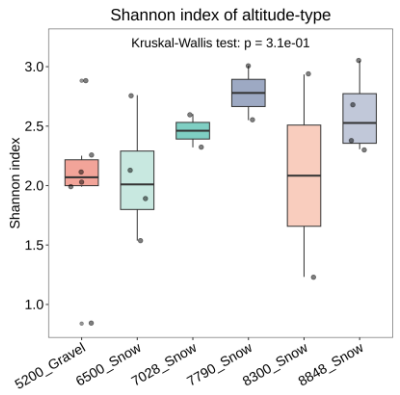
微生物组概况

微生物丰度表及其常见分析方法

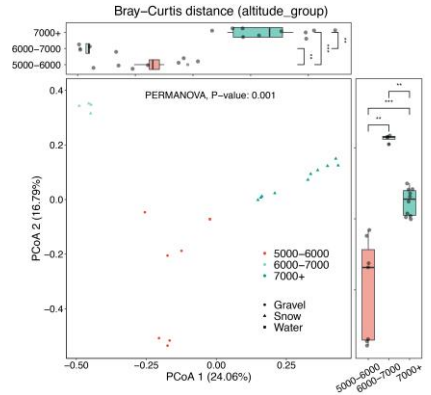
微生物分类

A	B	C
clade_name	QML0000-240606-...	QML0000-240606-...
k__Bacteria|p__Acti...	0	0
k__Bacteria|p__Acti...	0	0
k__Bacteria|p__Acti...	0	0.0001292
k__Bacteria|p__Acti...	0	0
k__Bacteria|p__Acti...	0.0000072	0
k__Bacteria|p__Acti...	0.0000485	0
k__Bacteria|p__Acti...	0.0002707	0
k__Bacteria|p__Acti...	0	0
k__Bacteria|p__Acti...	0	0
k__Bacteria|p__Acti...	0.0006176	0.0009362
k__Bacteria|p__Firm...	0	0
k__Bacteria|p__Firm...	0	0
k__Bacteria|p__Firm...	0	0
k__Bacteria|p__Firm...	0	0
k__Bacteria|p__Firm...	0	0.000005

