Deep single-cell RNA-seq data clustering with graph prototypical contrastive learning

Jaehyeon Shin

2025.01.14

Paper Review

CONTENTS

BACKGROUND

- single-cell RNA-sequencing (scRNA-seq)
- Contrastive Learning (Instance-wise, Prototypical)
- Sampling bias
- Reconstruction-based representation learning
- ZINB distribution

BACKGROUND

single-cell RNA-seq

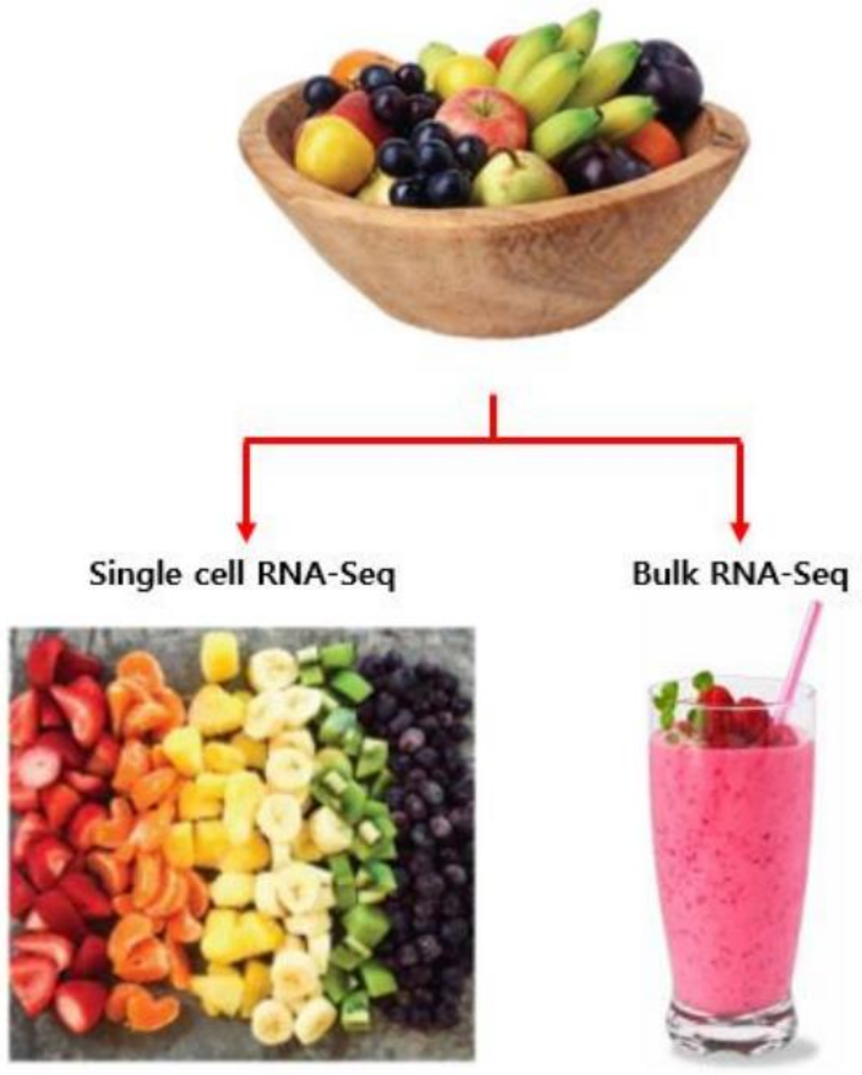
- Bulk RNA-seq

- Measure the average gene expression across the population of cells in a sample

- Single cell RNA-Seq

- Measure the gene expression of individual cells in a sample

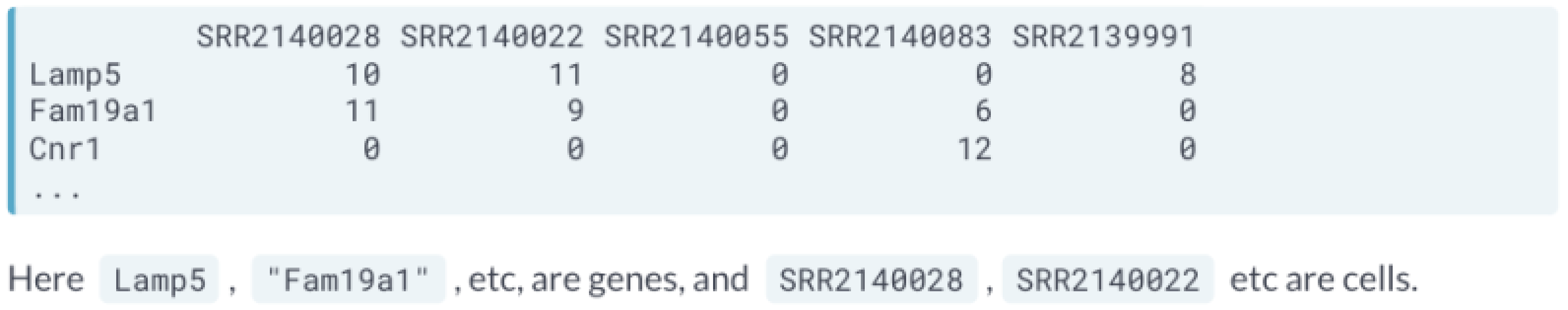
- Difference between obtaining detailed information at the cellular level and obtaining overall information.



single-cell RNA-seq

✓ Measure the gene expression of individual cells in a sample

▪ gene expression matrix



```
         SRR2140028 SRR2140022 SRR2140055 SRR2140083 SRR2139991
Lamp5            10         11          0          0          8
Fam19a1          11          9          0          6          0
Cnr1              0          0          0         12          0
...
```

Here `Lamp5`, `"Fam19a1"`, etc, are genes, and `SRR2140028`, `SRR2140022` etc are cells.

▪ Many zeros observed in these count matrices.

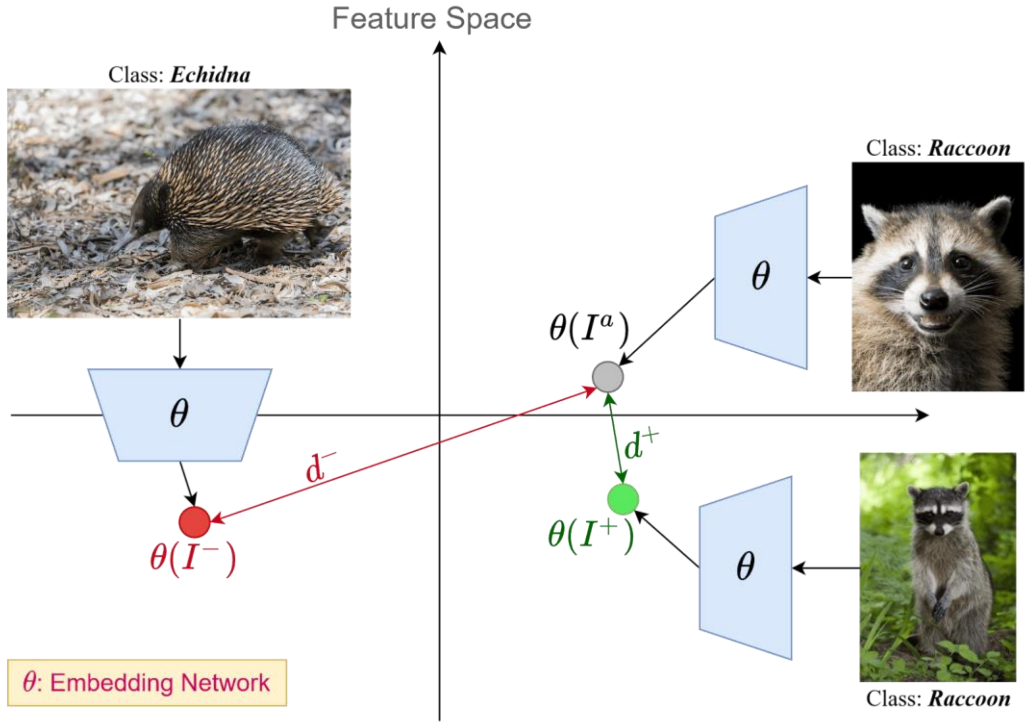
1) Biological zeros
2) Technical false zeros

BACKGROUND

Contrastive Learning

- Goal: To learn meaningful representations by:

Pulling positive pairs closer in the embedding space (similar data).
Pushing negative pairs farther apart (dissimilar data).



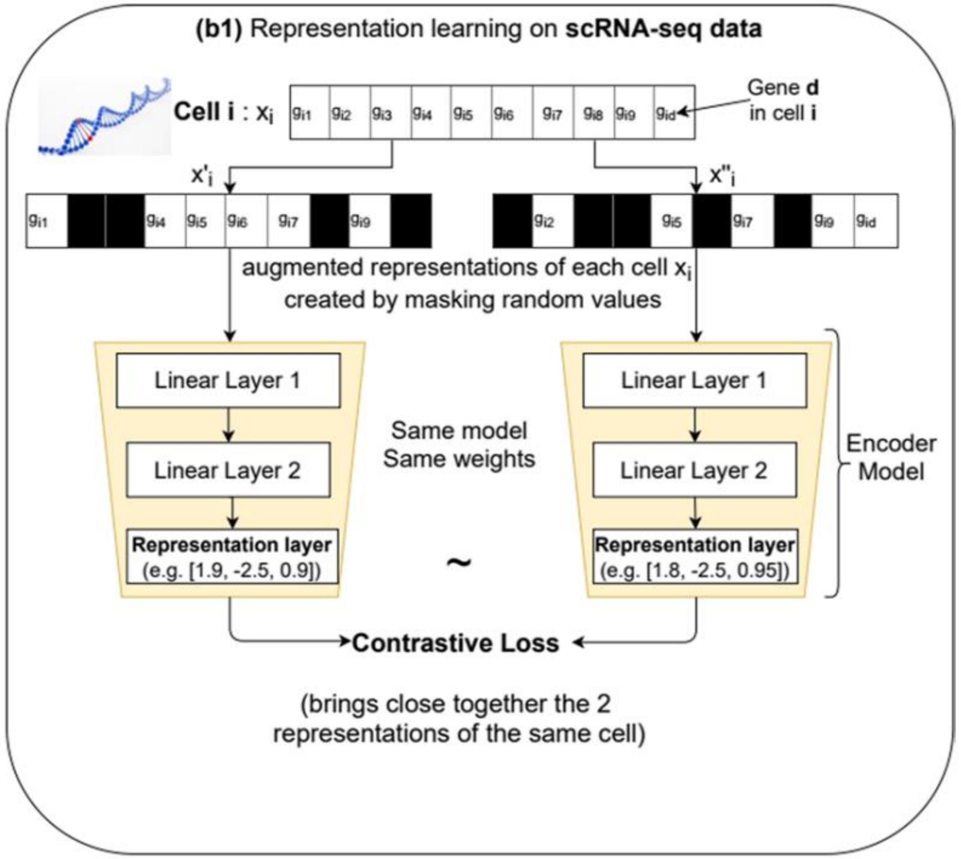
BACKGROUND

Instance-wise Contrastive Learning

For each **cell instance**, two distinct augmented views are created to represent the same cell:

- Positive Pairs : Generated by applying augmentations (random masking of gene expression values) to the same cell
- Negative Pairs : All other cells are treated as negative instances

Problem: Sampling bias


(b1) Representation learning on scRNA-seq data
Cell i : x_i
g_{i1} g_{i2} g_{i3} g_{i4} g_{i5} g_{i6} g_{i7} g_{i8} g_{i9} g_{id}
Gene d in cell i
x'_i
x''_i
augmented representations of each cell x_i created by masking random values
Linear Layer 1
Linear Layer 2
Representation layer (e.g. [1.9, -2.5, 0.9])
Same model Same weights
~
Linear Layer 1
Linear Layer 2
Representation layer (e.g. [1.8, -2.5, 0.95])
Encoder Model
Contrastive Loss
(brings close together the 2 representations of the same cell)

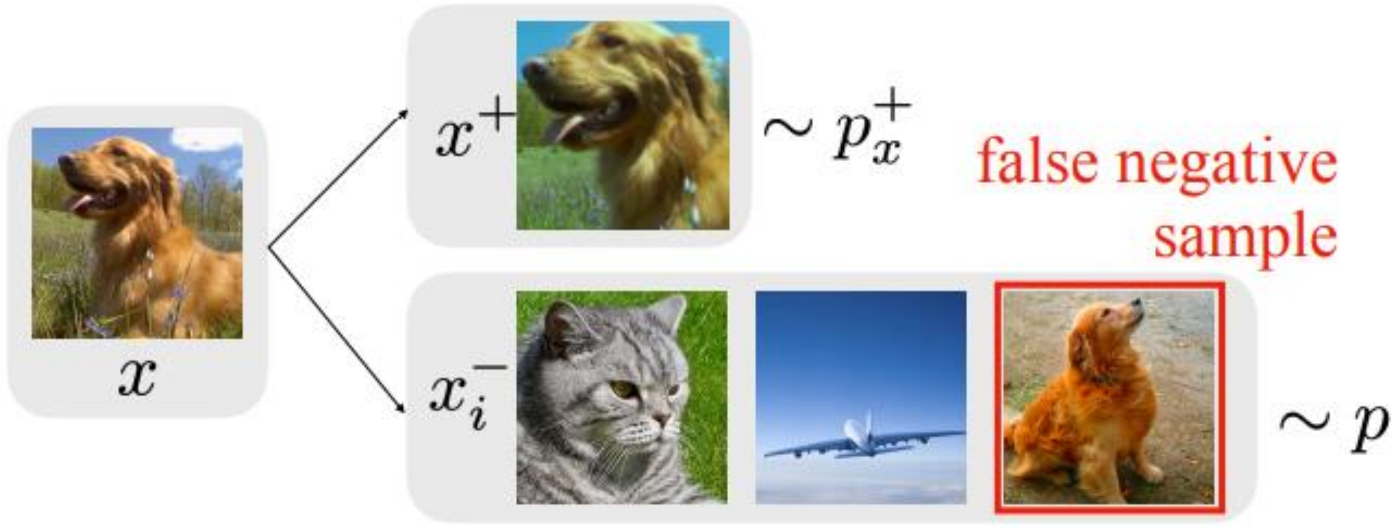
Ciortan, Madalina, and Matthieu Defrance. "Contrastive self-supervised clustering of scRNA-seq data." *BMC bioinformatics* 22.1 (2021): 280.

