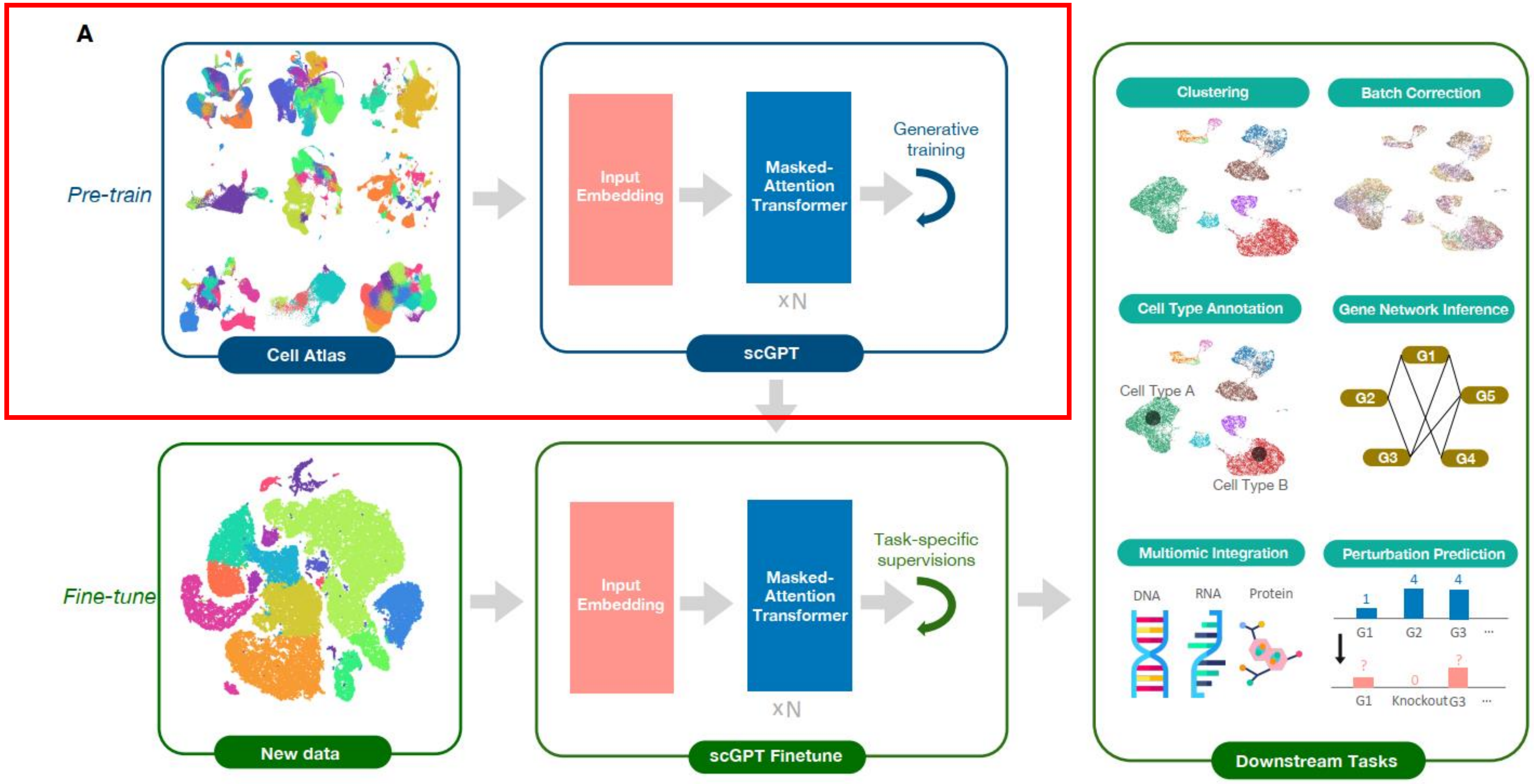