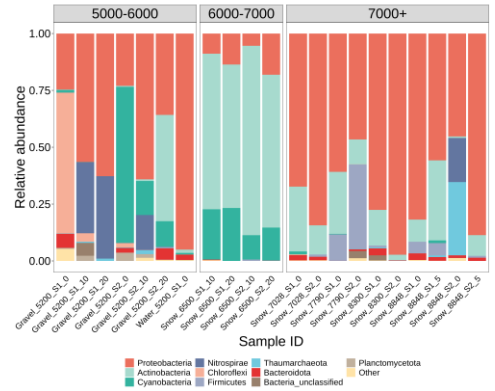
样本名



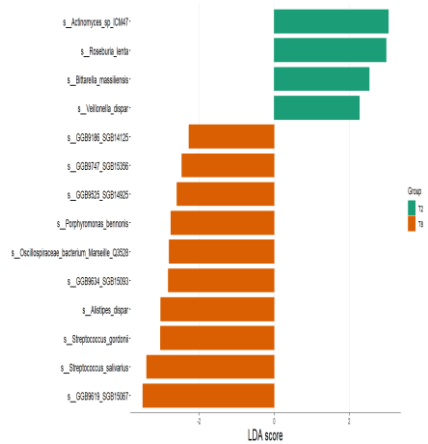
Alpha-diversity



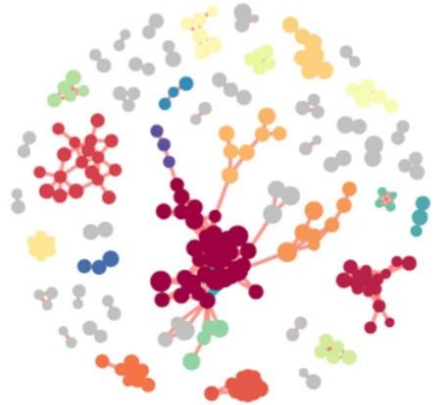
Beta-diversity



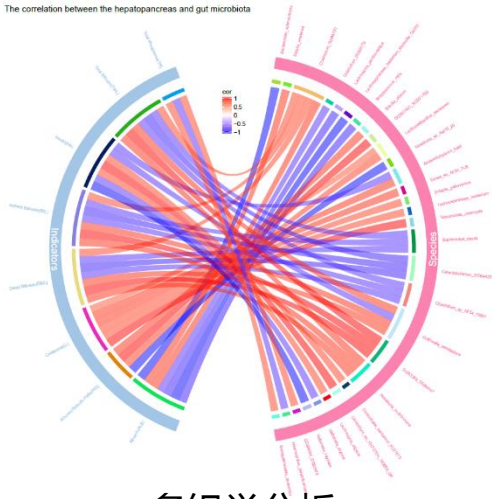
微生物丰度



差异微生物

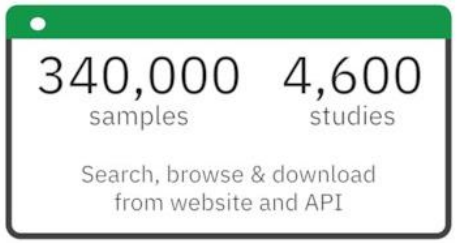


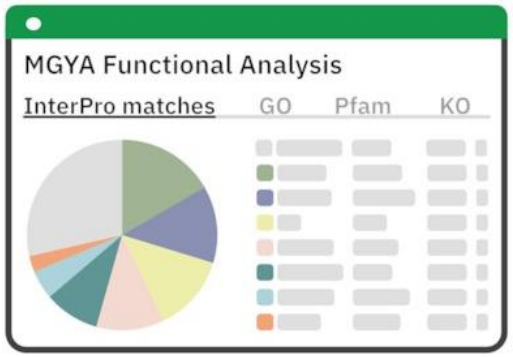
微生物网络



多组学分析

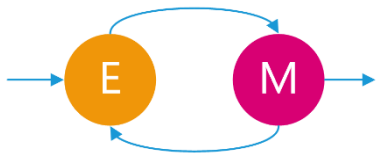
微生物组概况

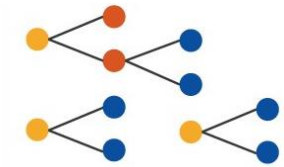




MGnify

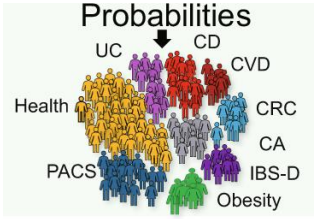
微生物组数据量爆炸增长

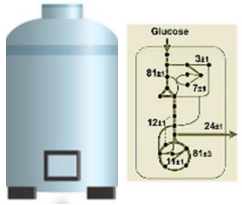




EM算法 **随机森林** **神经网络**

传统机器学习方法





疾病诊断 **污染源监测** **发酵工艺优化**

微生物组数据挖掘

微生物丰度表数据获取

使用aspera工具下载16S rRNA序列数据 ，并使用QIIME2工具包进行分析，了解微生物群落的组成和分类情况，得到微生物的丰度信息

1. 首先，我们需要从EBI SRA数据库下载测序数据。Aspera是一款高性能文件传输工具，适用于大文件下载:

```
#aspera安装：使用conda安装aspera。
cona install -c hcc aspera-cli
#下载：
ascp -v -QT -l 400m -P33001 -k1 \
    -i ~/miniconda3/envs/etc/asperaweb_id_dsa.openssh \
    --user era-fasp \
    --mode recv \
    --host fasp.sra.ebi.ac.uk \
    --file-list path.txt ./
```

参数：
- -v：打开详细模式，显示更多下载信息。
- -QT：关闭加密传输。
- -l 400m：限制传输速度为400Mbps。
- -P33001：使用端口33001。
- -k1：如果传输中断，从中断处继续。
- -i：指定Aspera的密钥文件。可通过 which ascp 查找 asperaweb_id_dsa.openssh 的路径。
- --user era-fasp：使用era-fasp用户进行连接。
- --mode recv：设置为接收模式。
- --host fasp.sra.ebi.ac.uk：指定主机地址。
- --file-list path.txt：指定包含下载文件路径的文本文件。
- `./：下载到当前目录。

path.txt 的示例内容：
/vol1/fastq/ERR428/006/ERR4281016/ERR4281016_1.fastq.gz
/vol1/fastq/ERR428/006/ERR4281016/ERR4281016_2.fastq.gz

微生物丰度表数据获取

2. 安装并激活QIIME2环境

```
# 在开始QIIME2分析之前，需要安装并激活QIIME2环境。
# 命令:
conda env create -n qiime2-amplicon-2024.5 \
    --file https://data.qiime2.org/distro/amplicon/qiime2-amplicon-2024.5-py39-linux-conda.y
conda activate qiime2-amplicon-2024.5
```

参数：

- conda env create -n qiime2-amplicon-2024.5 --file https://data.qiime2.org/distro/amplicon/qiime2-amplicon-2024.5-py39-linux-con: 从QIIME2官方网站下载并创建名为qiime2-amplicon-2024.5的环境。
- conda activate qiiime2-amplicon-2024.5：激活QIIME2环境。

微生物丰度表数据获取

3. 制作一个manifest表

在分析微生物组数据时，创建一个 Manifest 文件是关键的一步。这个文件详细描述了每个样本的文件路径及其对应的测序方向，从而帮助 QIIME2 确认并读取的原始测序数据。Manifest 文件不仅是数据管理的基础，也能保证后续分析的准确性。

```
# 创建一个manifest文件，告诉QIIME2数据的位置和方向。manifest文件是一个CSV文件，包含以下列：
# 示例格式
sample-id,absolute-filepath,direction
ERR4281016,/data2/public/zhanghaohong/5infant_data/ERR4281016_1.fastq.gz,forward
ERR4281016,/data2/public/zhanghaohong/5infant_data/ERR4281016_2.fastq.gz,reverse
```

- sample-id : 样本的唯一标识符。
- absolute-filepath : 原始FASTQ文件的绝对路径。
- direction : 测序读数的方向，forward表示正向读数，reverse表示反向读数。

微生物丰度表数据获取

4. 数据导入

将数据导入QIIME2进行处理。导入步骤将原始FASTQ文件转换为QIIME2能够处理的.qza格式。

```
# 导入数据
qiime tools import \
    --type 'SampleData[PairedEndSequencesWithQuality]' \
    --input-path manifest.txt \
    --output-path paired-end-demux.qza \
    --input-format PairedEndFastqManifestPhred33
```

- qiime tools import : QIIME2的数据导入工具。
- --type 'SampleData[PairedEndSequencesWithQuality]' : 指定导入的数据类型为配对末端序列数据，且包含质量信息。
- --input-path manifest.txt : 指定manifest文件路径。
- --output-path paired-end-demux.qza : 指定导入数据的输出文件路径和名称。
- --input-format PairedEndFastqManifestPhred33 : 指定输入文件格式，Phred33是FASTQ文件的质量编码格式。

微生物丰度表数据获取

5. 拼接序列

将双端测序数据的正向和反向读数进行拼接。通过这一步骤，我们可以获得更完整的序列，从而提高下游分析的准确性。

```
# 拼接序列
# 将双端测序数据的正向和反向读数进行拼接。通过这一步骤，我们可以获得更完整的序列，从而提高下游分析的准确性。
qiime vsearch merge-pairs \
    --i-demultiplexed-seqs paired-end-demux.qza \
    --o-merged-sequences data-merged.qza \
    --o-unmerged-sequences data-unmerged.qza
```

- qiime vsearch merge-pairs : 使用vsearch工具的merge-pairs命令。
- --i-demultiplexed-seqs paired-end-demux.qza : 指定输入的.qza文件，包含配对末端的序列数据。
- --o-merged-sequences data-merged.qza : 指定输出的拼接后序列文件。
- --o-unmerged-sequences data-unmerged.qza : 指定输出未能成功拼接的序列文件。

微生物丰度表数据获取

6. 可视化查看数据质量

```
# 可视化查看数据质量
qiime demux summarize \
    --i-data data-merged.qza \
    --o-visualization data-merged.qzv
```