BACKGROUND

Sampling Bias

- Some negative pairs might belong to the same type. (False Negative Samples)

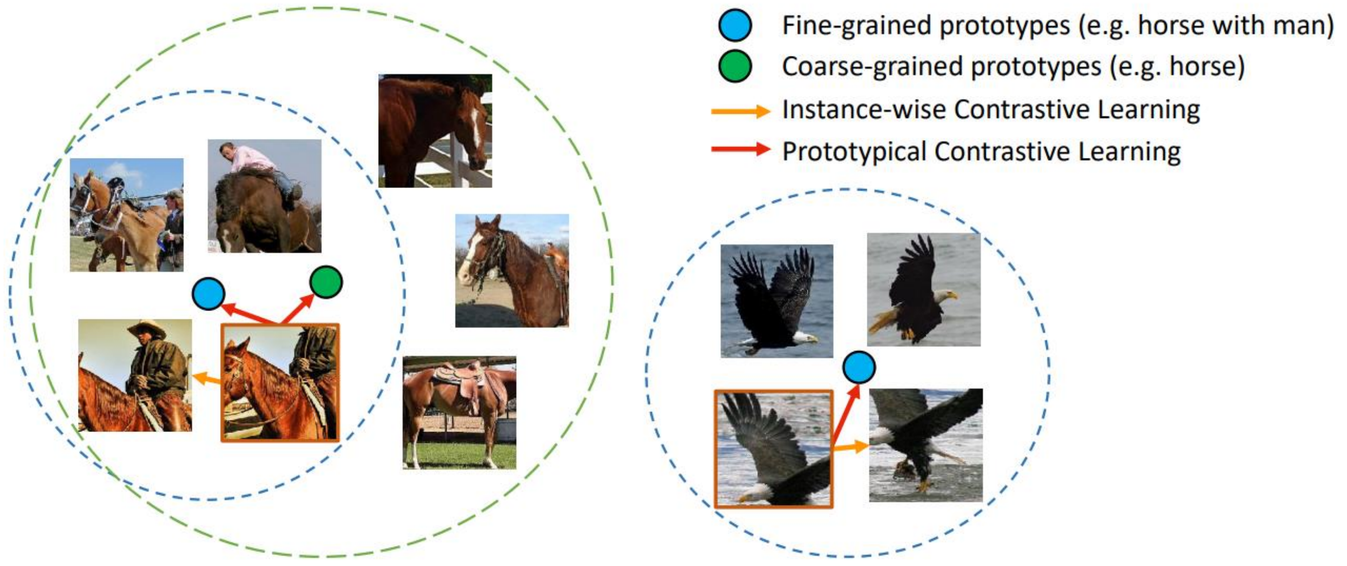
-> Performance Drop due to incorrect separation of semantically similar instances.



Chuang, Ching-Yao, et al. "Debiased contrastive learning." *Advances in neural information processing systems* 33 (2020): 8765-8775.

Prototypical Contrastive Learning

- Prototype : A representative embedding for a group of semantically similar instances.
- Each instance is assigned to multiple prototypes with different granularity
- Construct a contrastive loss which enforce the embedding of a sample to be more similar to its corresponding prototypes compared to other prototypes.


Fine-grained prototypes (e.g. horse with man)
Coarse-grained prototypes (e.g. horse)
Instance-wise Contrastive Learning
Prototypical Contrastive Learning

BACKGROUND

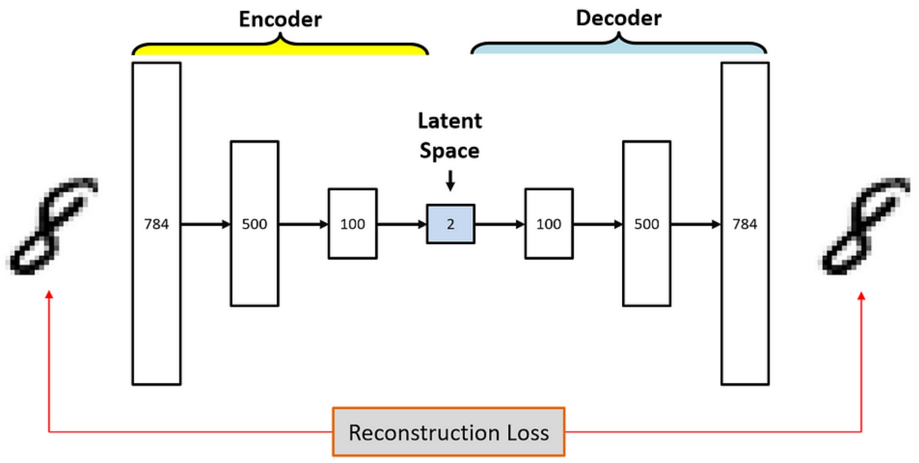
Reconstruction-based representation learning

- Step 1 : Compression (Encoding)
The input data is compressed into a latent space

- Step 2 : Reconstruction (Decoding)
The latent representation is used to reconstruct the original data

- Step 3 : Loss function
Measure the difference between the input and reconstructed output

Ex: $\mathcal{L} = ||x - \hat{x}||^2$



BACKGROUND

ZINB Distribution : Zero-inflated negative binomial distribution

- NB Distribution : $\mathrm{NB}(X^{\text{count}} \mid \mu, \theta) = \dfrac{\Gamma(X^{\text{count}} + \theta)}{X^{\text{count}}!\,\Gamma(\theta)} \left(\dfrac{\theta}{\theta+\mu}\right)^{\theta} \left(\dfrac{\mu}{\theta+\mu}\right)^{X^{\text{count}}}$

- Zero-inflated model : Statistical model based on a zero-inflated probability distribution

(a distribution that allows for frequent zero-valued observations)

- The mean, the dispersion of the negative binomial distribution and with an additional coefficient(pi) : the weight of the point mass of probability at zero.

$$\mathrm{ZINB}(X^{\text{count}} \mid \pi, \mu, \theta) = \pi\delta_0(X^{\text{count}}) + (1-\pi)\,\mathrm{NB}(X^{\text{count}} \mid \mu, \theta)$$

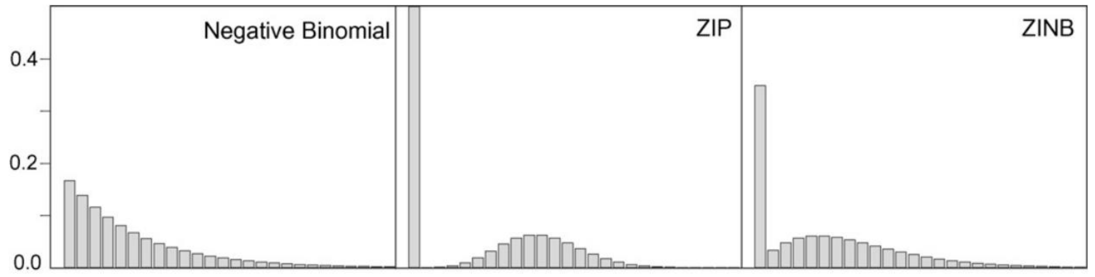


Fig. 2. Probability functions of the negative binomial ($\mu = 5$, $\theta = 1$), ZIP ($\mu = 10$, $p = 0.5$) and ZINB ($\mu = 7.5$, $p = 1/3$, $\theta = 3$) with the same overall mean and variance.

M. Minami et al. / Fisheries Research 84 (2007) 210–221

MOTIVATION

Existing method

- Dimensionality reduction techniques: PCA, t-SNE, UMAP

1. scRNA-seq data contains high-dimensional features
-> Curse of Dimensionality -> Poor clustering performance.

2. Pervasive dropout phenomenon in scRNA-seq data
-> False zero counts -> Hard to analyze scRNA-seq data

MOTIVATION

Existing method

- Deep Neural Networks:

Powerful feature extractors for dimensionality reduction of clustering using Autoencoder Network

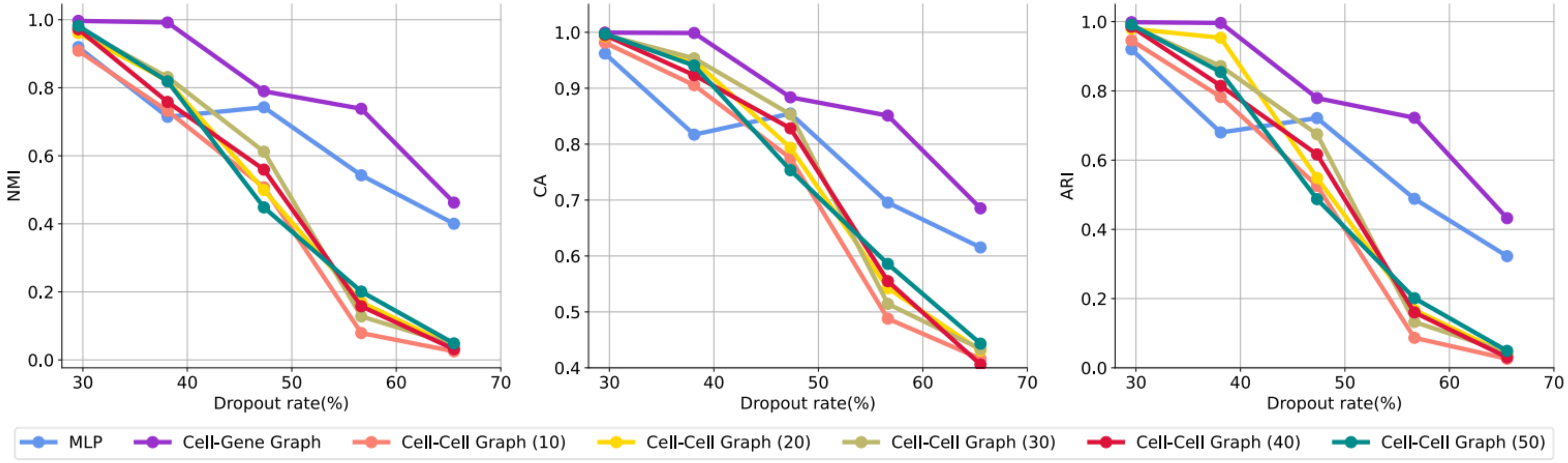
Example : DCA (Eraslan et al., 2019) and scDeepCluster (Tian et al, 2019).

- Problem :

Works well when input features are informative enough
However, gene expression matrix is highly sparse

MOTIVATION

Existing method



- To leverage the relational information between cells

-> Construct a cell-cell graph

- Limitation

Cell-cell graphs are built using cell representations or raw gene expression values.

The sparsity of the gene expression matrix results in low-quality graphs

-> Fail to accurately capture cell-to-cell relationships

MOTIVATION

scGPCL (Graph-base Prototypical Contrastive Learning method)

- Goal : clustering cells in scRNA-seq by fully leveraging relational information between cells.

- Components

1. Bipartite cell-gene graph
- Connects cell and gene nodes if a gene is expressed in the cell
- Preserve the inherent relationships in scRNA-seq data -> maintain graph quality

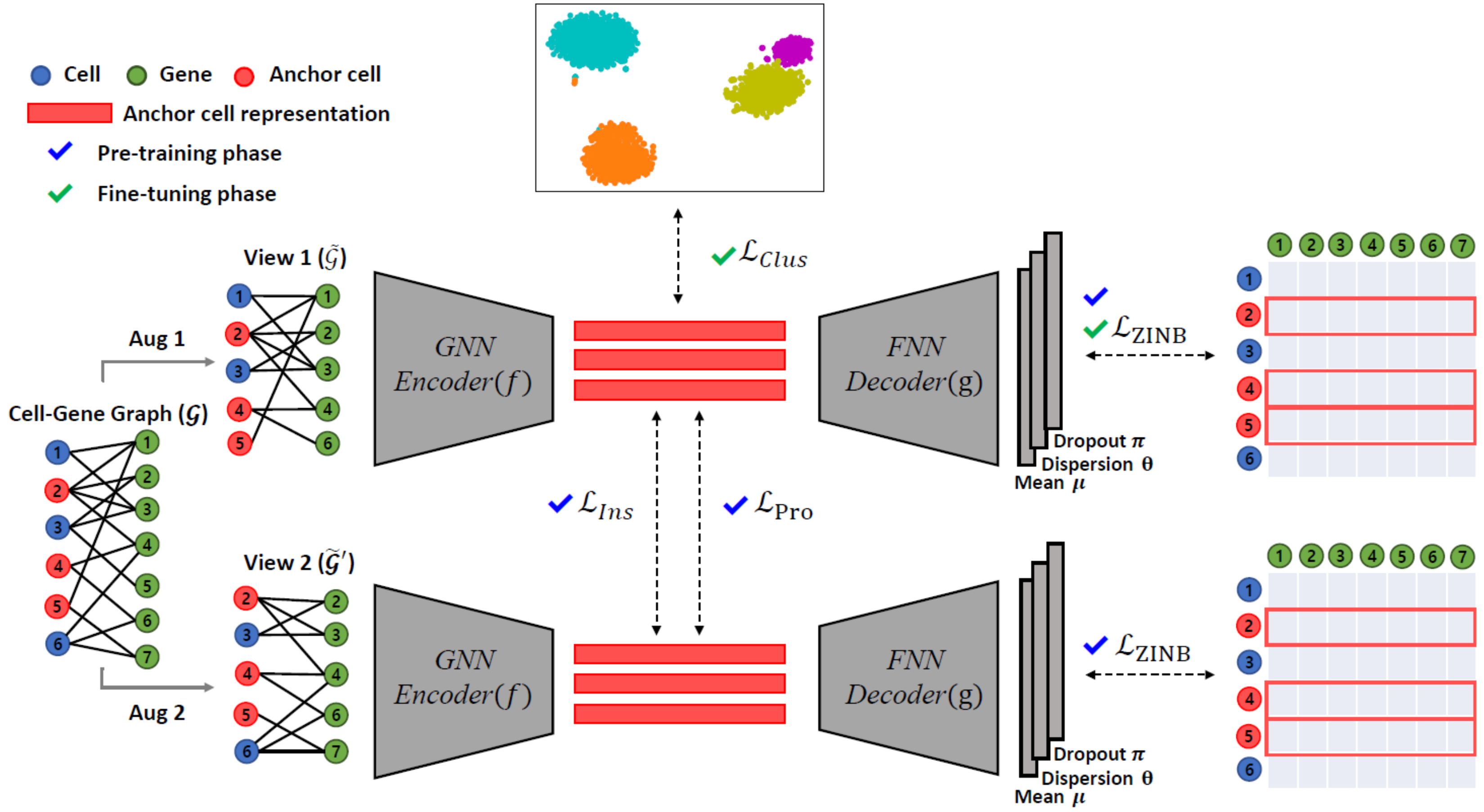
2. Instance-wise Contrastive Learning
- Use augmentation techniques to mimic the characteristics of scRNA-seq data

3. Prototypical Contrastive Learning
- Learn cluster-specific(cell type) information
- Reduce sampling bias by linking anchor cells to corresponding cluster prototypes.

