

Chat with scGNN in scRNA-seq analysis

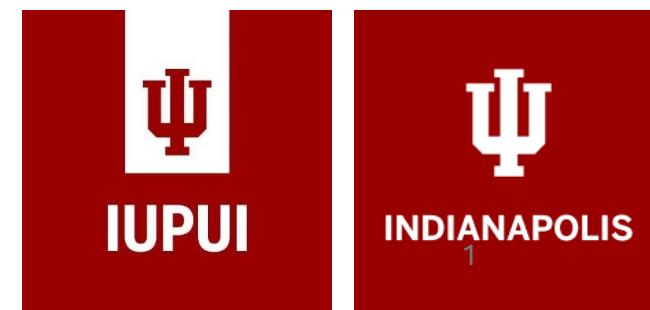
Juexin Wang

Department of BioHealth Informatics

Luddy School of Informatics, computing, and engineering

Indiana University Purdue University Indianapolis

07/12/2024



Data Challenges in Single cell RNA-seq

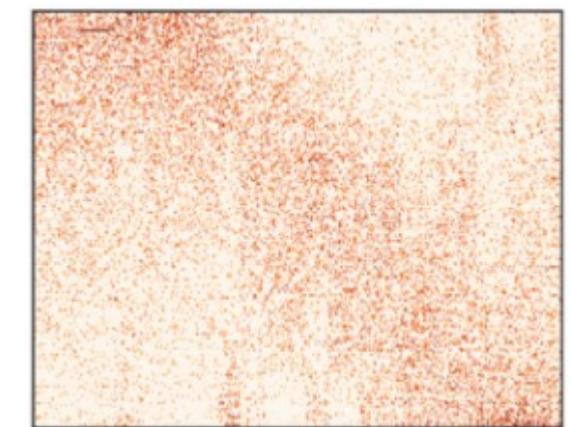
Big data challenge Containing up to millions of cells at a time

Complex differential patterns Highly heterogeneous

Sequencing sparsity Facing high dropout issue and data noise

- **Dropouts:** the data only captures a small fraction of the transcriptome of each cell
- Possible explanation of dropout:
 - Technical error
 - None zero expression will be identified as 0 with a certain probability
 - Low resolution
 - True expression level lower than a **resolution** cannot be identified

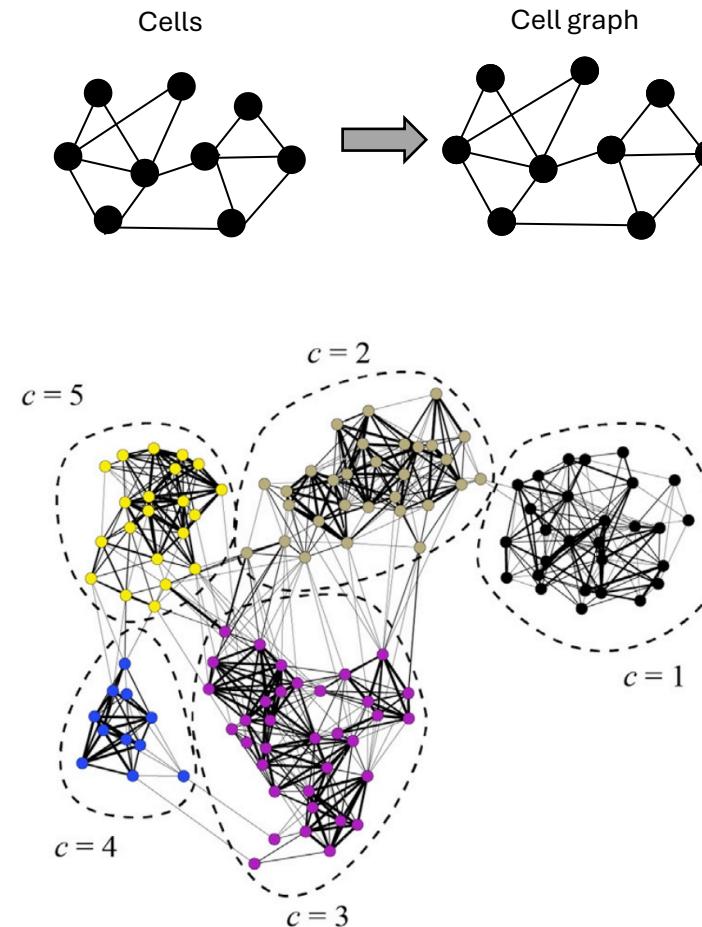
Undercounting
reads in
scRNA-seq



Qiu, *Nature communications*, (2020):11(1), 1-9.

Classical approach I: KNN graph

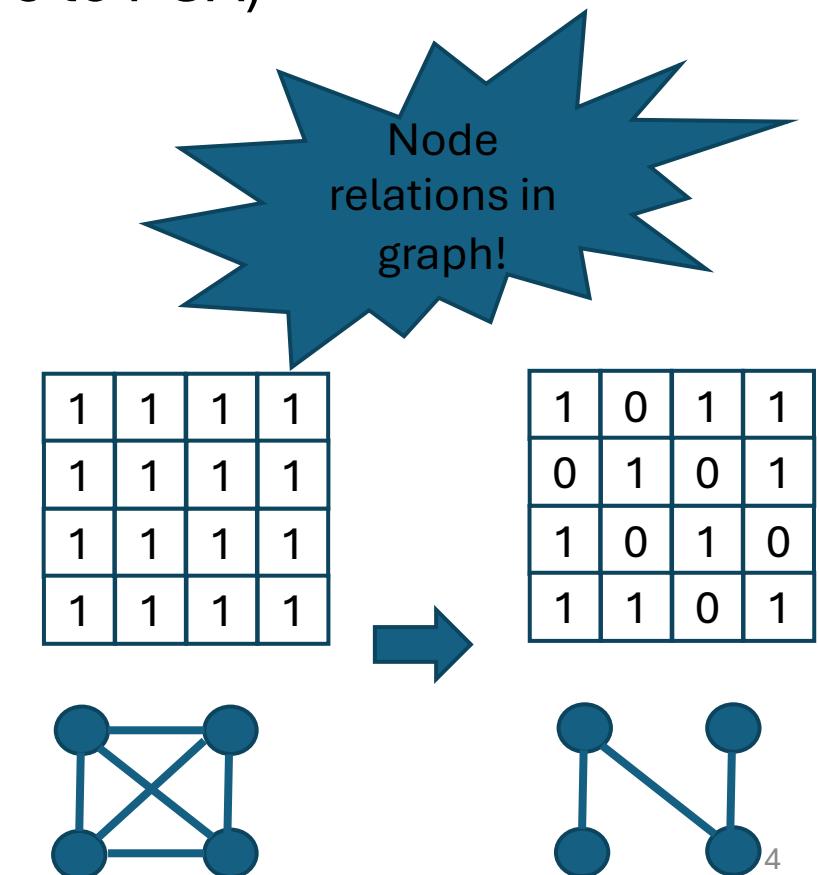
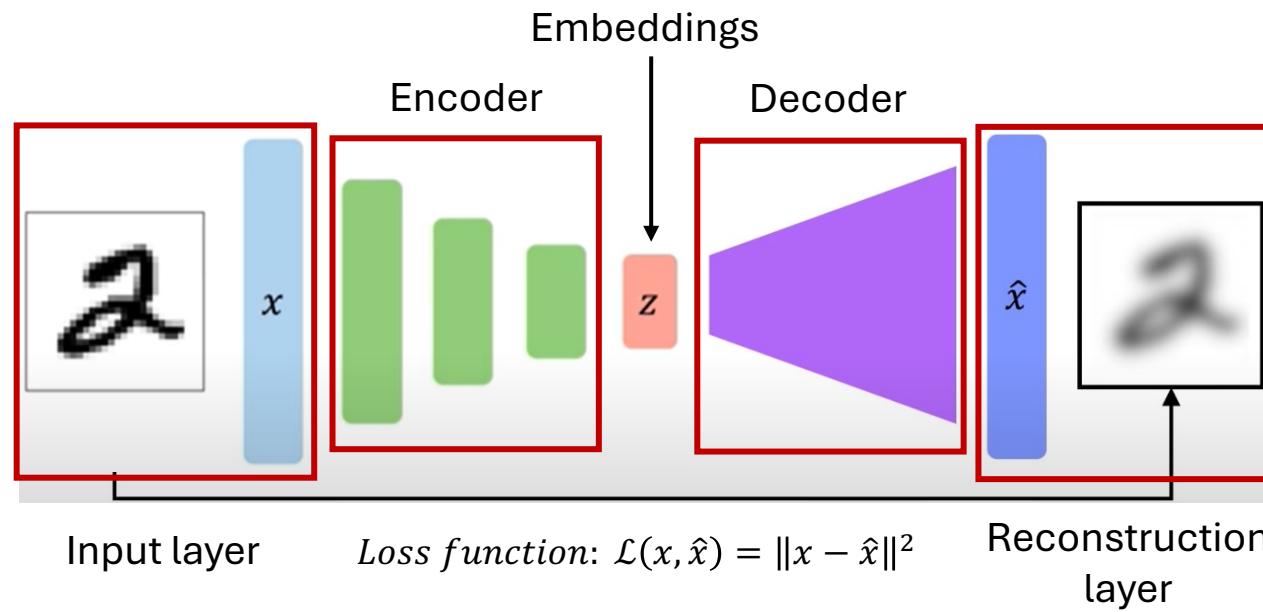
- **K-Nearest-Neighbor(KNN) graph:**
 - Nodes are individual single-cells
 - Edges are relations between cells
 - Each node finds its neighbors within the **K** shortest distances
 - Louvain clustering to find clusters
- Increase signal/noise ratio
- Adopted by: Seurat, Phenograph, MAGIC,
...
- May over-simplify the complex cell and gene relationships of the global cell population.



Levine, et al., *Cell*, (2015), 162(1), pp.184-197.

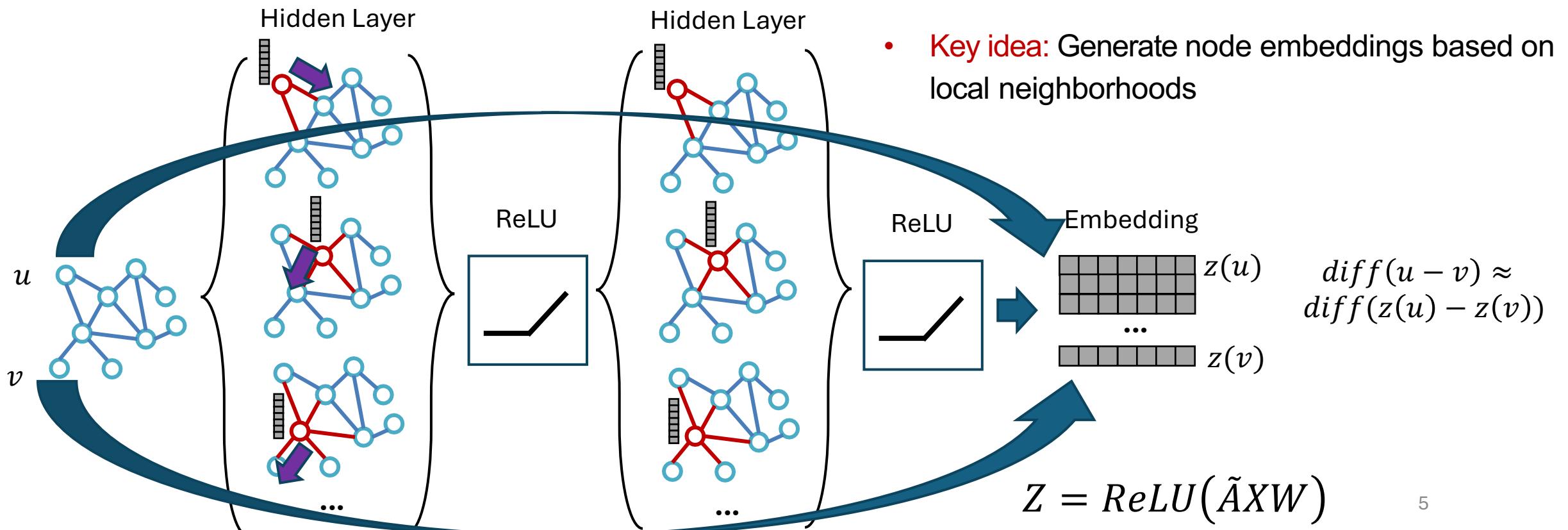
Classical approach II: Autoencoder

- Unsupervised framework for learning lower-dimensional features in deep learning.
 - **Non-linear** dimensional reduction (Alternative to PCA)
 - Data denoising and missing value recovery
- SAUCIE/DeeplImpute/DCA

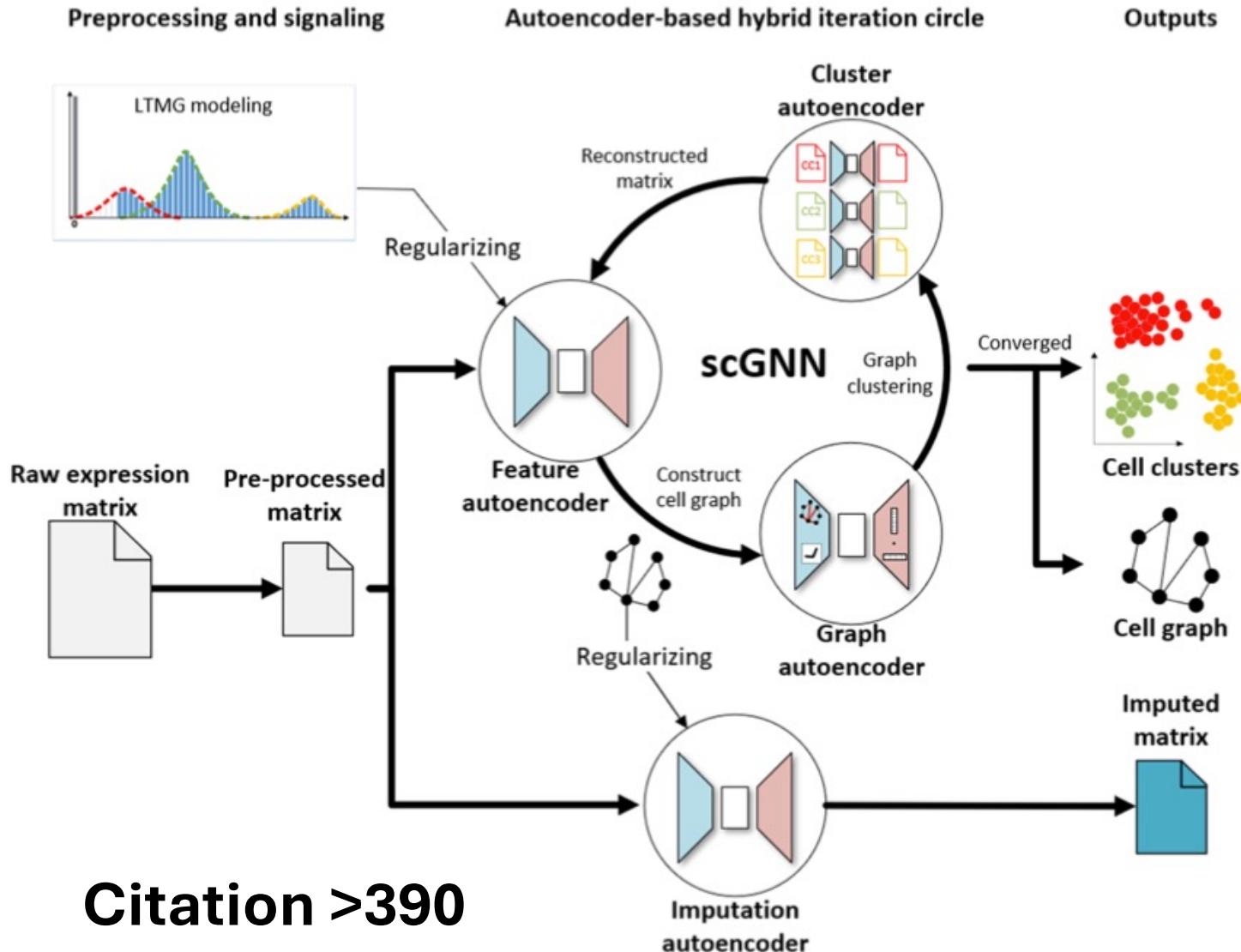


Learn representation in a graph using GNN

- Graph Neural Networks (**GNN**) learns a task-independent representation of a graph by deconvoluting node relationships through neighbor information propagation in a deep learning architecture.



