bio-protocol

Open Access

Published: Feb 05, 2025

# Model Architecture Analysis and Implementation of TENET for Cell–Cell Interaction Network Reconstruction Using Spatial Transcriptomics Data

Ziyang Wang[1, #], Yujian Lee[2, #, §], Yongqi Xu[3], Peng Gao[2], Chuckel Yu[4] and Jiaxing Chen[2, *]

[1]Dept/Center, Guangdong Medical University, Dongguan, China
[2]Guangdong Provincial Key Laboratory IRADS, BNU-HKBU UIC, Zhuhai, China
[3]Department of Computer Science and Technology, Guangdong University of Technology, Guangzhou, China
[4]Independent researcher, Guangzhou, China
[§]Technical contact: yujianlee1119@gmail.com
[#]Contributed equally to this work
*For correspondence: jiaxingchen@uic.edu.cn

## Abstract

Cellular communication relies on the intricate interplay of signaling molecules, which come together to form the cell–cell interaction (CCI) network that orchestrates tissue behavior. Researchers have shown that shallow neural networks can effectively reconstruct the CCI from the abundant molecular data captured in spatial transcriptomics (ST). However, in scenarios characterized by sparse connections and excessive noise within the CCI, shallow networks are often susceptible to inaccuracies, leading to suboptimal reconstruction outcomes. To achieve a more comprehensive and precise CCI reconstruction, we propose a novel method called triple-enhancement-based graph neural network (TENET). The TENET framework has been implemented and evaluated on both real and synthetic ST datasets. This protocol primarily introduces our network architecture and its implementation.

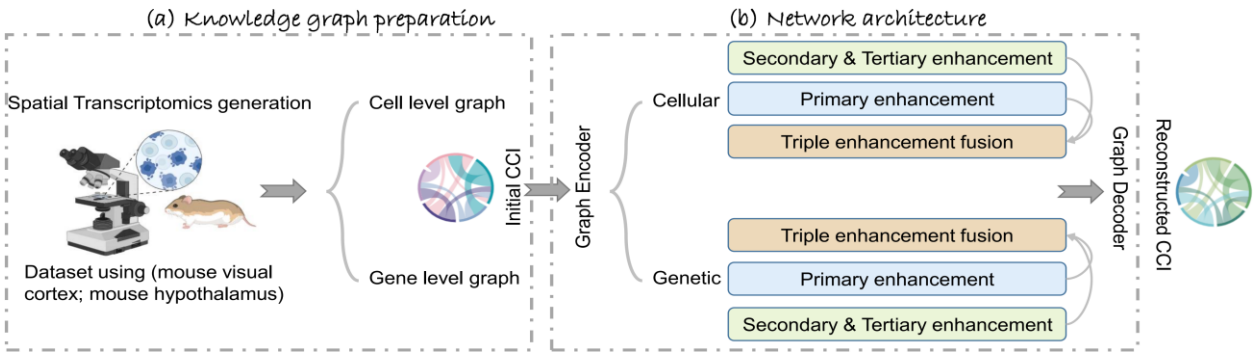
## Key features

- Cell–cell reconstruction network using ST data.
- To facilitate the implementation of a more holistic CCI, we incorporate diverse CCI modalities into consideration.
- To further enrich the input information, the downstream gene regulatory network (GRN) is also incorporated as an input to the network.
- The network architecture considers global and local cellular and genetic features rather than solely leveraging the graph neural network (GNN) to model such information.

**Keywords:** Cell–cell interaction network (CCI) reconstruction, Gene regulatory network (GRN), Spatial transcriptomics (ST) data, Graph neural network (GNN), Attention mechanism

**This protocol is used in:** J Mol Biol (2024), DOI: 10.1016/j.jmb.2024.168543

## Graphical overview




**Graphical abstract of TENET, including (a) the knowledge graph preparation on both cell and gene levels and (b) the network architecture.**

## Background

Understanding cellular communication is crucial for constructing a cell–cell interaction network (CCI), which allows researchers to investigate the roles of different cells in biological processes and diseases. A common method for analyzing CCI involves studying the interactions between secreted ligands and their corresponding receptors (LR pairings), as these interactions are essential for signal transmission.

However, CCIs also occur through direct cell–cell contact, the extracellular matrix (ECM), and the secretion of signaling molecules [1]. Focusing solely on LR pairings overlooks the spatial context in which these interactions occur. With the advent of spatial transcriptomics (ST) data, incorporating spatial information can help address this limitation. Several methods for reconstructing CCI using ST data have been developed. For instance, Giotto [2] constructed a spatial grid based on cell coordinates to model proximal interactions but did not account for distal interactions. MISTy [3] utilizes multiple perspectives (intrinsic, local, tissue) to enhance cell knowledge for CCI inference. DeepLinc [4] employs a graph neural network (GNN) [5] to create a spatial proximity graph, using KNN [6] to identify both proximal and distal interactions. Despite these advances, modeling ST data solely at the cell level can lead to false positives and negatives due to a lack of downstream information. Incorporating downstream data, such as intracellular signaling, gene regulation, and protein changes can provide crucial insights for understanding actual CCI; CLARIFY [7] combines cellular and genetic information to build knowledge graphs for GNN inputs, effectively capturing structural features important for CCI inference. However, solely using GNNs is limiting, as they often aggregate local information, hindering the capture of global context. GNNs also struggle with similar nodes, which may lead to a loss of features' specificity.

To overcome these challenges, we propose a novel approach called triple-enhancement-based graph neural network (TENET), designed as a comprehensive architecture for CCI reconstruction. TENET employs a graph convolution network (GCN) [8] backbone to extract global features from the data at multiple resolutions, generating robust latent feature embeddings. These embeddings then undergo a triple-enhancement mechanism that restores cell specificity that may have been lost during the GCN's global aggregation. The first enhancement mitigates the GCN's over-smoothing problem, while the secondary and tertiary enhancements refine the embeddings both locally and globally. Finally, a denoising loss function is introduced to tackle the challenges posed by noisy, low-quality data often encountered in small molecule experiments. This multi-scale enhancement strategy enables TENET to accurately and robustly reconstruct cell–cell interactions, addressing the limitations of previous methods and opening new avenues for researchers to explore the complexities of CCI.

## Software and datasets

### Software

1. 2.90 GHz Intel i7-10700F CPU and NVIDIA A100 graphics card; nvcc: NVIDIA (R) Cuda compiler driver; Copyright 2005–2024 NVIDIA Corporation; Built on Tue_Feb_27_16:19:38_PST_2024; Cuda compilation tools, release 11.7, V11.7.99; Build cuda_11.7.r11.7/compiler.33961263_0

### Input data

1. Three ST datasets are utilized, namely seqFISH, MERFISH, and scMultiSim [9], which can be acquired at https://bitbucket.org/qzhu/smfish-hmrf/src/master/, https://datadryad.org/stash/dataset/10.5061/dryad.8t8s248, and https://github.com/ZhangLabGT/scMultiSim. The hyperlinks direct to the datasets utilized in TENET. The specific analysis of the datasets is mentioned in the following sections.

## Procedure

The procedure includes two parts: the input preparation in section A and the network architecture in section B. The inputs of the network are the knowledge graphs from the cell and gene levels. Graphs encapsulate both the nodal feature representation V as well as the corresponding adjacency matrix (edge list) Adj. In section A, we elucidate the respective construction for the cell-level graph and the gene-level graph, denoted as Gc and Gg, respectively. In section B, we present the corresponding operations to the constructed knowledge graphs. Table 1 lists the frequently used symbols in sections A and B, improving overall readability.

