Supplementary Notes

November 6, 2024

1 List of notations

1.1 Model input

- The expression matrix $\mathrm{Exp} \in \mathbb{R}^{k \times c}$, where k represents the number of cells in the neighborhood and c is the number of measured genes in this dataset.
- The distance matrix distance $\in \mathbb{R}^{k \times k}$, describing the pairwise distances between the cells.
- The cell type vector $T \in \mathbb{R}^{k}$, where each entry corresponds to one of the t cell types identified in this dataset.

1.2 Embedding module

- The function of multi-layer perceptron (MLP), suppose the input is a tensor $x \in \mathbb{R}^{* \times c_1}$, where $*$ can be any values or sequence of values, c_1 is its number of features. And we want the output $\in \mathbb{R}^{* \times c_2}$, where c_2 is the number of dimensions we want to project x on. The MLP uses two linear transformations with the GELU activation function to make the encoding. Specifically, output $= \mathrm{GELU}(xW_1)W_2$, where W_1 and W_2 are the weight matrix of the two linear transformation.
- Node features are represented by the matrix $X \in \mathbb{R}^{k \times D_V}$, where k corresponds to the number of cells in a neighborhood, and D_V denotes the embedding dimension of the nodes. Analogous to the input tensor of a language model, each cell in this context is analogous to a word (token), and the entire neighborhood of k cells is treated as a sentence comprising k words.
- Edge features are represented by the tensor $E \in \mathbb{R}^{k \times k \times D_E}$, where D_E represents the embedding dimension of the edges. This tensor characterizes the interactions between every pair of cells within the k-cell neighborhood.
- Distance embeddings are represented by the tensor $DE_1 \in \mathbb{R}^{k \times k \times D_E}$, $DE_2 \in \mathbb{R}^{k \times k \times \#L}$, and $DE_3 \in \mathbb{R}^{k \times k \times D_E}$, detailing the influence of the spatial distance $d \in \mathbb{R}^{k \times k}$ between each pair of cells in the neighborhood.

1.3 Downstream analysis

1.3.1 Processing of the influence tensor

- Influence tensor $I \in \mathbb{R}^{N \times (k-1) \times c}$, where N is the number of cells in the analyzed slide (tissue section), c is the number of measured genes in the dataset and k-1 is the number of cells in the cell neighborhood used for prediction. This tensor characterizes how each cell's each gene's state is influenced by its top k-1 nearest neighbor cells. We can select the top 5 items along the second dimension for the UMAP visualization of CCI pairs (How one sender cell influences each gene's state in the corresponding receiver cell that forms an interaction cell pair).
- Proportionally normalized influence tensor $I_p \in \mathbb{R}^{N \times (k-1) \times c}$, which is the absolute value of the influence tensor I with all items within the same second dimension sum up to 1. I_p represents the proportional influence from the top k-1 nearest neighbors targeting each cell's each target gene.
- Proportionally normalized influence tensor averaged across all genes $I_s \in \mathbb{R}^{N \times (k-1)}$, which is the absolute value of the influence tensor I with all items within the same second dimension sum up to 1. I_s quantifies how each cell is proportionally influenced by its top k-1 nearest neighbors averaged across all target genes.
- Aggregated influence tensor $\mathbb{I} \in \mathbb{R}^{N \times t \times c}$, where N refers to the number of cells in the analyzed slide, t refers to the number of unique cell types in this slide, and c donates the number of measured genes. This tensor characterizes how each cell's each gene's state is influenced by other cell types.

1.3.2 Statistical inference of CCI network

- Partial linear regression model for interaction from cell type A to cell type B targeting gene i: $Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$, where Y_i represents the cell state expression of gene i of type B cells, X denotes the proportional influence of type A cells on type B targeting gene i extracted from the aggregated influence tensor $\mathbb{I}$, and the error term ϵ_i is assumed to be independently and identically distributed (i.i.d.) following a normal distribution with zero mean and unknown variance σ^2.
- Hypothesis test. $\hat{\beta}_1$ refers to the estimated value of true value in the partial linear regression model β_1.
- Z-score in the hypothesis test of partial linear model z.
- Number of cells with cell type B in the slide n_b.
- The cumulative distribution function of the normal distribution Φ.
- Type I error cutoff (p-value) α.
- Type II error cutoff (1 - power) β.
- Number of slides (tissue sections) in the dataset S.

2 Details of distance embedding

Given a distance matrix d that characterizes the distance between each cell in the input cell neighborhood, we can calculate the distance feature tensor DF by:

$$DF = \text{stack}\left(\frac{1}{1+\mathrm{d}}, \frac{1}{1+\mathrm{d}^2}, \frac{1}{1+\sqrt{\mathrm{d}}}, e^{-\mathrm{d}}, e^{-\mathrm{d}^2}\right). \tag{1}$$

where $DF \in \mathbb{R}^{k\times k\times 5}$ and "stack" means unsqueeze the last dimension for all the tensors and concatenate them along that dimension.