- qiime demux summarize : 使用demux插件的summarize命令生成数据质量报告。
- --i-data data-merged.qza : 指定输入的拼接后序列文件。
- --o-visualization data-merged.qzv : 指定输出的可视化文件。

微生物丰度表数据获取

7. 去噪

为了确保数据的准确性，去除测序过程中产生的噪音和低质量数据是至关重要的一步。选择合适的去噪方法能够显著提升下游分析的可靠性。

使用Deblur去噪

```
# 使用Deblur去噪
qiime deblur denoise-16S \
    --i-demultiplexed-seqs data-merged.qza \
    --p-trim-length 251 \
    --o-table table.qza \
    --o-representative-sequences rep_set.qza \
    --o-stats stats.qza \
    --p-sample-stats
```

使用DADA2去噪

```
# 使用DADA2去噪
time qiime dada2 denoise-paired \
    --i-demultiplexed-seqs paired-end-demux.qza \
    --o-table table.qza \
    --o-representative-sequences rep-set.qza \
    --o-denoising-stats stats.qza \
    --p-trim-left-f 0 \
    --p-trim-left-r 0 \
    --p-trunc-len-f 250 \
    --p-trunc-len-r 250
```

微生物丰度表数据获取

8. 下载分类器

从QIIME2官网下载分类器文件，分类器用于对特征序列进行物种注释。

```
# 下载分类器
wget -O "gg-13-8-99-515-806-nb-classifier.qza" \
    "https://data.qiime2.org/2021.4/common/gg-13-8-99-515-806-nb-classifier.qza"
```

- wget：下载文件的命令行工具。
- -O "gg-13-8-99-515-806-nb-classifier.qza"：将下载的文件命名为 gg-13-8-99-515-806-nb-classifier.qza 。
- "https://data.qiime2.org/2021.4/common/gg-13-8-99-515-806-nb-classifier.qza"：文件下载地址。

微生物丰度表数据获取

9. 物种注释

使用qiime2官方提供的分类器对特征序列进行物种注释。

```
# 物种注释
qiime feature-classifier classify-sklearn \
    --i-classifier gg-13-8-99-515-806-nb-classifier.qza \
    --i-reads rep-set.qza \
    --o-classification taxonomy.qza
```

- qiime feature-classifier classify-sklearn : 使用feature-classifier插件的classify-sklearn命令进行物种注释。
- --i-classifier gg-13-8-99-515-806-nb-classifier.qza : 指定分类器文件。
- --i-reads rep-set.qza : 指定输入的代表性序列文件。
- --o-classification taxonomy.qza : 指定输出的物种分类文件。

微生物丰度表数据获取

10. 查看物种注释结果

将物种注释结果进行可视化，生成易于理解的报告。

```
# 物种注释结果
qiime metadata tabulate \
    --m-input-file taxonomy.qza \
    --o-visualization taxonomy.qzv
```

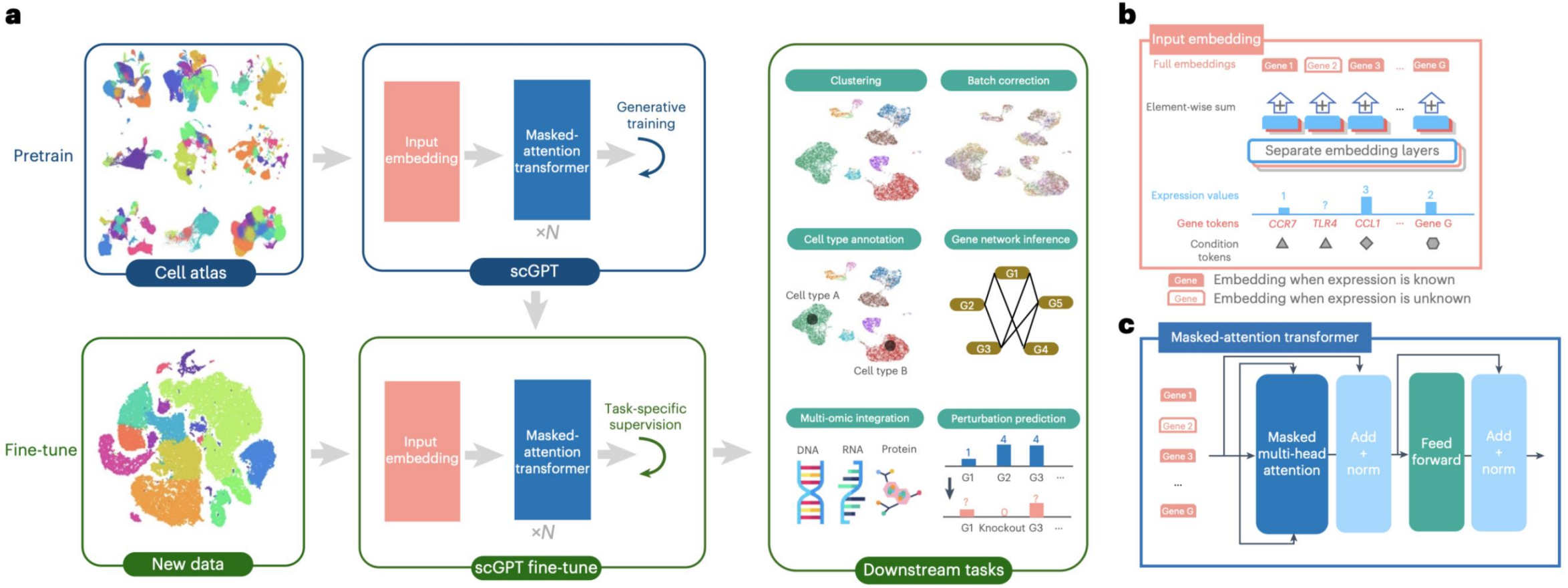
- qiime metadata tabulate : 使用metadata插件的tabulate命令生成可视化报告。
- --m-input-file taxonomy.qza : 指定输入的物种分类文件。
- --o-visualization taxonomy.qzv : 指定输出的可视化文件。