**Table 1. Frequently used symbols**

| Symbol | Explanation |
|---|---|
| $G_{c(g)} = (V_{c(g)}, Adj_{c(g)})$ | Input cell (gene) knowledge graph, containing nodes with features and the adjacency matrix. |
| $V_{c(g)} \in \mathbb{R}^{N_{c(g)} \times F_{c(g)}}$ | Nc(g) number of nodes with Fc(g) feature dimensions. |
| $Adj_{c(g)} \in \mathbb{R}^{N_{c(g)} \times N_{c(g)}}$ | Adjacency matrix representing the establishment of interactions among nodes. |
| $K_p$ | The numbers of cells' indexes denoting the establishment of proximal interaction. |
| $Adj_p \in \mathbb{R}^{N_c \times N_c}$ | The proximal interaction adjacency matrix. |
| $K_{ir}$ | The numbers of cells' indexes denoting the establishment of intermediate range interaction. |
| $Adj_{ir} \in \mathbb{R}^{N_c \times N_c}$ | The intermediate range interaction adjacency matrix. |
| $Z_{c(g)} \in \mathbb{R}^{N_{c(g)} \times F_{c(g)}}$ | Latent feature embeddings after the input graph Gc(g) traverse through the GCN encoder. |
| $Z_{c+g} \in \mathbb{R}^{N_c \times F_{c+g}}$ | An integration of Zc and Zg. |
| $Z'_{c+g} \in \mathbb{R}^{N_c \times F'_{c+g}}$, $Z'_g \in \mathbb{R}^{N_c \times F'_g}$ | Enhancing Z(c+g), Zg using CECB. |
| $Z''_{c+g} \in \mathbb{R}^{N_c \times F''_{c+g}}$, $Z'_g \in \mathbb{R}^{N_c \times F''_g}$ | Enhancing Z(c+g)' and Zg' using EEB and SEB, respectively. |
| $Z'''_{c+g} \in \mathbb{R}^{N_c \times F'''_{c+g}}$, $Z'_g \in \mathbb{R}^{N_c \times F'''_g}$ | Integration of the previous result {Z(c+g), Z(c+g,)' Z(c+g)'"},{Zg, Z(g,)' Zg'"}. |

### A. Input preparation

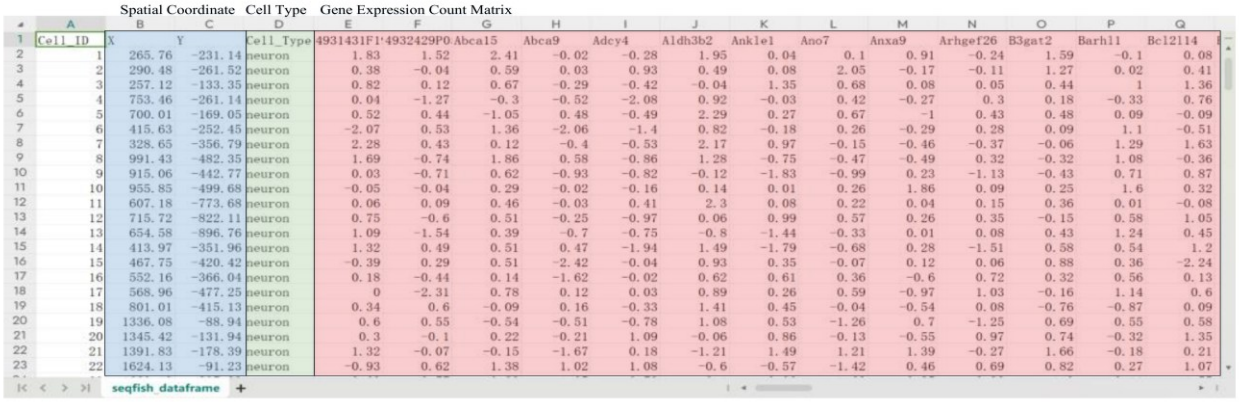
The construction of Gc and Gg is described in subsections a and b, respectively.

1. Cell-level graph construction

From the aforementioned links, there are four .csv files corresponding to each dataset. These files encompass the spatial coordinates of individual cells, the cell type classifications, the gene expression count matrices for each cell, and the adjacency matrix delineating cellular connectivity. Except for the adjacency matrix file, the other three files are encapsulated into a data frame, representing the cells' features. Figure 1 depicts the cell features organized into a data frame format; using the seqFISH dataset as an example, there are Nc number of cells with Fc number of features, denoted as $Vc \in R^{(Nc \times Fc)}$. In this feature representation, the blue overlay delineates the spatial coordinates of individual cells, the green overlay denotes the cell type classifications, and the red overlay represents the gene expression matrix. (Due to space limitation, the red overlay is a partial visualization of the gene expression profile.) To build a cell-level adjacency matrix Adjc, our procedure is not simply to use the provided adjacency matrix but to simulate actual cell communication in the real biological world by integrating the three interaction modes into consideration. These three secretion modes exhibit a different range of interactions, where the cell–cell direct contact is defined to be the proximal interaction, ECM to be the intermediate interaction, and the secreting signaling molecule to be the distal interaction. Given that our approach is based on the neighbor

search algorithm and that the input data consists solely of the spatial coordinates of the cells, considering the distal interaction will introduce error to the constructed CCI. Therefore, the initial CCI includes the proximal and intermediate-range interaction modes, and our algorithm combines two neighbor clustering methods, both of which have different purposes. The first is the hierarchical navigable small world (HNSW) algorithm [10], which is good at processing large numbers of data points and speeding up the construction of our spatial connectivity networks. The second is the Louvain algorithm [11], which can merge small clusters into larger ones, helping the search for intermediate-range cellular interactions. The implementations are summarized as follows:

- Finding proximal interacting cells Kp.
- Finding intermediate-range interacting cells Kir.
- Integration of the initial CCI.




| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Cell_ID | X | Y | Cell_Type | 4931431F1 | 4932429P0 | Abca15 | Abca9 | Adcy4 | Aldh3b2 | Ankle1 | Ano7 | Anxa9 | Arhgef26 | B3gat2 | Barh11 | Bcl2114 |
| 2 | 1 | 265.76 | −231.14 | neuron | 1.83 | 1.52 | 2.41 | −0.02 | −0.28 | 1.95 | 0.04 | 0.1 | 0.91 | −0.24 | 1.59 | −0.1 | 0.08 |
| 3 | 2 | 290.48 | −261.52 | neuron | 0.38 | −0.04 | 0.59 | 0.03 | 0.93 | 0.49 | 0.08 | 2.05 | −0.17 | −0.11 | 1.27 | 0.02 | 0.41 |
| 4 | 3 | 257.12 | −133.35 | neuron | 0.82 | 0.12 | 0.67 | −0.29 | −0.42 | −0.04 | 1.35 | 0.68 | 0.08 | 0.05 | 0.44 | 1 | 1.36 |
| 5 | 4 | 753.46 | −261.14 | neuron | 0.04 | −1.27 | −0.3 | −0.52 | −2.08 | 0.92 | −0.03 | 0.42 | −0.27 | 0.3 | 0.18 | −0.33 | 0.76 |
| 6 | 5 | 700.01 | −169.05 | neuron | 0.52 | 0.44 | −1.05 | 0.48 | −0.49 | 2.29 | 0.27 | 0.67 | −1 | 0.43 | 0.48 | 0.09 | −0.09 |
| 7 | 6 | 415.63 | −252.45 | neuron | −2.07 | 0.53 | 1.36 | −2.06 | −1.4 | 0.82 | −0.18 | 0.26 | −0.29 | 0.28 | 0.09 | 1.1 | −0.51 |
| 8 | 7 | 328.65 | −356.79 | neuron | 2.28 | 0.43 | 0.12 | −0.4 | −0.53 | 2.17 | 0.97 | −0.15 | −0.46 | −0.37 | −0.06 | 1.29 | 1.63 |
| 9 | 8 | 991.43 | −482.35 | neuron | 1.69 | −0.74 | 1.86 | 0.58 | −0.86 | 1.28 | −0.75 | −0.47 | −0.49 | 0.32 | −0.32 | 1.08 | −0.36 |
| 10 | 9 | 915.06 | −442.77 | neuron | 0.03 | −0.71 | 0.62 | −0.93 | −0.82 | −0.12 | −1.83 | −0.99 | 0.23 | −1.13 | −0.43 | 0.71 | 0.87 |
| 11 | 10 | 955.85 | −499.68 | neuron | −0.05 | −0.04 | 0.29 | −0.02 | −0.16 | 0.14 | 0.01 | 0.26 | 1.86 | 0.09 | 0.25 | 1.6 | 0.32 |
| 12 | 11 | 607.18 | −773.68 | neuron | 0.06 | 0.09 | 0.46 | −0.03 | 0.41 | 2.3 | 0.08 | 0.22 | 0.04 | 0.15 | 0.36 | 0.01 | −0.08 |
| 13 | 12 | 715.72 | −822.11 | neuron | 0.75 | −0.6 | 0.51 | −0.25 | −0.97 | 0.06 | 0.99 | 0.57 | 0.26 | 0.35 | −0.15 | 0.58 | 1.05 |
| 14 | 13 | 654.58 | −896.76 | neuron | 1.09 | −1.54 | 0.39 | −0.7 | −0.75 | −0.8 | −1.44 | −0.33 | 0.01 | 0.08 | 0.43 | 1.24 | 0.45 |
| 15 | 14 | 413.97 | −351.96 | neuron | 1.32 | 0.49 | 0.51 | 0.47 | −1.94 | 1.49 | −1.79 | −0.68 | 0.28 | −1.51 | 0.58 | 0.54 | 1.2 |
| 16 | 15 | 467.75 | −420.42 | neuron | −0.39 | 0.29 | 0.51 | −2.42 | −0.04 | 0.93 | 0.35 | −0.07 | 0.12 | 0.06 | 0.88 | 0.36 | −2.24 |
| 17 | 16 | 552.16 | −366.04 | neuron | 0.18 | −0.44 | 0.14 | −1.62 | −0.02 | 0.62 | 0.61 | 0.36 | −0.6 | 0.72 | 0.32 | 0.56 | 0.13 |
| 18 | 17 | 568.96 | −477.25 | neuron | 0 | −2.31 | 0.78 | 0.12 | 0.03 | 0.89 | 0.26 | 0.59 | −0.97 | 1.03 | −0.16 | 1.14 | 0.6 |
| 19 | 18 | 801.01 | −415.13 | neuron | 0.34 | 0.6 | −0.09 | 0.16 | −0.33 | 1.41 | 0.45 | −0.04 | −0.54 | 0.08 | −0.76 | −0.87 | 0.09 |
| 20 | 19 | 1336.08 | −88.94 | neuron | 0.6 | 0.55 | −0.54 | −0.51 | −0.78 | 1.08 | 0.53 | −1.26 | 0.7 | −1.25 | 0.69 | 0.55 | 0.58 |
| 21 | 20 | 1345.42 | −131.94 | neuron | 0.3 | −0.1 | 0.22 | −0.21 | 1.09 | −0.06 | 0.86 | −0.13 | −0.55 | 0.97 | 0.74 | −0.32 | 1.35 |
| 22 | 21 | 1391.83 | −178.39 | neuron | 1.32 | −0.07 | −0.15 | −1.67 | 0.18 | −1.21 | 1.49 | 1.21 | 1.39 | −0.27 | 1.66 | −0.18 | 0.21 |
| 23 | 22 | 1624.13 | −91.23 | neuron | −0.93 | 0.62 | 1.38 | 1.02 | 1.08 | −0.6 | −0.57 | −1.42 | 0.46 | 0.69 | 0.82 | 0.27 | 1.07 |

**Figure 1. Demonstration of the seqFISH cell data features, where the blue overlay delineates the spatial coordinates of individual cells, the green overlay denotes the cell type classifications, and the red overlay represents the gene expression levels within each cell.**

It is highly recommended that researchers combine Algorithm 1 with Figure 2 for a better understanding.

**Algorithm 1: Construction of the cell adjacency matrix**

**Input:** Cell spatial coordinates
**Output:** Cell adjacency matrix
**Step 1:** Find Kp proximal interacting cells.
1.1 Construct the spatial connectivity network.
1.2 kNNquery indexing Kp.
1.3 Form the proximal interaction adjacency matrix $Adj_p$.
Adjp: $[[N_1, N_2, \ldots, N_c]_1,[N_1, N_2, \ldots, N_c]_2,\ldots[N_1, N_2, \ldots, N_c]_{(Nc)}]$
**Step 2:** Find $K_{ir}$ intermediate-range interacting cells.
2.1 Coalesce small clusters into a larger cluster.

$$\Delta Q = \left[\frac{\sum in + k_{i,in}}{2m} - \left(\frac{\sum total + k_i}{2m}\right)^2\right] - \left[\frac{\sum in}{2m} - \left(\frac{\sum total}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right], \quad (1)$$

2.2 Iterate until ΔQ converge indexing.
2.3 Form the inter-mediate range interaction adjacency matrix Adjir.
**Step 3:** Construct the cell adjacency matrix Adjc by combining Adjp, Adjir.

In Algorithm 1, in step 1.1, based on the input coordinates of the cells, a spatial connectivity network is constructed using the spatial proximity and geometric relationship. The result is shown in Figure 2A. The subsequent $kNN_{query}$ in step 1.2 uses the spatial connectivity network to find $K_p$ proximal interacting cells, defined as having potential direct cell–cell contact with the query cell. The green circle in Figure 2B is a demonstration of the randomly selected cell (red dot) and its interacting entities. The output of step 1 is the proximal interaction adjacency matrix $Adj_p \in R^{(Nc \times Nc)}$, where $N_c$ is denoted as the number of input cells, the corresponding $K_p$ index is 1, and the others are 0. Step 2 is the construction of the intermediate range interaction adjacency matrix. In step 2.1, the process involves grouping each cell with its neighbors and evaluating whether the maximum gain of modularity (ΔQ) is greater than 0. If the gain is positive, the cell is assigned to the adjacent cell with

the largest modular increment. Step 2.2 is the iteration of step 2.1, until ΔQ in Eq. (1) in Algorithm 1 no longer changes, being defined to be convergence. The result indicates that the cells within the same community are identified as potentially having intermediate-range interactions (ECM) with respect to the query cell, denoted as $K_{ir}$. As a result, the number of potential interacting cells is $K_p + K_{ir}$. The green circle in Figure 3C is the demonstration of a randomly picked cell (red dot) with its proximal interacting cells $K_p$ and intermediate-range interacting cells $K_{ir}$ (green dots). In step 3, we can construct the adjacency matrix by

$$Adj_c(i,j) = 1,\ if\ Adj_p(i,j) = 1\ \cup\ Adj_{ir}(i,j) = 1;\ 0,\ if\ Adj_p(i,j) = 0\ \cap\ Adj_{ir}(i,j) = 0, \quad (2)$$

on the cell level, denoting whether two cells will establish interactions under the conditions of $Adj_p$ and $Adj_{ir}$. Then, the cell-level graph $G_c$ can be constructed using $V_c$ and $Adj_c$.

In summary, Algorithm 1 distinguishes itself from the existing approaches relying on the kNN method, in which only $K_p$ is considered, resulting in the time complexity of $O(N_c)^2$. Instead, taking both $K_p$ and $K_{ir}$ into account, the potential ignorance of interactions that arise from various secretion patterns is prevented [12,13]. The overall time complexity is $O(N_c \log N_c + I \cdot (E+N_c))$, where $I$ denotes the iterations in step 2.2 and denotes the number of edges constructed by the spatial connectivity network in step 1.1. Researchers can easily construct the cell level graph $G_c$ by installing the packages "hnswlib" and "community" in Python.