METHOD

Architecture

Phase1	→	Phase2
Pre-training		Fine-tuning



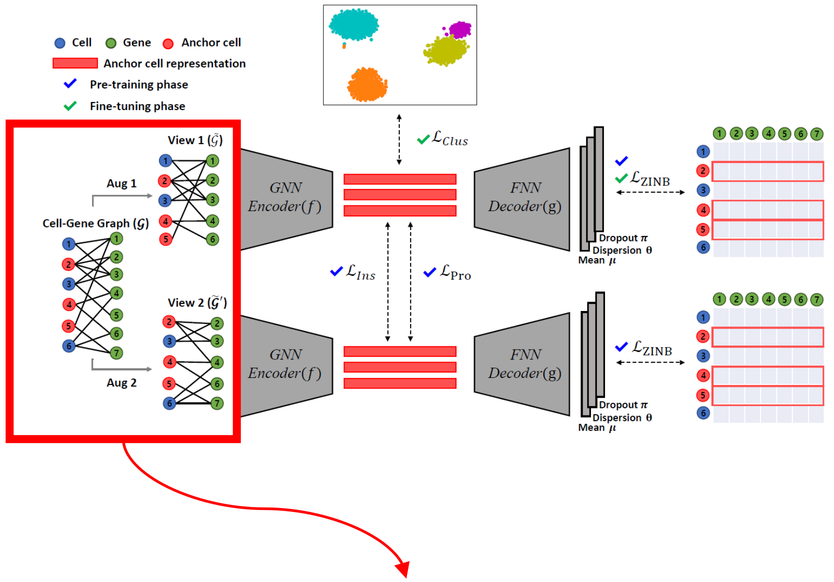
Cell
Gene
Anchor cell
Anchor cell representation
Pre-training phase
Fine-tuning phase
View 1 ($\tilde{\mathcal{G}}$)
Aug 1
Cell-Gene Graph ($\mathcal{G}$)
Aug 2
View 2 ($\tilde{\mathcal{G}}'$)
GNN Encoder(f)
FNN Decoder(g)
✓ $\mathcal{L}_{Clus}$
✓ $\mathcal{L}_{Ins}$
✓ $\mathcal{L}_{\text{Pro}}$
✓ ✓ $\mathcal{L}_{\text{ZINB}}$
✓ $\mathcal{L}_{\text{ZINB}}$
Dropout π
Dispersion θ
Mean μ

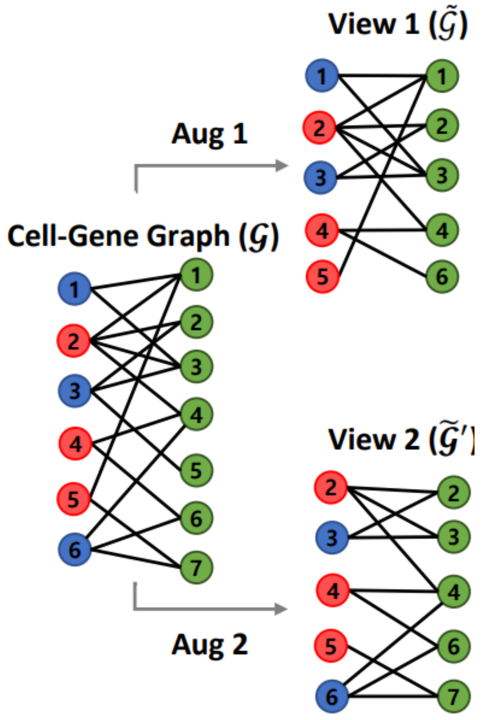
Phase 1 : Pre-training

Augmentation

- Two augmented view by Using two ways
- Subgraph Sampling
- Feature Masking

✓ To Mimic the technical limitations in sequencing
only a fraction of the gene expression is detected for each cell. :
Single-cell RNA-seq processes can result in substantial cell loss or selective isolation

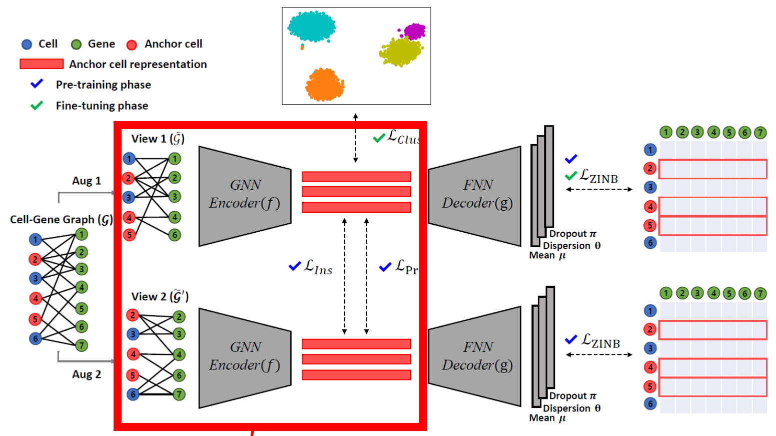


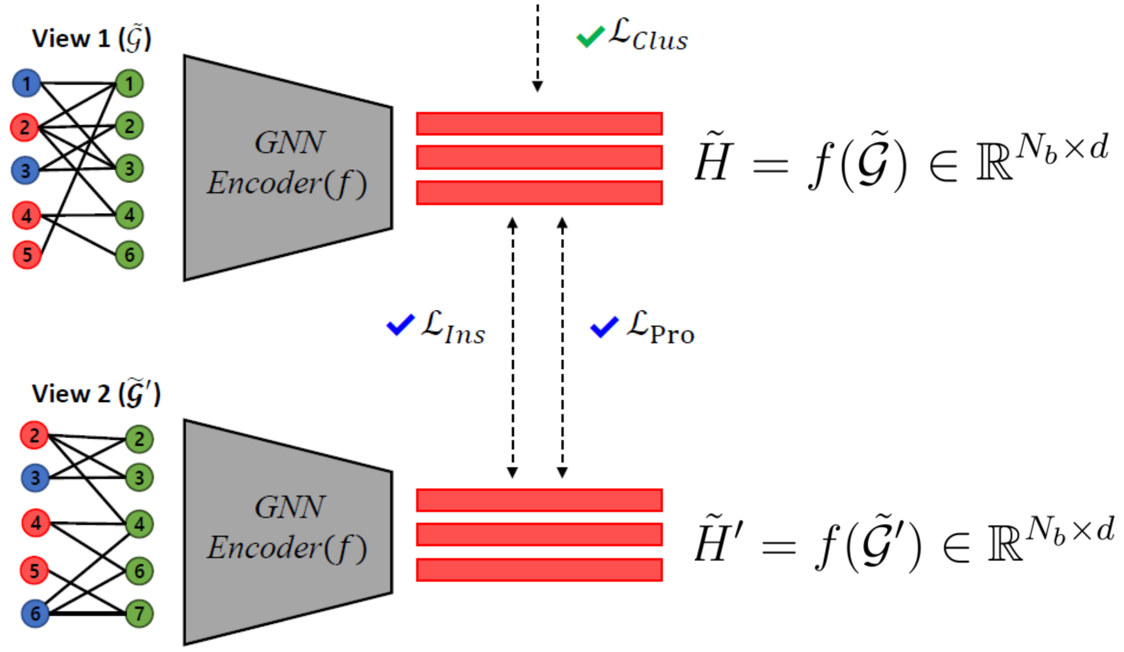

View 1 ($\tilde{\mathcal{G}}$)
Aug 1
Cell-Gene Graph ($\mathcal{G}$)
View 2 ($\tilde{\mathcal{G}}'$)
Aug 2

Phase 1 : Pre-training

GNN Encoder

- Two anchor cell representations is obtained by passing them through GNN Encoder(f)




View 1 ($\tilde{\mathcal{G}}$)
View 2 ($\tilde{\mathcal{G}}'$)
GNN Encoder(f)
GNN Encoder(f)
$\mathcal{L}_{Clus}$
$\mathcal{L}_{Ins}$
$\mathcal{L}_{Pro}$
$\tilde{H} = f(\tilde{\mathcal{G}}) \in \mathbb{R}^{N_b \times d}$
$\tilde{H}' = f(\tilde{\mathcal{G}}') \in \mathbb{R}^{N_b \times d}$

Phase 1 : Pre-training

Instance-wise Contrastive Learning

- After the encoding process, apply the contrastive learning framework.

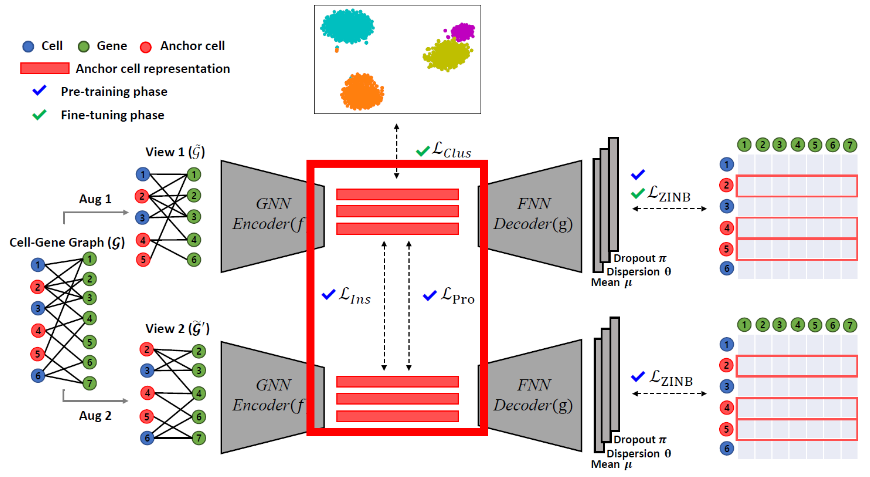
- the infoNCE object for each positive node pair :

$$l_{\text{Ins}}(\tilde{h}_i, \tilde{h}_i^{'}) = \log \frac{e^{(\text{sim}(\tilde{h}_i,\tilde{h}_i^{'})/\tau)}}{\sum_{j=1}^{N_b} \mathbb{1}_{[i\neq j]} e^{(\text{sim}(\tilde{h}_i,\tilde{h}_j)/\tau)} + \sum_{j=1}^{N_b} e^{(\text{sim}(\tilde{h}_i,\tilde{h}_j^{'})/\tau)}}$$

- The instance-wise contrastive loss :

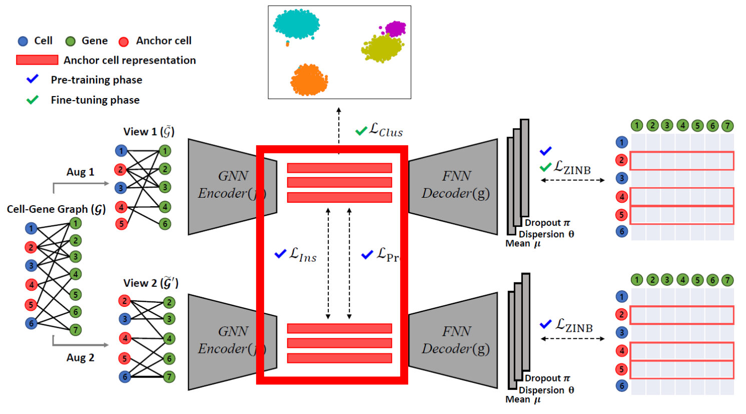
$$\mathcal{L}_{\text{Ins}} = -\frac{1}{2N_b}\sum_{i=1}^{N_b}[l_{\text{Ins}}(\tilde{h}_i, \tilde{h}_i^{'}) + l_{\text{Ins}}(\tilde{h}_i^{'}, \tilde{h}_i)].$$

- Learn the cell representations by pulling together positive pairs and pushing apart negative pairs in the cell representation space



Phase 1 : Pre-training

Prototypical Contrastive Learning



- Problem of Instance-wise contrastive loss
: sampling bias

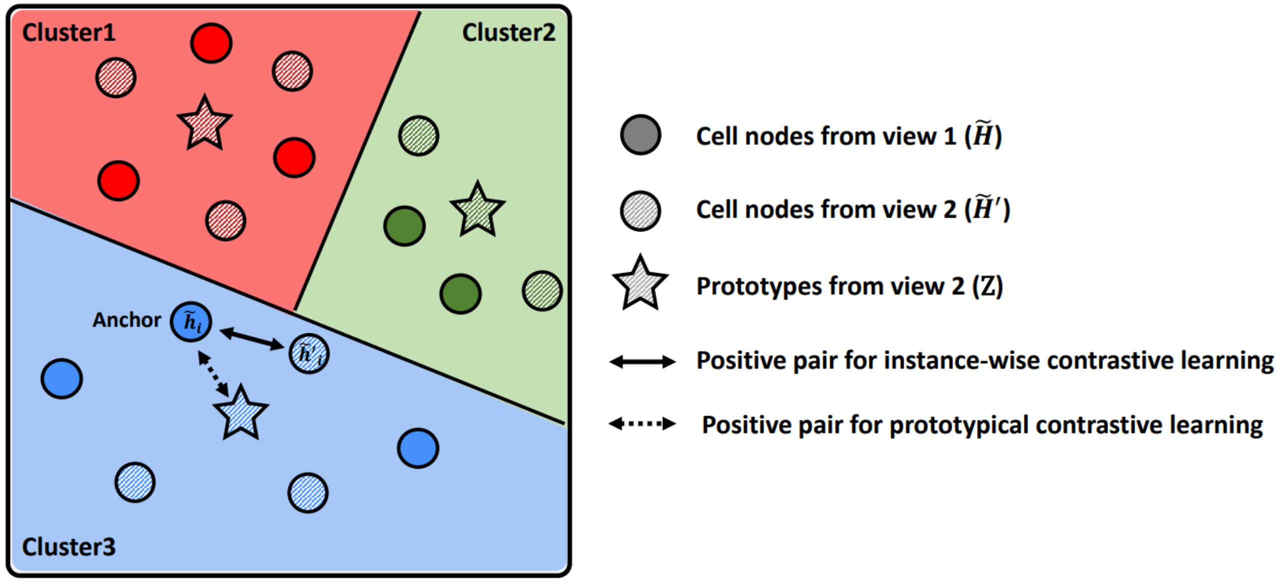
- To alleviate this
-> Prototypical Contrastive learning framework

- Loss for a particular cell i : $l_{\text{Pro}}(\tilde{h}_i) = \frac{1}{T}\sum_{t=1}^{T}\sum_{s=1}^{K_t}\mathbb{1}_{(\tilde{h}_i \in z_s^t)} \log \frac{e^{(\text{sim}(\tilde{h}_i, z_s^t)/\tau)}}{\sum_{j=1}^{K_t} e^{(\text{sim}(\tilde{h}_i, z_j^t)/\tau)}}$

- The Prototypical contrastive loss:

$$\mathcal{L}_{\text{Pro}} = -\frac{1}{N_b}\sum_{i=1}^{N_b} l_{\text{Pro}}(\tilde{h}_i).$$

cf.