微生物丰度表数据获取

11. 利用biom完成丰度表

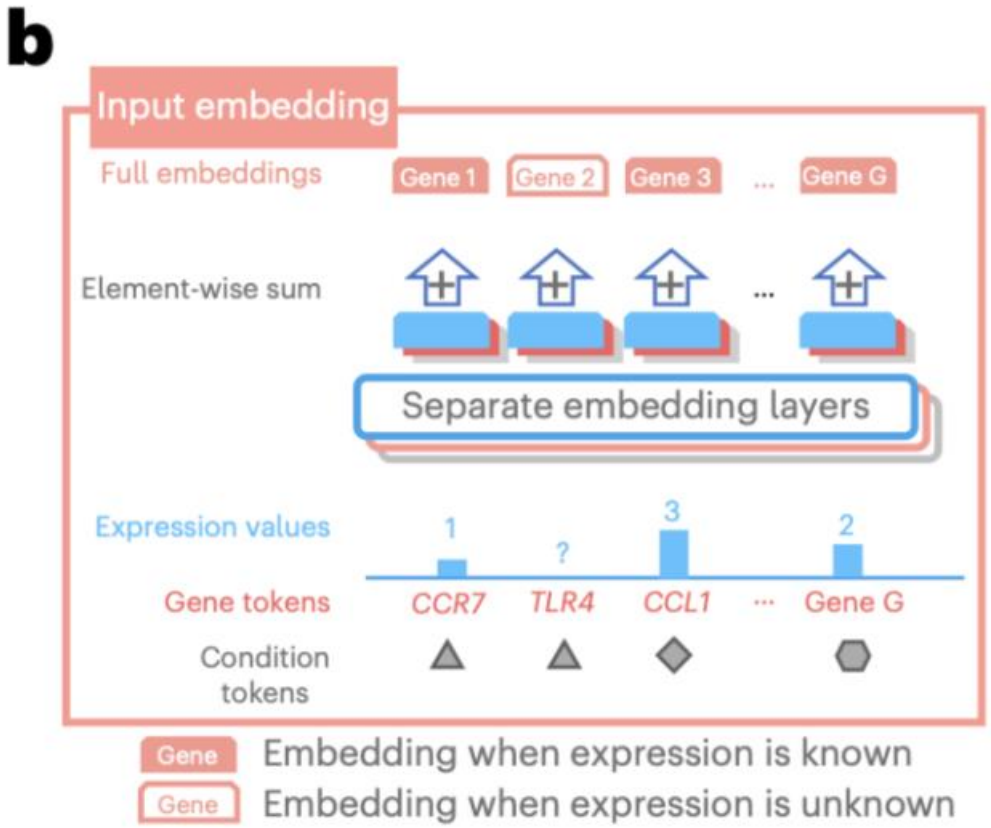
```
# Biom格式转换与物种注释整合
# 导出物种注释
qiime tools export \
    --input-path taxonomy.qza \
    --output-path taxa

# 为了确保物种注释文件能够顺利与 Biom 格式兼容，我们需要对其进行简单的修改。
sed -i -e '1 s/Feature/##Feature/' -e '1 s/Taxon/taxonomy/' taxa/taxonomy.tsv

# 将特征表导出为biom格式。
qiime tools export \
    --input-path table.qza \
    --output-path table_exported

# 将物种注释信息添加到biom表中，生成包含物种注释信息的丰度表。
biom add-metadata \
    -i table_exported/feature-table.biom \
    -o table_exported/feature-table_w_tax.biom \
    --observation-metadata-fp taxa/taxonomy.tsv \
    --sc-separated taxonomy


# 为了便于进一步分析和处理，我们将生成的 Biom 表转换为更为直观的 TSV 格式。
biom convert \
    -i table_exported/feature-table_w_tax.biom \
    -o table_exported/feature-table_w_tax.txt \
    --to-tsv \
    --header-key taxonomy

# 通过上述步骤可将完成16S rRNA基因测序数据的处理和分析，生成微生物群落的分类和丰度表，便于后续研究和分析。
```

单细胞大模型实践（以scGPT为例）



单细胞大模型实践

scGPT数据嵌入方式



$$\boldsymbol{t}_g^{(i)} = \left[\mathrm{id}(g_1^{(i)}), \mathrm{id}(g_2^{(i)}), \ldots, \mathrm{id}(g_M^{(i)})\right],$$

基因嵌入

$$x_j^{(i)} = \begin{cases} k, \text{ if } X_{i,j} > 0 \text{ and } X_{i,j} \in [b_k, b_{k+1}] \\ 0, \text{ if } X_{i,j} = 0. \end{cases}$$

表达量嵌入

$$\boldsymbol{t}_c^{(i)} = \left[t_{c,1}^{(i)}, t_{c,2}^{(i)}, \ldots, t_{c,M}^{(i)}\right]$$

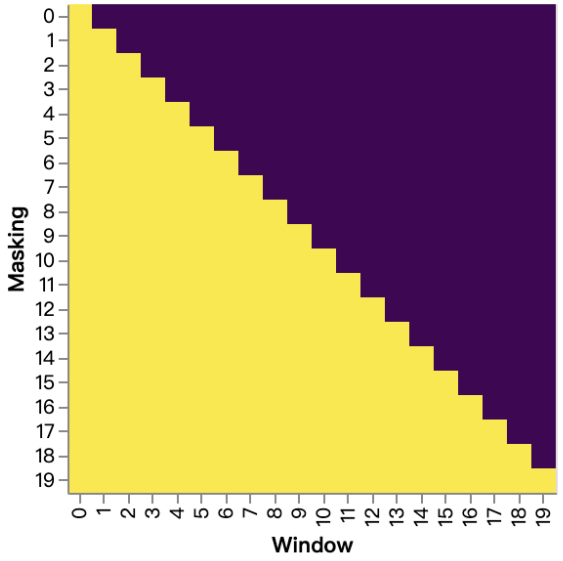
条件嵌入（不同批次、不同组学等）

$$\boldsymbol{h}^{(i)} = \mathrm{emb}_g\left(\boldsymbol{t}_g^{(i)}\right) + \mathrm{emb}_x\left(\boldsymbol{x}^{(i)}\right) + \mathrm{emb}_c\left(\boldsymbol{t}_c^{(i)}\right).$$