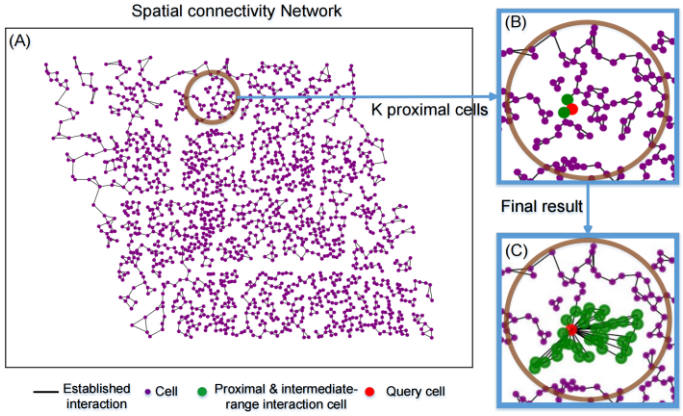



**Figure 2. Demonstration of the initial CCI construction using the seqFISH dataset.** A. The purple dots represent the spatial locations of cells, and the spatial connectivity network generates the linkages between them. B. After step 1 in Algorithm 1, the randomly selected cell is in red in our demonstration, $K_p$ ($K_p = 2$) returns in green and in the same size as the query cell does, denoted as the proximal interacting cells. (C) Cells in green inside the green circle ($K_p+K_{ir}$) are defined to have potential interactions with the query cell in red, denoted as the proximal and intermediate-range interacting cells.

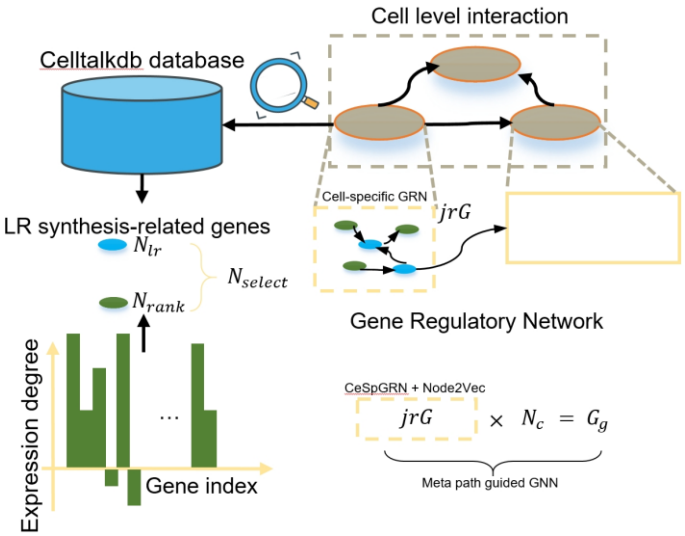



**Figure 3. Construction process of $G_g$**

2. Gene-level graph construction

The construction of the gene-level graph Gg is presented in Figure 3 and involves two main steps:

a. Construction of the cell-specific GRN:

- Select $N_{select}$ ($N_{select}$= $N_{lr}$+ $N_{rank}$) genes, comprising $N_{lr}$ genes related to LR synthesis using standardized LR databases provided by Celltalkdb [14] and $N_{rank}$ highly expressed genes.
- Use the CeSpGRN backbone [15] to obtain the cell-specific gene interaction network, represented as cell-specific gene adjacency matrices ($N_c$ graphs).
- Apply Node2Vec [16] to generate features $F_{select}$ for the selected genes in each cell-specific graph.

b. Construction of the global GRN:

- Combine the cell-specific graphs ($N_c$jrG) into a global gene graph Gg.
- Enrich the gene features by preserving the connections from the cell-specific graphs and using a meta-path-guided graph neural network [17,18] to derive new features $F'_{select}$.
- The final gene features $F_g$ in $G_g$ are obtained by integrating $F_{select}$ and $F'_{select}$.

Researchers can use the following links to customize the gene level graph: https://github.com/PeterZZQ/CeSpGRN, https://github.com/eliorc/node2vec, and https://github.com/zhiqiangzhongddu/PM-HGNN.

## B. Network architecture

Triple-enhancement-based graph neural network (TENET) is an end-to-end structure that could be utilized to retrieve ST data features as well as the graph structure to learn and reproduce. $G_c$ and $G_g$ are separately encoded in the graph convolution network (GCN) encoder in the first stage, obtaining the latent feature embeddings, denoted as $Z_c$ and $Z_g$. $Z_g$ will traverse through the channel enhancement compensation block (CECB) and synergistic enhancement block (SEB). Before $Z_c$ moves on to the next stage, we integrate $Z_c$ and $Z_g$ to become $Z_{(c+g)}$. Then, $Z_{(c+g)}$ will traverse through CECB and the elective enhancement block (EEB). The whole process is illustrated in Figure 4.

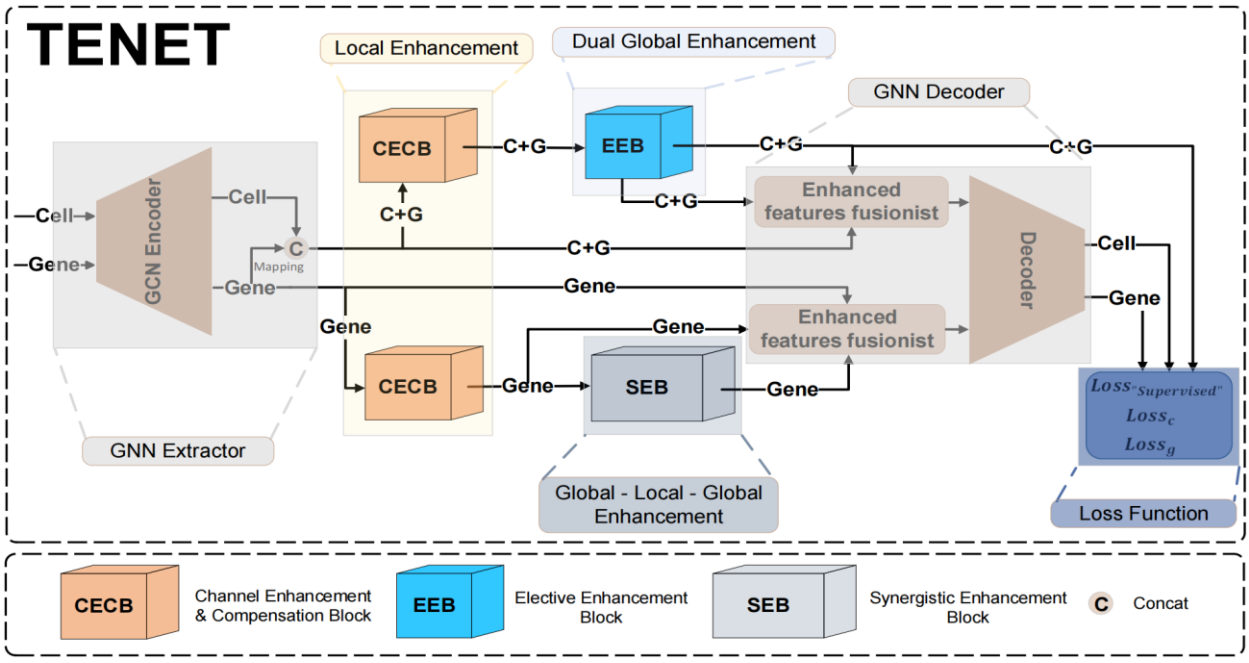



**Figure 4. TENET workflow.** The GCN encoder extracts latent feature embeddings $Z_c$ and $Z_g$ from $G_c$ and $G_g$. Before $Z_c$ is input into the subsequent blocks, we have an integration process converting $Z_c$ into $Z_{(c+g)}$. The subsequent triple-enhancement mechanism can be summarized as local-global-local enhancement. By doing so, the network can not only extract feature representations from the node itself but also from the graph structure. Before the decoding process, we fuse the latent feature embeddings from the enhanced block to have an accumulated effect. For the loss function, we employ the binary cross entropy (BCE) loss function for both cell and gene. Besides, due to the utilization of the triple-enhancement mechanism, if the model is to be subjected to noise or wrongly enhanced during the enhancement process, which is undesirable, it would struggle to converge. Consequently, we have the $Loss_{"supervised"}$ function that enables the model to combat noise and enhance its resilience effectively.