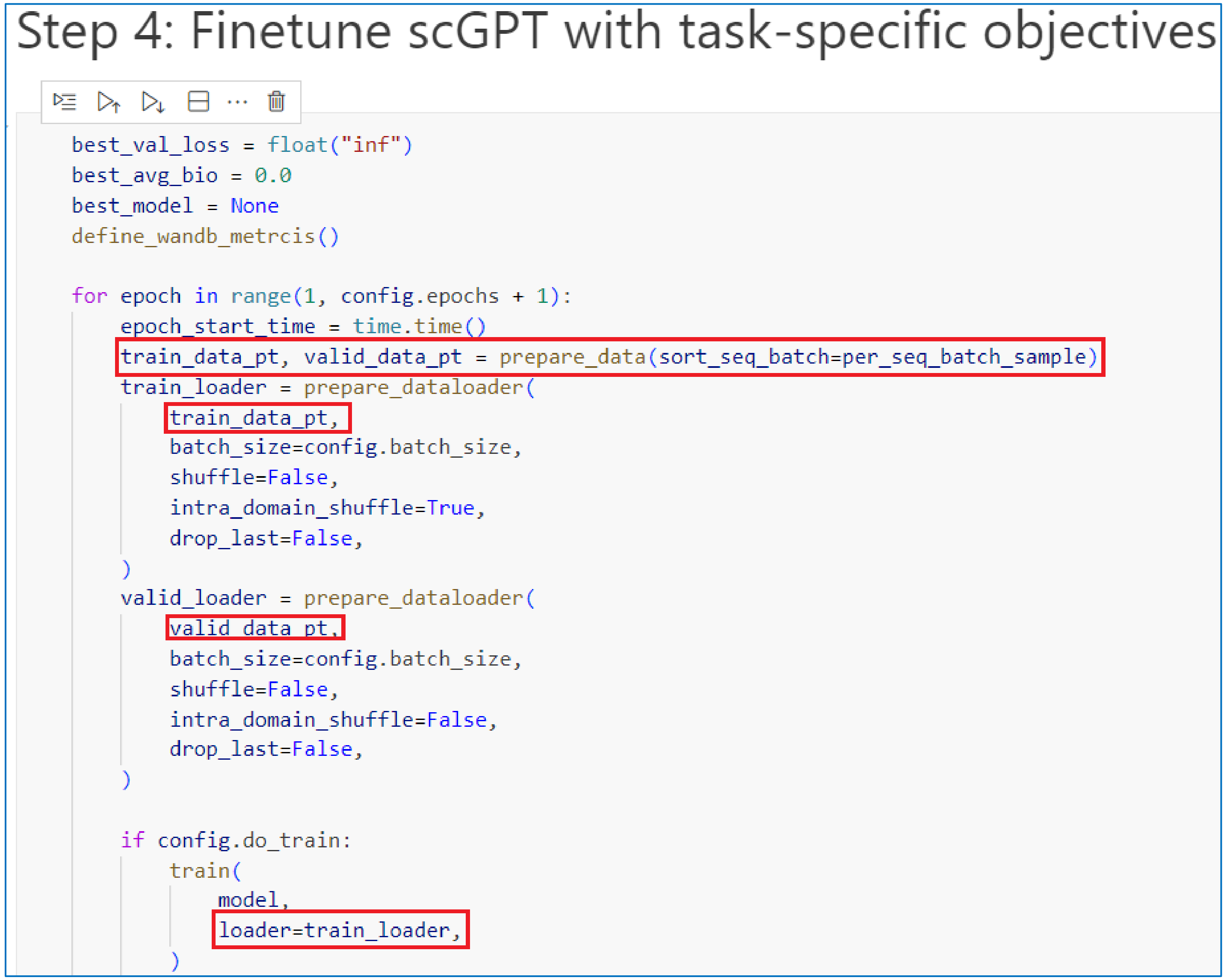
Overview of scGPT