Cluster1
Cluster2
Cluster3
Anchor
$\tilde{h}_i$
$\tilde{h}'_i$
Cell nodes from view 1 ($\tilde{H}$)
Cell nodes from view 2 ($\tilde{H}'$)
Prototypes from view 2 (Z)
Positive pair for instance-wise contrastive learning
Positive pair for prototypical contrastive learning

Phase 1 : Pre-training

ZINB-based Reconstruction Loss

(Assumption) The gene expression matrix follows a zero-inflated negative binomial(ZINB) distribution.

- scRNA-seq data distribution is parameterized by the ZINB distribution

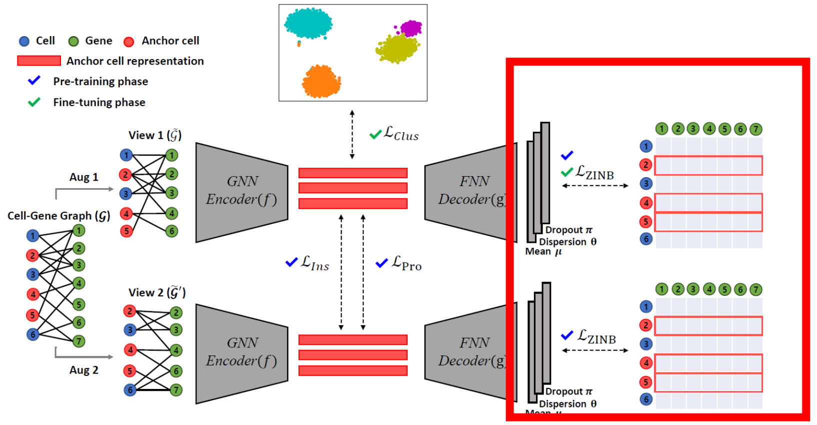
Estimate the parameters of the ZINB distribution

- Negative log-likelihood of ZINB Distribution :

$$l_{\text{ZINB}}(\Pi, M, \Theta) = \frac{1}{N_b \times N_g} \sum_{i=1}^{N_b} \sum_{j=1}^{N_g} -\log\left(\text{ZINB}\left(X_{ij}^{\text{count}} \mid \Pi_{ij}, M_{ij}, \Theta_{ij}\right)\right)$$

- The ZINB-based reconstruction loss :

$$\mathcal{L}_{\text{ZINB}}^{\text{Pre}} = \frac{1}{2}\left[l_{\text{ZINB}}(\tilde{\Pi}, \tilde{M}, \tilde{\Theta}) + l_{\text{ZINB}}(\tilde{\Pi}', \tilde{M}', \tilde{\Theta}')\right]$$



Phase 1 : Pre-training

Final Objectives of Pre-training Phase

$$\mathcal{L}_{\mathrm{Pre}} = \lambda_1 \mathcal{L}_{\mathrm{Ins}} + \lambda_2 \mathcal{L}_{\mathrm{Pro}} + \mathcal{L}_{\mathrm{ZINB}}^{\mathrm{Pre}}$$

instance-wise contrastive loss

prototypical contrastive loss

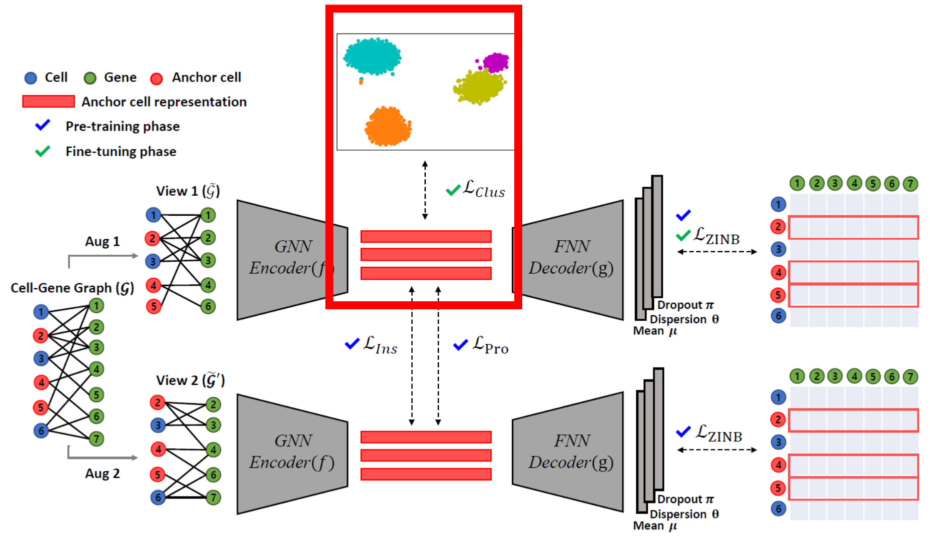
ZINB-based Reconstruction Loss

λ_1 , λ_2 : Balance coefficients

- By minimizing $\mathcal{L}_{Pre}$, we can learn cell representations

Phase 2 : Fine-tuning

Clustering Task-oriented Loss



- Assign each cell to a cluster with high confidence
- Minimize KL divergence D_{KL}

$$\mathcal{L}_{\text{Cluster}} = D_{\text{KL}}(P\|Q) = \sum_{i=1}^{N_b}\sum_{k=1}^{K} p_{ik} \log \frac{p_{ik}}{q_{ik}}$$

q_{ik} : Soft cluster probability (cell i to cluster k)

p_{ik} : Sharpened q_{ik} for better clustering

$$p_{ik} = \frac{q_{ik}^2/f_k}{\sum_{j=1}^{K} q_{ij}^2/f_j}$$

Phase 2 : Fine-tuning

Final Objectives of Fine-tuning phase

$$\mathcal{L}_{\text{Fine}} = \underbrace{\mathcal{L}_{\text{Cluster}}}_{\text{Clustering Task-Oriented Loss}} + \lambda_3 \underbrace{\mathcal{L}^{\text{Fine}}_{\text{ZINB}}}_{\text{ZINB-based Reconstruction Loss}}$$

λ_3 : Balance coefficients

- scGPCL maintains the reconstruction-based loss to preserve the local structure of data

EXPERIMENTS

Performance comparison – on the simulated datasets

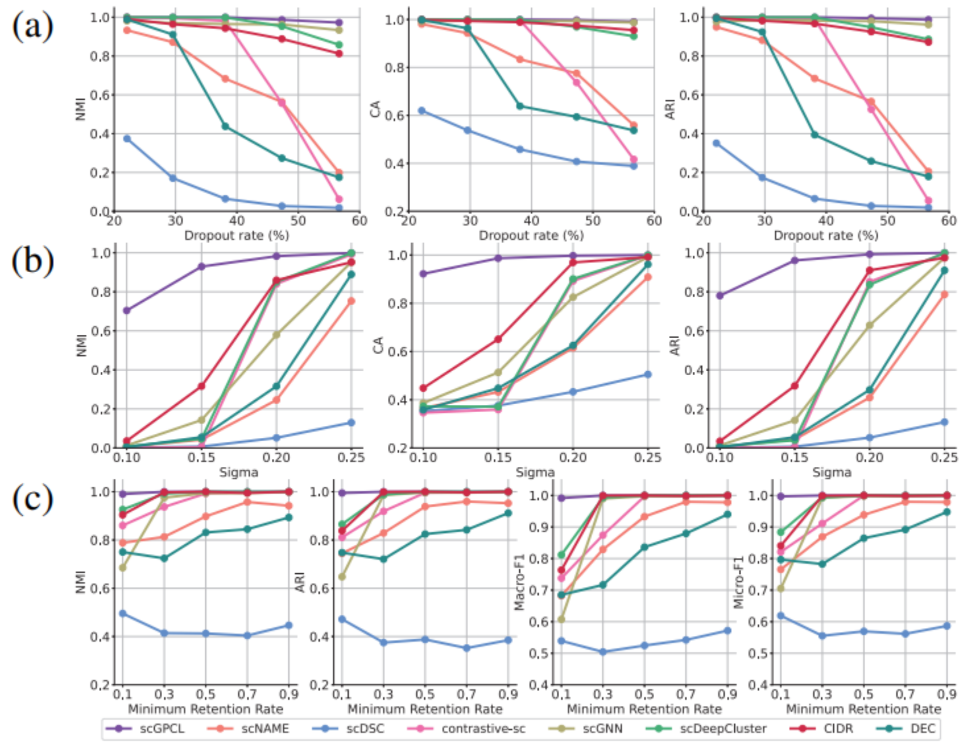


Figure 2. Performance comparisons of scGPCL and other baselines on the simulated dataset. (a), (b), and (c) represent the performance over the various dropout rates, sigmas (small sigma indicates low signals for clustering), and minimum retention rates (small value indicates more imbalanced data), respectively.

(a) Case 1: Gene expression matrix is highly sparse

(b) Case 2: Gene expression values contain relatively low signal strength required for clustering

(c) Case 3: The size of cell clusters is imbalanced in number

EXPERIMENTS

Evaluation of scGPCL on real scRNA-seq datasets

Data	Sequencing platform	# of Cells	# of Genes	# of Subgroups
Camp	SMARTer	777	19,020	7
Mouse ES cells	inDrop	2,717	24,047	4
Mouse bladder cells	Microwell-seq	2,746	19,771	16
Zeisel	STRT-seq UMI	3,005	19,972	9
Worm neuron cells	sci-RNA-seq	4,186	13,488	10
10X PBMC	10X	4,340	19,773	8
Human kidney cells	10X	5,685	25,215	11
Baron	inDrop	8,569	20,125	14
Shekhar mouse retina cells	Drop-seq	27,499	13,166	19

Table 1. Statistics for real datasets used for experiments.

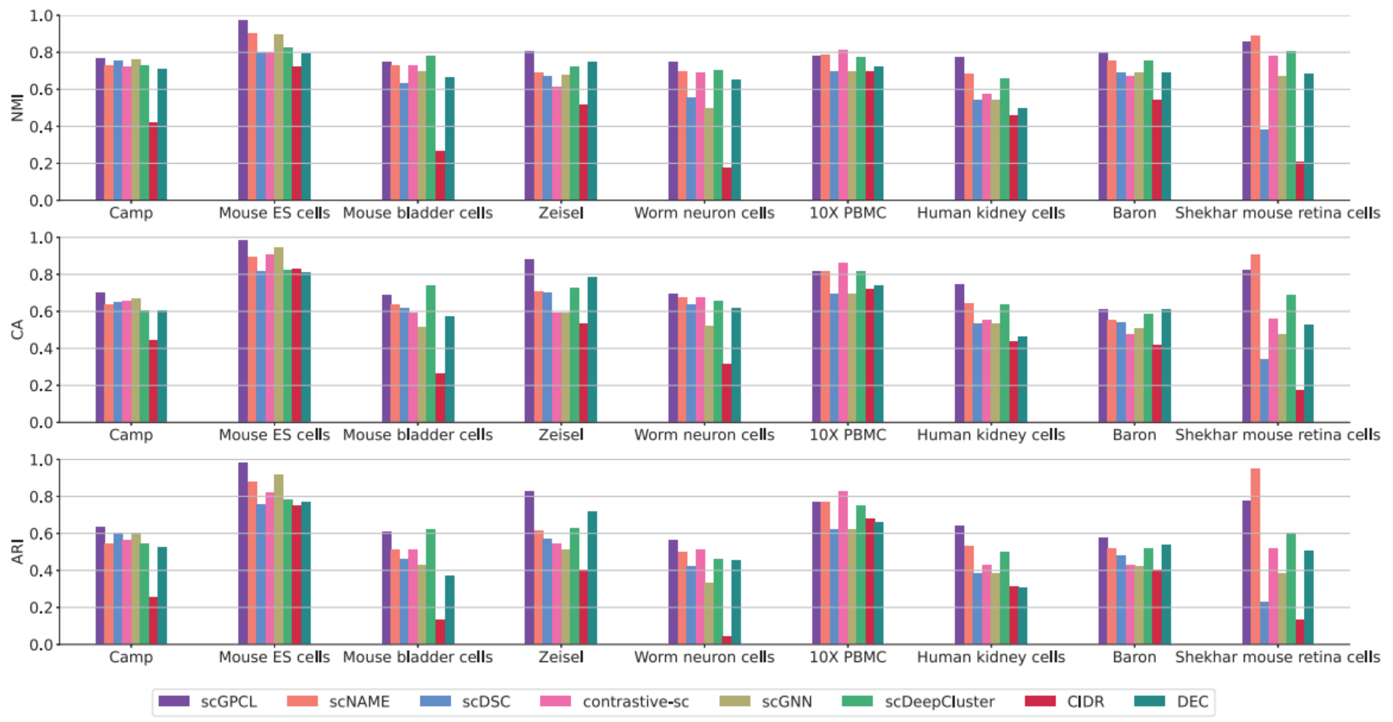
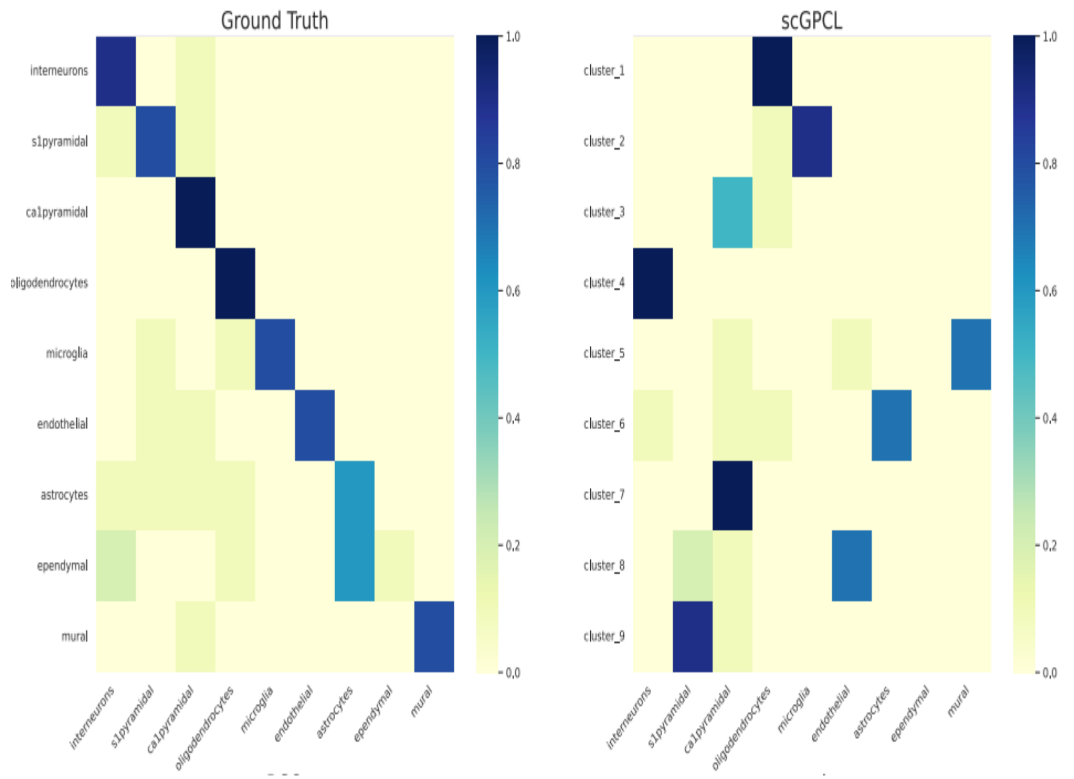