1. GCN encoder

The core of TENET's architecture is built upon the GCN [8], which serves as the foundation for extracting meaningful node representations from the input graph $G_c$ and $G_g$ and outputs the latent feature embeddings of cells and genes denoted as $Z_c$ and $Z_g$, respectively. The calculation process of the two-layer GCN encoder is presented as

$$H_{c(g)}^{layer_n} = \begin{cases} V_{c(g)}(Original\ node\ features, & n = 0; \\ ReLU(\widetilde{D}^{-\frac{1}{2}}\widetilde{Adj}_{c(g)}\widetilde{D}^{-\frac{1}{2}}H^{layer_{n-1}}W^{layer_{n-1}}, & n > 0, \end{cases}$$
$$where\ \widetilde{Adj}_{c(g)} = Adj_{c(g)} + Identity,$$
$$Z_{c(g)} = H_{c(g)}^{layer_n} \quad (3)$$

In Eq. (3), the nonlinear activation function ReLU introduces nonlinearity into the node representations. $\widetilde{D}$ is the degree of each node. $\widetilde{Adj}_{c(g)}$ is denoted as the adjacency matrix with self-loop (a linkage connected to the node itself). $H^{layer_{n-1}}$ is denoted as the features extracted from the last layer; specifically, if layer = 0, $H^{layer_{n-1}}$ are the original node features. $W^{layer_{n-1}}$ is the learnable weight matrix from the previous layer, enabling the model to learn feature-specific transformations and effectively aggregate information from a node's local neighborhood. When inputting $Z_c$ to the next block, we first convert it into $Z_{(c+g)}$, introducing cross-resolution embedding to extend the model's ability to capture different data patterns.

2. Channel enhancement and compensation block (CECB)

The CECB in Algorithm 2 is a streamlined transition block; $Z_{(c+g)}$ and $Z_g$ are enhanced in the channel dimension, aiming to emphasize the relevant feature embeddings and mitigate the over-smoothing problem caused by the GCN encoder [19].

**Algorithm 2. Channel enhancement and compensation block (CECB)**

**Input:** Input feature: $Z_{(c+g)}$, $Z_g$; Operation: Normalization: Batch normalization (BN), Activation function (ϕ,ω); Two fully connected layers [$FC_1$, $FC_2$].
**Output:** $Z'_{(c+g)}$, $Z'_g$.
The results of the intermediate operations are represented using "temp" followed by different subscripts.
**if** $Z_{(c+g)}$ **then**
$temp_{c+g} = Z_{(c+g)}$-BN($Z_{c+g}$)
**else**
$temp_g = Z_g$
**Endif**
Parallel operation for $temp_{(c+g)1}$, $temp_{(g)1}$.
$temp_{(c+g)2} = \phi(FC_1(temp_{(c+g)1}))$, $temp_{(g)2} = \omega(FC_2(temp_{(g)1}))$
$temp_{(c+g)3} = \phi(FC_1(temp_{(c+g)2}))$, $temp_{(g)3} = \omega(FC_2(temp_{(g)2}))$
$temp_{(c+g)4} = temp_{(c+g)2} * temp_{(c+g)3}$, $temp_{(g)4} = temp_{(g)2} * temp_{(g)3}$

**if** $temp_{(c+g)4}$ **then**
$Z'_{c+g} = temp_{(c+g)1} + temp_{(c+g)4}$
**else**
$Z'_g = BN(temp_{(g)1} + temp_{(g)4}$
**Endif**
**return Output:** $Z'_{(c+g)}$, $Z'_g$.

3. Synergistic enhancement block (SEB)

The enhancement of $Z'_g$ in the SEB is done jointly by the global branch enhancement and the local branch enhancement. In Algorithm 3, the global branch enhancement utilizes the graph attention network (GAT) [20], and the local branch enhancement acts as a buffer to integrate the feature embeddings extracted from the GAT.

**Algorithm 3. Synergistic enhancement block (SEB)**

**Input:** Input features: $Z'_g$; Operation: Graph Attention Network (GAT), Convolution ($Conv_1$), Batch normalization (BN), Activation function (ϕ,σ); Linear transformation (MLP, contains three fully connected layers [$FC_3$,$FC_4$,$FC_5$]); Kernel list: K = [1, 3, 5]; Group Size: G; Latent dimension: $L_{g1}$,$L_l$ ,$L_{g2}$; Storage list: sl = []; Attention weight list: awl = [].

**Output:** Z"g.

The result of the intermediate operations are represented using "global"/"local" followed by different subscripts.

**Global enhancement operation:**

$$global_{enhance1} = \mathrm{GAT}(Z'_g)[\text{in channel } (F'_g), L_{g1}, \text{attention heads}].$$

The feature dimension turns into $global_{enhance1} \in \mathbb{R}^{N_g \times (L_{g1} * attention\ heads)}$.

**Local enhancement operation:**

**for** k in K **do:**

$$local_k = \sigma(\mathrm{BN}\left(Conv_1\left(global_{enhance1}, padding = \frac{k}{2}\right)\right)).$$

sl.append ($local_k$).

**End for**

$local_{feature} = stack(sl)$.

S=sum(sl).mean

**for** FC in MLP **do**

awl.append(FC(S))

**End for**

$$local_{enhance} = \phi(\mathrm{stack(awl)}) * local_{feature}.$$

The feature dimension turns into $local_{enhance} \in R^{Ng \times Ll}$.

**Global enhancement operation:**

$$global_{enhance2} = \mathrm{GAT}(local_{enhance})[\text{in channel } (L_l), L_{g2}, \text{attention heads}].$$

The feature dimension turns into $global_{enhance2} \in \mathbb{R}^{N_g \times (L_{g2} * attention\ heads)}$.

**return Output:** Z"g = $global_{enhance2}$.

4. Elective enhancement block (EEB)

The EEB in Algorithm 4 amplifies the latent feature embeddings by augmenting the spatial dimensions and leveraging adaptive feature selection [21], thereby bolstering the richness and caliber of the embeddings. The adaptive enhanced feature election mechanism in EEB bears a resemblance to the local branch in SEB.

**Algorithm 3. Elective enhancement block (EEB)**

**Input:** Input features: $Z'_{(c+g)}$; Operation: Adaptive average pooling(AAP), Convolution ($Conv_2$,$Conv_3$), Batch normalization (BN), Activation function (θ,ω); Two fully connected layers [$FC_6$,$FC_7$]); Latent dimension: L.

**Output:** $Z''_{(c+g)}$

The results of the intermediate operations are represented using "temp" followed by different subscripts.

$$temp_1, temp_2 = Duplicate(Z'_{c+g}).$$

Parallel operation for $temp_1$, $temp_2$.

$temp_3$ = AAP($temp_1$), $temp_4$ = AAP($temp_2$)

$temp_5$=Concat($temp_3$, $temp_4$).

**Elective Enhancement operation:**

$temp_6$ = θ(BN($Conv_2$ ($temp_5$)))

The feature dimension turns into $temp_6 \in \mathbb{R}^{N_C \times L}$.

$temp_7$, $temp_8$ = split($temp_6$)

Parallel operation for $temp_7$, $temp_8$.

$temp_9$ = ω($FC_6$($Conv_3$ ($temp_7$))), $temp_{10}$ = ω($FC_6$($Conv_3$($temp_8$))).

The feature dimension turns into $temp_{10} \in R^{NC \times F''c+g}$.