Then, the three distance embedding DE_1, DE_2, and DE_3 can be calculated by:

$$DE_1 = \text{MLP1}(DF) \tag{2}$$
$$DE_2 = \text{Sigmoid}(\text{MLP2}(DF)) \tag{3}$$
$$DE_3 = \text{Sigmoid}(\text{MLP3}(DF)) \tag{4}$$
$$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{5}$$

where MLP1 and MLP3 transform the last dimension of features from 5 to the D_E, and MLP2 transform the last dimension of features from 5 to the number of measured ligands in the dataset #L.

3 Details of edge embedding

The edge embeddings take in 3 components: ligand expressions, DE_1, and DE_2. It contains 3 steps: calculation of ligand level, ligand transformation, and integration with DE_1.

Firstly, for different ligands, we compute the corresponding ligand level using the measured expressions of the gene(s) that make up this ligand. Similar to Neuronchat, AND logic (i.e., geometric mean) is applied among different groups of genes; since the genes within the same group are redundant for the same function, the OR logic (i.e., arithmetic mean) is applied.

The computed ligand level, denoted as ligand_level, initially has dimensions $\mathbb{R}^{k\times \#L}$. It is then expanded along the second dimension to the dimensions of $\mathbb{R}^{k\times k\times \#L}$, whereby each element along this second dimension replicates the original values. Subsequently, it is encoded by a linear transformation:

$$\text{Ligand\_score} = \text{Linear}(\text{Ligand\_level} \odot DE_2). \tag{6}$$

where $\odot$ denotes element-wise multiplication of tensors using the Hadamard product and Linear donates a linear transformation from the dimension of $\#L$ to D_E.

Then, the edge features is crafted by integrating the ligand score with another transformation of the distance features DE_1:

$$E = W(\text{concatenate}(\text{Ligand\_score}, DE_1)) \times \sqrt{D_E} \tag{7}$$

where $\mathrm{W} \in \mathbb{R}^{2D_E\times D_E}$ represents a weight matrix that transforms the concatenated tensor with shape $\mathbb{R}^{k\times k\times 2D_E}$ to $\mathbb{R}^{k\times k\times D_E}$. The purpose of multiplying by $\sqrt{D_E}$ is to appropriately scale the edge features, thereby ensuring their comparability with the attention matrix that is subsequently computed within the encoder.

4 Details of graph attention calculation

We first calculate the attention matrix generated from the node features:

$$K = XW_K.\text{reshape}(k, D_A, 2D_E).\text{permute}(2,0,1) \tag{8}$$
$$Q = XW_Q.\text{reshape}(k, D_A, 2D_E).\text{permute}(2,1,0) \tag{9}$$
$$\text{KQ} = (QK).\text{permute}(1,2,0) \tag{10}$$

Here, node features are transformed into key (K), query (Q), and value (V), each has the shape of $\mathbb{R}^{k\times(2D_E\times D_A)}$, via linear transformations using learnable weight matrices $W_K, W_Q \in \mathbb{R}^{C\times(2D_E\times D_A)}$. Here, D_A donates the dimension required for one-head attention computation, which is 8 as the default. These matrices are then reshaped and permuted to create a multi-head attention matrix $\text{KQ} \in \mathbb{R}^{2D_E\times k\times k}$, akin to the approach in the original attention mechanism paper. In multi-dimensional arrays, the "permute" operation rearranges the order of the dimensions. This is analogous to reordering the axes in a coordinate system, which changes how we access and interpret the data, but does not alter the data itself. After the permutation, the attention matrix is permuted to align with the shape of the edge features $\in \mathbb{R}^{k\times k\times D_E}$. This matrix is designed to encapsulate latent relational information not directly measured in the dataset but potentially inferable. For example, if gene B produces a ligand not measured in the single-cell-resolution spatial transcriptomics dataset, yet its expression closely correlates with gene A (which is measured), this methodology facilitates the incorporation of gene B's information into the cellular communication model. We preserve the original dot-product attention, as opposed to the additive attention used in GRIT. This choice is due to the "AND" logic inherent in ligand-receptor (LR) pairs: the absence of either a receptor or ligands nullifies the signaling pathway, despite the nonzero sum of ligand and receptor levels.