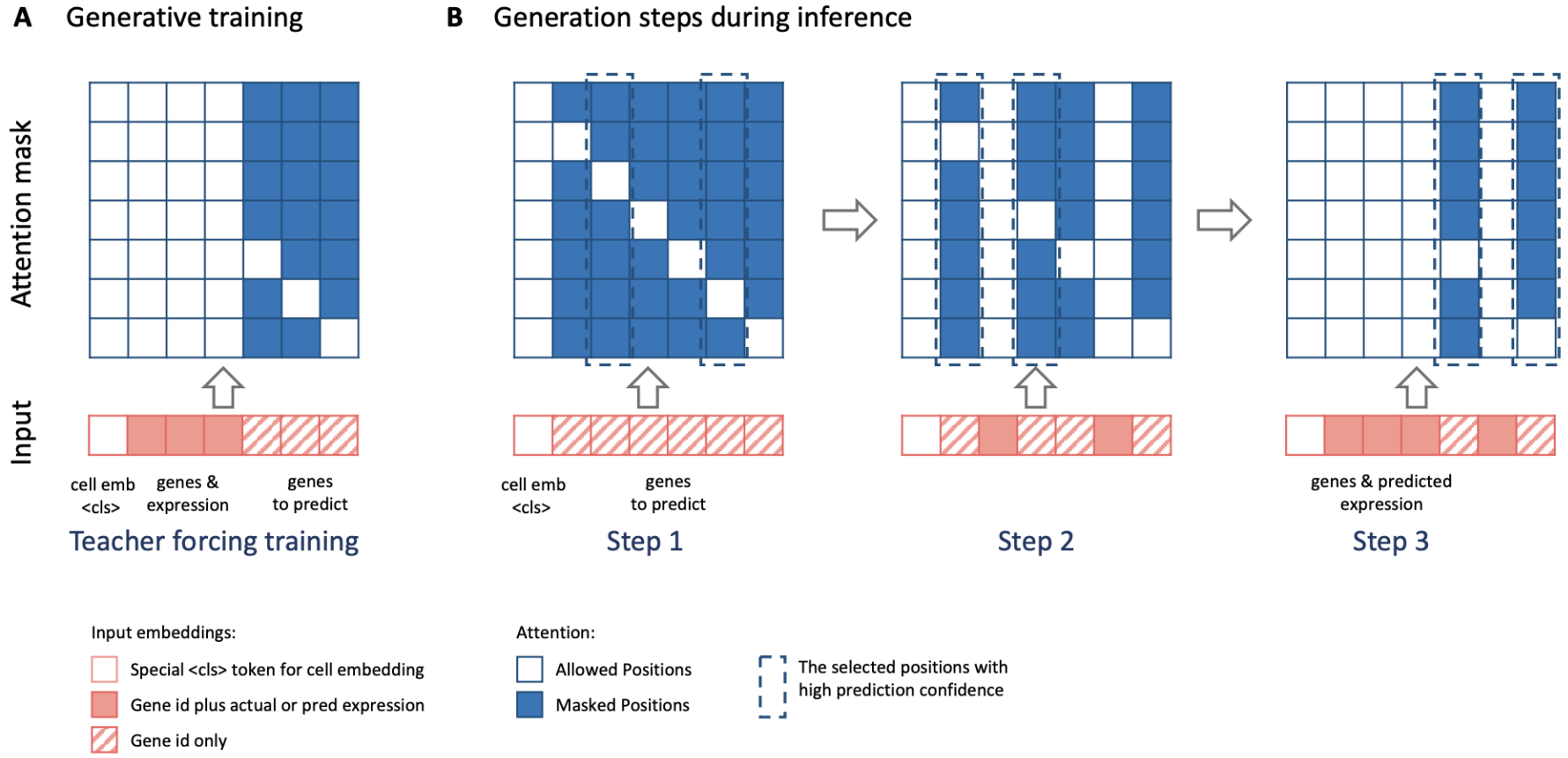
单细胞大模型实践

scGPT预训练

传统因果语言建模采用的下三角注意力掩码



基因表达矩阵无前后排列关系，根据已知表达量基因预测未知表达量基因



单细胞大模型实践

实操：微调scGPT进行细胞注释

使用pip安装scGPT

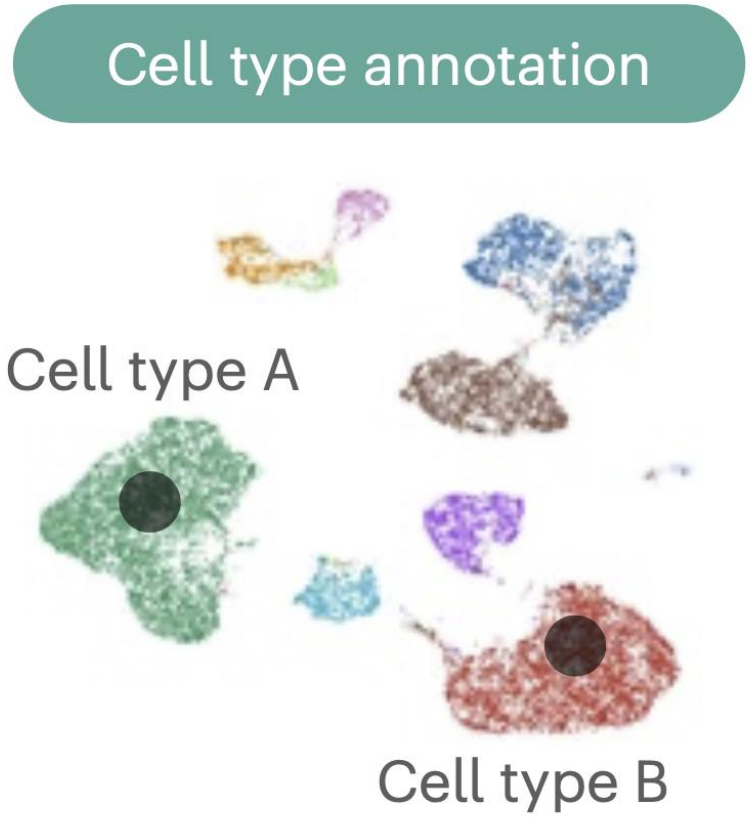
```
pip install scgpt
```

解压实操资源

```
unzip scgpt_resource.zip
```

包含预训练模型与示例数据

```
scgpt_resource
├── ms
│   ├── c_data.h5ad
│   └── filtered_ms_adata.h5ad
└── scGPT_human
    ├── args.json
    ├── best_model.pt
    └── vocab.json
```