**return Output:** $Z''_{c+g}$ = $FC_7$ ($Z'_{c+g}$) * $temp_9$*$temp_{10}$.

5. Triple-enhancement fusionist decoder (TEFD)

Triple-enhancement fusionist decoder (TEFD) fuses the enhanced latent feature embeddings from the previous blocks as input to the inner product decoder [22]. The two decoders for cell and gene are parallel, denoted as $D_c$ and $D_g$. The fusion mechanisms for the latent feature embeddings of cells and genes are:

$\text{Fusion}_c = \text{sum}(Z_{c+g}, Z'_{c+g}, Z''_{c+g})$. (4)
$\text{Fusion}_g = \text{sum}(Z_g, Z'_g, Z''_{c+g})$. (5)

The first terms in Eq. (4) and Eq. (5) $Z_{(c+g)}$ and $Z_g$ represent the raw latent feature embeddings obtained from the GCN encoder in subsection a. The second terms $Z'_{(c+g)}$ and $Z'_g$ undergo the CECB (primary enhancement) in subsection b. The third term $Z''_g$ is dual enhanced embeddings from SEB (secondary enhancement and tertiary enhancement) in subsection c. Analogously, $Z''_{(c+g)}$ is also a dual enhanced embedding when exported from EEB (secondary enhancement and tertiary enhancement) in subsection d. Each enhancement module incrementally amplifies inherent feature embeddings across different scales by progressively augmenting the channel dimensions, leading to a cumulative effect in $\text{Fusion}_{c(g)}$. To integrate the fusion features a step further, we have

$$Z'''_{c(g)} = \text{FC}_9(\text{ReLU}(\text{FC}_8(\text{Fusion}_{c(g)}))). \quad (6)$$

Such a process discerns and extracts the salient features crucial for the subsequent decoding process.

The decoding process of $D_c$ and $D_g$ are summarized as follows: $Z'''_{c(g)} \rightarrow Transpose \rightarrow Z'''^{T}_{c(g)} \rightarrow cdot\left(Z'''_{c(g)}, Z'''^{T}_{c(g)}\right) \rightarrow Sigmoid(Adj_{recon}^{N_{c(g)} \times N_{c(g)}})$. The usage of the sigmoid activation is to limit the interaction scores between 0 and 1, representing the probability of the existence of an interaction between the corresponding cells (genes). Generally, TEFD ensures the interconnection and information flow among the blocks. The output of each block not only serves as input for the subsequent enhancement block but also contributes to the final reconstruction of CCI. By incorporating multi-scale information, a more nuanced understanding of cellular (genetic) interacting patterns can be achieved, allowing for a more accurate determination of communication establishment among cells.

6. Loss function

We employ the binary cross entropy (BCE) loss and propose a supervised loss in Algorithm 4. The BCE loss can achieve refinement in both the cell graph and gene graph and encourages the production of well-calibrated probabilities by penalizing large errors in the predicted probabilities [23].

To further correctly augment cell features, we have the supervised loss ($\text{Loss}_{\text{"supervised"}}$). The inputs are the output of EEB and the output of the GCN encoder (after mapping and concatenating), the $Z'_{(c+g)}$ and $Z_{(c+g)}$, respectively, and its core concept is using the cosine similarity [24]. The embeddings at the same index are calculated in pairs, aiming to learn representations that bring similar cell embeddings in closer proximity [25] and dissimilar embeddings further apart. The triple-enhancement mechanism is employed to accentuate the specificity of the cells without significantly altering their representations. Therefore, a higher similarity value implies that the features are correctly enhanced and iteratively adjusted toward the desired enhancement, while a lower similarity value suggests the need for further adjustment to differentiate the features, leading to a modification of the cells' intrinsic properties. In addition to the embeddings being correctly enhanced, the triple-enhancement technique showcases remarkable robustness against noise. If noise is artificially simulated, the triple-enhancement structure can render the noise more prominent and distinguishable, and the cosine similarity between the enhanced and the original noise tends to be small. Therefore, TENET can effectively mitigate and suppress noise, thereby improving the reliability and stability of the model.

**Algorithm 4. Supervised loss function**

**Input:** Input features: $Z_{c+g}$, $Z''_{c+g}$.
**Output:** Loss value.
Emb=concat($Z_{c+g}$, $Z''_{c+g}$)
Construct the similarity matrix → Sim(Emb, $\text{Emb}^T$)
Elements of the same index → diagonal elements; Elements of different index → off-diagonal elements
**return Output:** Loss value $= -\log\frac{sum(e^{diagonal})}{sum(e^{off-diagonal})}$(Sim).

## Validation of protocol

Two hyper-parameter experiments A (train-test-split ratio) and B (tolerance to noisy data) are presented to validate the efficiency of TENET.

### A. Train-test-split ratio

To validate the protocol, TENET's model performance is compared with the two most related research DeepLinc and CLARIFY. For the validation, three ST datasets (seqFISH, MERFISH, and scMultiSim) are utilized, with a total of 120 iterations. Various train-test-split ratios (10%, 30%, 50%, 70%, and 90%) are employed to divide the data into training and testing subsets for evaluation. Figure 5A presents the segmentation results for a 70% train-test-split ratio (with a 30% proportion as the training set), where TENET is represented in red, CLARIFY in yellow, and DeepLinc in green. Figure 5B shows the AP results with the maximum and minimum values for each segmentation condition, with DeepLinc in green, CLARIFY in blue, and TENET in red. Notably, TENET consistently outperforms the other two models across all segmentation ratios on the three datasets.

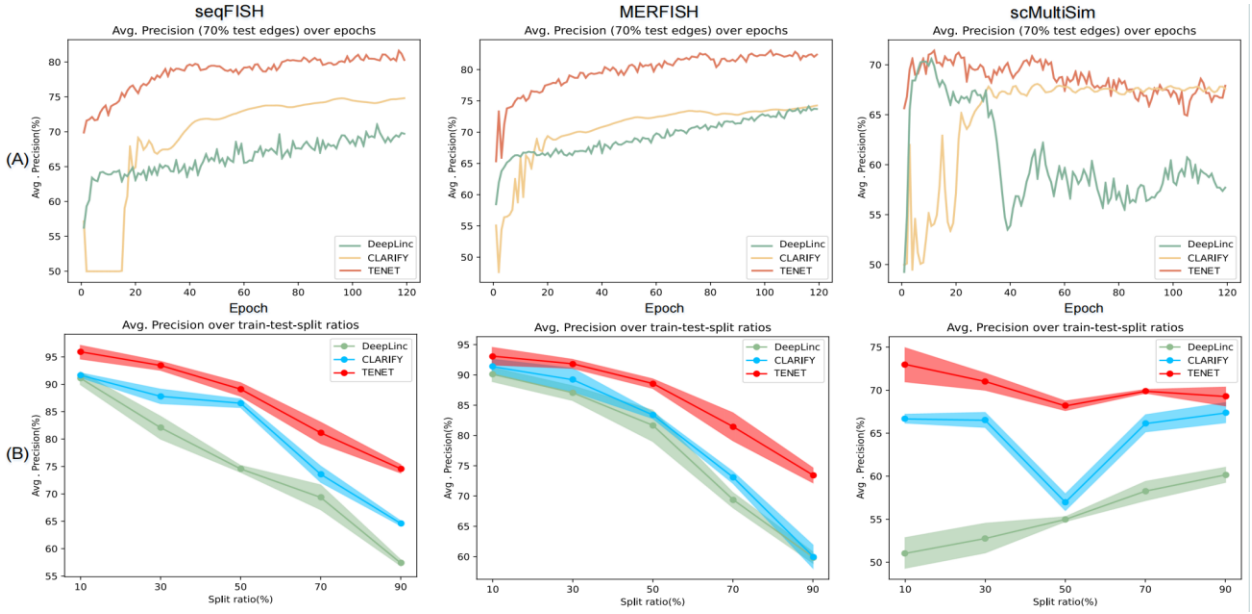



**Figure 5. Model performance comparison.** A. Experiment conducted on three datasets under the segmentation of 70% (TENET in red, CLARIFY in yellow, and DeepLinc in green). B. The shadow in the shadow plot with the upper and lower boundaries extends to the maximum and minimum values of the AP results. DeepLinc in green, CLARIFY in blue, and TENET in red. Results are obtained from three repeated experiments.