Once the attention matrix is derived from the node features, we update the edge features through a composite process. This update integrates the newly calculated attention matrix with previous edge features and the distance embedding. This dynamic update procedure refines the edge features, capturing the complexities of cell-cell interaction (CCI), which is influenced by multiple factors. These include spatial proximity, both observed and unobserved ligand-receptor (LR) pairs, and additional elements such as the extracellular matrix, potentially governed by specific genes. This can be mathematically represented as follows:

$$E = \text{GELU}(\rho(\text{KQ}[:,:,:D_E/2] \odot EW_{Ew}) + EW_{Eb} + \text{KQ}[:,:,D_E/2:] \odot DE_3) \tag{11}$$
$$\rho(x) = \sqrt{\text{relu}(x)} - \sqrt{\text{relu}(-x)} \tag{12}$$

where $[:,:,:D_E/2]$ donates slicing the tensor along the last dimension, ρ represents the signed-square-root function, which stabilizes training by moderating the impact of large inputs, and relu refers to the Rectified Linear Activation function. Learnable weight matrices W_{Ew} and W_{Eb} are dimensioned in $\mathbb{R}^{D_E\times D_E}$, and after updating, the edge features E reside in $\mathbb{R}^{k\times k\times D_E}$. Specifically, the term $\text{KQ}[:,:,:D_E/2]\odot EW_{Ew}$ captures the impact of ligands on the target cells, where the ECM and receptor information extracted from the node features ($\text{KQ}[:,:,:D_E/2]$) is multiplied by the ligands' representation EW_{Ew}. In addition, EW_{Eb} acts as an additional encoding layer for ligand information. Furthermore, $\text{KQ}[:,:,D_E/2:] \odot DE_3$ represents the LR information that might not be explicitly measured in the dataset but can be inferred from other measured genes in the node features.

Then, the attention score $\alpha \in \mathbb{R}^{k\times k}$ is computed by:

$$\alpha = \text{softmax}((EW_A).\text{squeeze}(\text{dim=-1}), \text{dim=-1}) \tag{13}$$

where $W_A \in \mathbb{R}^{D_E\times 1}$, and "squeeze" refers to removing the last dimension with length 1. After that, the softmax function is applied to normalize the attention score for each of the cells in the cell neighborhood (along the -1 dimension).

After that, the influences of the 2nd to the kth tokens (neighboring cells) on the central cell (0-th token, subscript starts from 0) from node features i_n and edge features i_e are calculated by the following equations:

$$\begin{aligned}
\alpha' &= \alpha[0, 1:] \\
i_n &= \alpha' X[1:, :]W_V \\
i_e &= \alpha' E[0, 1:, :]W_{En}
\end{aligned}$$

Initially, the attention scores α are truncated to produce a vector $\alpha' \in \mathbb{R}^{1\times(k-1)}$ that represents the impact of all neighboring cells on the central cell to predict. Subsequently, the node and edge features associated with these cells are extracted and transformed using the weight matrices $W_V \in \mathbb{R}^{C\times C}$ and $W_{En} \in \mathbb{R}^{D_E\times C}$, respectively.

Finally, the contributions from both node and edge features are combined and processed through an MLP head to yield the influence matrix $i \in \mathbb{R}^{(k-1)\times c}$. The matrix i is then summed along the first dimension to derive the final prediction of the cell state $\hat{y} \in \mathbb{R}^c$, which is then compared to the measured cell state expression $y \in \mathbb{R}^c$ by mean square error to update model parameter:

$$\begin{aligned}
i &= \text{MLP}(\text{concatenate}(i_n, i_e)) \\
\hat{y} &= \text{sum}(i, \text{dim} = 0)
\end{aligned}$$

Concatenating all influence matrix for all cells within one slide results in the influence tensor $I \in \mathbb{R}^{N\times(k-1)\times c}$, where N indicates the number of cells within a slide.

5 Details of benchmarking with other GNN backbone models

GCN. Graph Convolutional Network (GCN) is utilized to learn latent representations of nodes in a graph. The GCN takes the raw adjacency matrix $A \in \mathbb{R}^{n\times n}$ and the node features matrix $X \in \mathbb{R}^{n\times C}$ as inputs. The model learns to aggregate features from neighboring nodes to update node representations. Specifically, the update at the l-th layer of the GCN can be described by the following equation:

$$H^{(l)} = \sigma(\hat{A}H^{(l-1)}W^{(l)} + b^{(l)}) \tag{14}$$

where $\hat{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is the normalized adjacency matrix, and D is a diagonal matrix with $D_{ii} = \sum_j A_{ij}$. Here, $W^{(l)}$ and $b^{(l)}$ are the trainable weight matrix and bias vector for the l-th layer, respectively. The function $\sigma(\cdot)$ denotes a nonlinear activation function, which is ReLU in this study.

The initial layer inputs are set as $H^{(0)} = X$, representing the node features. The final output $H^{(L)}$ of the GCN after L layers captures the latent representations of the nodes, integrating both their features and the structural information of the graph. To construct the graph, we define an edge as existing between two cells if their distance is less than a specified threshold. This threshold is set at 48 for two MERFISH datasets and 70 for two CosMx datasets. For the hyperparameter settings, the dimension of node features is set to 256 and the number of layers of GCN is set to 3, which is the same as the dimension we used in our model. The optimizer and batch size are also set to the same as in our model.