scGNN (single cell Graph Neural Networks)

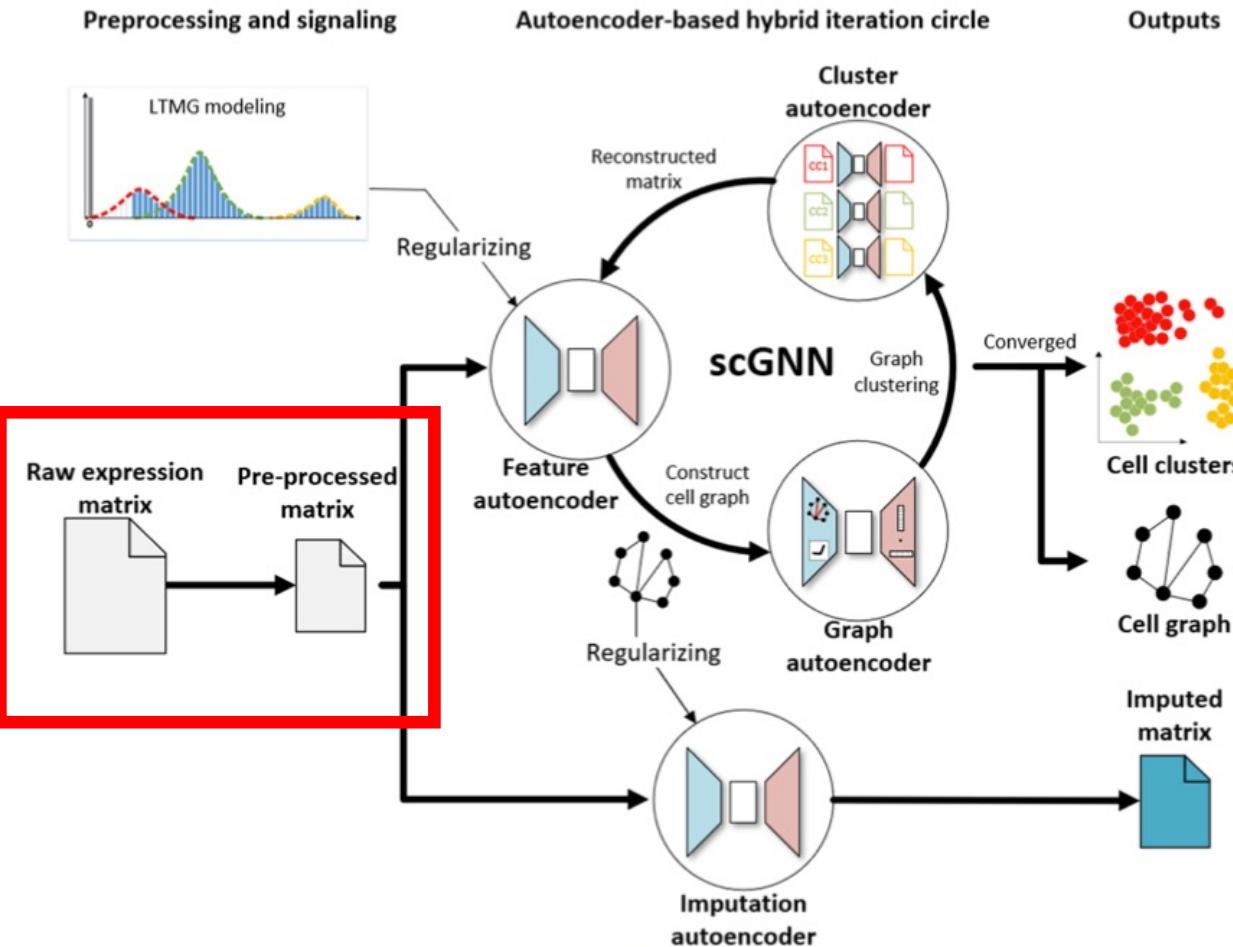


Citation >390

- **First GNN model on single cells**
- Simultaneous cell clustering and gene imputation
- Unsupervised multi-model deep learning framework
- Graph neural network + stacks of autoencoders
- Iterative build the cell graphs

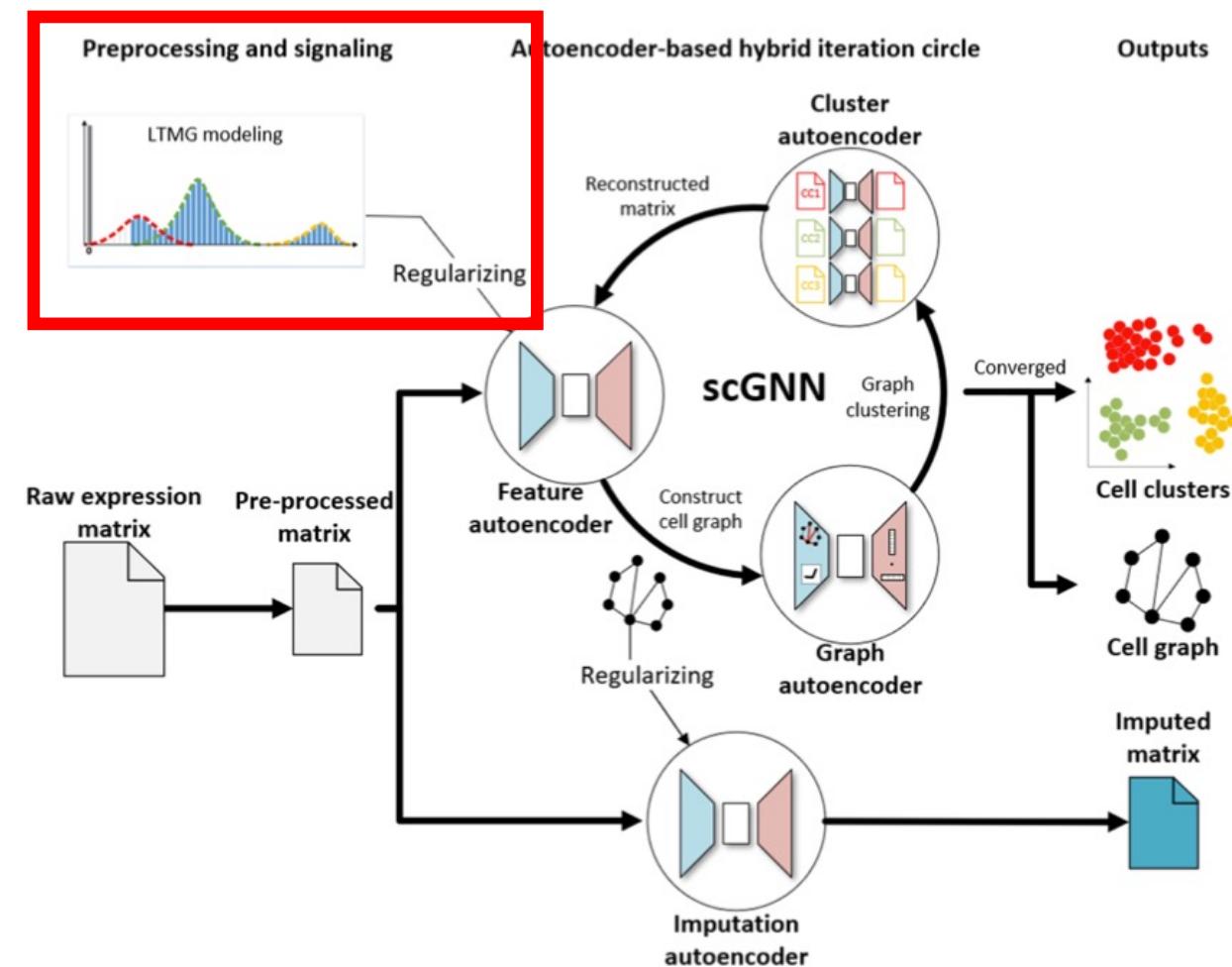
Juexin Wang*, Anjun Ma*, Yuzhou Chang, Jianting Gong, Yuexu Jiang, Ren Qi, Cankun Wang, Hongjun Fu, Qin Ma, and Dong Xu. "scGNN is a novel graph neural network framework for single-cell RNA-Seq analyses." *Nature communications* 12, no. 1 (2021): 1-11.

Preprocessing

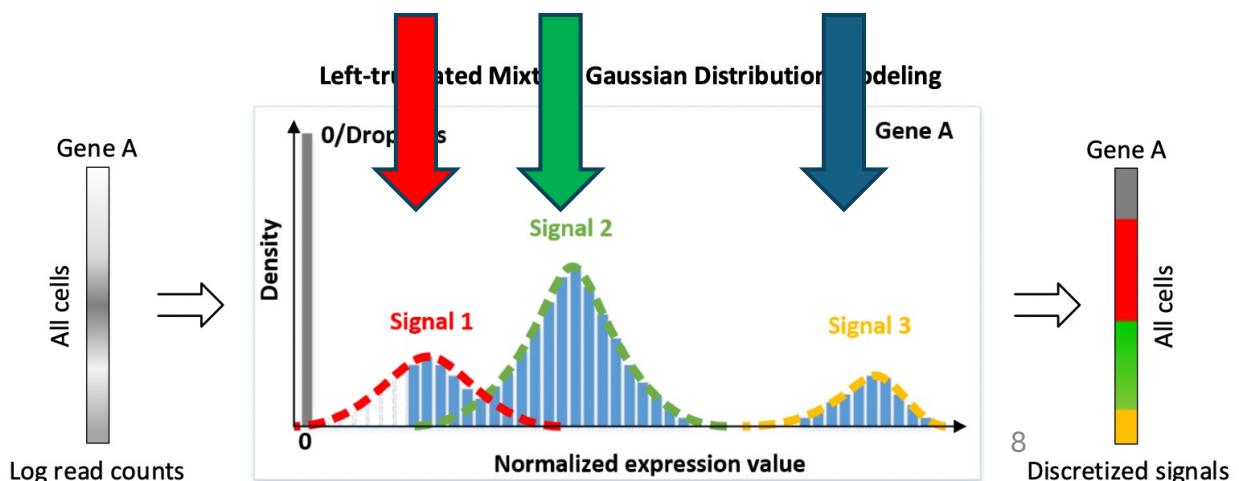


- scRNA-Seq gene expression profile as the input.
- Only genes expressed as nonzero in more than 1% of cells, and cells expressed as nonzero in more than 1% of genes are kept.
- Top 2,000 variant genes are used.
- All the data are log-transformed.

Integrating regulatory signals through LTMG

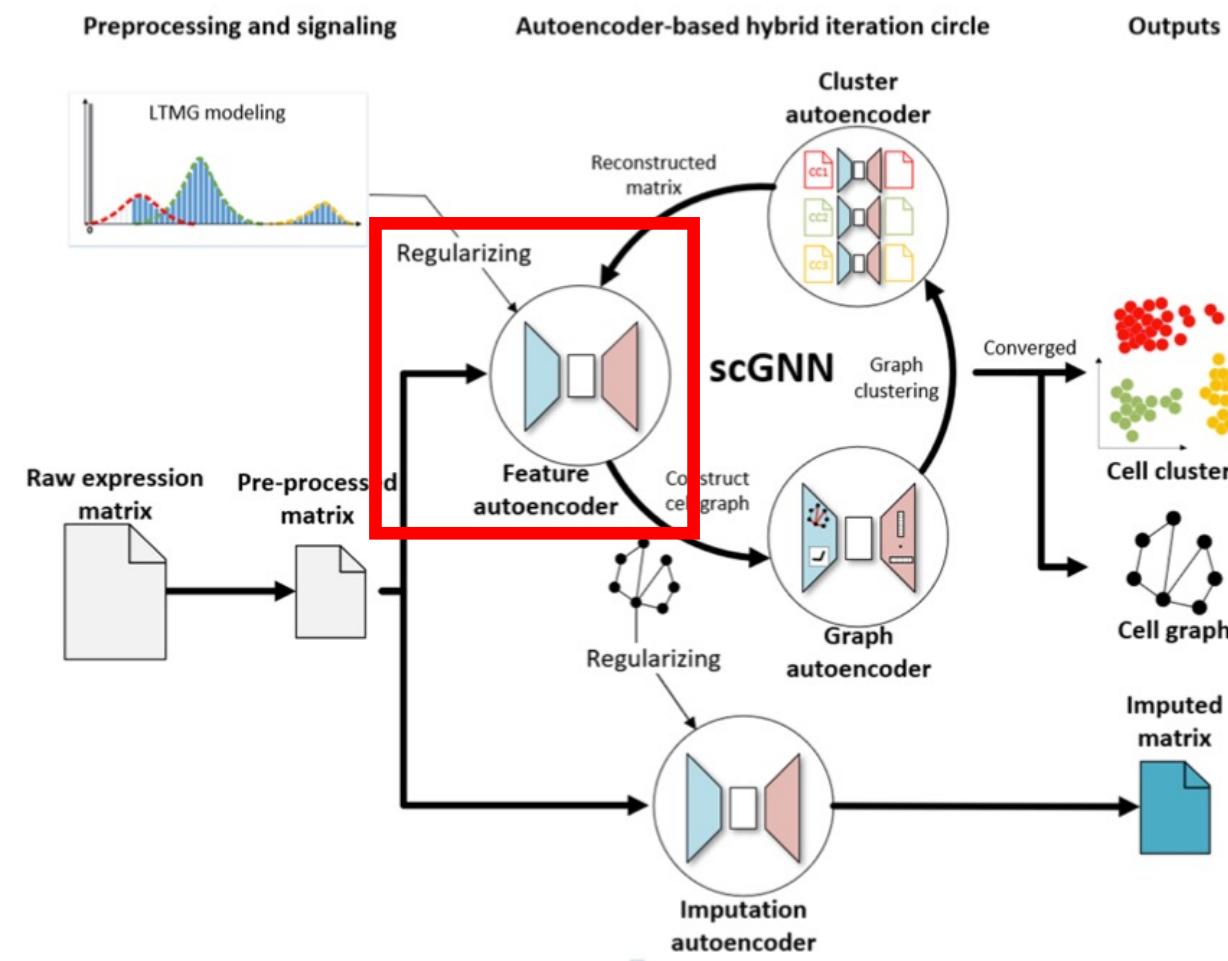


- **Left Truncation Mixed Gaussian** model is used to explore the regulatory signals from single cell gene expression.
- $\text{TRS} \in [1, 2, 3, \dots]$
- Regularizer: penalize genes/cells with different roles.