Figure 3. Performance comparisons of scGPCL and other baselines on the nine real scRNA-seq datasets.

EXPERIMENTS

Marker gene identification



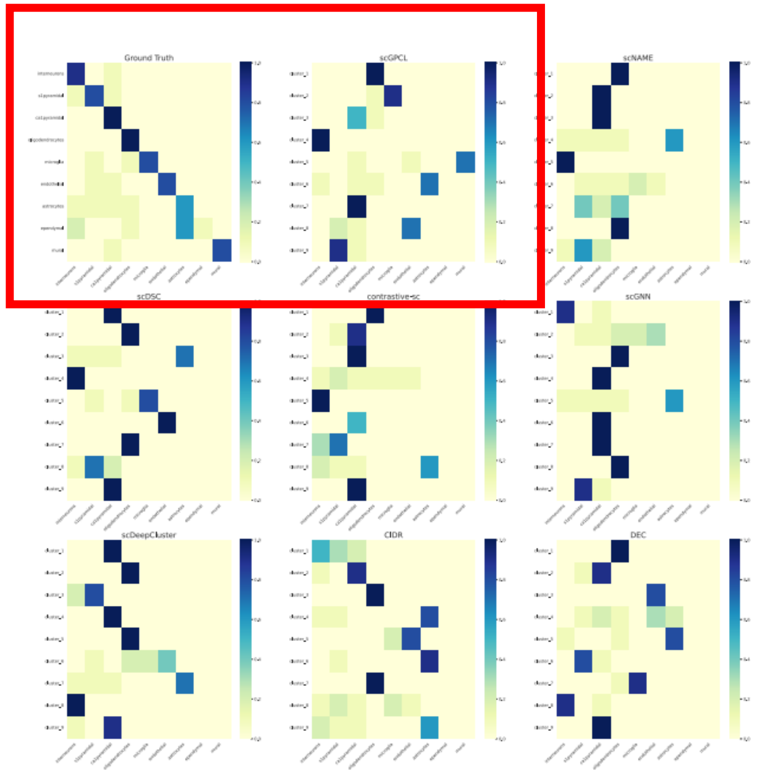


Figure 4. Overlap between gold standard cell types and the top 10 DEGs in clusters detected by ground truth cell type on scGPCL, and baseline methods.

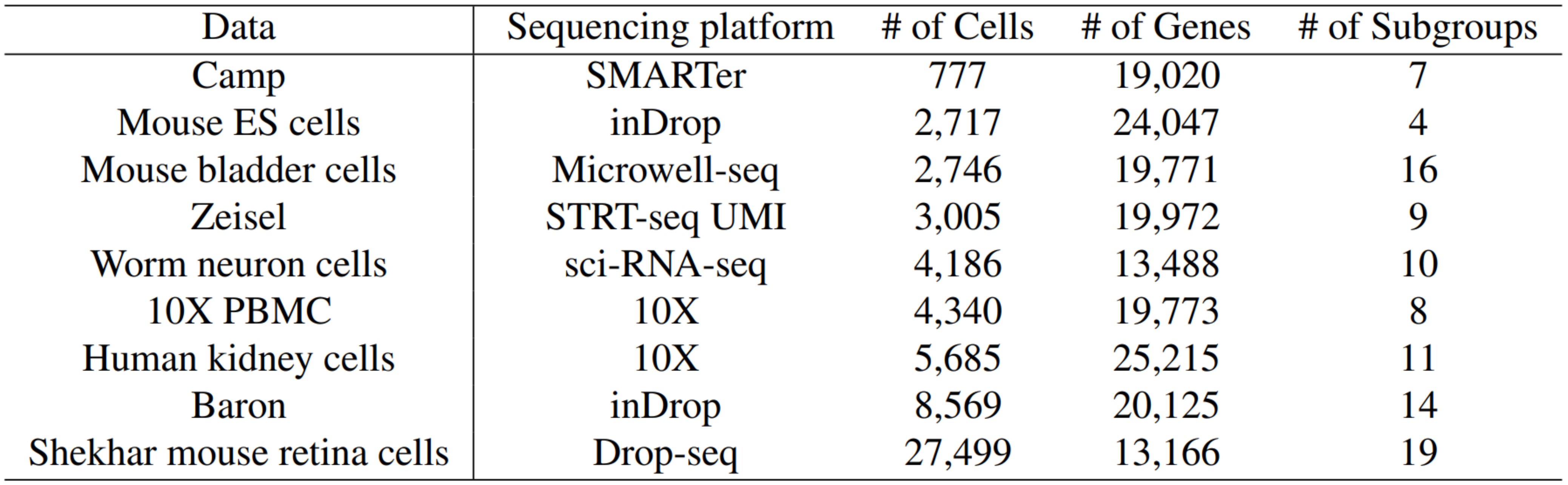
- ✓ Compute the overlap with the gold standard cell types on the Zeisel dataset.
- ✓ scGPCL succeeds in learning clusters with 'mural' cell type that belongs to a minority class with a small number of cells.

Implementation

- Performance on nine real scRNA-seq datasets
  - Augmentation
  - Phase1 : Pre-training
  - Phase2 : Fine-tuning
- Hyperparameter Analysis

Implementation – Performance on nine real scRNA-seq datasets

Datasets



Data	Sequencing platform	# of Cells	# of Genes	# of Subgroups
Camp	SMARTer	777	19,020	7
Mouse ES cells	inDrop	2,717	24,047	4
Mouse bladder cells	Microwell-seq	2,746	19,771	16
Zeisel	STRT-seq UMI	3,005	19,972	9
Worm neuron cells	sci-RNA-seq	4,186	13,488	10
10X PBMC	10X	4,340	19,773	8
Human kidney cells	10X	5,685	25,215	11
Baron	inDrop	8,569	20,125	14
Shekhar mouse retina cells	Drop-seq	27,499	13,166	19

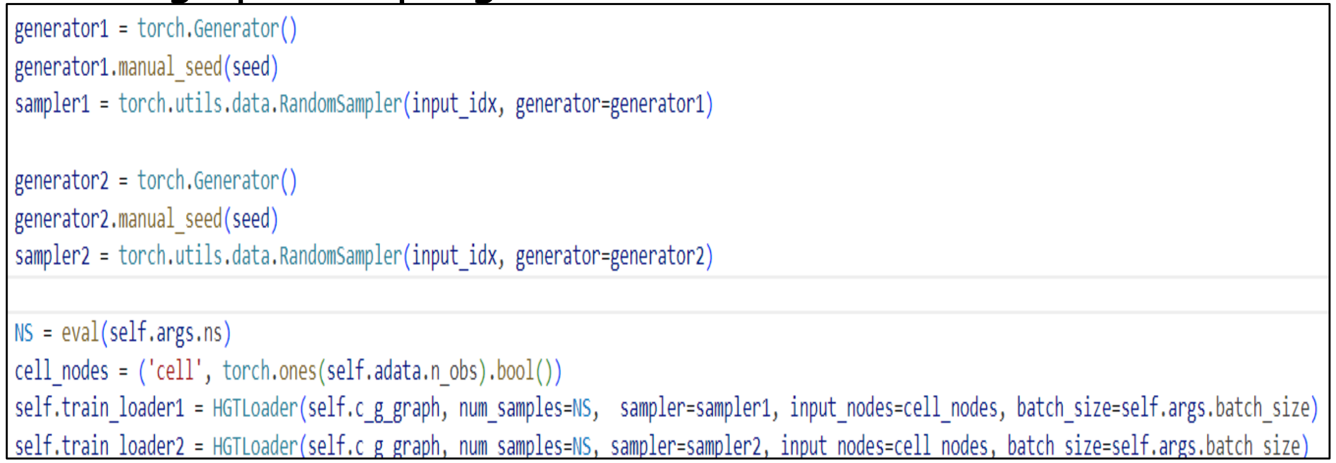
Table 1. Statistics for real datasets used for experiments.

- ✓ For each data, Matched the number of clusters with # of subgroups.
- ✓ The experiments were conducted three times

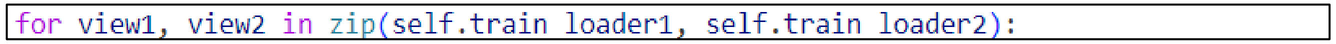
Implementation - Performance on nine real scRNA-seq datasets

Augmentation

✓ Subgraph Sampling

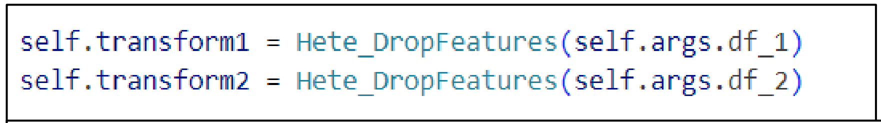


```
generator1 = torch.Generator()
generator1.manual_seed(seed)
sampler1 = torch.utils.data.RandomSampler(input_idx, generator=generator1)

generator2 = torch.Generator()
generator2.manual_seed(seed)
sampler2 = torch.utils.data.RandomSampler(input_idx, generator=generator2)

NS = eval(self.args.ns)
cell_nodes = ('cell', torch.ones(self.adata.n_obs).bool())
self.train_loader1 = HGTLoader(self.c_g_graph, num_samples=NS,  sampler=sampler1, input_nodes=cell_nodes, batch_size=self.args.batch_size)
self.train_loader2 = HGTLoader(self.c_g_graph, num_samples=NS, sampler=sampler2, input_nodes=cell_nodes, batch_size=self.args.batch_size)
```

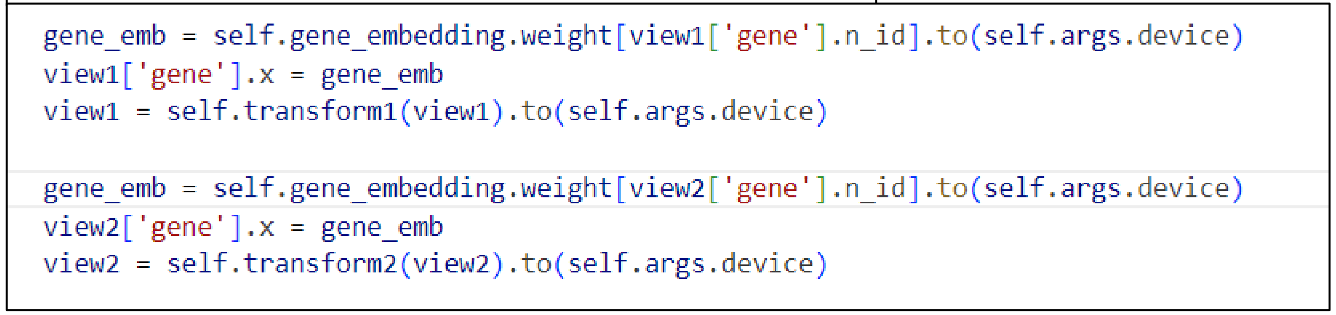


```
for view1, view2 in zip(self.train_loader1, self.train_loader2):
```