GAT. Graph Attention Network (GAT) leverages the mechanism of attention in graph neural networks to focus on important neighbors. Like GCNs, GATs operate on graphs, but instead of treating all neighbors equally, they learn to weigh the importance of each neighbor's features dynamically. The key update rule in a GAT can be formulated as follows:

$$H^{(l)} = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{(l)} W^{(l)} H_j^{(l-1)}\right) \tag{15}$$

where $H^{(l)}$ denotes the node representations at layer l, and $H_j^{(l-1)}$ is the representation of the j-th neighbor node from the previous layer. The coefficients $\alpha_{ij}^{(l)}$ are attention scores computed for each edge connecting node i and its neighbor j at layer l, determining the importance of node j's features to node i. These attention scores are computed as:

$$\alpha_{ij}^{(l)} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[W^{(l)}H_i^{(l-1)} \| W^{(l)}H_j^{(l-1)}]))}{\sum_{k\in\mathcal{N}_i}\exp(\text{LeakyReLU}(\mathbf{a}^T[W^{(l)}H_k^{(l-1)} \| W^{(l)}H_j^{(l-1)}]))} \tag{16}$$

where $\mathbf{a}$ is a weight vector learned during the training process, $\|$ denotes concatenation, and the function $\sigma(\cdot)$ is LeakyReLU (cite here). Similarly, the dimensions and embeddings of node features and the optimizer are the same as in our model. We stacked two layers of GAT in our implementation and the way to build the sparse spatial graph is the same as in GCN.

Multi-view Graph Convolutional Network. In the implementation of the Multi-view Graph Convolutional Network (mv-GCN), we adopt the same embedding and model architecture as used in HoloNet. However, we deliberately omit the receptor expression information of the target cell in our predictions and use the ligand expression and the distance scaler purely derived from the chemical diffusion process as edge features. This exclusion is strategic in that it prevents the model from merely learning to decode from the product of ligand and receptor interactions (see supplementary files) to predict the expression of the cell to predict. Other methods like scaling the edge features by cosine similarity and edge features normalization were retained. The embedding dimension and the optimizer were the same as in our model.

To demonstrate the superiority of our embedding techniques, we implemented our embedding methods to generate node and edge features within HoloNet, replacing its original embedding methods (mv-GCN1). This substitution aimed to assess whether our approach could enhance prediction performance. We maintained consistency in the experimental conditions by using the same hyperparameter settings as illustrated above.

Original Graph Transformer. We apply the same model architecture as used in SiGra. To be more specific, the Graph Transformer employs a convolutional layer with a multihead attention mechanism as its core component. For a given node v_i, the propagation from layer l to layer $l+1$

in a Graph Transformer is defined by the following equation:

$$h_i^{(l+1)} = \text{ReLU}\left(W^{(l)} h_i^{(l)} + \sum_{v_j \in \mathcal{N}(v_i)} \alpha_{ij} V_j^{(l)}\right) \tag{17}$$

where ReLU is the rectified linear unit used as the nonlinear activation function. In this formulation, $h_i^{(l+1)}$ denotes the updated feature vector of node v_i at layer $l+1$, and $h_i^{(l)}$ is the feature vector from the previous layer. The attention mechanism is crucial for weighing the influence of each neighbor v_j in the neighborhood $\mathcal{N}(v_i)$. The attention coefficients α_{ij} are computed as follows:

$$\alpha_{ij} = \text{softmax}\left(\frac{Q_i^{(l)} (K_j^{(l)})^T}{\sqrt{\dim(h_i^{(l)})}}\right) \tag{18}$$

where the query $Q_i^{(l)}$, key $K_j^{(l)}$, and value $V_j^{(l)}$ for the nodes are computed by linear transformation's weight matrix W and b as:

$$Q_i^{(l)} = W_Q^{(l)} h_i^{(l)} + b_Q^{(l)} \tag{19}$$

$$K_j^{(l)} = W_K^{(l)} h_j^{(l)} + b_K^{(l)} \tag{20}$$

$$V_j^{(l)} = W_V^{(l)} h_j^{(l)} + b_V^{(l)} \tag{21}$$

The attention scores are normalized using the softmax function, ensuring they sum to one. The Graph Transformer applies the multihead attention mechanism, where multiple sets of Q, K, and V are learned and concatenated, allowing the model to attend to information from different representation subspaces at different positions in the graph simultaneously. In our implementation, the embedding dimension h_i, the methods to construct the sparse spatial graph and acquire the neighbors of each cell, the number of graph transformer layers, and the optimizer are the same as those used in our model.