Wan, et al., 2019. *Nucleic acids research*, 47(18), pp.e111-e111.

Feature autoencoder



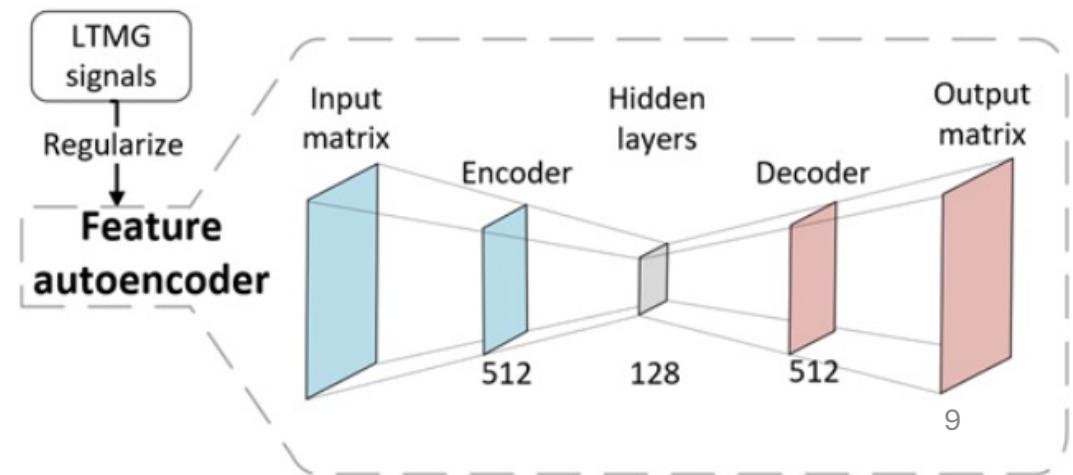
- The objective is to achieve a maximum similarity between the original and reconstructed expression through minimizing the loss function:

$$Loss = (1 - \alpha) \sum (X - \hat{X})^2 + \alpha \sum ((X - \hat{X})^2 \circ TRS)$$

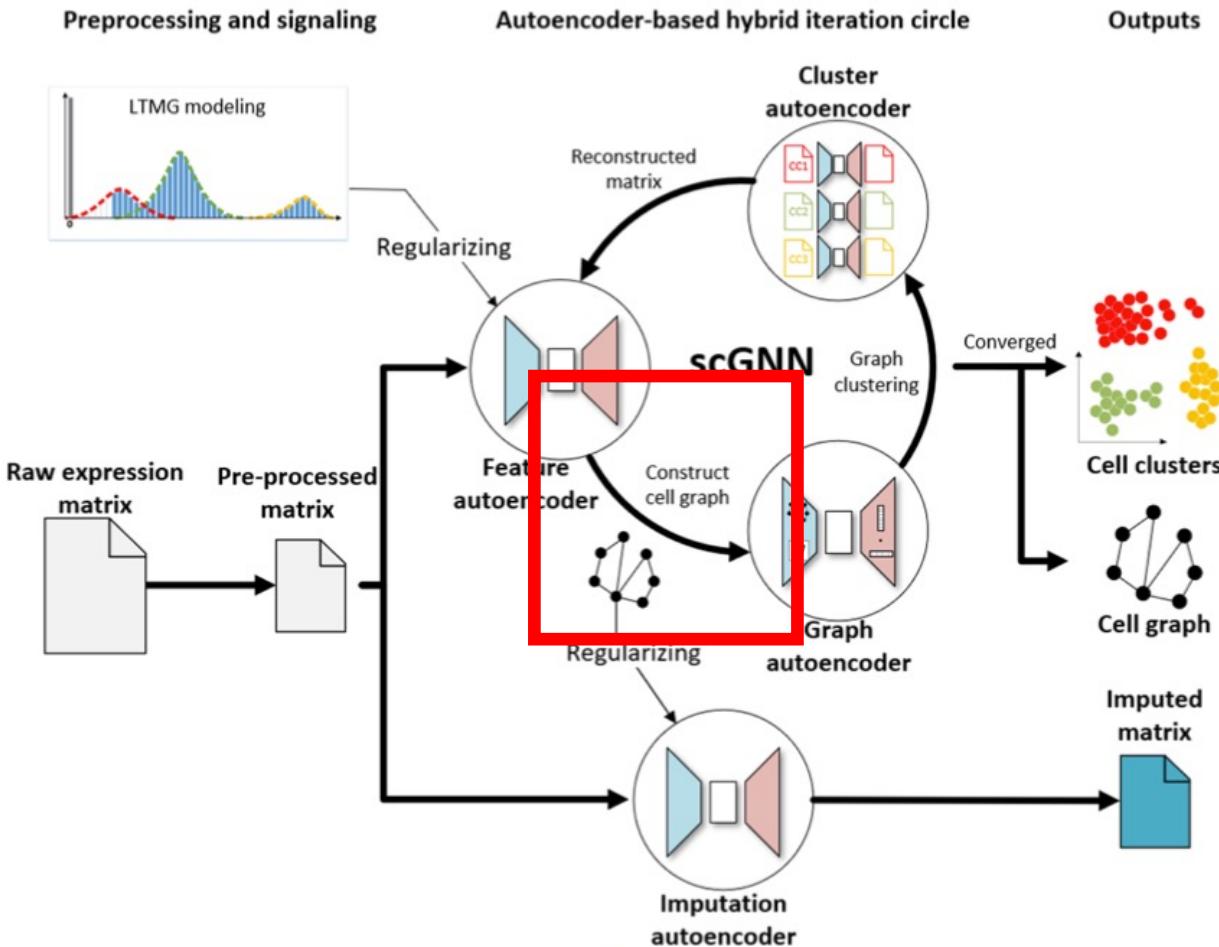
MSE

LTMG

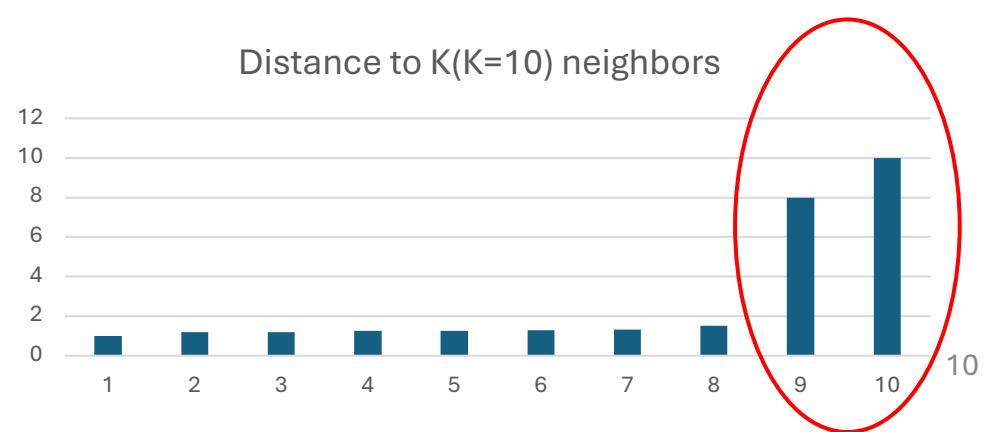
α controls the strength of gene regulation regularization; $\alpha \in [0,1]$.



Construct cell graph and pruning

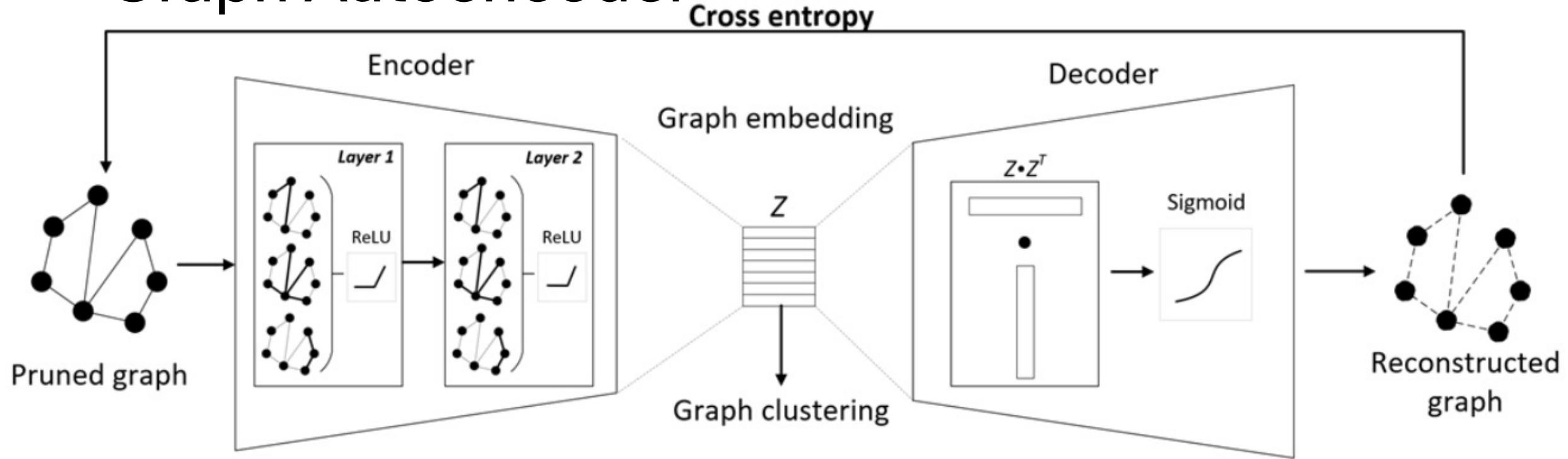


- The cell graph is built from a **KNN graph** by Euclidean distance
- Pruning process selects an adaptive number of neighbors for each node.
- **Isolation Forest** builds individual random forest to check distances from the node to all K neighbors and only disconnects the outliers.



Cell graph formulates cell-cell relations

Graph Autoencoder



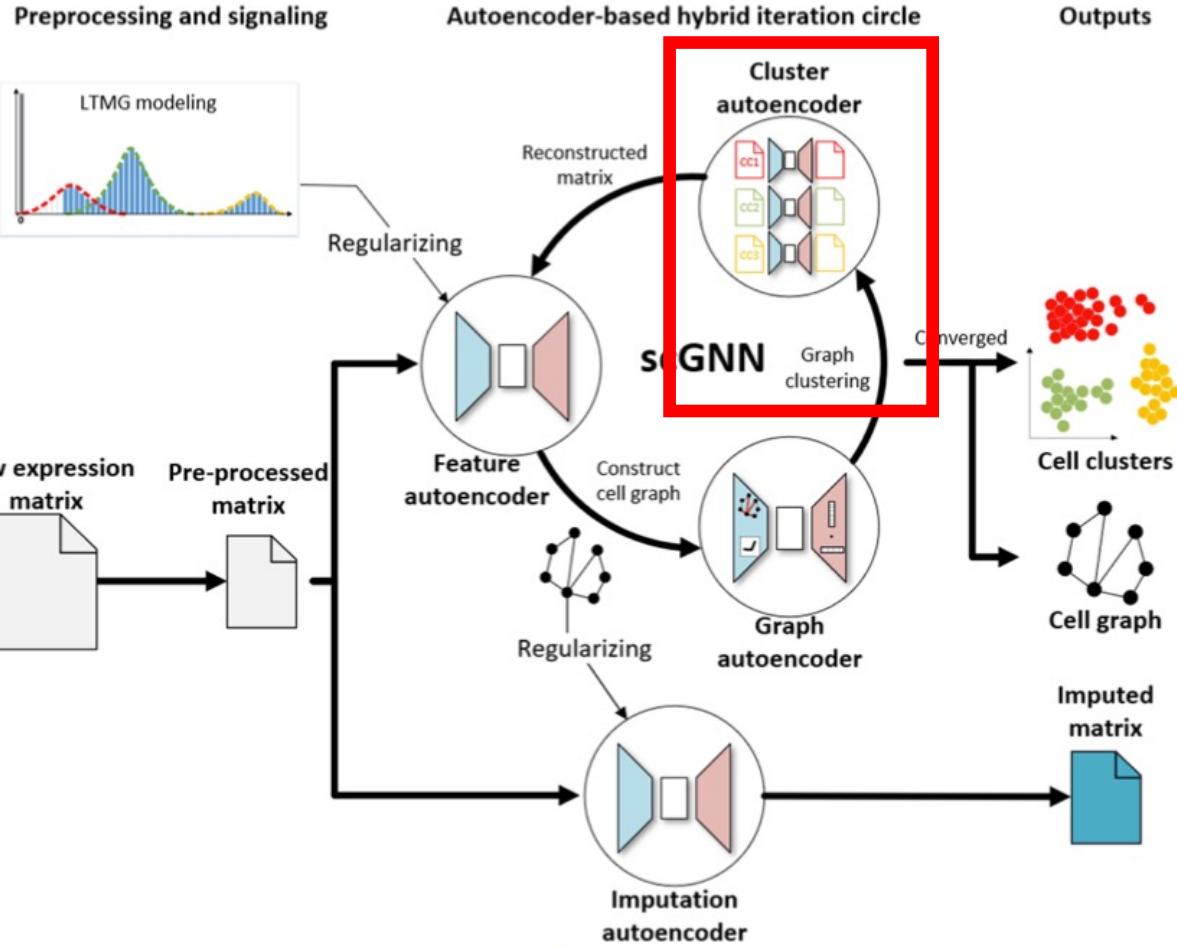
**Encode cell relationship
into vectors**