根据细胞的基因表达特征进行细胞类型分类

参考：
https://github.com/bowang-lab/scGPT/blob/main/tutorials/Tutorial_Annotation.ipynb

单细胞大模型实践

数据导入与预处理

```
# 数据导入与预处理
import scanpy as sc # 单细胞数据常用处理库
from scgpt.preprocess import Preprocessor
from scgpt.tokenizer.gene_tokenizer import GeneVocab
import json
config = json.load(open('scgpt_resource/scGPT_human/args.json', 'r'))

adata = 'scgpt_resource/ms/c_data.h5ad'
adata = sc.read_h5ad(adata)
adata_test = 'scgpt_resource/ms/filtered_ms_adata.h5ad'
adata_test = sc.read_h5ad(adata_test)
adata.var.set_index(adata.var["gene_name"], inplace=True)
adata_test.var.set_index(adata.var["gene_name"], inplace=True)
adata_test_raw = adata_test.copy()
adata = adata.concatenate(adata_test)
```

单细胞表达数据一般保存为h5格式，使用scanpy库进行快速读写与预处理

单细胞大模型实践

数据导入与预处理

```
# 细胞类型 构建标签
label_col = "Factor Value[inferred cell type - authors labels]"
adata.obs["celltype"] = adata.obs[label_col].astype("category")
adata_test.obs["celltype"] = adata_test.obs[label_col].astype("category")
celltype_id_labels = adata.obs["celltype"].astype("category").cat.codes.values
id2type = dict(enumerate(adata.obs["celltype"].astype("category").cat.categories))
adata.obs["celltype_id"] = celltype_id_labels

# 基因词表    仅保留预训练中存在的基因
vocab = GeneVocab.from_file('scgpt_resource/scGPT_human/vocab.json')
adata.var['id_in_vocab'] = [ 1 if gene in vocab.gene2id else -1 for gene in adata.var_names ]
# 过滤掉不在词表中的基因
adata = adata[:, adata.var['id_in_vocab'] ⩾ 0]
```

单细胞大模型实践

数据导入与预处理

采取与scgpt一致的预处理策略，将每个基因的表达量根据大小分到51个区间中

```
# 预处理
preprocessor = Preprocessor(
    use_key="X",  # 表达矩阵所在键
    normalize_total=1e4, # 每个细胞归一化到1e4
    result_normed_key="X_normed", # 归一化结果存储键
    log1p=False,    # 是否对归一化结果进行log1p变换，示例数据已经是log1p变换后的数据
    result_log1p_key="X_log1p", # log1p结果存储键
    binning=51,  # 分箱数
    result_binned_key="X_binned", # 分箱结果存储键
)
adata_test = adata[adata.obs['str_batch'] == '1']
adata = adata[adata.obs['str_batch'] == '0']
preprocessor(adata)
preprocessor(adata_test)
```

单细胞大模型实践

数据导入与预处理

```
# 创建数据集
from sklearn.model_selection import train_test_split
import numpy as np
import torch
from torch.utils.data import DataLoader, Dataset
from scgpt.tokenizer import tokenize_and_pad_batch
all_counts = adata.layers["X_binned"]
celltypes_labels = adata.obs["celltype_id"].values
train_counts, val_counts, train_labels, val_labels = train_test_split(
    all_counts,
    celltypes_labels,
    test_size=0.1,
    random_state=42,
    stratify=celltypes_labels
)
gene_ids = np.array(vocab(adata.var['gene_name'].tolist()), dtype=int)
```

根据不同的细胞类型进行层次采样，划分训练集与验证集

单细胞大模型实践

数据导入与预处理

```
tokenized_train = tokenize_and_pad_batch(
    train_counts,
    gene_ids,
    max_len=config['max_seq_len'],
    vocab=vocab,
    pad_token=config['pad_token'],
    pad_value=config['pad_value'],
    append_cls=True,  # append <cls> token at the beginning
)
tokenized_valid = tokenize_and_pad_batch(
    val_counts,
    gene_ids,
    max_len=config['max_seq_len'],
    vocab=vocab,
    pad_token=config['pad_token'],
    pad_value=config['pad_value'],
    append_cls=True,  # append <cls> token at the beginning
    include_zero_gene=False,
)
```

将基因根据词表进行标量映射（tokenize），并将长度不足的样本使用<pad>补齐，在样本的最开始加上<cls>进行细胞全局信息的捕捉

单细胞大模型实践

数据导入与预处理

```
class SeqDataset(Dataset):
    def __init__(self, data):
        self.data = data

    def __len__(self):
        return self.data["gene_ids"].shape[0]

    def __getitem__(self, idx):
        return {k: v[idx] for k, v in self.data.items()}

train_dataset = SeqDataset({
    'gene_ids': tokenized_train['genes'],
    'values': tokenized_train['values'],
    'target_values': tokenized_train['values'],
    'batch_labels': torch.zeros(len(train_labels), dtype=torch.long),
    'celltype_labels': torch.tensor(train_labels, dtype=torch.long)
})

val_dataset = SeqDataset({
    'gene_ids': tokenized_valid['genes'],
    'values': tokenized_valid['values'],
    'target_values': tokenized_valid['values'],
    'batch_labels': torch.zeros(len(val_labels), dtype=torch.long),
    'celltype_labels': torch.tensor(val_labels, dtype=torch.long)
})
train_loader = DataLoader(train_dataset, batch_size=config['batch_size'], shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=config['batch_size'], shuffle=False)
```