### B. Tolerance to noisy data

The tolerance to noisy data experiment is designed to simulate poor data quality, including error-detection issues and missing or incorrectly detected edges in the experimental setup. In $Noise_1$ experiment, a certain percentage of false edges are intentionally added to $G_c$ and $G_g$. In $Noise_2$ experiment, the existing edges are removed in proportion. We aim to mimic these scenarios and assess the model's ability to handle such challenges. These experiments are conducted under the condition of train-test-split ratios of 10%, 30%, 50%, 70%, and 90% on the dataset of seqFISH. The proportion of $Noise_1$ and $Noise_2$ increases from 0.0 to 0.5 with steps of 0.1, resulting in 36 AP values in each subplot in Figure 6. The values outside the bracket in Table 2 can represent the stability of the model with the increment of noise ($Noise_1$ and $Noise_2$) ratios. The value outside the bracket on one train-test-split ratio represents the elevation difference between $Value_1$ ($Noise_1$ and $Noise_2$ = 0.0) and $Value_2$ ($Noise_1$ and $Noise_2$ = 0.5). The value inside the bracket at one train-test-split ratio represents the average value of 36 AP, used to facilitate comparison among models.

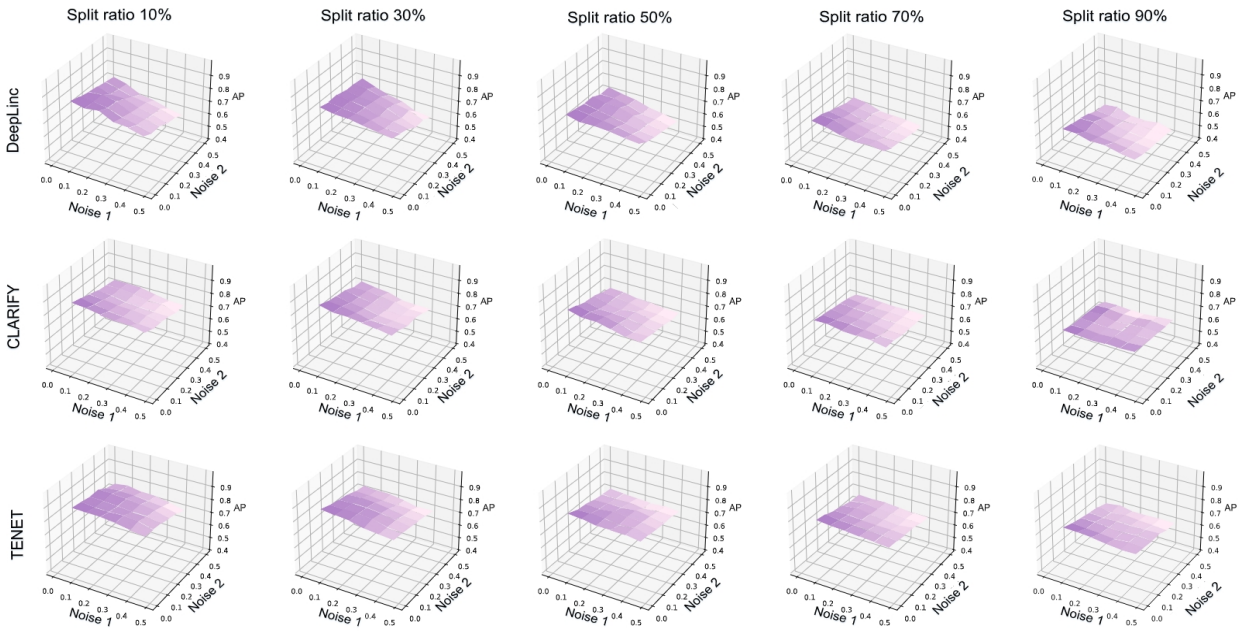



**Figure 6. Experiment conducted on three of the models using the dataset of seqFISH aiming to evaluate the model's tolerance to noise (Noise$_1$ and Noise$_2$).** At each train-test-split ratio, the ratio of Noise$_1$ and Noise$_2$ gradually increases from 0 to 0.5 steps with a size of 0.1. The result in each subgraph is the average of three independent experiments under the condition of 120 epochs execution.

**Table 2. AP elevation difference of Noise$_1$, Noise$_2$ = 0.0, and (Noise$_1$ and Noise$_2$) = 0.5 on the same train-test-split ratio. The value in the bracket is the average AP from all proportions of the two noises on the same train-test-split ratio (36 AP values on one train-test-split ratio).**

| AP elevation difference (Avg. AP) | Train-test-split ratio | | | | |
|---|---|---|---|---|---|
| Model | 10 | 30 | 50 | 70 | 90 |
| DeepLinc | 30.22 (77.39) | 26.91 (74.01) | 23.74 (66.21) | 22.09 (57.86) | 17.27 (49.83) |
| CLARIFY | 17.20 (83.86) | 19.12 (81.04) | 18.33 (75.13) | **10.13** (69.22) | **6.75** (61.64) |
| TENET | **16.25 (88.89)** | **16.73 (85.37)** | **12.73(82.17)** | 12.12 **(75.39)** | 7.02 **(67.58)** |

* The best result for each model on each split ratio is highlighted in bold, and the second-best result is underlined.

# General notes and troubleshooting

## General notes

In Figure 7, there are two approaches to setting up the TENET environment. The simplest method is to utilize the provided environment.yml file (A); however, some dependencies may not be easily installed. Therefore, the recommended approach is the second way (B). Initially, create a Python virtual environment, then activate it and start installing the packages using the *requirement.txt* file. This file contains packages that can be downloaded without significant difficulty. Before installing the remaining packages, verify the CUDA version you are employing. If the CUDA version differs from this, the corresponding URL should be modified accordingly.

A. Use the provided .yml file to install the dependencies and create the TENET environment.

```
conda env create -f environment.yml
```

To activate the environment:

```
conda activate TENET
```

B.

Create a Python environment.

```
conda create --name TENET python=3.8
```

Activate the TENET environment, and start installing the packages.

```
conda activate TENET
pip install -r requirements.txt
```

To install the other packages, check the Cuda version first.

```
# To check the Cuda version
nvcc -V
```

Install the rest of the packages according to the Cuda version.

```
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu117
pip install xxx.whl
```

**Figure 7. Environment preparation**

To start training the model, use *cd TENET* to enter the program directory and use *python main.py.* The hyper-parameters can be adjusted according to the actual demand. The explanations of hyper-parameters in Figure 8 are as follows:

- *-m:* "preprocess", this mode will construct the cell-level graph and gene-level graph using the provided method. This is necessary to generate the required graph structures before training the model. "train" this mode assumes that the cell-level graph and gene-level graph have already been constructed. It will use the provided graph structures to start the CCI reconstruction process.
- *-i:* the input data frame path.
- *-o:* the output results' path.
- *-t:* train-test split ratio, the value ranges from 0 to 1. If *default = 0.7*, the test edges account for 70% percent of all.
- *-n, -k:* these two hyper-parameters will be used when the mode (*-m*) is set to be "preprocess, where *-n* specifies the number of selected genes to be used when constructing the gene-level graph within each cell, *-k* controls the number of proximal interaction cells to be used.
- *-fp, -fn:* the two noise ratios, their values range from 0 to 1. Introducing artificial noise to the constructed cell level graph can examine the model's tolerance to noises.
- *-lr:* the value of the learning rate.

## Troubleshooting

The PyTorch version in TENET is 2.0.1 and the CUDA version is 11.7.
When installing PyTorch-related dependencies (*torch-cluster, torch-scatter, torch-sparse, torch-spline-conv,* and *torch-geometric*), they can be challenging to install and may require extended installation times. The recommendation is to download the respective *.whl* file via https://pytorch-geometric.com/whl (ensure you check the PyTorch version and CUDA version) and proceed with the installation.