Classical Autoencoder: Node feature X
Graph Autoencoder: Topology A

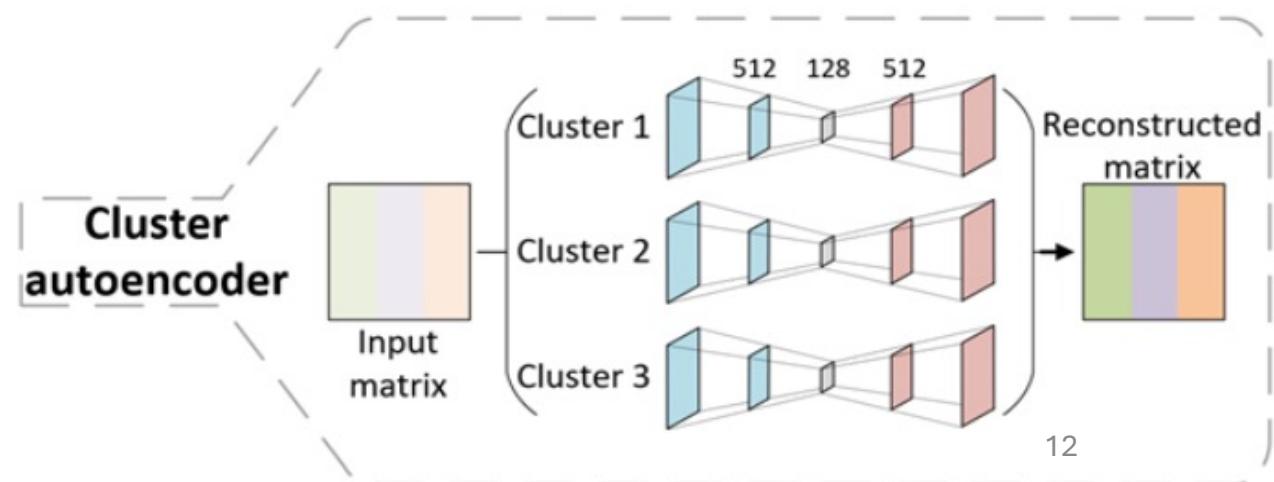
Goal: Minimize cross-entropy L between input adjacency matrix A and reconstructed matrix \hat{A} :

$$L(A, \hat{A}) = -\frac{1}{N \times N} \sum_{i=1}^N \sum_{j=1}^N (a_{ij} * \log(\hat{a}_{ij})) + (1 - a_{ij}) * \log(1 - \hat{a}_{ij})$$

Clustering and Clustering autoencoder

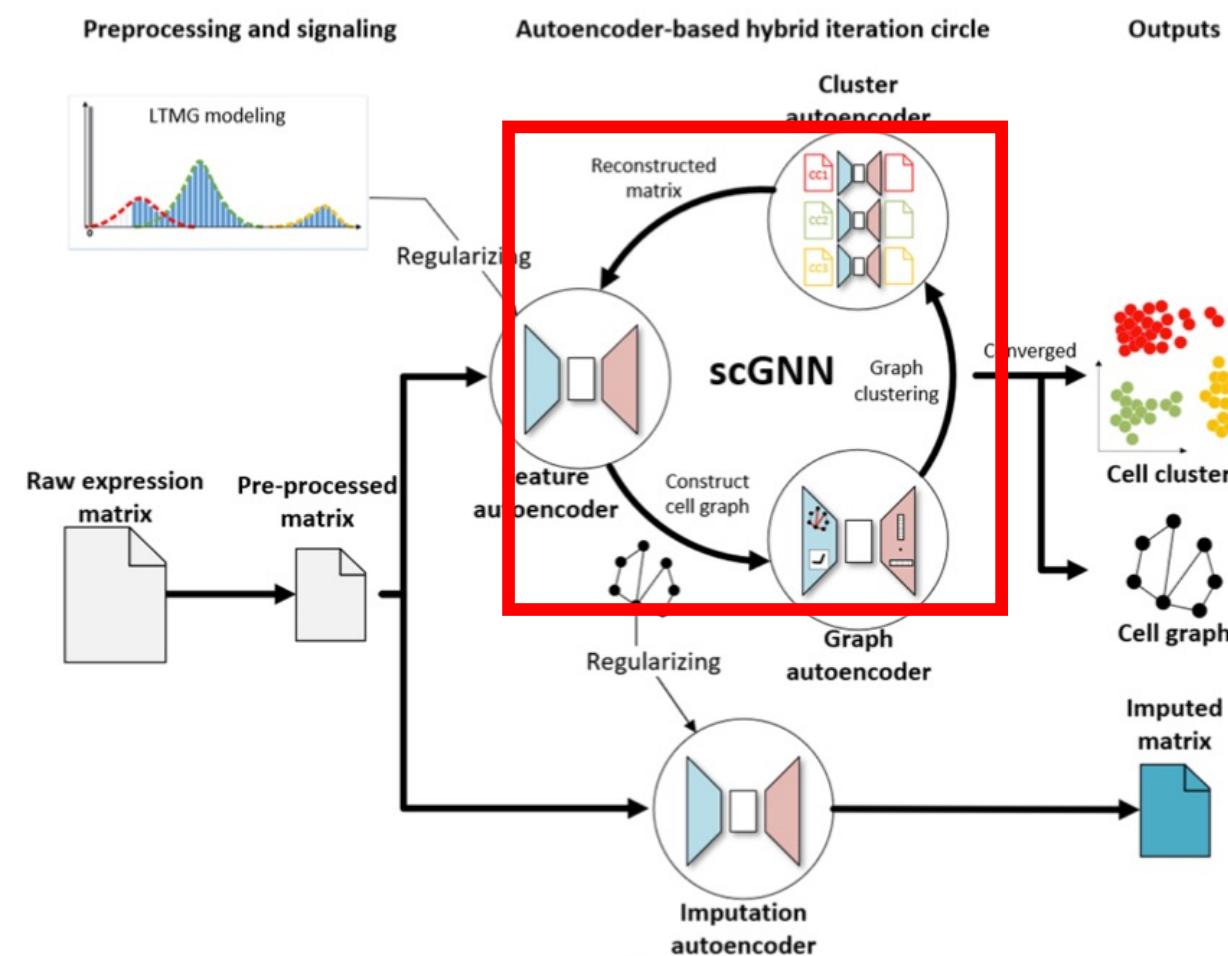


- Louvain determines cluster number K, use **K-means** clustering on graph autoencoder embedding.
- Cluster autoencoder treats different cell types specifically and regenerates expression in the same cell cluster.
- Leverages the inferences between **global** and **cell-type-specific** representation learning.



Build cell-type specific embedding

Iterative process

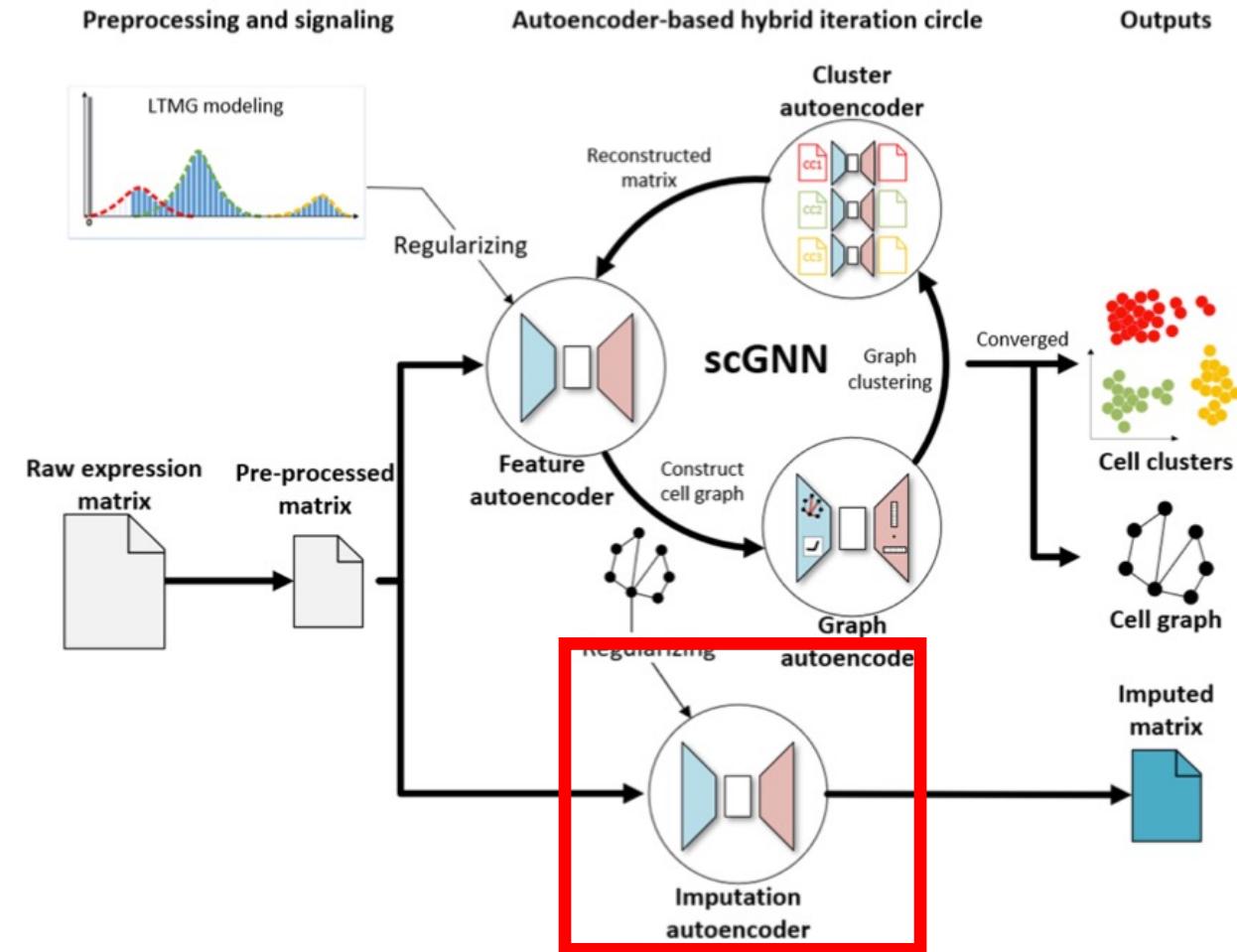


- Iterative process can be defined as:

$$\tilde{A} = \lambda L_0 + (1 - \lambda) \frac{A_{ij}}{\sum_j A_{ij}}$$

- L_0 : normalized adjacency matrix of the initial pruned graph
- λ : control the converging speed.
- Stop if the **adjacency matrix converges** or the inferred **cell types are similar enough**

Imputation by imputation autoencoder



- Imputation autoencoder imputes and denoises the raw expression matrix within the inferred cell-cell relationship.

Same as feature autoencoder

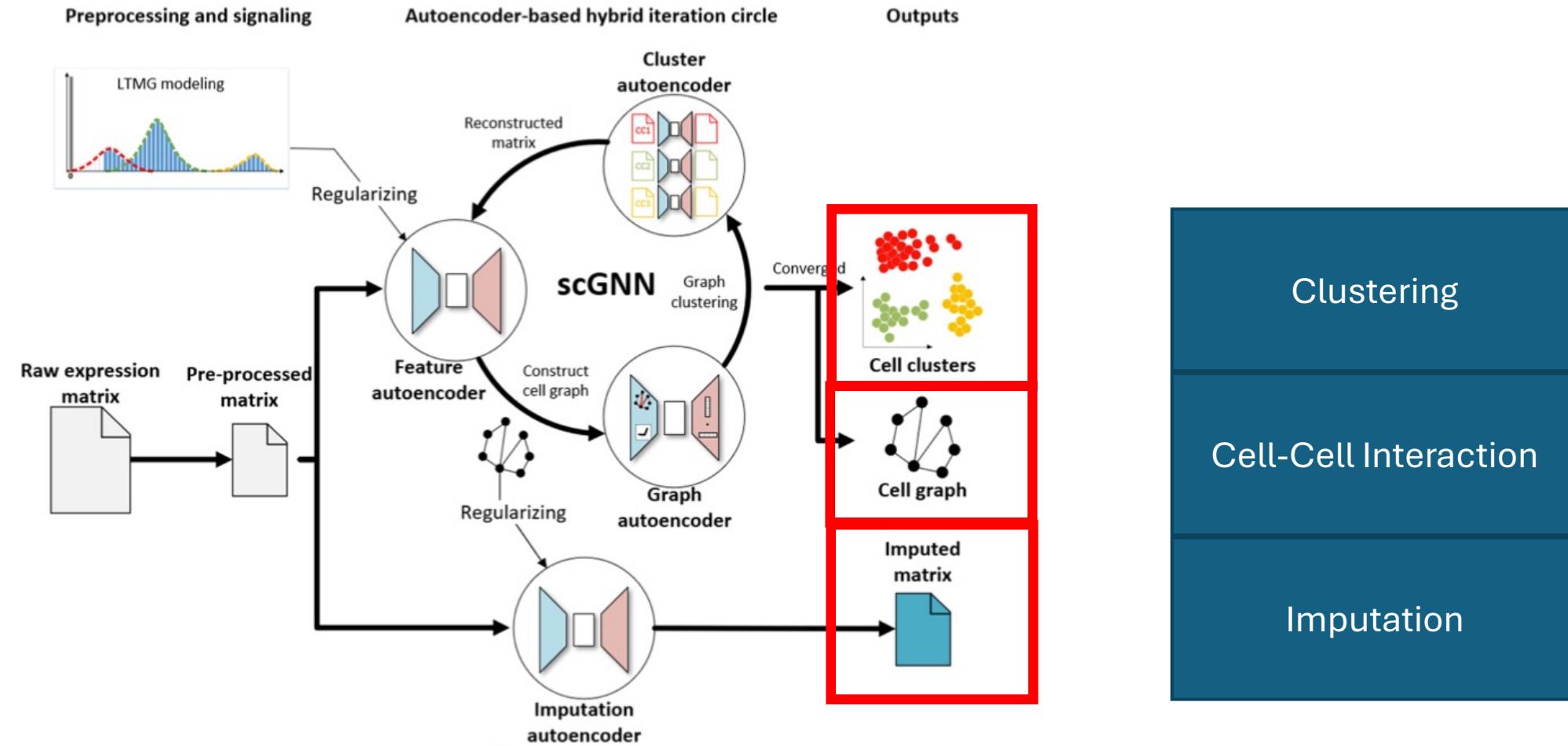
$$\text{Loss} = \boxed{(1 - \alpha) \sum (X - \hat{X})^2 + \alpha \sum ((X - \hat{X})^2 \circ TRS)}$$