构建数据集，匹配模型接收的数据格式：

gene_id: 经过tokenize后的基因

values: 对应基因的表达分箱

target_values: 需预测的表量，仅在预训练中需要

batch_labels: 批次id，处理多批次数据时需要

celltype_labels: 细胞类型标签

导入预训练模型

使用scGPT_human作为基础模型

```
# 导入预训练模型
from scgpt.model import TransformerModel

model = TransformerModel(ntoken=len(vocab),
                         d_model=config['embsize'],
                         nhead=config['nheads'],
                         d_hid=config['d_hid'],
                         nlayers=config['nlayers'],
                         n_cls=len(adata.obs['celltype'].unique()),
                         vocab=vocab,
                         dropout=config['dropout'],
                         pad_token=config['pad_token'],
                         pad_value=config['pad_value'])
model_file = 'scgpt_resource/scGPT_human/best_model.pt'
model.load_state_dict(torch.load(model_file), strict=False)
```

strict=False 避免分类层参数无法匹配产生冲突

单细胞大模型实践

预训练模型微调

```
# 细胞类型注释微调
import torch.optim as optim
import torch.nn as nn
from sklearn.metrics import accuracy_score
import numpy as np
import os
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4)
num_epochs = 5
best_val_acc = 0.0
save_path = './celltype_annotation_model'
os.makedirs(save_path, exist_ok=True)
```

使用交叉熵作为分类任务的损失函数

最大训练轮数

初始化最佳acc

单细胞大模型实践

预训练模型微调

```
for epoch in range(num_epochs):
    model.train()
    train_losses = []
    for batch in train_loader:
        gene_ids = batch['gene_ids'].to(device)
        values = batch['values'].to(device)
        celltype_labels = batch['celltype_labels'].to(device)

        optimizer.zero_grad()  清空梯度
        outputs = model(gene_ids,
                        values,
                        src_key_padding_mask=gene_ids.eq(vocab[config['pad_token']]),
                        CLS=True)
        loss = criterion(outputs['cls_output'], celltype_labels)
        loss.backward()  反向传播计算梯度
        optimizer.step()  更新模型参数
        train_losses.append(loss.item())

    avg_train_loss = np.mean(train_losses)
```

单细胞大模型实践

预训练模型微调

```
model.eval()
val_losses = []
all_preds = []
all_labels = []
with torch.no_grad():
    for batch in val_loader:
        gene_ids = batch['gene_ids'].to(device)
        values = batch['values'].to(device)
        celltype_labels = batch['celltype_labels'].to(device)
        src_key_padding_mask = gene_ids.eq(vocab[config['pad_token']])
        outputs = model(gene_ids,
                        values,
                        src_key_padding_mask=src_key_padding_mask,
                        CLS=True)
        loss = criterion(outputs['cls_output'], celltype_labels)
        val_losses.append(loss.item())

        preds = torch.argmax(outputs['cls_output'], dim=1).cpu().numpy()
        all_preds.extend(preds)
        all_labels.extend(celltype_labels.cpu().numpy())

avg_val_loss = np.mean(val_losses)
val_acc = accuracy_score(all_labels, all_preds)
```

在验证集上推理，并计算模型在验证集上的准确性表现

```
    print(f"Epoch {epoch+1}/{num_epochs}, \
          Train Loss: {avg_train_loss:.4f}, \
          Val Loss: {avg_val_loss:.4f}, \
          Val Acc: {val_acc:.4f}")
```

每一轮训练后打印训练损失，验证损失与验证准确性

```
if val_acc > best_val_acc:
    best_val_acc = val_acc
    torch.save(model.state_dict(), os.path.join(save_path, 'best_model.pt'))
    print(f"New best model saved with Val Acc: {best_val_acc:.4f}")
```

根据验证集的表现更新最佳验证准确性，并保存在验证集上表现最好的模型

单细胞大模型实践

测试集评估

```
# 构建测试集
test_counts = adata_test.layers["X_binned"]
test_labels = adata_test.obs["celltype_id"].values
gene_ids = np.array(vocab(adata_test.var['gene_name'].tolist()), dtype=int)
tokenized_test = tokenize_and_pad_batch(
    test_counts,
    gene_ids,
    max_len=config['max_seq_len'],
    vocab=vocab,
    pad_token=config['pad_token'],
    pad_value=config['pad_value'],
    append_cls=True,  # append <cls> token at the beginning
    include_zero_gene=False,
)
test_dataset = SeqDataset({
    'gene_ids': tokenized_test['genes'],
    'values': tokenized_test['values'],
    'target_values': tokenized_test['values'],
    'batch_labels': torch.zeros(len(test_labels), dtype=torch.long),
    'celltype_labels': torch.tensor(test_labels, dtype=torch.long)
})
test_loader = DataLoader(test_dataset, batch_size=config['batch_size'], shuffle=False)
```

单细胞大模型实践

测试集评估

```
model.load_state_dict(torch.load(os.path.join(save_path, 'best_model.pt')))
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
    for batch in test_loader:
        gene_ids = batch['gene_ids'].to(device)
        values = batch['values'].to(device)
        celltype_labels = batch['celltype_labels'].to(device)
        src_key_padding_mask = gene_ids.eq(vocab[config['pad_token']])
        outputs = model(gene_ids,
                        values,
                        src_key_padding_mask=src_key_padding_mask,
                        CLS=True)
        preds = torch.argmax(outputs['cls_output'], dim=1).cpu().numpy()
        all_preds.extend(preds)
        all_labels.extend(celltype_labels.cpu().numpy())
test_acc = accuracy_score(all_labels, all_preds)
print(f"Test Acc: {test_acc:.4f}")
```