Cui, H., Wang, C., Maan, H., Pang, K., Luo, F., & Wang, B. (2023). *scGPT: Towards Building a Foundation Model for Single-Cell Multi-omics Using Generative AI* [Preprint]. Bioinformatics. https://doi.org/10.1101/2023.04.30.538439

To build trainer from the code of finetuning

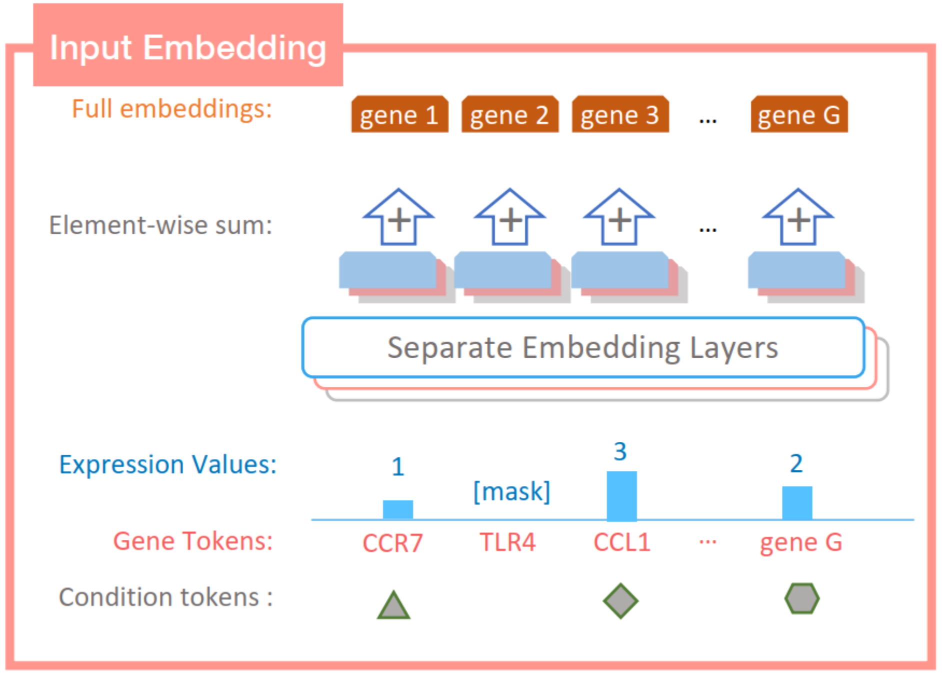


Step 4: Finetune scGPT with task-specific objectives

```
best_val_loss = float("inf")
best_avg_bio = 0.0
best_model = None
define_wandb_metrcis()

for epoch in range(1, config.epochs + 1):
    epoch_start_time = time.time()
    train_data_pt, valid_data_pt = prepare_data(sort_seq_batch=per_seq_batch_sample)
    train_loader = prepare_dataloader(
        train_data_pt,
        batch_size=config.batch_size,
        shuffle=False,
        intra_domain_shuffle=True,
        drop_last=False,
    )
    valid_loader = prepare_dataloader(
        valid_data_pt,
        batch_size=config.batch_size,
        shuffle=False,
        intra_domain_shuffle=False,
        drop_last=False,
    )

    if config.do_train:
        train(
            model,
            loader=train_loader,
        )
```

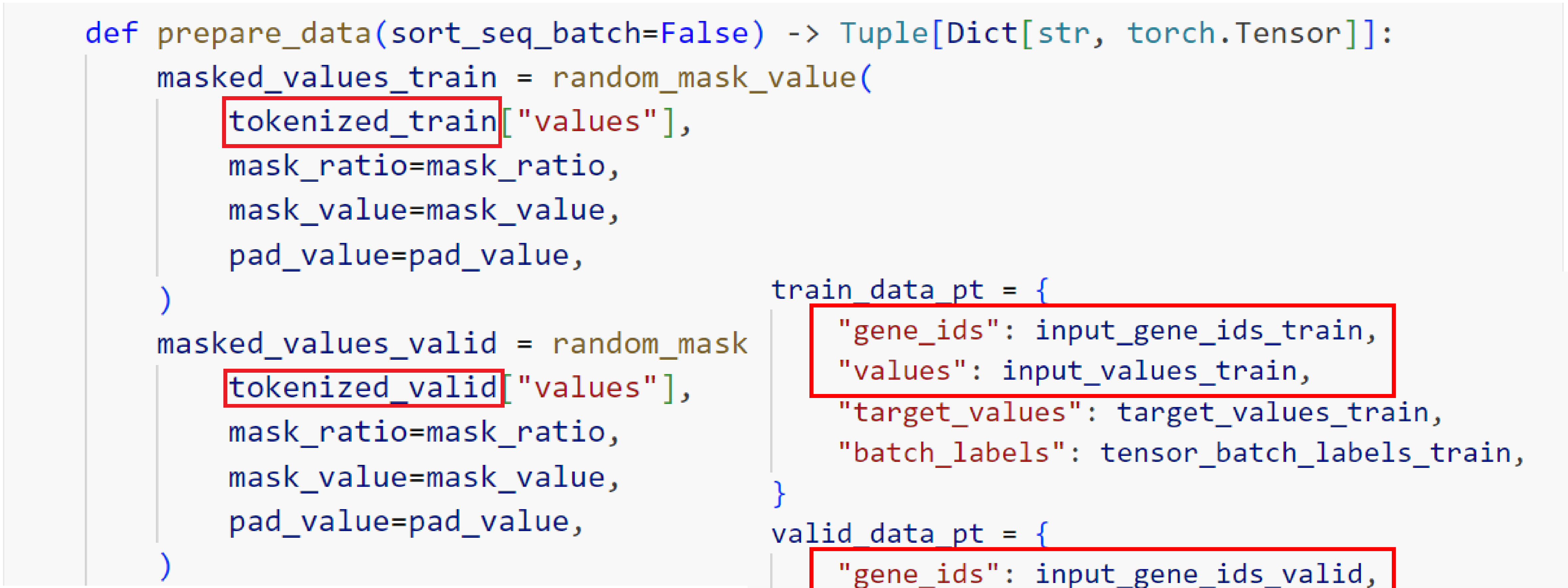
Main loop:

1. prepare_data(): To prepare input embedding for scGPT
2. prepare_dataloader()
3. train
4. valid

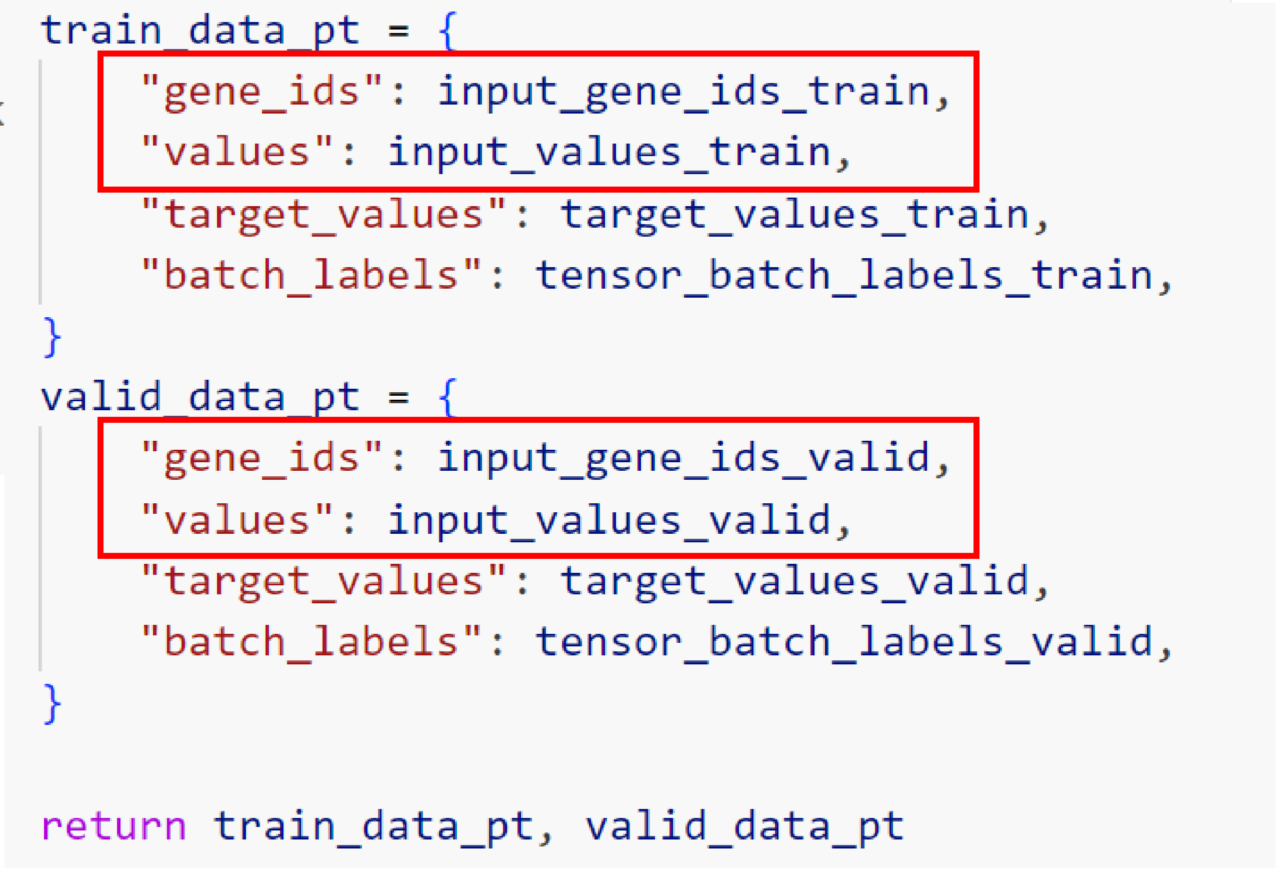


(Cui et al., 2023)

prepare_data()