Enter into the TENET directory, run the second instruction to start model training.

```
cd TENET
python main.py
```

Hyperparameters are as follows, modify them according to your need.

```
def parse_arguments():
    parser = argparse.ArgumentParser(description='GAE arguments')
    parser.add_argument("-m", "--mode", type=str, default = "train",
        help="GAE mode: preprocess,train")
    parser.add_argument("-i", "--inputdirpath", type=str,default = "../dataset/seqFISH/seqfish_dataframe.csv",
                    help="Input directory path where ST data is stored")
    parser.add_argument("-o", "--outputdirpath", type=str,default = "../output/seqfish",
                    help="Output directory path where results will be stored ")
    parser.add_argument("-s", "--studyname", type=str,default='TENET',
                     help="study name")
    parser.add_argument("-t", "--split", type=float,default = 0.7,
        help="# of test edges [0,1)")
    parser.add_argument("-n", "--numgenespercell", type=int, default = 60,
                help="Number of genes in each gene regulatory network")
    parser.add_argument("-k", "--nearestneighbors", type=int, default = 2,
                help="Number of nearest neighbors for each cell")
    parser.add_argument("--fp", type=float, default=0,
        help="(experimentation only) # of false positive test edges [0,1)")
    parser.add_argument("--fn", type=float, default=0,
        help="(experimentation only) # of false negative test edges [0,1)")
    parser.add_argument("-a", "--ownadjacencypath", type=str,default = None,
        help="Using your own cell level adjacency (give path)")
    parser.add_argument("-lr",type=float,default = 0.8,
        help="Using your lr (learning rate)")
    args = parser.parse_args()
    return args
```

**Figure 8. Hyper-parameters settings**

## Acknowledgments

This work is supported by the Guangdong Provincial Department of Education (2022KTSCX152), the Key Laboratory IRADS, Guangdong Province (2022B1212010006, R040000122), Guangdong Higher Education Upgrading Plan (2021-2025) with UIC Research Grant UICR0400025-21. The icons of the graphical abstract are created from https://BioRender.com.

## Competing interests

The authors declare no conflicts of interest.

Received: July 18, 2024; Accepted: December 25, 2024; Available online: January 21, 2025; Published: February 05, 2025

## References

1. Schwager, S. C., Taufalele, P. V. and Reinhart-King, C. A. (2018). Cell–Cell Mechanical Communication in Cancer. *Cell Mol Bioeng.* 12(1): 1–14. https://doi.org/10.1007/s12195-018-00564-x

2. Dries, R., Zhu, Q., Dong, R., Eng, C. H., Li, H., Liu, K., Fu, Y., Zhao, T., Sarkar, A., Bao, F., et al. (2021). Giotto: a toolbox for integrative analysis and visualization of spatial expression data. *Genome Biol.* 22(1): 1–31. https://doi.org/10.1186/s13059-021-02286-2
3. Tanevski, J., Flores, R. O. R., Gabor, A., Schapiro, D. and Saez-Rodriguez, J. (2022). Explainable multiview framework for dissecting spatial relationships from highly multiplexed data. *Genome Biol.* 23(1): 1–31. https://doi.org/10.1186/s13059-022-02663-5
4. Li, R. and Yang, X. (2022). De novo reconstruction of cell interaction landscapes from single-cell spatial transcriptome data with DeepLinc. *Genome Biol.* 23(1): 1–24. https://doi.org/10.1186/s13059-022-02692-0
5. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. and Monfardini, G. (2008). The Graph Neural Network Model. *IEEE Transactions on Neural Networks*. 20(1): 61–80. https://ieeexplore.ieee.org/document/4700287
6. Peterson, L. E. (2009). K-nearest neighbor. Scholarpedia, 4(2):1883.
7. Bafna, M., Li, H. and Zhang, X. (2023). CLARIFY: cell–cell interaction and gene regulatory network refinement from spatially resolved transcriptomics. *Bioinformatics* 39: i484–i493. https://doi.org/10.1093/bioinformatics/btad269
8. Kipf, T. N. and Welling, M (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 https://doi.org/10.48550/arXiv.1609.02907
9. Li, H., Zhang, Z., Squires, M., Chen, X., Zhang, X. (2023). scMultiSim: simulation of multi-modality single cell data guided by cell-cell interactions and gene regulatory networks. *Res Sq*. 15: rs.3.rs-2675530. https://doi.org/10.21203/rs.3.rs-2675530/v1
10. Zhang, X., Niu, X., Fournier-Viger, P. and Wang, B. (2022). Two-Stage Traffic Clustering Based on HNSW. *Lect Notes Comput Sci.* 609–620. https://doi.org/10.1007/978-3-031-08530-7_51
11. Zhang, J., Fei, J., Song, X. and Feng, J. (2021). An Improved Louvain Algorithm for Community Detection. *Math Probl Eng.* 2021: 1–14. https://doi.org/10.1155/2021/1485592
12. Cang, Z., Zhao, Y., Almet, A. A., Stabell, A., Ramos, R., Plikus, M. V., Atwood, S. X. and Nie, Q. (2023). Screening cell–cell communication in spatial transcriptomics via collective optimal transport. *Nat Methods.* 20(2): 218–228. https://doi.org/10.1038/s41592-022-01728-4
13. McCoy-Simandle, K., Hanna, S. J. and Cox, D. (2016). Exosomes and nanotubes: Control of immune cell communication. *Int J Biochem Cell Biol.* 71: 44–54. https://doi.org/10.1016/j.biocel.2015.12.006
14. Shao, X., Li, C., Yang, H., Lu, X., Liao, J., Qian, J., Wang, K., Cheng, J., Yang, P., Chen, H., et al. (2022). Knowledge-graph-based cell-cell communication inference for spatially resolved transcriptomic data with SpaTalk. Nat Commun. 13(1): 4429. https://doi.org/10.1038/s41467-022-32111-8
15. Zhang, Z., Han, J., Song, L. and Zhang, X. (2022). CeSpGRN: Inferring cell-specific gene regulatory networks from single cell multi-omics and spatial data. *bioRxiv* : e482887. https://doi.org/10.1101/2022.03.03.482887
16. Grover.A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pages 855–864.
17. Zhong, Z., Li, C. T. and Pang, J. (2022). Personalised meta-path generation for heterogeneous graph neural networks. *Data Min Knowl Discovery.* 36(6): 2299–2333. https://doi.org/10.1007/s10618-022-00862-z
18. Zhang, J. and Zhu, Y. (2021). Meta-path Guided Heterogeneous Graph Neural Network For Dish Recommendation System. *J Phys Conf Ser.* 1883(1): 012102. https://doi.org/10.1088/1742-6596/1883/1/012102
19. Zhang, Y., Yan, Y., Li, J. and Wang, H. (2023). Mrcn: A novel modality restitution and compensation network for visible-infrared person re-identification. *arXiv.* 2303.14626. https://doi.org/10.48550/arXiv.2303.14626
20. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P. and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv*:1710.10903, 2017. https://doi.org/10.48550/arXiv.1710.10903
21. Hu, J., Shen, L. and Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7132–7141. https://ieeexplore.ieee.org/document/8578843
22. Kipf, T. N. and Welling, M. (2016). Variational graph auto-encoders. *arXiv:* 1611.07308. https://doi.org/10.48550/arXiv.1611.07308
23. Ruby, U. and Yendapalli, V. (2020). Binary cross entropy with deep learning technique for image classification. *Int J Adv Trends Comput Sci Eng.* 9(10). http://dx.doi.org/10.30534/ijatcse/2020/175942020
24. Nguyen, H. V. and Bai, L. (2011). Cosine Similarity Metric Learning for Face Verification. *Lect Notes Comput Sci.*: 709–720. https://doi.org/10.1007/978-3-642-19309-5_55
25. Sohn, K. (2016). Improved deep metric learning with multi-class n-pair loss objective. *Adv Neural Inf Process Syst*. 29: 1857–1865.