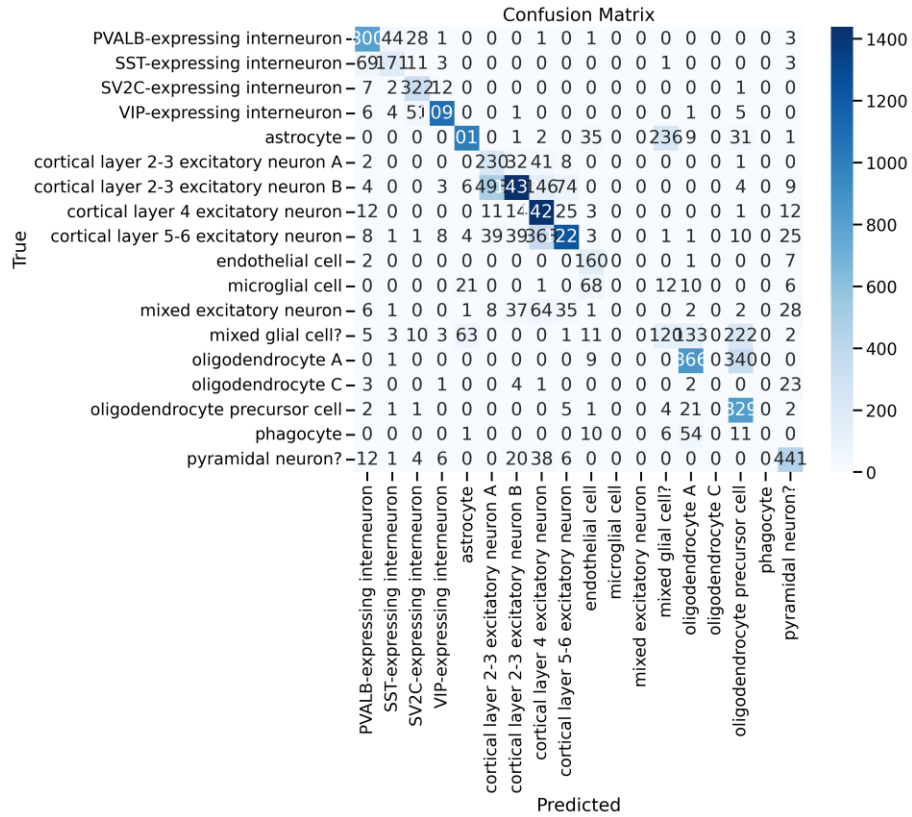
在关闭梯度计算的条件下对每个测试集样本进行推理

单细胞大模型实践

测试集评估

混淆矩阵可视化

```python
# 混淆矩阵
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=[id2type[i] for i in range(len(id2type))],
            yticklabels=[id2type[i] for i in range(len(id2type))])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.savefig('confusion_matrix.png', dpi=300, bbox_inches='tight')
plt.close()
```

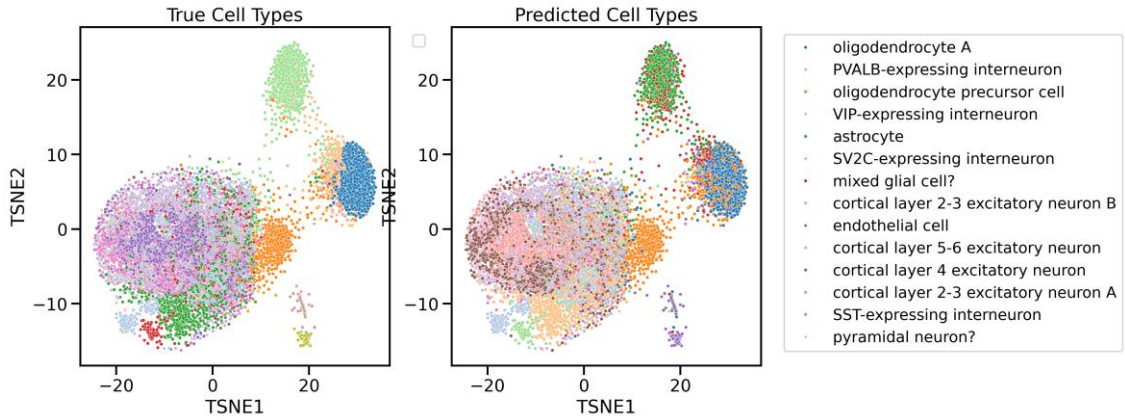


单细胞大模型实践

测试集评估

TSNE降维可视化

```
from sklearn.manifold import TSNE
import pandas as pd
adata_test_raw.obs['true_celltype'] = [id2type[i] for i in all_labels]
adata_test_raw.obs['pred_celltype'] = [id2type[i] for i in all_preds]
X = adata_test_raw.X.toarray()
tsne = TSNE(n_components=2, random_state=42, n_jobs=-1)
X_tsne = tsne.fit_transform(X)
true_df = pd.DataFrame(X_tsne, columns=['TSNE1', 'TSNE2'])
true_df['celltype'] = adata_test_raw.obs['true_celltype'].values
pred_df = pd.DataFrame(X_tsne, columns=['TSNE1', 'TSNE2'])
pred_df['celltype'] = adata_test_raw.obs['pred_celltype'].values
plot_df = pd.concat([true_df, pred_df], axis=0)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.scatterplot(data=true_df, x='TSNE1', y='TSNE2', hue='celltype', palette='tab20', s=10, legend=False)
plt.title('True Cell Types')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize='small')
plt.subplot(1, 2, 2)
sns.scatterplot(data=pred_df, x='TSNE1', y='TSNE2', hue='celltype', palette='tab20', s=10)
plt.title('Predicted Cell Types')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize='small')
plt.savefig('tsne_celltype.png', dpi=300, bbox_inches='tight')
plt.close()
```



课堂作业

- 1. 思考题：组学大模型（如scGPT）相比传统组学数据分析方法有哪些优势？请从数据嵌入方式、预训练策略以及在细胞类型注释等下游任务中的表现等方面进行分析
- 2. 实践题：选择以下任一任务完成：
  - 微生物组分析：使用QIIME2分析16S rRNA测序数据，完成从原始数据下载到物种注释的全流程，并生成微生物丰度表和多样性分析报告
  - 单细胞分析：使用scGPT对单细胞表达数据进行细胞类型注释，完成数据预处理、模型微调和结果评估，提交混淆矩阵和t-SNE可视化结果

课外学习

- 参考资料：
  - 1. scGPT: toward building a foundation model for single-cell multi-omics using generative AI (Nature Methods, 2024)
  - 2. Reproducible, interactive, scalable and extensible microbiome data science using QIIME 2 (Nature Biotechnology, 2019)
  - 3. QIIME2: https://qiime2.org/
  - 4. 单细胞Scanpy流程学习和整理