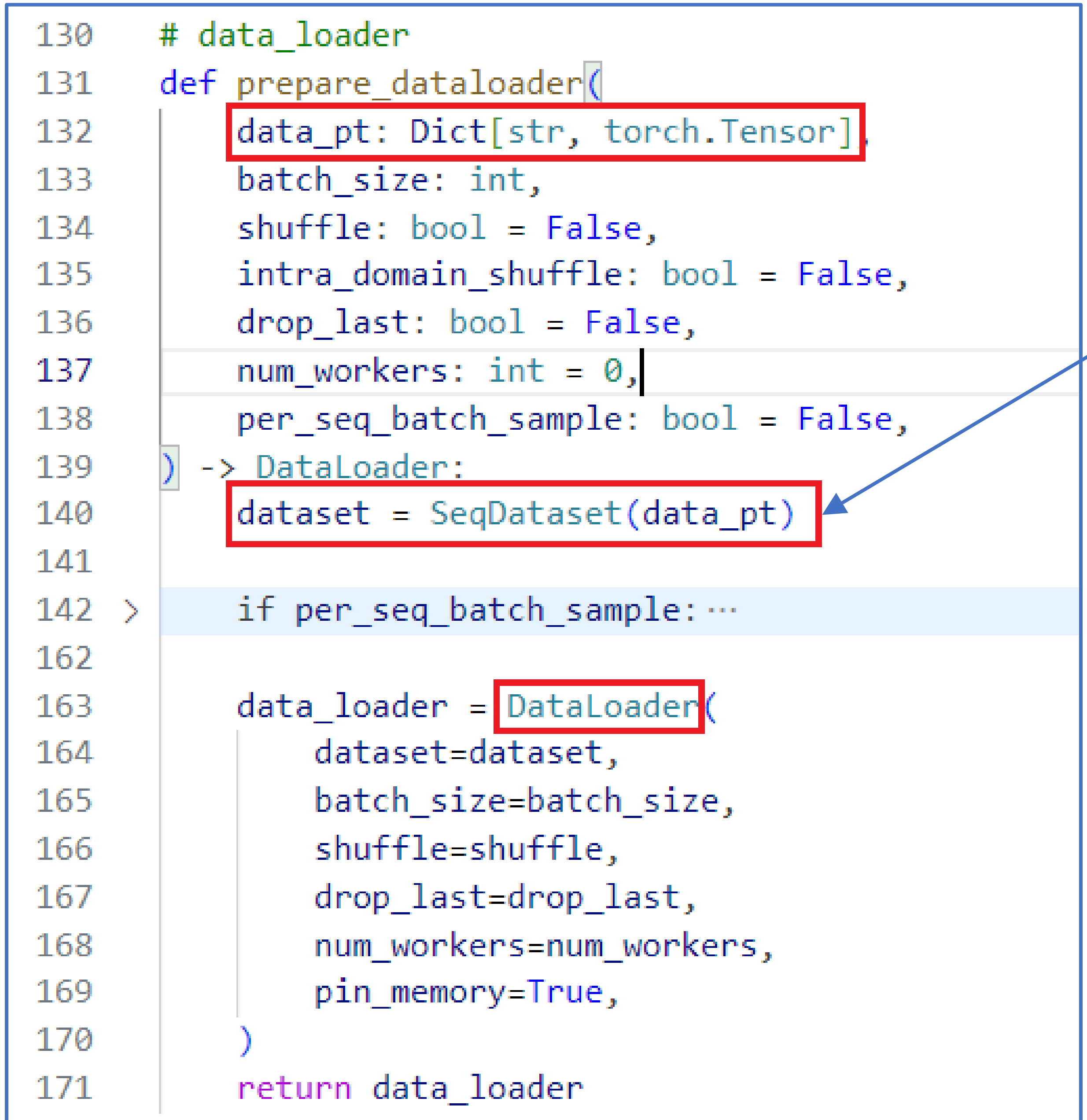
```
def prepare_data(sort_seq_batch=False) -> Tuple[Dict[str, torch.Tensor]]:
    masked_values_train = random_mask_value(
        tokenized_train["values"],
        mask_ratio=mask_ratio,
        mask_value=mask_value,
        pad_value=pad_value,
    )
    masked_values_valid = random_mask
        tokenized_valid["values"],
        mask_ratio=mask_ratio,
        mask_value=mask_value,
        pad_value=pad_value,
    )
```