$+ \beta \sum |w|$ L1 term to reduce non-zeros

$+ \gamma_1 \sum (A \cdot (X - \hat{X})^2)$ Cell graph regularizer

$+ \gamma_2 \sum (B \cdot (X - \hat{X})^2)$ Cell type regularizer

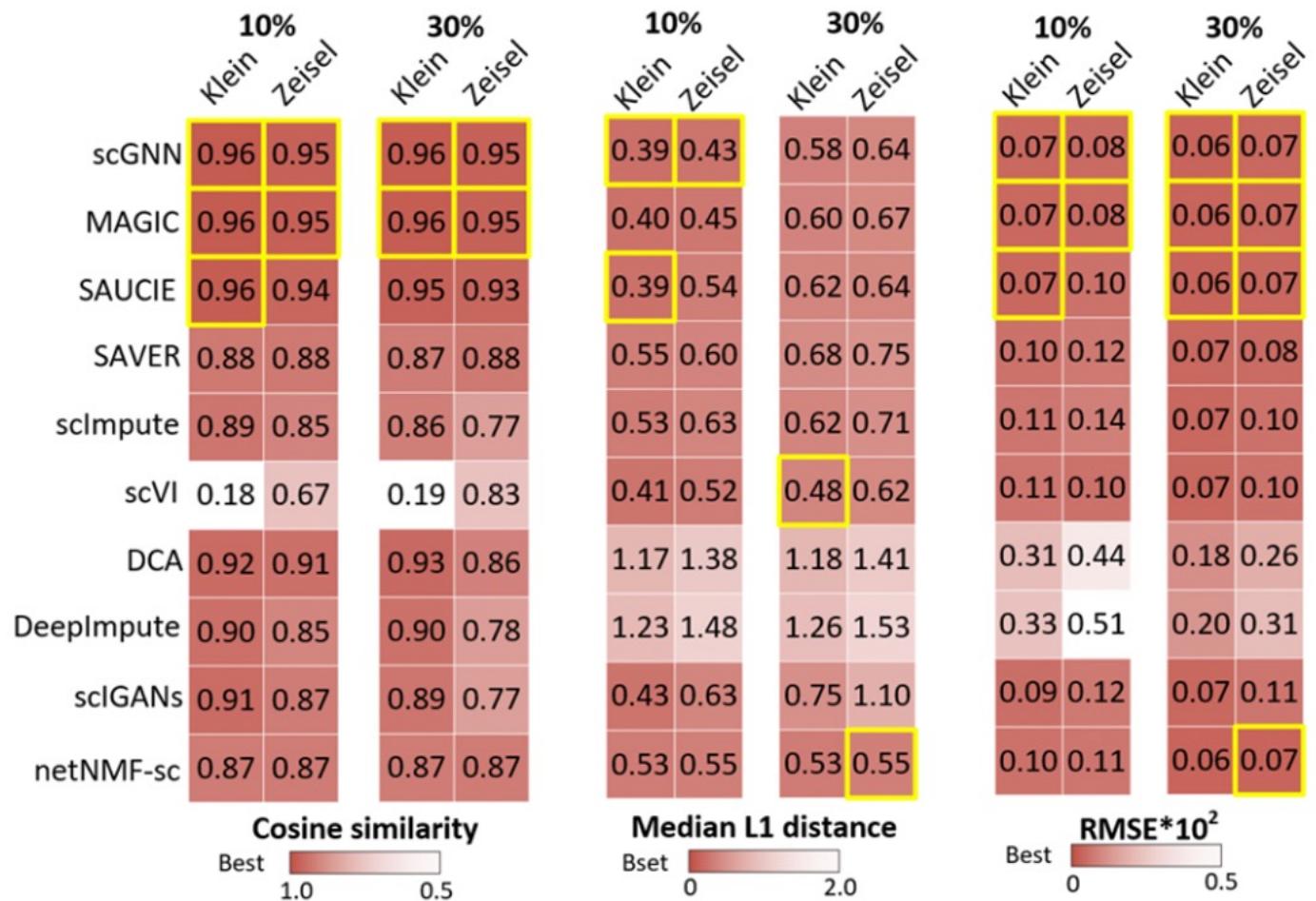
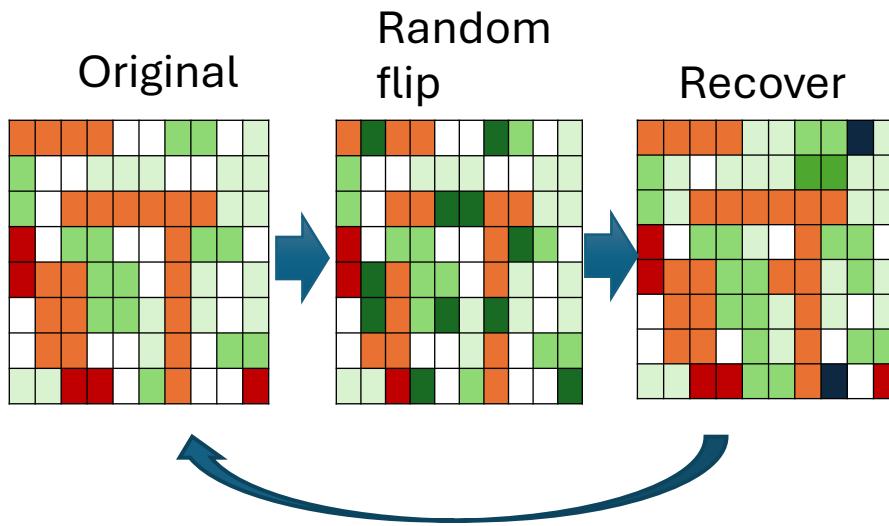
Outputs of scGNN



Datasets and tools for comparison

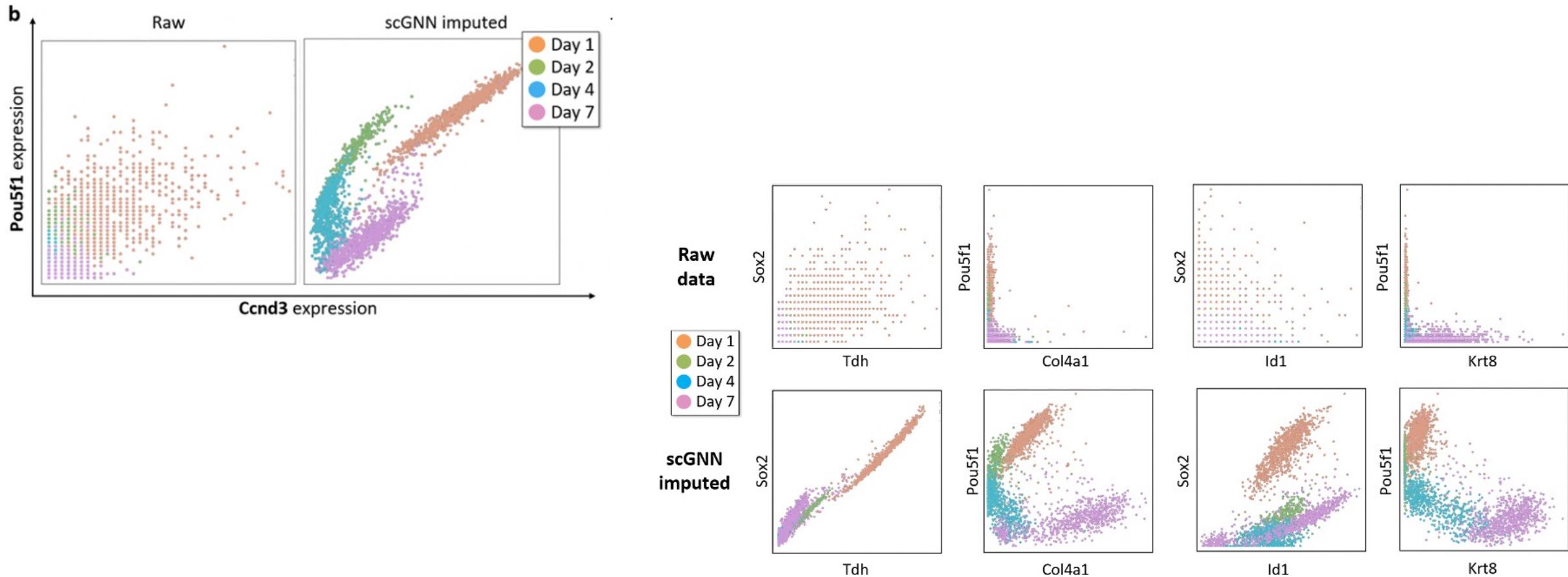
Datasets used in paper				
Dataset	# of genes	# of cells	# clusters	Cell Source
Klein's	24,175	2,717	4	Human Embryo
Zeisel's	19,972	3,005	9	Mouse cortex
Chung	57,915	317	4	Human breast cancer
Kolodziejczy	38,653	704	3	Mouse Embryo
AD data	-	13,214	8	GSE138852
Tools for performance comparison				
	Tool name	Version	Implementation	Parameters
Imputation tools	MAGIC	2.0	R & Python	Default
	SAUCIE	-	Python	
	SAVER	1.1.2	R	
	sclImpute	0.0.9	R	
	scVI	0.6.5	Python	
	DCA	-	Python	
	DeepImpute	1.1	Python	
	scIGANs	-	Python	
	netNMF-sc	-	Python	
Clustering tools	Seurat	3.1	R	
	CIDR	0.1.5	R & C++	
	Monocle	3.0	R	
	RaceID	-	R	

scGNN get better performances in imputation on synthetic dropout effects

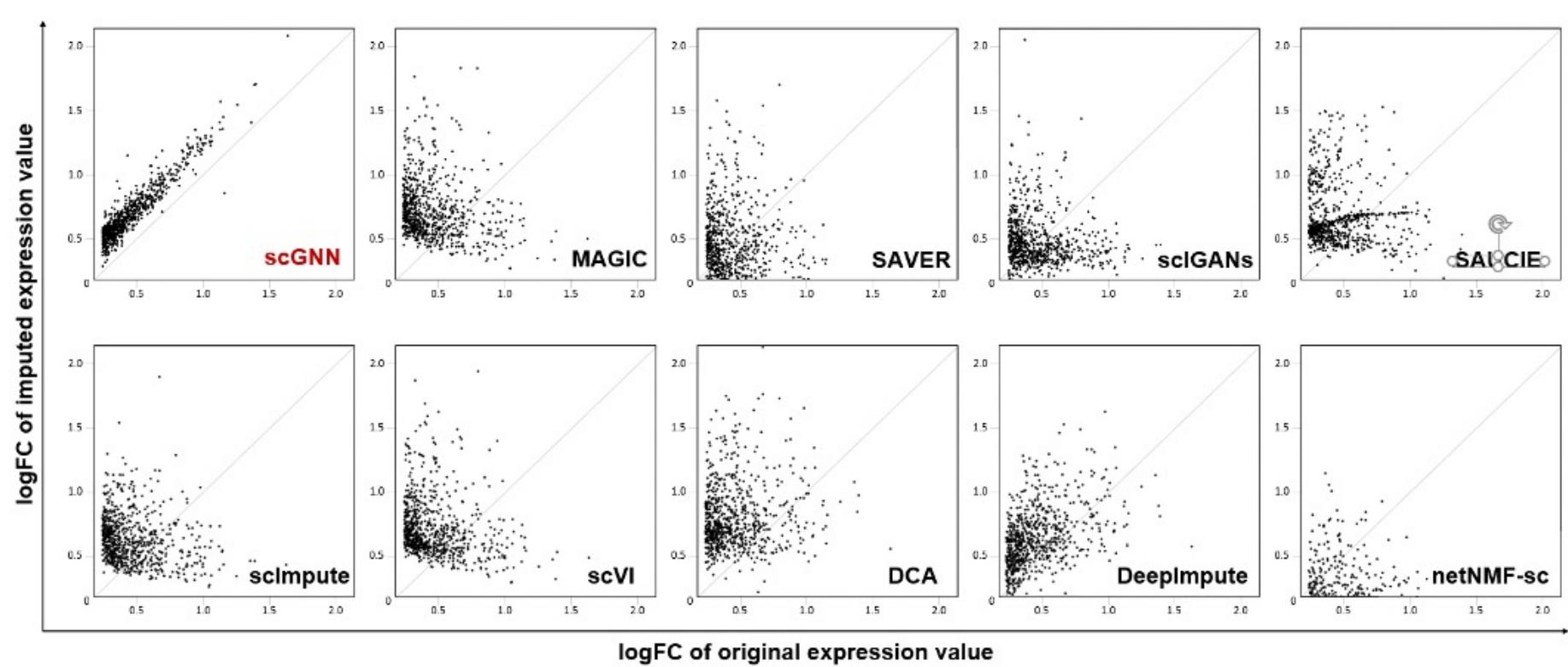
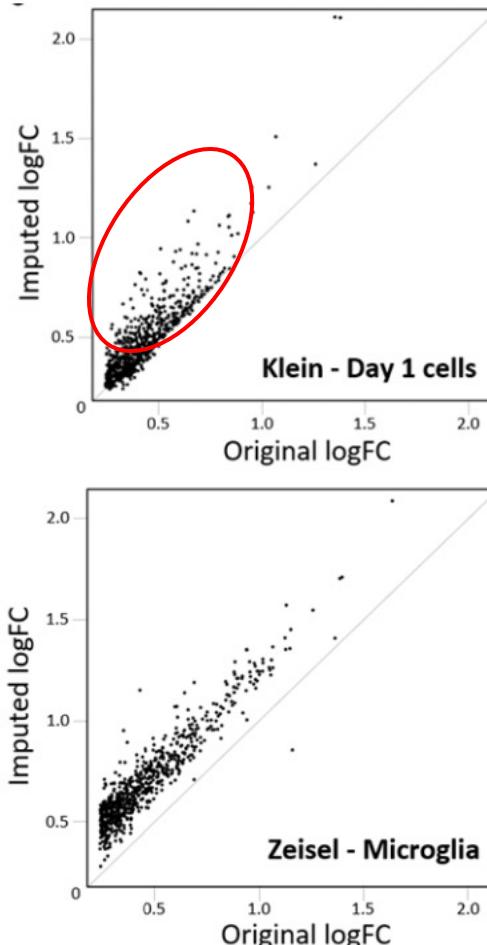


- Imputation on randomly flip nonzero values to zero with probability 10% and 30%

scGNN can recover the underlying gene-gene relationships missed in the raw expression data due to the sparsity of scRNA-Seq

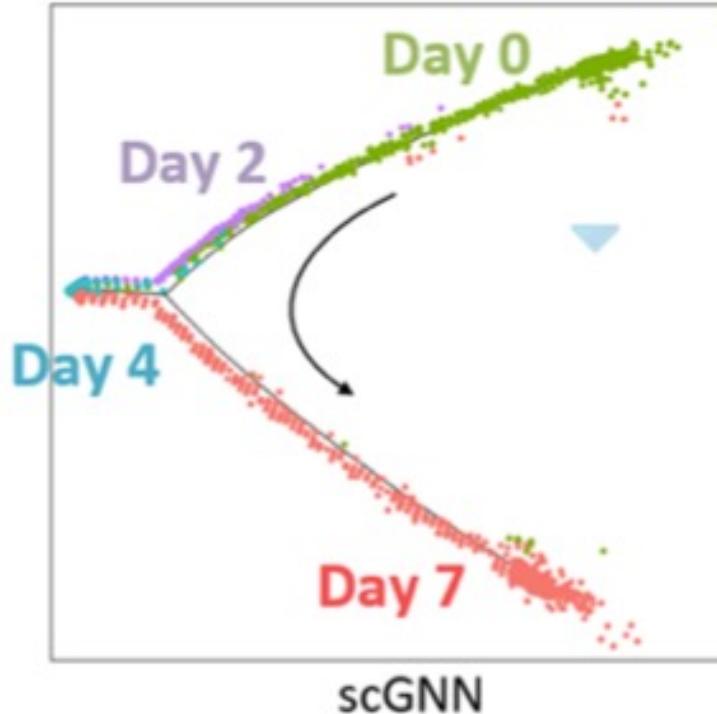
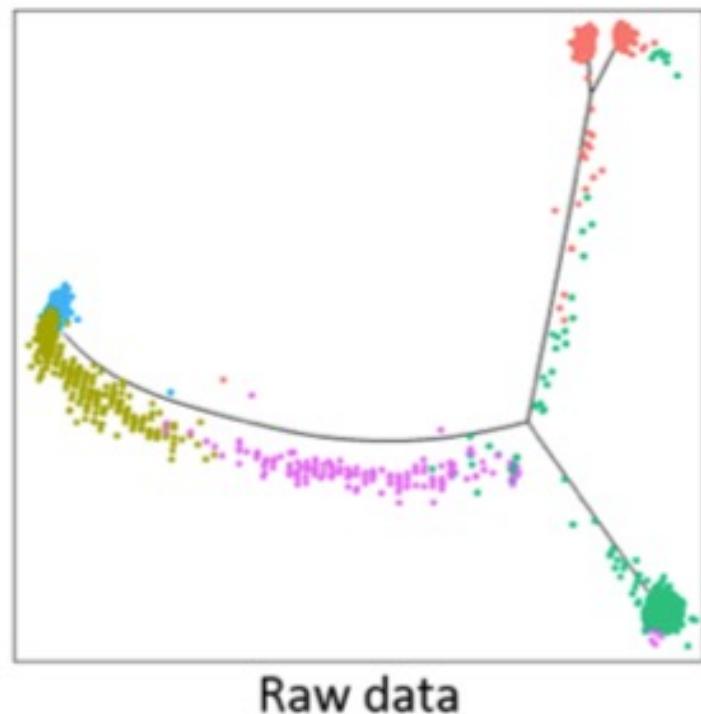


scGNN amplifies differentially expressed genes (DEGs) signals with a higher fold change than the original