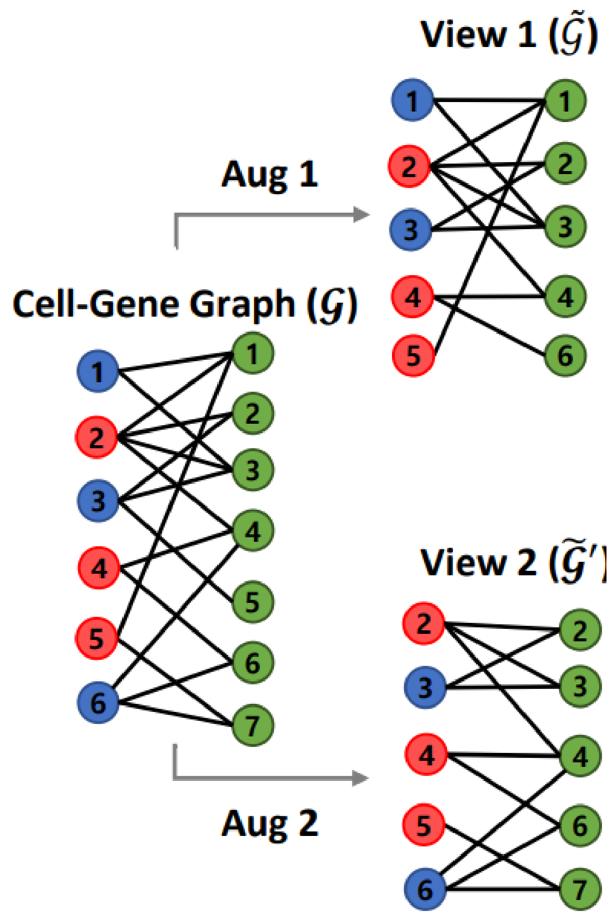
✓ Feature Masking(Dropout)



```
self.transform1 = Hete_DropFeatures(self.args.df_1)
self.transform2 = Hete_DropFeatures(self.args.df_2)
```



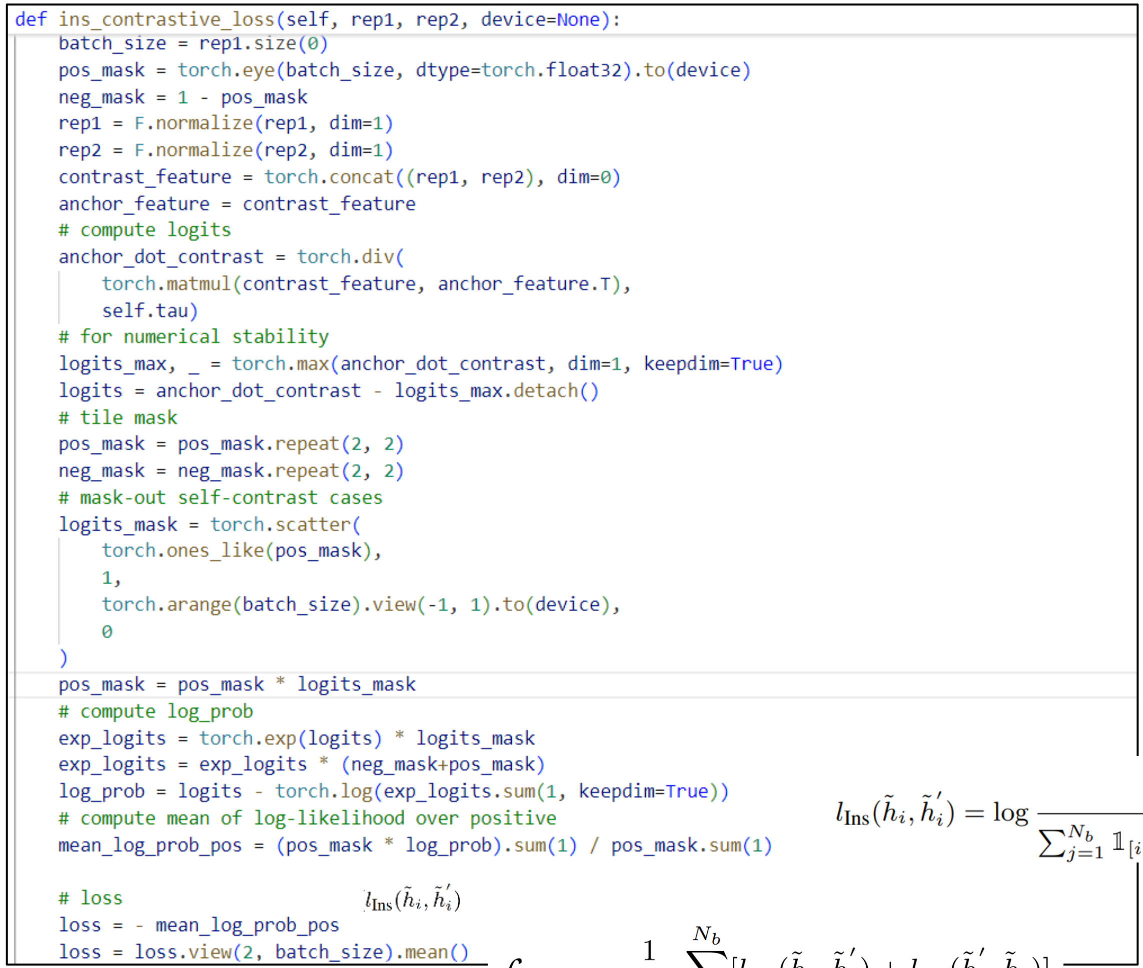
```
  gene_emb = self.gene_embedding.weight[view1['gene'].n_id].to(self.args.device)
  view1['gene'].x = gene_emb
  view1 = self.transform1(view1).to(self.args.device)

  gene_emb = self.gene_embedding.weight[view2['gene'].n_id].to(self.args.device)
  view2['gene'].x = gene_emb
  view2 = self.transform2(view2).to(self.args.device)
```


View 1 ($\tilde{\mathcal{G}}$)
Aug 1
Cell-Gene Graph ($\mathcal{G}$)
View 2 ($\tilde{\mathcal{G}}'$)
Aug 2
1
2
3
4
5
6
7

Implementation - Performance on nine real scRNA-seq datasets

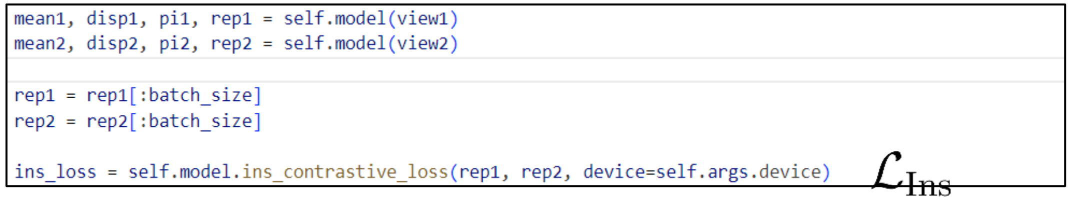
Phase 1: Pre-training

✓ Instance-wise Contrastive Loss



```
def ins_contrastive_loss(self, rep1, rep2, device=None):
    batch_size = rep1.size(0)
    pos_mask = torch.eye(batch_size, dtype=torch.float32).to(device)
    neg_mask = 1 - pos_mask
    rep1 = F.normalize(rep1, dim=1)
    rep2 = F.normalize(rep2, dim=1)
    contrast_feature = torch.concat((rep1, rep2), dim=0)
    anchor_feature = contrast_feature
    # compute logits
    anchor_dot_contrast = torch.div(
        torch.matmul(contrast_feature, anchor_feature.T),
        self.tau)
    # for numerical stability
    logits_max, _ = torch.max(anchor_dot_contrast, dim=1, keepdim=True)
    logits = anchor_dot_contrast - logits_max.detach()
    # tile mask
    pos_mask = pos_mask.repeat(2, 2)
    neg_mask = neg_mask.repeat(2, 2)
    # mask-out self-contrast cases
    logits_mask = torch.scatter(
        torch.ones_like(pos_mask),
        1,
        torch.arange(batch_size).view(-1, 1).to(device),
        0
    )
    pos_mask = pos_mask * logits_mask
    # compute log_prob
    exp_logits = torch.exp(logits) * logits_mask
    exp_logits = exp_logits * (neg_mask+pos_mask)
    log_prob = logits - torch.log(exp_logits.sum(1, keepdim=True))
    # compute mean of log-likelihood over positive
    mean_log_prob_pos = (pos_mask * log_prob).sum(1) / pos_mask.sum(1)

    # loss
    loss = - mean_log_prob_pos
    loss = loss.view(2, batch_size).mean()
```

$l_{\text{Ins}}(\tilde{h}_i, \tilde{h}_i')$



```
mean1, disp1, pi1, rep1 = self.model(view1)
mean2, disp2, pi2, rep2 = self.model(view2)

rep1 = rep1[:batch_size]
rep2 = rep2[:batch_size]

ins_loss = self.model.ins_contrastive_loss(rep1, rep2, device=self.args.device)
```

$\mathcal{L}_{\text{Ins}}$

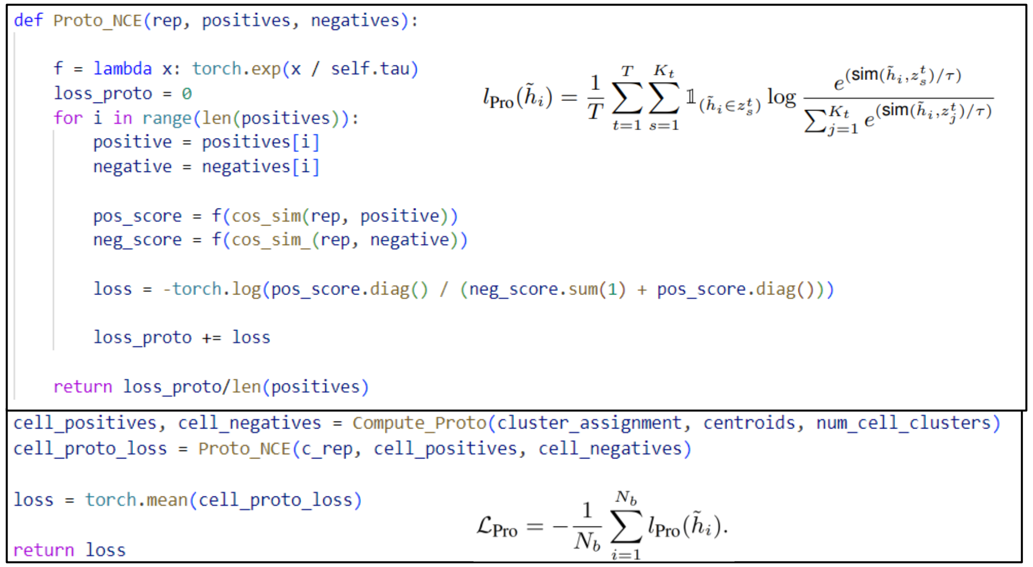
$$l_{\text{Ins}}(\tilde{h}_i, \tilde{h}_i') = \log \frac{e^{(\mathsf{sim}(\tilde{h}_i, \tilde{h}_i')/\tau)}}{\sum_{j=1}^{N_b} \mathbb{1}_{[i \neq j]} e^{(\mathsf{sim}(\tilde{h}_i, \tilde{h}_j)/\tau)} + \sum_{j=1}^{N_b} e^{(\mathsf{sim}(\tilde{h}_i, \tilde{h}_j')/\tau)}}$$

$$\mathcal{L}_{\text{Ins}} = -\frac{1}{2N_b} \sum_{i=1}^{N_b} [l_{\text{Ins}}(\tilde{h}_i, \tilde{h}_i') + l_{\text{Ins}}(\tilde{h}_i', \tilde{h}_i)].$$

Implementation - Performance on nine real scRNA-seq datasets

Phase 1: Pre-training

✓ Prototypical Contrastive Learning

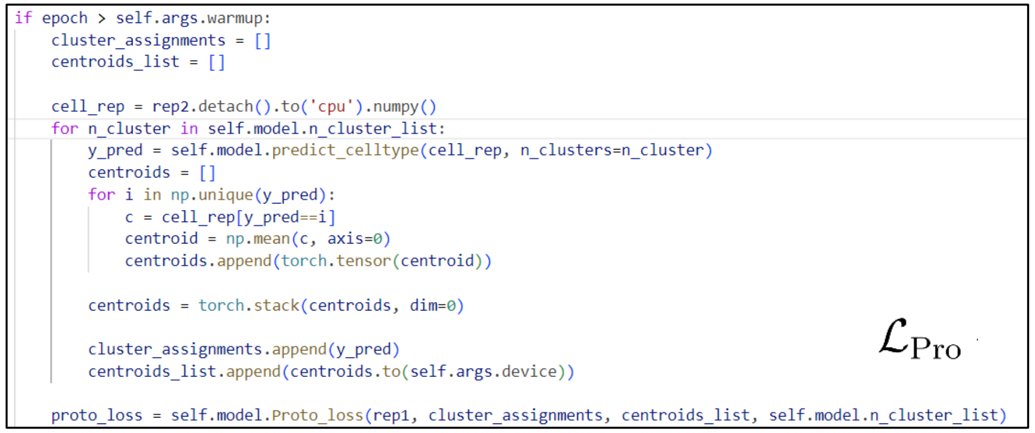


```
def Proto_NCE(rep, positives, negatives):

    f = lambda x: torch.exp(x / self.tau)
    loss_proto = 0
    for i in range(len(positives)):
        positive = positives[i]
        negative = negatives[i]

        pos_score = f(cos_sim(rep, positive))
        neg_score = f(cos_sim_(rep, negative))

        loss = -torch.log(pos_score.diag() / (neg_score.sum(1) + pos_score.diag()))

        loss_proto += loss

    return loss_proto/len(positives)
```

$$l_{\text{Pro}}(\tilde{h}_i) = \frac{1}{T}\sum_{t=1}^{T}\sum_{s=1}^{K_t}\mathbb{1}_{(\tilde{h}_i \in z_s^t)} \log \frac{e^{(\text{sim}(\tilde{h}_i, z_s^t)/\tau)}}{\sum_{j=1}^{K_t} e^{(\text{sim}(\tilde{h}_i, z_j^t)/\tau)}}$$

```
cell_positives, cell_negatives = Compute_Proto(cluster_assignment, centroids, num_cell_clusters)
cell_proto_loss = Proto_NCE(c_rep, cell_positives, cell_negatives)

loss = torch.mean(cell_proto_loss)

return loss
```