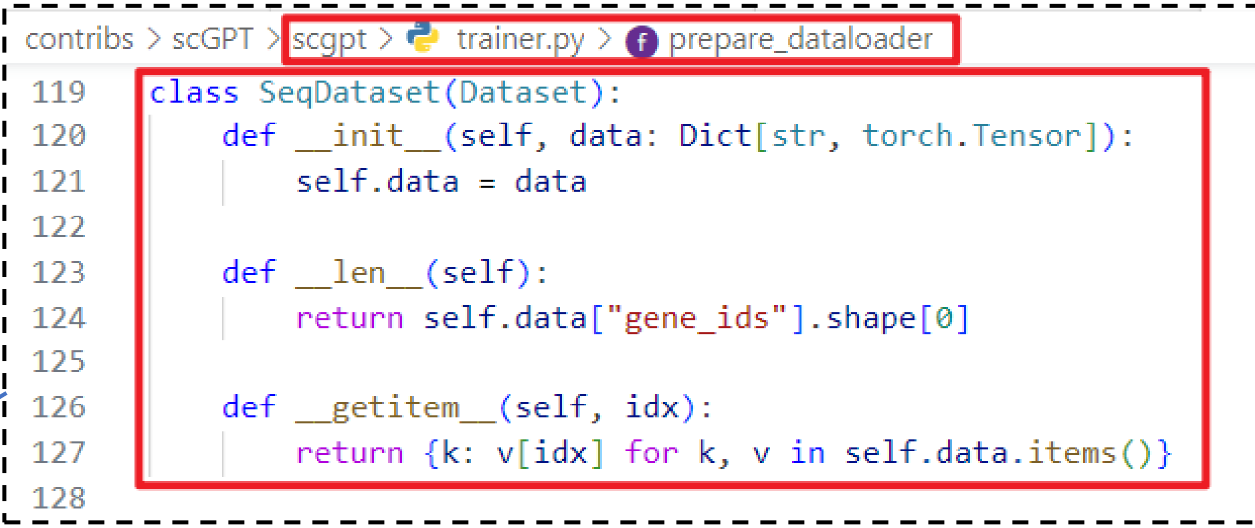
```
train_data_pt = {
    "gene_ids": input_gene_ids_train,
    "values": input_values_train,
    "target_values": target_values_train,
    "batch_labels": tensor_batch_labels_train,
}
valid_data_pt = {
    "gene_ids": input_gene_ids_valid,
    "values": input_values_valid,
    "target_values": target_values_valid,
    "batch_labels": tensor_batch_labels_valid,
}

return train_data_pt, valid_data_pt
```

Getting the tokenized data, and returning the data whose values are random masked .

prepare_dataloader()



```python
# data_loader
def prepare_dataloader(
    data_pt: Dict[str, torch.Tensor],
    batch_size: int,
    shuffle: bool = False,
    intra_domain_shuffle: bool = False,
    drop_last: bool = False,
    num_workers: int = 0,
    per_seq_batch_sample: bool = False,
) -> DataLoader:
    dataset = SeqDataset(data_pt)

    if per_seq_batch_sample: ···

    data_loader = DataLoader(
        dataset=dataset,
        batch_size=batch_size,
        shuffle=shuffle,
        drop_last=drop_last,
        num_workers=num_workers,
        pin_memory=True,
    )
    return data_loader
```



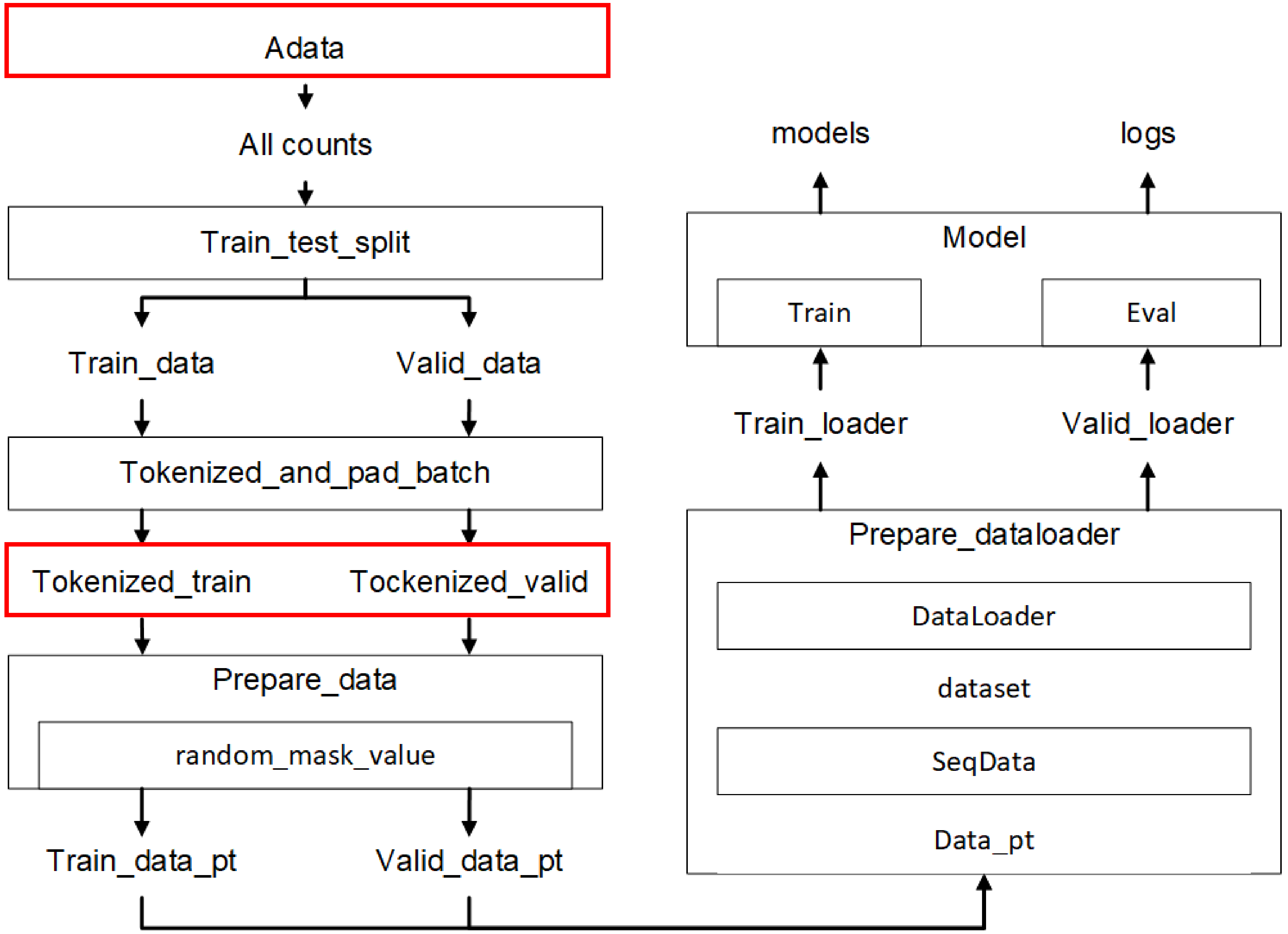
contribs > scGPT > scgpt > trainer.py > prepare_dataloader