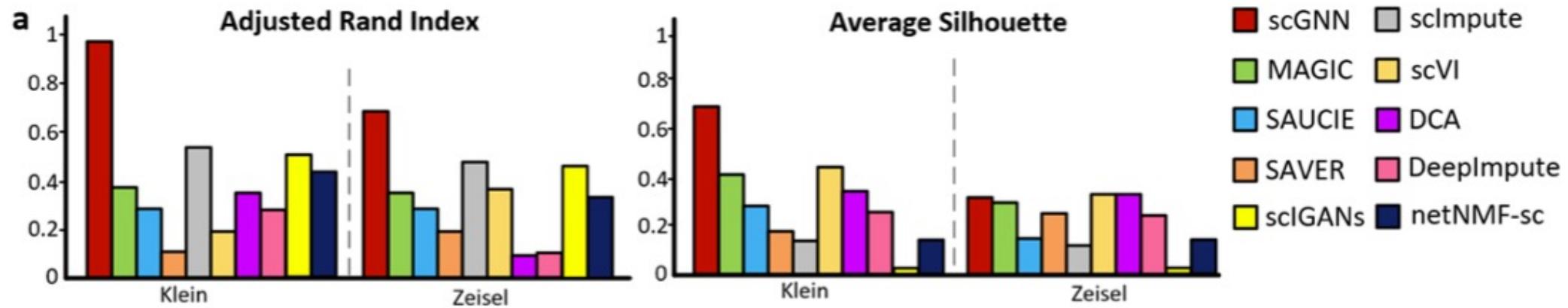
Others showed weaker coefficients and signals in some of the genes were decreased, indicating potential imputation bias in these tools

scGNN recovered a complex structure that was not well represented by the raw data



- Heterogeneity along with embryonic stem cell development in Zeisel data
- After imputation, scGNN shows a well-aligned trajectory path of cell development from Day 0 to Day 7
- Trajectory inferred with monocle

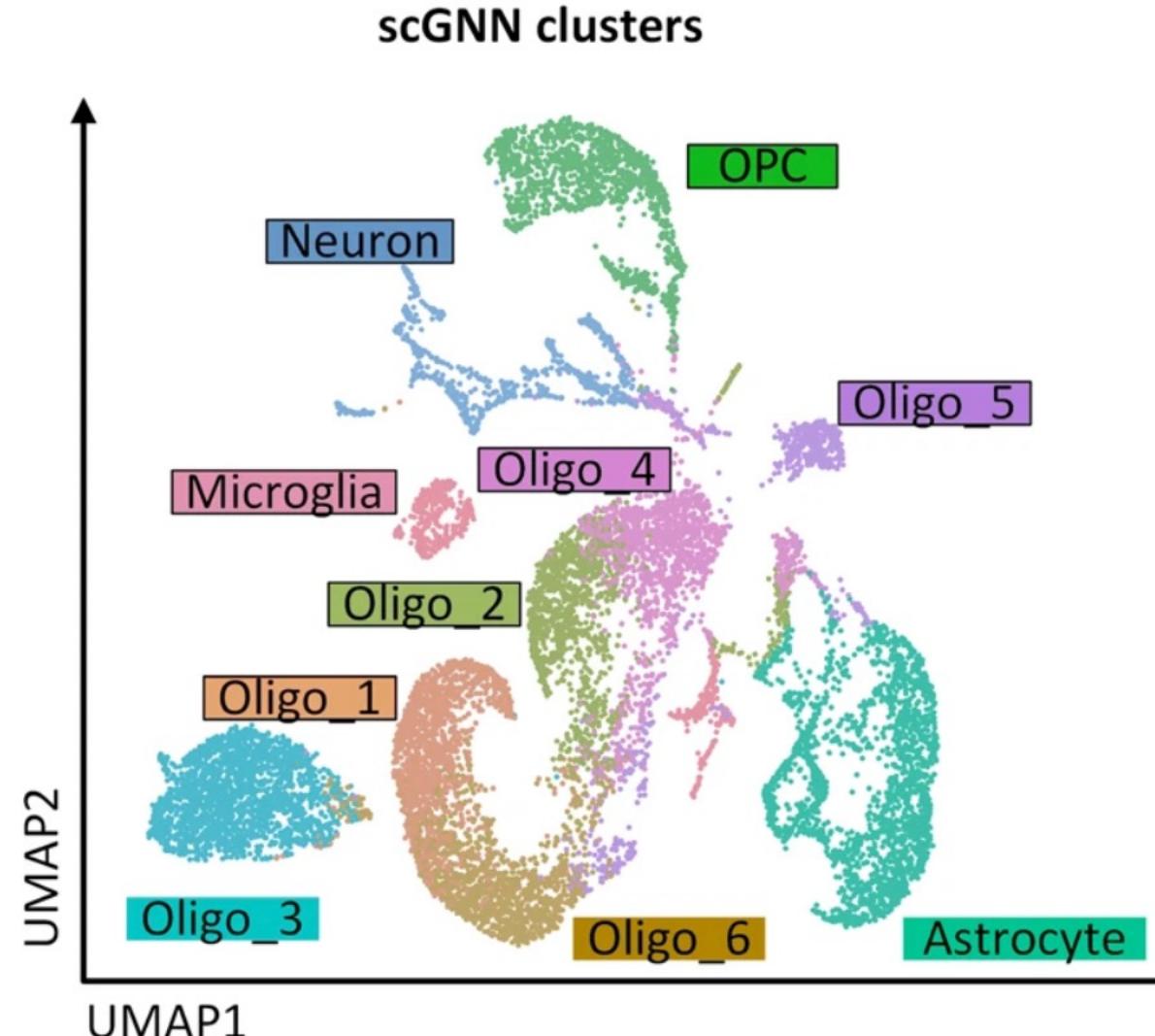
scGNN has better clustering performances than other methods



Imputation can help handling data heterogeneity by clustering

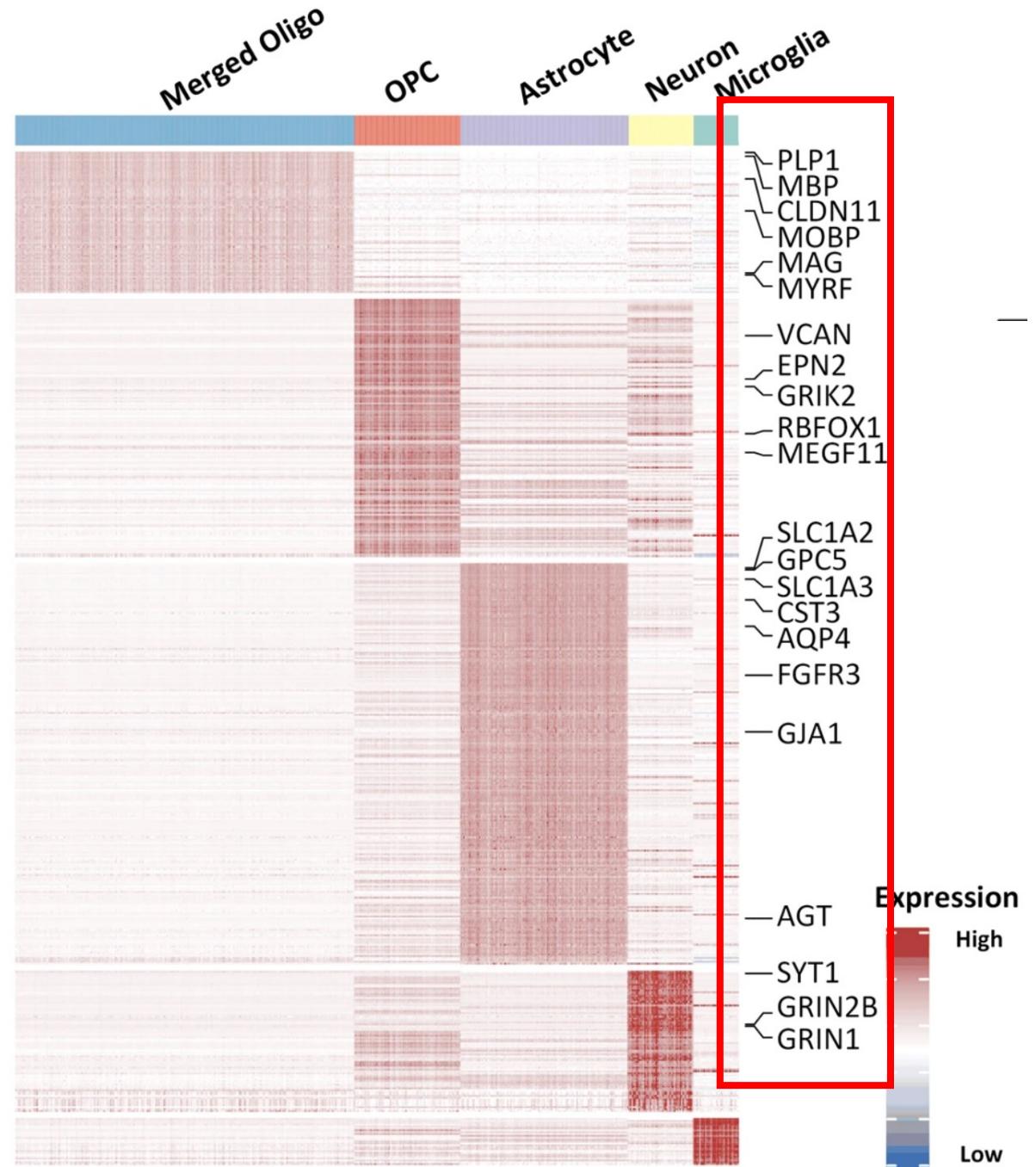
Case study: scGNN identified cell types in Alzheimer's Disease (AD)

- GSE138852 containing 13,214 single nuclei collected from six AD and six control brains
- Identified: microglia, neurons, oligodendrocyte progenitor cells (OPCs), astrocytes, and six sub-clusters of oligodendrocytes.

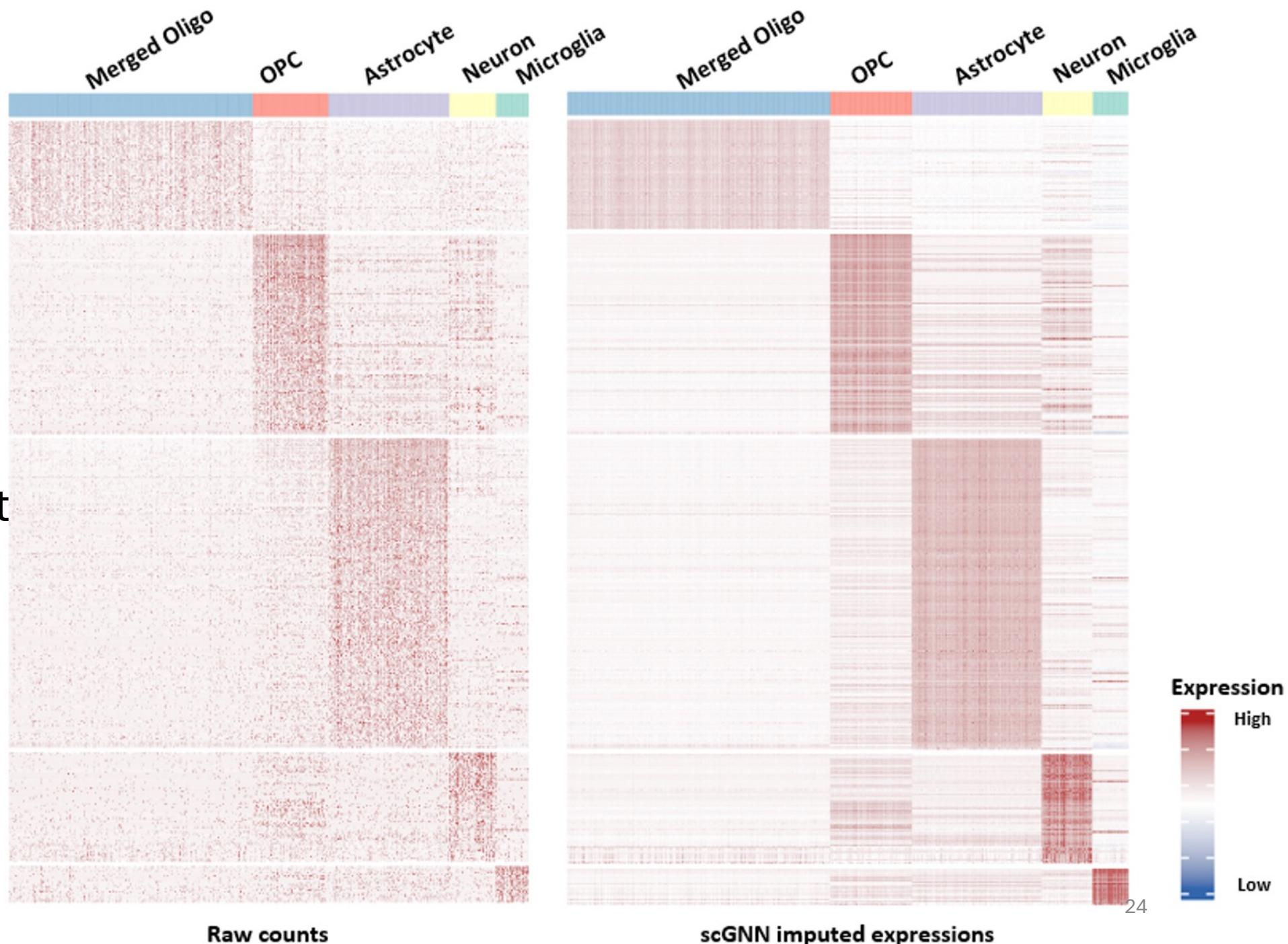


scGNN identifies cell-type-specific markers

- We confirmed 22 genes as cell-type-specific markers for astrocytes, OPCs, oligodendrocytes, and neurons.