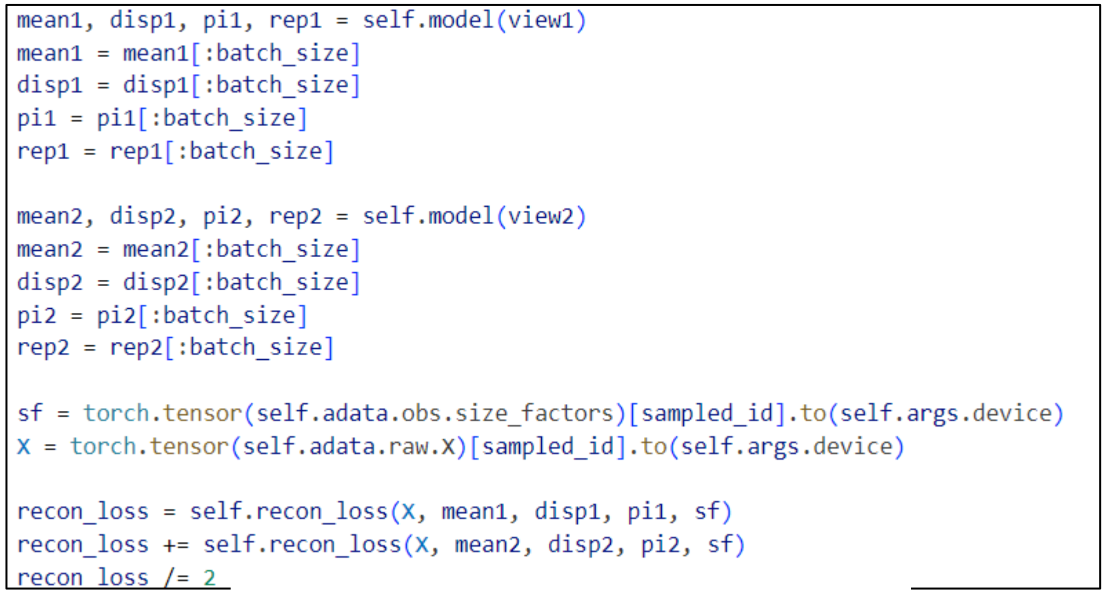
$$\mathcal{L}_{\text{Pro}} = -\frac{1}{N_b}\sum_{i=1}^{N_b} l_{\text{Pro}}(\tilde{h}_i).$$



```
if epoch > self.args.warmup:
    cluster_assignments = []
    centroids_list = []

    cell_rep = rep2.detach().to('cpu').numpy()
    for n_cluster in self.model.n_cluster_list:
        y_pred = self.model.predict_celltype(cell_rep, n_clusters=n_cluster)
        centroids = []
        for i in np.unique(y_pred):
            c = cell_rep[y_pred==i]
            centroid = np.mean(c, axis=0)
            centroids.append(torch.tensor(centroid))

        centroids = torch.stack(centroids, dim=0)

        cluster_assignments.append(y_pred)
        centroids_list.append(centroids.to(self.args.device))

    proto_loss = self.model.Proto_loss(rep1, cluster_assignments, centroids_list, self.model.n_cluster_list)
```

$\mathcal{L}_{\text{Pro}}$

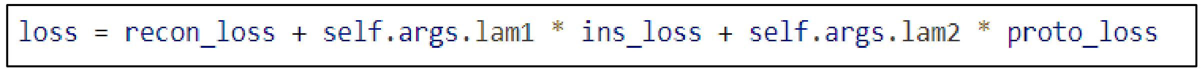
✓ ZINB-based Reconstruction Loss



```
mean1, disp1, pi1, rep1 = self.model(view1)
mean1 = mean1[:batch_size]
disp1 = disp1[:batch_size]
pi1 = pi1[:batch_size]
rep1 = rep1[:batch_size]

mean2, disp2, pi2, rep2 = self.model(view2)
mean2 = mean2[:batch_size]
disp2 = disp2[:batch_size]
pi2 = pi2[:batch_size]
rep2 = rep2[:batch_size]

sf = torch.tensor(self.adata.obs.size_factors)[sampled_id].to(self.args.device)
X = torch.tensor(self.adata.raw.X)[sampled_id].to(self.args.device)

recon_loss = self.recon_loss(X, mean1, disp1, pi1, sf)
recon_loss += self.recon_loss(X, mean2, disp2, pi2, sf)
recon_loss /= 2
```

$$\mathcal{L}_{\text{ZINB}}^{\text{Pre}} = \frac{1}{2}[l_{\text{ZINB}}(\tilde{\Pi}, \tilde{M}, \tilde{\Theta}) + l_{\text{ZINB}}(\tilde{\Pi}', \tilde{M}', \tilde{\Theta}')]$$

✓ Final Objectives of Fine-tuning Phase



```
loss = recon_loss + self.args.lam1 * ins_loss + self.args.lam2 * proto_loss
```

$$\mathcal{L}_{\text{Pre}} = \lambda_1\mathcal{L}_{\text{Ins}} + \lambda_2\mathcal{L}_{\text{Pro}} + \mathcal{L}_{\text{ZINB}}^{\text{Pre}}$$

Implementation - Performance on nine real scRNA-seq datasets

Phase 1: Pre-training – Results(AVG)

Hyperparameter	Value
Epoch	200
Warm-up	100
Learning Rate	0.0001
λ_1	1.0
λ_2	0.05

Data	Time(m)	Recon	ins-wise	proto
Camp	4.62	2.3741	3.8001	0.9113
Mouse ES cells	7.23	1.5999	4.0216	0.7791
Mouse Bladder cells	18.91	0.5596	3.899	1.6193
Zeisel	14.8	1.5074	4.0414	1.0882
Worm neuron cells	26.89	0.2161	3.9773	1.2303
10X PBMC	20.01	0.7074	3.9526	1.1718
Human kidney cells	40.02	0.6595	3.9165	1.3002
Baron	49.62	0.9186	3.9583	1.3442
Shekhar mouse retina cells	194.41	0.5999	4.1101	1.6186

Implementation - Performance on nine real scRNA-seq datasets

Phase 2: Fine-tuning

✓ Clustering Task-Oriented Loss

```
def soft_assign(self, z):
    q = 1.0 / (1.0 + torch.sum((z.unsqueeze(1) - self.mu)**2, dim=2) / self.alpha)
    q = q**((self.alpha+1.0)/2.0)
    q = (q.t() / torch.sum(q, dim=1)).t()
    return q
def target_distribution(self, q):
    p = q**2 / q.sum(0)
    return (p.t() / p.sum(1)).t()
```

$p_{ik} = \frac{q_{ik}^2/f_k}{\sum_{j=1}^{K} q_{ij}^2/f_j}$

```
def cluster_loss(self, p, q):
    def kld(target, pred):
        return torch.mean(torch.sum(target*torch.log(target/(pred+1e-6)), dim=-1))
    kldloss = kld(p, q)
    return kldloss
```

$D_{\text{KL}}(P\|Q)$

```
latent = self.model.predict_full_cell_rep(self.eval_loader, self.gene_embedding)
q = self.model.soft_assign(torch.tensor(latent).to(self.args.device))
p = self.model.target_distribution(q).data
qbatch = self.model.soft_assign(rep)
pbatch = p[sampled_id]
target = Variable(pbatch).to(self.args.device)

cluster_loss = self.model.cluster_loss(target, qbatch)
```

$\mathcal{L}_{\text{Cluster}}$

✓ ZINB-based Reconstruction Loss

```
for batch in self.train_loader1:
    self.model.train()
    batch_size = batch['cell'].batch_size
    sampled_id = batch['cell'].n_id[:batch_size]

    gene_emb = self.gene_embedding.weight[batch['gene'].n_id].to(self.args.device)
    batch['gene'].x = gene_emb
    batch = self.transform1(batch).to(self.args.device)

    mean, disp, pi, rep = self.model(batch)
    mean = mean[:batch_size]
    disp = disp[:batch_size]
    pi = pi[:batch_size]
    rep = rep[:batch_size]
```

```
sf = torch.tensor(self.adata.obs.size_factors)[sampled_id].to(self.args.device)
X = torch.tensor(self.adata.raw.X)[sampled_id].to(self.args.device)
recon_loss = self.recon_loss(X, mean, disp, pi, sf)
```

$\mathcal{L}_{\text{ZINB}}^{\text{Fine}}$

✓ Final Objectives of Fine-tuning Phase

```
loss = recon_loss + self.args.lam3 * cluster_loss
```

$$\mathcal{L}_{\text{Fine}} = \mathcal{L}_{\text{Cluster}} + \lambda_3 \mathcal{L}_{\text{ZINB}}^{\text{Fine}}$$

Implementation - Performance on nine real scRNA-seq datasets

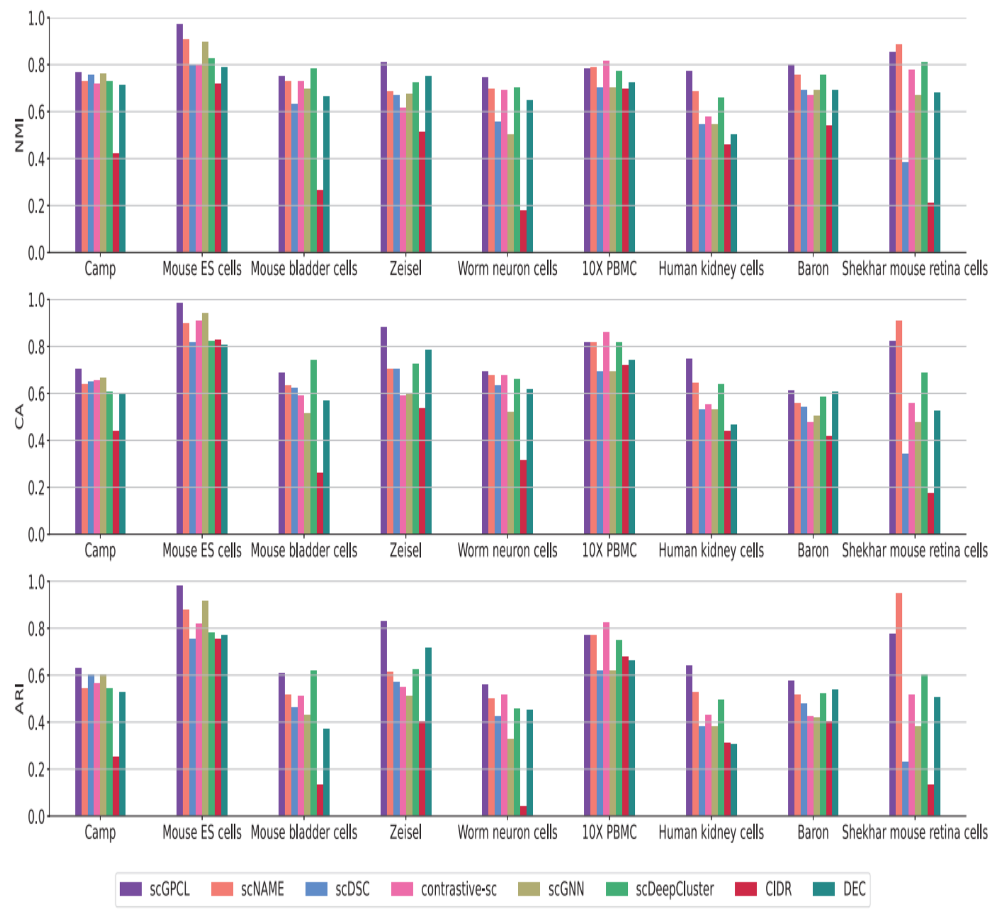
Phase 2: Fine-tuning - results

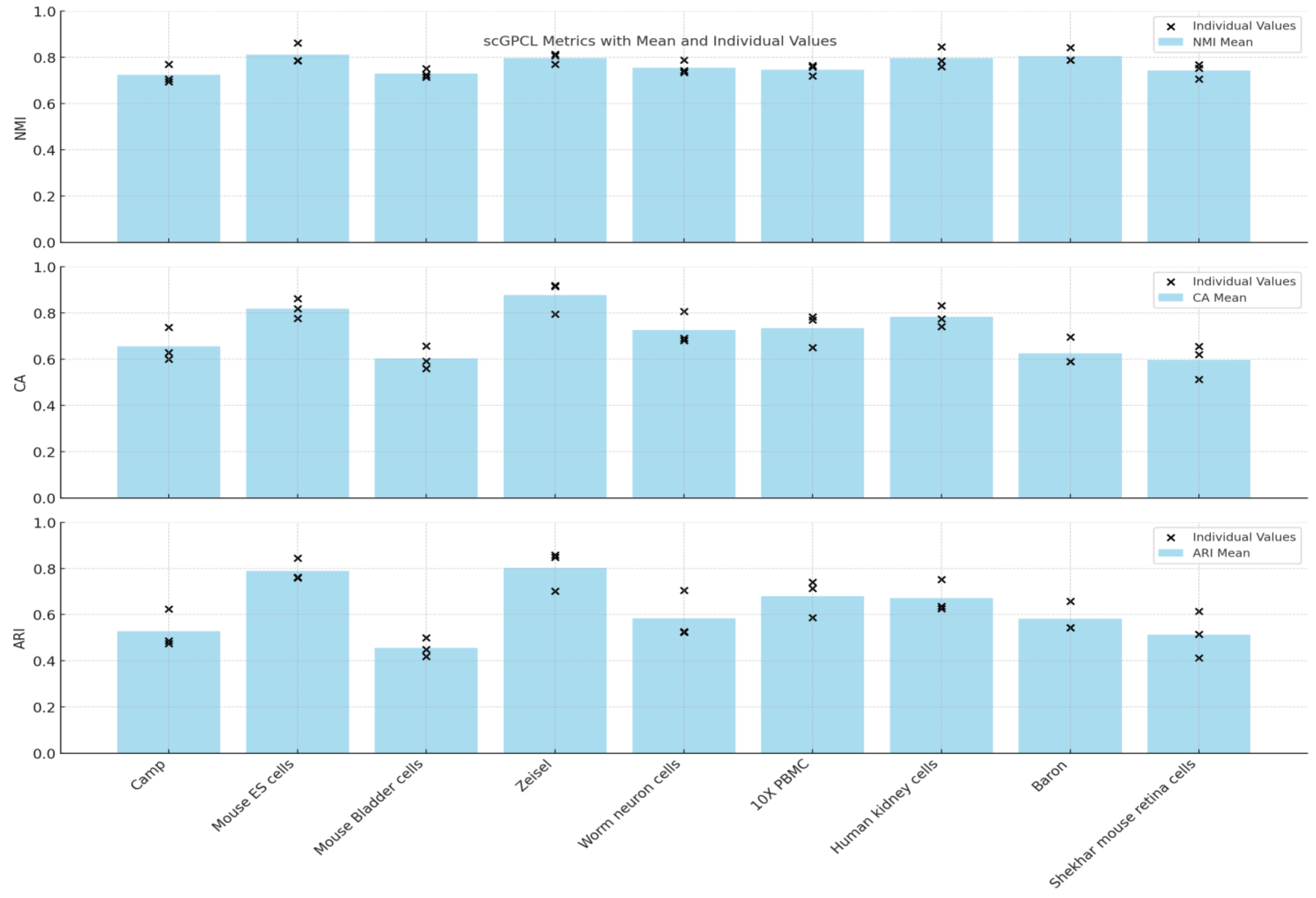
Hyperparameter	Value
Epoch	200
λ_3	1.0

Data	Time(m)	NMI	CA	ARI
Camp	0.71	0.7237	0.6551	0.5289
Mouse ES cells	5.48	0.8112	0.8187	0.7883
Mouse Bladder cells	9.09	0.7296	0.6022	0.4562
Zeisel	12.88	0.7965	0.8761	0.803
Worm neuron cells	7.42	0.755	0.7258	0.5848
10X PBMC	14.82	0.7468	0.7341	0.6805
Human kidney cells	24.84	0.797	0.7821	0.6716
Baron	42.24	0.8057	0.6243	0.5817
Shekhar mouse retina cells	142.74	0.7425	0.5958	0.5137