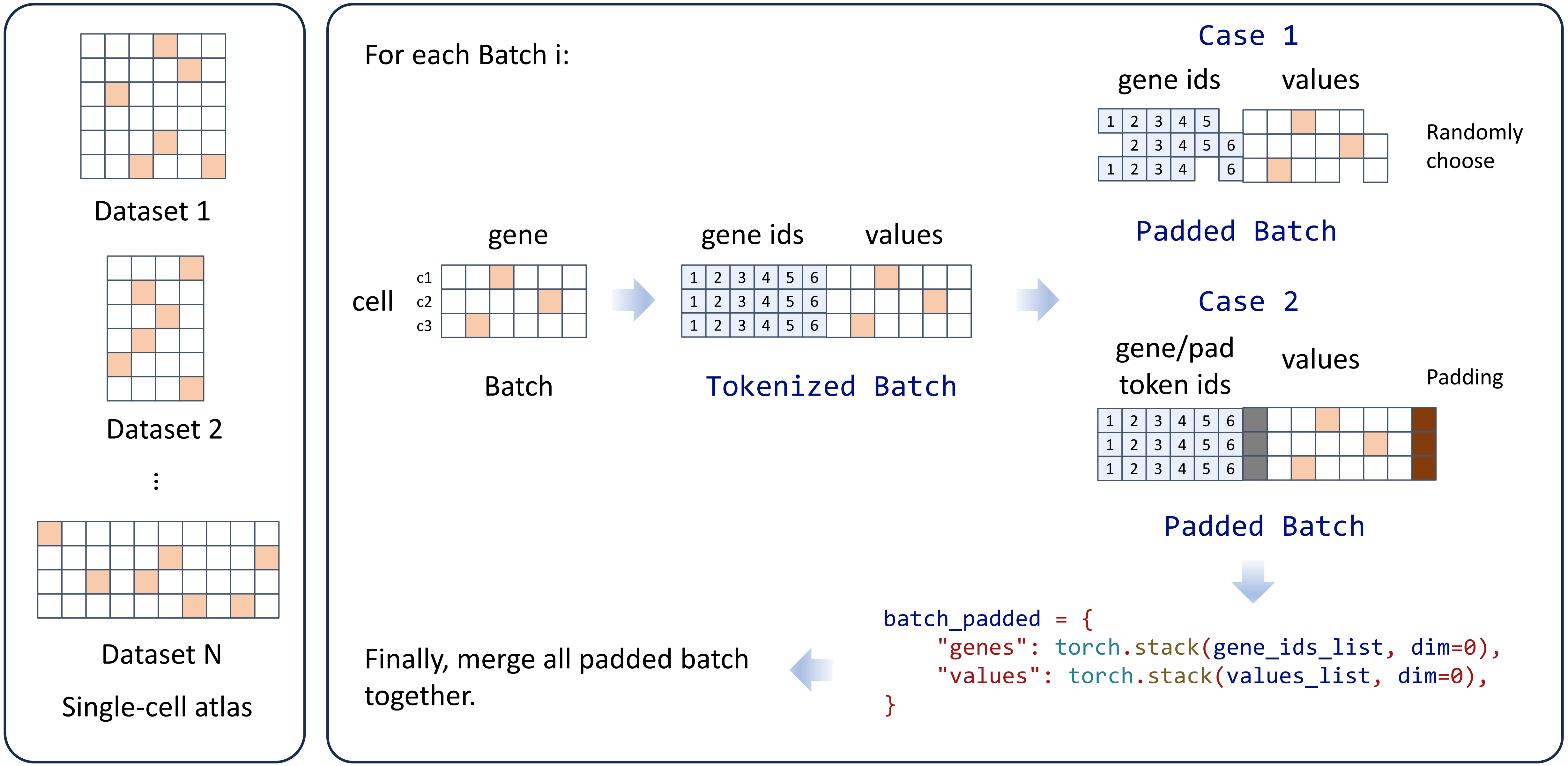
```python
class SeqDataset(Dataset):
    def __init__(self, data: Dict[str, torch.Tensor]):
        self.data = data

    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()}