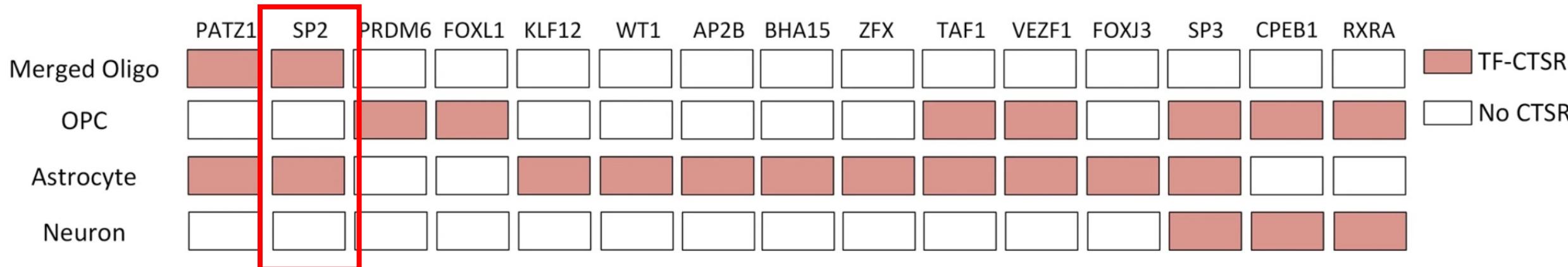


- scGNN can significantly increase true signals in the raw dataset, DEG patterns are more explicit



Underlying TFs are responsible for the cell-type-specific gene regulations

- Identified 21 cell-type-specific regulons (CTSR) in five cell types using IRIS3*.
- SP2 regulates *ABCA7* gene
- SP2 CTSR in astrocytes regulate marker genes *APOE*, *AQP4*, *SLC1A2*, *GJA1*, and *FGFR3*.



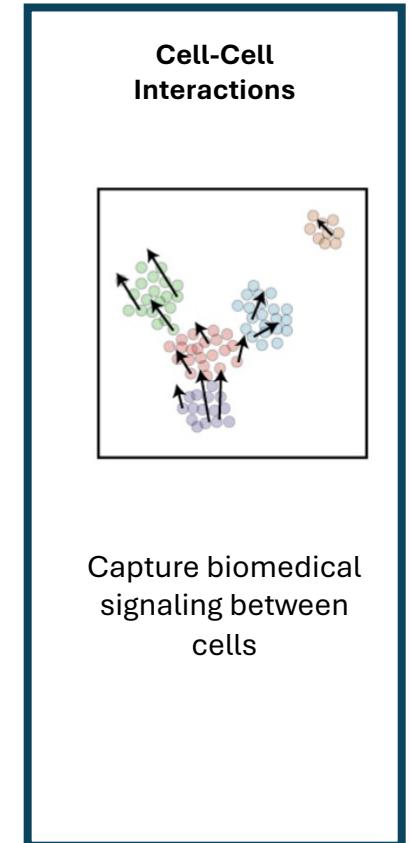
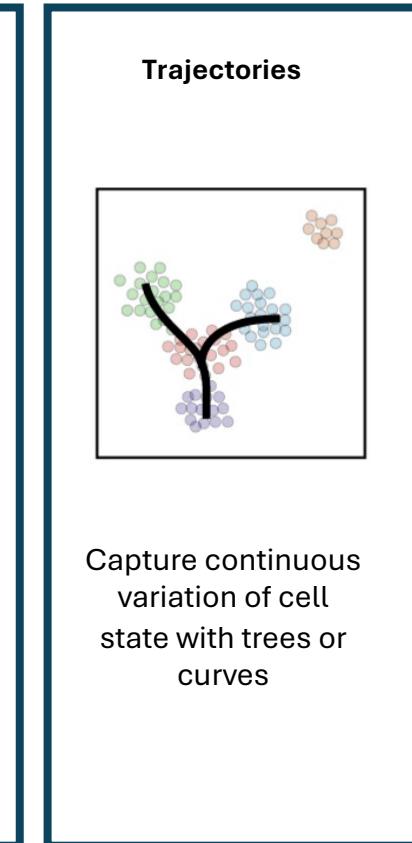
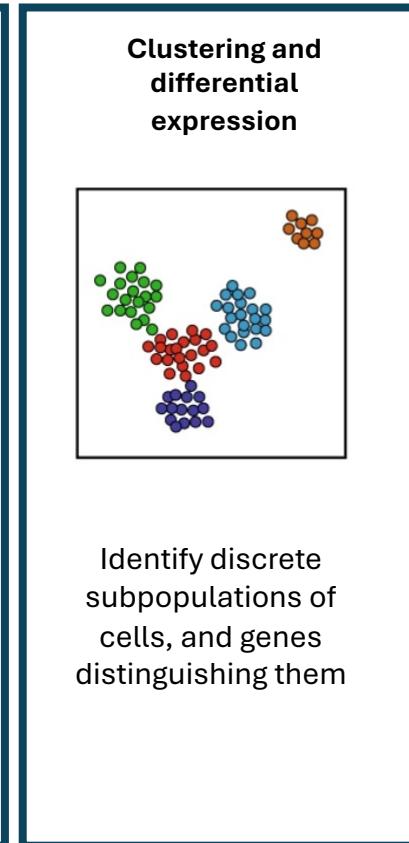
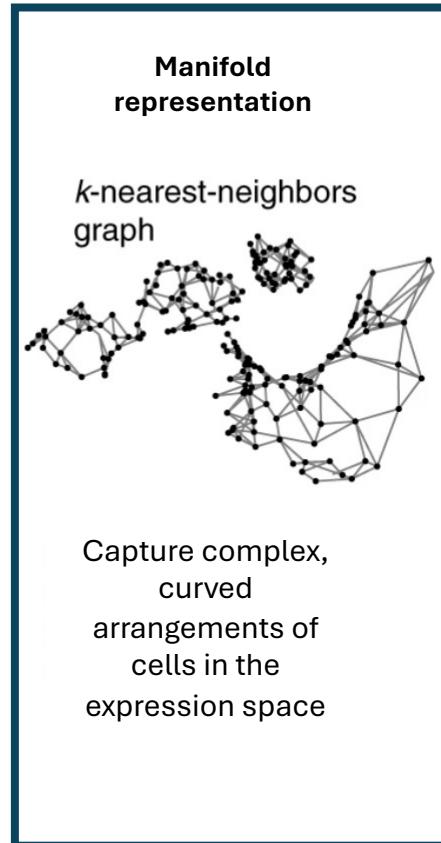
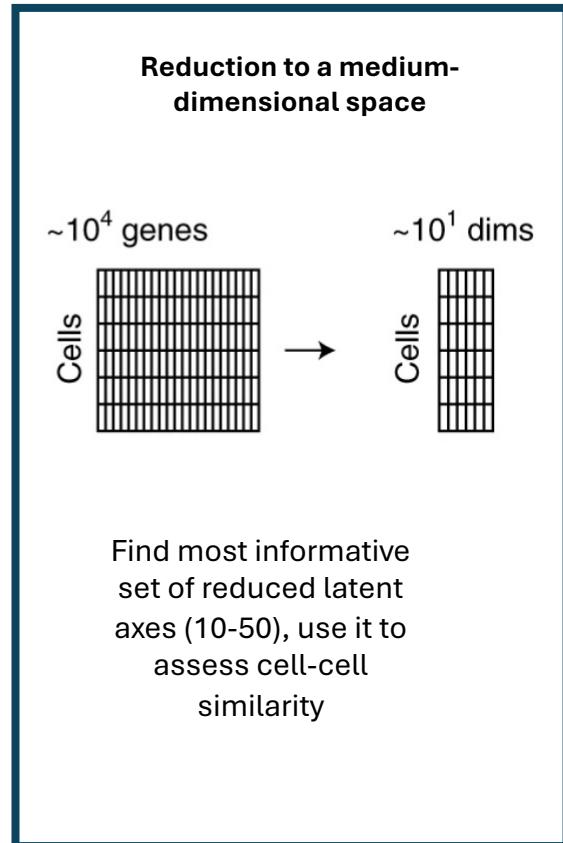
*Ma, et al. *Nucleic acids research* 48.W1 (2020): W275-W286.

scGNN contributes to key steps in scRNA-seq analysis

Kharchenko, *Nature Methods* 18.7 (2021): 723-732.

Step

Aim



Feature Autoencoder
Regularized by LTMG

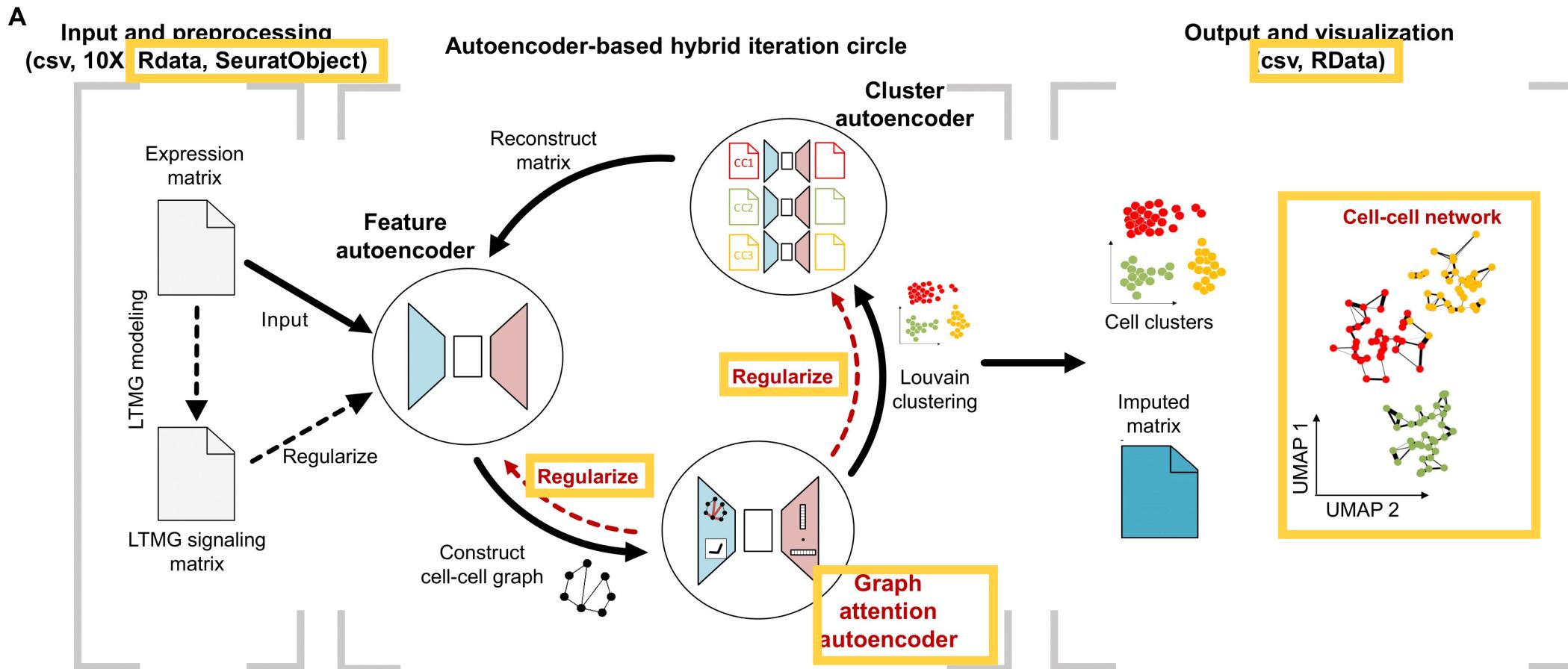
Graph Autoencoder

Clustering Autoencoder
& **Iterative process**
& Imputation

Imputation

Cell Graph

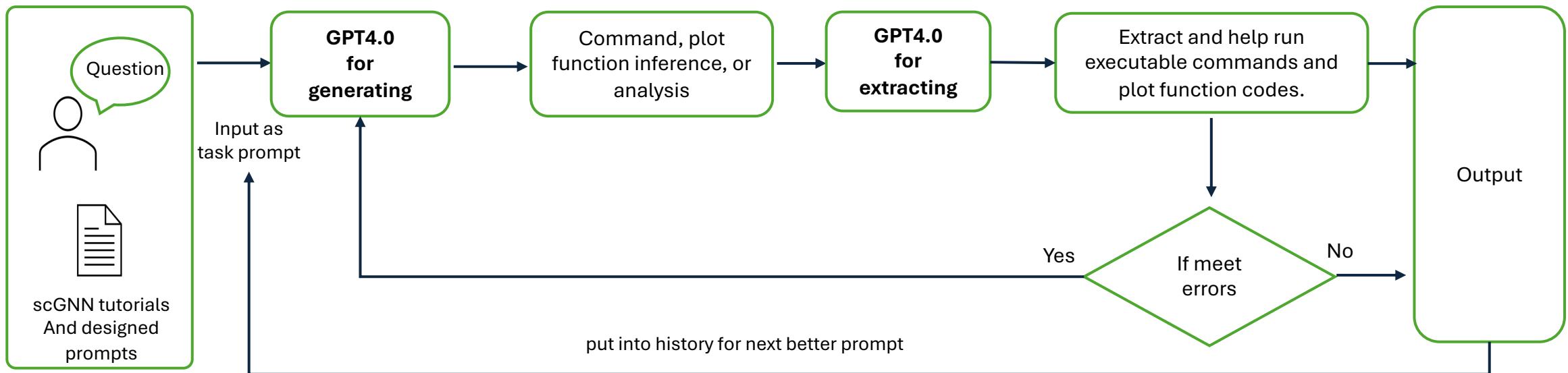
scGNN 2.0



Gu H, Cheng H, Ma A, Li Y, Wang J, Xu D, Ma Q. scGNN 2.0: a graph neural network tool for imputation and clustering of single-cell RNA-Seq data. Bioinformatics. 2022 Oct 17.

scGNN 3.0 (Chat with scGNN)

Combine ChatGPT with scGNN 2.0



- Use two GPTs, one is to **generate** commands, plot function codes, and provide analysis, the other one is to **extract** executable commands or plot function codes with a comprehensive prompt.
- The prompt list of every output is saved as chat history, which is used in the context of chat.
- The final output without running errors doesn't represent the question asked by the user is well-answered.