Implementation - Performance on nine real scRNA-seq datasets

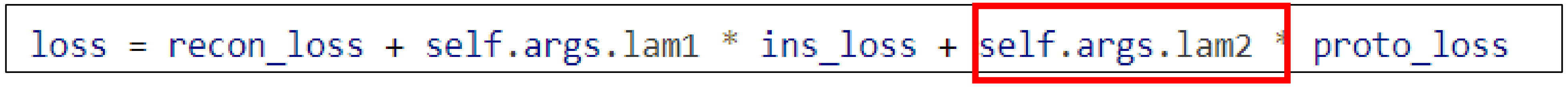
Results - Graph





Implementation2

Hyperparameter Analysis



```
loss = recon_loss + self.args.lam1 * ins_loss + self.args.lam2 * proto_loss
```

$$\mathcal{L}_{\text{Pre}} = \lambda_1 \mathcal{L}_{\text{Ins}} + \lambda_2 \mathcal{L}_{\text{Pro}} + \mathcal{L}_{\text{ZINB}}^{\text{Pre}}$$

λ_1 (lam1) : 1.0
λ_2 (lam2) : 0.05

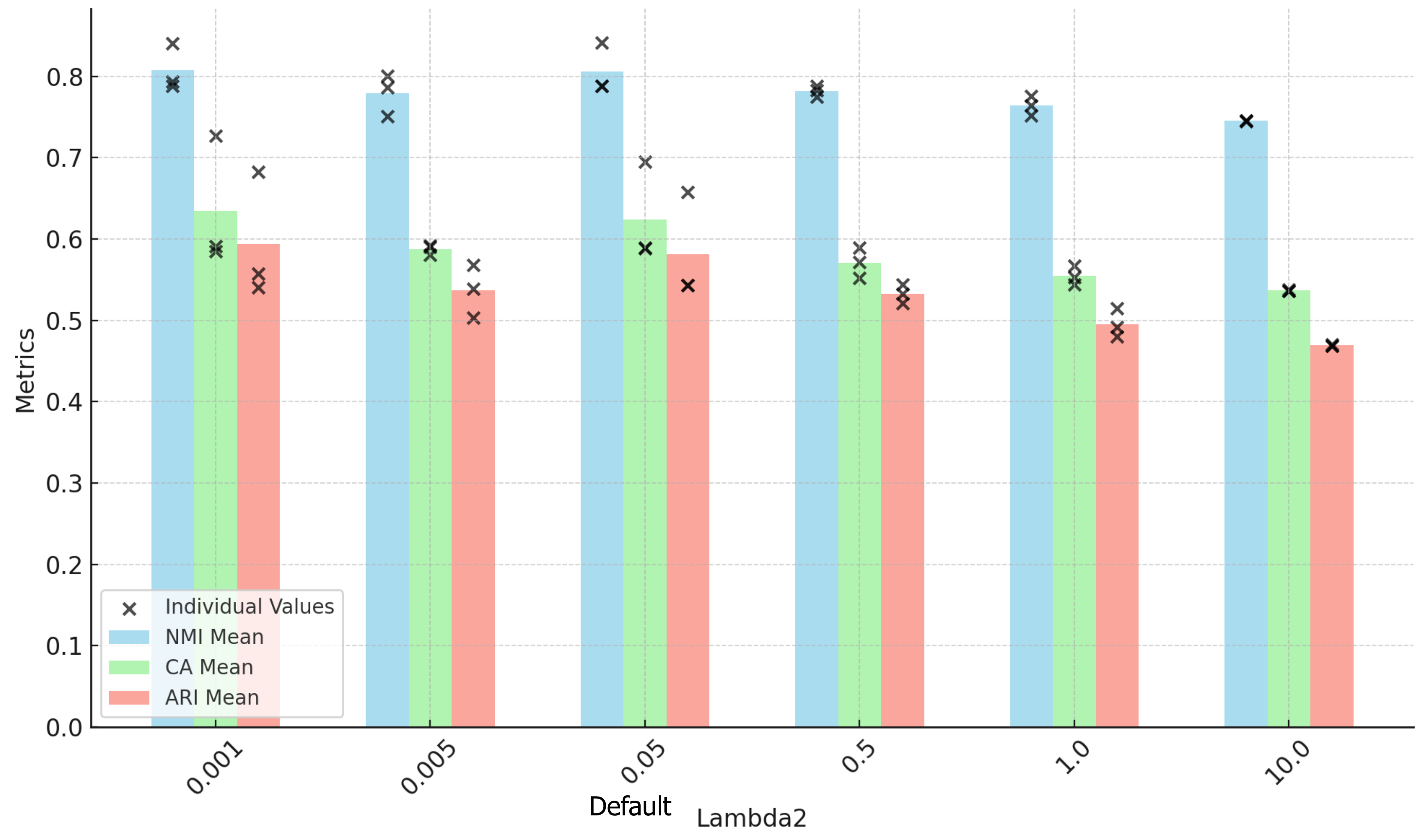
- Why $\lambda_1 >> \lambda_2$?

-> Conducted experiments three times for each λ_2 value using the Baron dataset (0.001, 0.005, 0.05(default), 0.5, 1.0, 10)

- Check Effect of λ_2

Implementation2

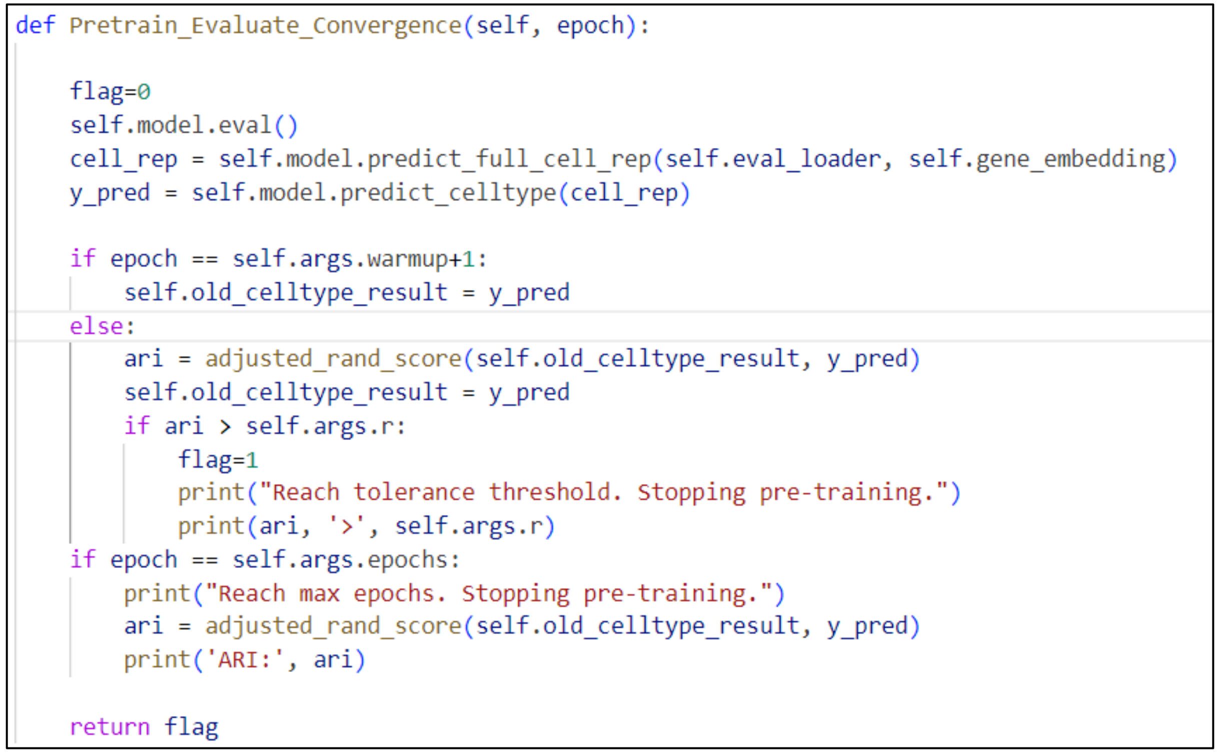
Hyperparameter Analysis



Implementation2

Hyperparameter Analysis

λ_2	Pre-training epoch			Avg
0.001	200	200	200	200
0.005	200	200	200	200
0.05 (Default)	200	200	118	173
0.5	115	115	107	112
1.0	114	111	106	110
10.0	108	109	108	108

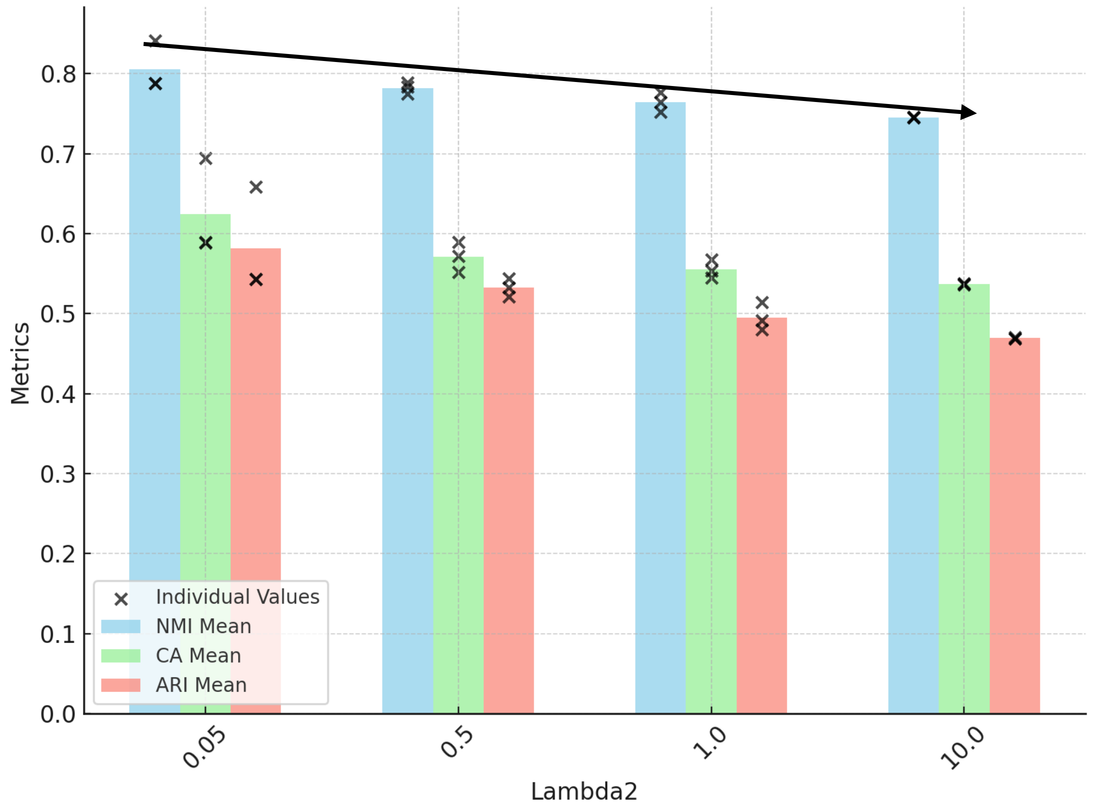


```python
def Pretrain_Evaluate_Convergence(self, epoch):

    flag=0
    self.model.eval()
    cell_rep = self.model.predict_full_cell_rep(self.eval_loader, self.gene_embedding)
    y_pred = self.model.predict_celltype(cell_rep)

    if epoch == self.args.warmup+1:
        self.old_celltype_result = y_pred
    else:
        ari = adjusted_rand_score(self.old_celltype_result, y_pred)
        self.old_celltype_result = y_pred
        if ari > self.args.r:
            flag=1
            print("Reach tolerance threshold. Stopping pre-training.")
            print(ari, '>', self.args.r)
    if epoch == self.args.epochs:
        print("Reach max epochs. Stopping pre-training.")
        ari = adjusted_rand_score(self.old_celltype_result, y_pred)
        print('ARI:', ari)

    return flag
```

- When λ_2 is larger, the pre-training terminated earlier.
- Early stopping criteria : the clustering results from the current epoch are similar to those of the previous epoch (ARI > 0.99)

Implementation2

Hyperparameter Analysis



This may suggest that...

Prototypical contrastive stabilizes clusters by minimizing the distance to prototypes

-> Early Convergence

-> Cell representations are less learned -> Performance Drop

Conclusion

- Using GNNs on a cell-gene bipartite graph, scGPCL effectively captures relational information and combines instance-wise and prototypical contrastive learning to improve clustering
- Experiments show its robustness and effectiveness.

Reference

- ebiogen. *Single-cell RNA Sequencing 분석 기술 Technical Note*. Mar. 2021
- M. Minami et al. / Fisheries Research 84 (2007) 210–221
- Ciortan, Madalina, and Matthieu Defrance. "Contrastive self-supervised clustering of scRNA-seq data." *BMC bioinformatics* 22.1 (2021): 280.
- Chuang, Ching-Yao, et al. "Debiased contrastive learning." *Advances in neural information processing systems* 33 (2020): 8765-8775.
- Li, Junnan, et al. "Prototypical contrastive learning of unsupervised representations." *arXiv preprint arXiv:2005.04966* (2020).