```

Getting the prepared data (embedding) and returning a pytorch Dataloader for training scGPT.

Solution 1: To prepare a whole anndata. (Out of memory)

Solution 2: To prepare the tokenized data from a series of anndata.





```
batch_padded = {
    "genes": torch.stack(gene_ids_list, dim=0),
    "values": torch.stack(values_list, dim=0),
}
```

Finally, merge all padded batch together.

Tokenize and pad batch

Pretraining on the CRE data:
Small:
notebook/pretrain_all_in_one_0801.ipynb
https://wandb.ai/qiliu-ghddi/Pretraining%20scGPT/runs/h2wj495n

0.7M Cells:
notebook/pretrain_all_in_one_0802.ipynb
https://wandb.ai/qiliu-ghddi/Pretraining%20scGPT%20on%20the%20cre329_tokenized_merged_numds10/runs/zdaqhx27

To build the cell atlas from the cell x gene census

```
- The workflow is:

  1. Build the cell index files based on
query
  2. download the dataset in
partitions(chunks)
  3. transform the `AnnData` into `scb`

- `query_list.txt` records the query for
retrieving the cell atlas from the cell x
gene census.
- `build_soma_idx.sh` builds index for all
all healthy human cells collected by the
census.
- `download_partition.sh` downloads the
dataset in partitions(chunks) with the
given index file, max partition size is
200000 cells per file by default.
```

Python scripts

1. build_soma_idx.py: This script is used to retrieve cell soma ids from cellxgene census
2. build_large_scale_data.py: build large-scale data in scBank format from a group of AnnData objects
3. data_config.py: 很多VALUE_FILTER
4. download_partitions.py: Download a given partition cell of the query in h5ad
5. expand_gene_list.py: To create new_gene_list (default_census_vocab)
6. process_allcounts.py: load or make the dataset w/ <cls> appended at the beginning

build_large_scale_data.py

```
main_table_key = "counts"
token_col = "feature_name"
for f in files:
   adata = sc.read(f, cache=True)
   adata = preprocess(adata, main_table_key, N = args.N)
   print(f“read {adata.shape} valid data from {f.name}”)
    # BUILD SCBANK DATA
    db = scbank.DataBank.from_anndata(
        adata,
        vocab=vocab,
        to=output_dir / f"{f.stem}.scb",
        main_table_key=main_table_key,
        token_col=token_col,
        immediate_save=False,
    )
    db.meta_info.on_disk_format = "parquet"
    # sync all to disk
    db.sync()
```

对于每个preprocessed adata，将其转成scb (.parquet) 格式的文件，单独保存

process_allcounts.py

```python
# load or make the dataset w/ <cls> appended at the beginning
cls_prefix_datatable = Path(args.data_source) / "cls_prefix_data.parquet"
if not cls_prefix_datatable.exists():
    print("preparing cls prefix dataset")
    raw_dataset = load_dataset(
        "parquet",
        data_files=parquet_files,
        split="train",
        cache_dir=str(cache_dir),)
    raw_dataset = _map_append_cls(raw_dataset)
    raw_dataset.to_parquet(str(cls_prefix_datatable))

raw_dataset = load_dataset(
    "parquet",
    data_files=str(cls_prefix_datatable),
    split="train",
    cache_dir=str(cache_dir),
)
```

对每个scb (.parquet) 格式的文件，每个都进行tokenize & padding，借助hugging face datasets，返回PyTorch Dataset类对象

TODO

[ ] To create a new Dataset not loading to the memory

`build_large_scale_data.py`, 将`anndata`处理成 `parquet` 格式， 随后`process_allcounts.py`, 将所有的`.Parquet` 最后创建成一个PyTorch Dataset类对象. 获取这个Dasetset类对象后， 取代之前的`prepare_dataloader`

[ ] To create a DataLoader for pretraining scGPT