GPTs + Tutorial prompt + Auto-executing = Chat with scGNN!
<https://bmblx.bmi.osumc.edu/scgnn3>



Prompt engineering techniques

Tutorial is one of prompt

- We refine the scGNN tutorial and input it before users' questions

```
Prompt:  
You are a data scientist working on a project. You need to communicate with the AI assistant to complete the project.  
-----  
The user have these files:  
{file_name_list}  
  
The location folder of files is './data/{job_id}'  
-----  
The user may have these tasks: (1) generate scGNN command, (2) generate plot function codes, (3) provide the analysis of output data  
When you run scGNN you can set the output file folder into './results/task_output/'  
-----  
The scGNN package is in ./scGNN  
If you want to run scGNN, you need first get into this folder:  
  
cd ./scGNN  
  
And then run the scGNN command as following tutorial.  
-----  
Here are some context about the scGNN tutorial:  
  
{tutorial_context}
```

For running errors of scGNN and generating code

- We collect errors into chat history and provide correct running cases

```
"role": "system", "content":  
"Task: \n"+  
"I will provide some content which maybe include some descriptive words, some commands, or the executable python codes. "+  
"You need to first distinguish whether there are commands or some executable python codes. As the result, you need provide me a json object, which includes some keys: if_commands, "+  
"If there are commands, you need to provide 'true' for the key of if_commands. Then you need to extract all the commands inside the content and output only the commands for the ke  
"The same thing, if there are some executable python codes, you need to provide 'True' for the key of if_python_codes. Then you need to extract all the python script inside the co  
"-----\n"+  
"Environment: \n"+  
"You need to run the python script in the environment: /home/jiangyi/.conda/envs/yi/envs/scgnnEnv/bin/python\n"+  
"-----\n"+  
"Prompt: \n"+  
"Now I will give you four examples for you to understand the task. \n"+  
"Example 1 (commands): \n"+  
"content: 'If you want to get the result, you need to run the task: '+  
"----  
```bash  
mkdir
output\n python -W ignore scGNN_v2.py --given_cell_type_labels --load_use_benchmark --load_dataset_dir ./sampleData --load_dataset_name Goolam --output_dir ./outputs+
```  
"----  
"\nYou need to provide the json object: {'if_commands': True, 'if_python_codes': False, 'commands': 'mkdir outputs\n /home/jiangyi/.conda/envs/yi/envs/scgnnEnv/bin/python -W ignor  
"\n"+  
"  
"Example 2 (python script): \n"+  
"content: 'If you want to get the result, you need to run the python script: '+  
"----  
```python  
import os+
success = os.system('mkdir outputs')
if success != 0:
 print('error')
else:
 print('success')
```
```

Refine the scGNN tutorial and added it as a prompt

scGNN3.0

About

This is the manual for users to run scGNN and generate visualization results

Overview

- Workflow:** scGNN3.0 is an advanced web server designed to facilitate the analysis and visualization of single-cell genomics data using the scGNN model. Built on the powerful framework of ChatGPT-4.0, scGNN3.0 offers a unique interactive prompting interface that enhances user experience and efficiency in data processing and exploration. Users start sessions by inputting commands or queries, and each session is uniquely identified with a job ID for tracking and reference.
- Upload input file:** Users must upload their primary dataset, which typically includes an scRNA-seq expression matrix. Optional files include bulk expression data and ground truth cell type labels. Once data is uploaded, users can customize various model parameters through interactive prompts.
- Get results:** Based on the user's data and specified parameters, scGNN3.0 generates executable commands. These commands initiate the scGNN model processing in the background, associated with a unique task ID for each job.
- Visualization:** After model execution, users can query the system using the task ID to review the analysis results.

Upload files

There are three type of file you can upload to the scGNN-GPT server: 1.scRNA-Seq expression file -- the web server accepts '.csv', '.h5ad', '.mbx', and '.Rdata' format. If you want to run scGNN, you must upload scRNA-seq data as input.

2.(optional)cell labels -- the web server accepts ".csv" format This is the ground truth cell labels for comparing the labels that scGNN gets in Sankey diagram. In other words, cell label file is optional to upload. But if you want to get Sankey diagram, you need to upload it.

3.(optional)bulk RNA-Seq -- the web server accepts '.csv', '.h5ad', '.mtx', and '.Rdata' format. It is also an optional file. If you want to get the results with integrating bulk RNA-seq data, you can upload this file. Also, for getting sankey diagram, you also need one cell label result with integrating bulk data and other label result is without integrating bulk.

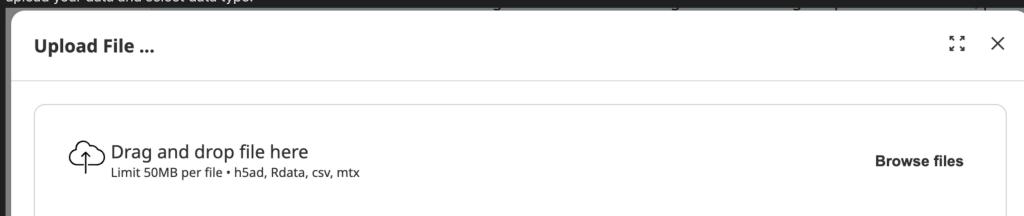
Customize parameters

If you want to set some customized parameters, please reference parameters below and set some parameters specific values in the prompt

| Parameter Name | Description |
|-------------------------------------|--|
| <code>given_cell_type_labels</code> | Type: boolean, default: False
If True, expect a cell type label file and will compute ARI against those labels. |
| <code>dropout_prob</code> | Type: float, default: 0.1
This is the dropout rate of scRNA-seq expression matrix. |

scGNN3.0-server usage example

- Upload files for scGNN running
 - Click this button:

- upload your data and select data type:


- The tutorial includes the whole pipeline of scGNN3.0 and details for functions in scGNN3.0
 - It can provide enough information for GPT-based workflow to generate commands and codes.
 - The users of scGNN can use this to provide more accurate questions.
- Also, the tutorial provides the guide of scGNN3.0-server.

An example by using scGNN3.0-server

1. upload your data

The screenshot shows the scGNN3.0-server interface. At the top left is a user icon. A message box says "Sure, please upload the data files first." Below it is a "Contextual Prompt". In the center, a modal window titled "Upload File ..." displays a file selection area with a cloud icon and the text "Drag and drop file here Limit 50MB per file • h5ad, Rdata, csv, mtx". It also shows a list item "dropout_top_expression.h5ad - 1127224 bytes" and checkboxes for "Data type: scRNA-Seq", "Cell Labels (Optional)", and "Bulk RNA-Seq (Optional)". An "Upload" button is at the bottom. At the bottom of the screen is a toolbar with various icons, and a text input field with placeholder text "Enter to send, Shift + Enter to wrap, / to search prompts, : to use commands" and a "Send" button.

2. When your data is ready, the chat robot will guide you to provide some order

The screenshot shows a message from a user icon: "You've uploaded data files, Now, tell me what do you want to do?". Below it is a timestamp "5/9/2024, 8:18:45 PM".

An example by using scGNN3.0-server

3. you can run scGNN by using this uploaded data



Certainly! Since you want to use the file 'dropout_top_expression.h5ad' to run scGNN, I will first generate the bash command for you:

```
cd ./scGNN
python scGNN.py --load_from_h5ad ../data/156/dropout_top_expression.h5ad \
--output_dir ../results/task_output/
```

This command changes the current directory to the scGNN folder, then executes the `scGNN.py` script with the `--load_from_h5ad` argument pointing to your input `.h5ad` file, and specifies the output directory for scGNN's results.

Remember to check the scGNN's specific settings or parameters you might need to adjust (such as `--total_epoch` if you want to set a custom number of epochs), and add those to the command line as needed.

After running this command, scGNN will process the provided input file and generate the output files in the specified output folder ('`../results/task_output/`').

Should you need any additional tasks, such as figure generation or specific analysis, please provide me with the task ID and the type of analysis or visualization you need, and I will generate the necessary Python script code for you.

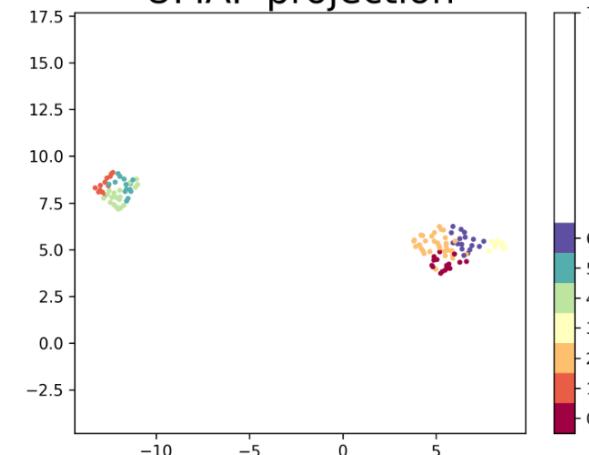
5/9/2024, 8:18:50 PM

4. When the task results are ready, the UMAP figure will be provided automatically



task 170 result: UMAP.png

UMAP projection



5/9/2024, 8:53:31 PM

An example by using scGNN3.0-server

5. Now, you can download the result data of scGNN output

The screenshot shows the 'Job Files' section with three items: 'python_script.py' (size: 924 Bytes), 'dropout_top_expression.h5ad' (size: 1.08 MB), and 'python_171_script.py' (size: 916 Bytes). Below it is the 'Tasks Files' section for task 170, listing nine files: 'imputed.csv', 'feature_embedding.csv', 'all_metrис.csv', 'clustering_embedding.csv', 'labels.csv', 'UMAP.png', 'graph_embedding.csv', and 'graph_edgeList.csv'. Each file has a 'Download' button.

6. The results are ready, you probably want to see some analysis figure: Ask the chat robot to draw a Heatmap figure by using the output of scGNN

A user message: "I want to draw heatmap according to the results of task 170" (5/9/2024, 8:53:59 PM). A response message: "To generate the heatmap code according to the results of task 170, you will need to specify the paths for the original dataset, imputed data, labels, and the destination of the resulting heatmap image. Here is the Python code snippet that you would use:"

```
from Viz.HeatMap import heatmap
# Define the task ID
task_id = '170'
# Define the original dataset path (the h5ad file provided by the user)
original_filepath = '/bmb1_data/jiangyi/scGNN-server-outline/scGNN-server_v3/data/156/dropout_top_expression.h5ad'
# Define the imputed dataset and label paths, based on the task ID and results directory
imputed_filepath = f'/bmb1_data/jiangyi/scGNN-server-outline/scGNN-server_v3/results/{task_id}/imputed.csv'
label_filepath = f'/bmb1_data/jiangyi/scGNN-server-outline/scGNN-server_v3/results/{task_id}/labels.csv'
# Define the save path for the heatmap
save_filepath = f'/bmb1_data/jiangyi/scGNN-server-outline/scGNN-server_v3/results/task_output/{task_id}_heatmap.png'
# Generate the heatmap
heatmap(
    original_filepath=original_filepath,
    imputed_filepath=imputed_filepath,
    label_filepath=label_filepath,
    save_filepath=save_filepath
)
```

Make sure you have the function `heatmap` correctly imported from the `Viz.HeatMap` module and all file paths correctly set according to your file structure. Also, ensure that the function `heatmap` accepts these parameters and you have access to the corresponding input files.

Two heatmap images are shown: 'task 171 result: 170_heatmaporiginal.png' (5/9/2024, 8:54:40 PM) and 'task 171 result: 170_heatmapimputed.png'.

7. When figures are ready, they will be provided automatically

An example by using scGNN3.0-server

8. You can also define customized colors

I want to draw heatmap according to the results of task 170 with a red-white colormap
5/9/2024, 9:05:05 PM

To generate the Python script for plotting a heatmap with a red-white colormap based on the results of task 170, we need to provide the file paths for the original expression dataset, the imputed dataset, the labels, and the save path for the heatmap image along with specifying the colormap. Below is the Python script that you can use:

```
from Viz.HeatMap import heatmap

# File paths for task 170
task_id = 170
data_folder = '/bmbL_data/jiangyi/scGNN-server-outline/scGNN-server_v3/data/156'
results_folder = f'/bmbL_data/jiangyi/scGNN-server-outline/scGNN-server_v3/results/{task_id}'
output_folder = '/bmbL_data/jiangyi/scGNN-server-outline/scGNN-server_v3/results/task_output'

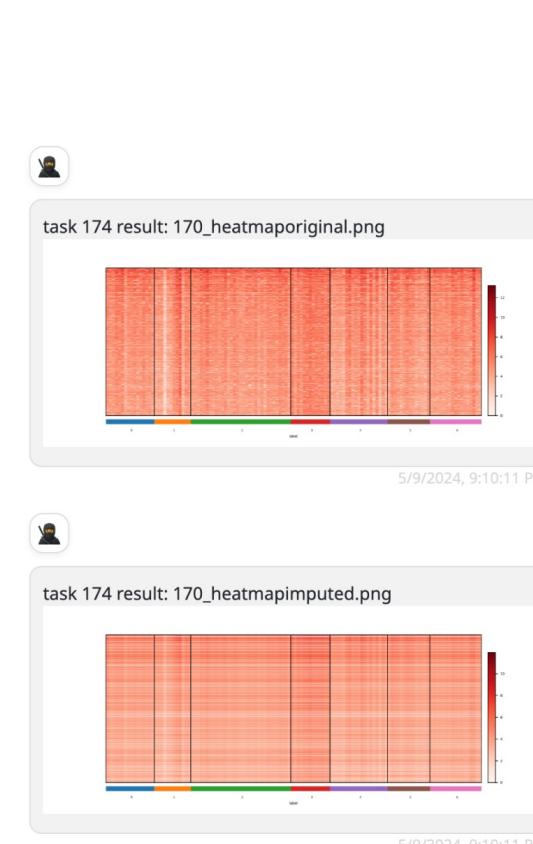
# File paths for heatmap function
original_filepath = f'{data_folder}/dropout_top_expression.h5ad'
imputed_filepath = f'{results_folder}/imputed.csv'
label_filepath = f'{results_folder}/labels.csv'
save_filepath = f'{output_folder}/heatmap.png'

# Colormap for heatmap
colormap = 'RdWh' # red-white colormap

# Generate heatmap
heatmap(original_filepath, imputed_filepath, label_filepath, save_filepath, colormap)

You would need to ensure that the Viz.HeatMap module and heatmap function are in the correct path and importable. Additionally, please replace the task_id with the actual task ID if it's different from 170, and ensure all file paths match the actual locations and structure of your directory.
```

5/9/2024, 9:05:05 PM



Acknowledgement



Lab Member

Mauminah Raina

Shuang Wang

ML and Data analysis

Dr. Dong Xu at University of Missouri

Jinpu Li

Yang Yu

Dr. Fei He

Skyler Kramer

Li Su

...

Dr. Qin Ma at OSU

Dr. Yuzhou Chang

Yi Jiang

...

Dr. Anjun Ma at OSU

scGNN

<https://github.com/juexinwang/scGNN>

scGNN2.0

<https://github.com/OSU-BMBL/scGNN2.0>

Server: <https://bmblx.bmi.osumc.edu/scgnn3>

Funding: NIH/NIDDK R01DK138504



National Institute of
Diabetes and Digestive
and Kidney Diseases



AD Study

Dr. Hongjun Fu at OSU



INDIANA UNIVERSITY
MELVIN AND BREN SIMON
COMPREHENSIVE CANCER CENTER