Supplementary Information

Spatially resolved transcriptomics and graph-based deep-learning improve accuracy of routine CNS tumor diagnostics

Michael Ritter[1,2,3], Christina Blume[1,2,3], Yiheng Tang[1,2], Areeba Patel[1,2,3], Bhuvic Patel[1,2,15], Natalie Berghaus[1,2], Jasim Kada Benotmane[5,6,7], Jan Kueckelhaus[5,6,7], Yabo Yahaya[5,6,7], Junyi Zhang[7,8,9], Elena Grabis[7,8,9], Giulia Villa[6,7], David Niklas Zimmer[8,9], Amir Khriesh[5], Philipp Sievers[1,2], Zaira Seferbekova[10], Felix Hinz[1,2], Vidhya M Ravi[8,9], Marcel Seiz-Rosenhagen[11], Miriam Ratliff[12], Christel Herold-Mende[13], Oliver Schnell[4], Juergen Beck[8,9], Wolfgang Wick[14], Andreas von Deimling[1,2], Moritz Gerstung[10], Dieter Henrik Heiland[5,6,7,8,9,15,16*§], Felix Sahm[1,2, *§]

[1] Dept. of Neuropathology, University Hospital Heidelberg, Heidelberg, Germany

[2] Clinical Cooperation Unit Neurooncology, German Consortium for Translational Cancer Research (DKTK), German Cancer Research Center (DKFZ), Heidelberg, Germany

[3] Department of Neurological Surgery, Washington University in St. Louis School of Medicine, St. Louis, USA

[4] Div. of Pediatric Neurooncology, German Consortium for Translational Cancer Research (DKTK), German Cancer Research Center (DKFZ), Heidelberg, Germany

[5]Department of Neurosurgery, University Hospital Erlangen, 91054 Erlangen, Germany

[6]Microenvironment and Immunology Research Laboratory, Friedrich-Alexander Universität Nürnberg-Erlangen, Erlangen, Germany

[7]Translational Neurosurgery, Friedrich-Alexander Universität Nürnberg-Erlangen, Erlangen, Germany

[8]Department of Neurosurgery, Medical Center - University of Freiburg, Freiburg, Germany

[9]Faculty of Medicine, Freiburg University, Germany

[10] Div. of AI in Oncology, German Cancer Research Center (DKFZ), Heidelberg, Germany

[11] Dept. of Neurosurgery, Hospital Memmingen, Memmingen, Germany

[12] Department of Neurosurgery, University Medicine Mannheim, University of Heidelberg, Mannheim, Germany

[13] Div. of Experimental Neurosurgery, University Hospital Heidelberg, Heidelberg, Germany

[14] Department of Neurology, University Hospital Heidelberg, and National Center for Tumor Diseases (NCT), University Hospital Heidelberg, and CCU Neurooncology, German Consortium for Translational Cancer Research (DKTK), German Cancer Research Center (DKFZ), Heidelberg, Germany

[15] Department of Neurological Surgery, Northwestern University Feinberg School of Medicine, Chicago, USA

[16] German Cancer Consortium (DKTK), partner site Freiburg

Overview of the Deep-learning Network

Graph neural networks: NePSTA

Data split

The graph neural network framework is part of the NePSTA project which aims to implement deep learning strategies to explore spatial resolved multi-omics. To assess the performance of NePSTA (Github – heilandd/ NePSTA) and comparative methods, we conducted evaluations on our Visum dataset. We adopted the following stratified procedure to partition the datasets into training and evaluation subsets:

Data Split: From the 107 patients characterized by EPIC, we split each dataset into a training segment and a validation segment. In samples containing multiple biopsies, we split the dataset into individual biopsy cores. For datasets with a single specimen, we split the spots by manual segmentation using the *createSegmentation* function of the SPATA2 package.

Training Dataset Construction: We created the training dataset (pytorch geometric library) by selecting a subset of up to 500 subgraphs from the training split. We incorporated clinical attributes such as tumor type (RTKI, RTKII, MES, …), histological region. This resulted in a comprehensive training set of 97,000 subgraphs. Further we included 12.000 Subgraphs from healthy control.

Evaluation Dataset Construction: For evaluation, we used the validation datasets to cover a spectrum of epigenetic classes. From each of these, we extracted up to 500 subgraphs using the 3-hop method.

Evaluation metrics

In the evaluation of our graph-neural network (GNN), we employed a comprehensive set of metrics to assess its classification performance on predicting clinical and histological parameters. We calculated the accuracy, which reflects the proportion of true results among the total number of cases examined, to provide a straightforward measure of the model's overall correctness.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Precision and recall were computed with a macro-average approach, which treats all classes equally, averaging the metrics without taking class imbalance into account. Precision measures the model's exactness by calculating the ratio of true positive predictions to the total number of positive predictions, while recall (or sensitivity) assesses the model's completeness by determining the ratio of true positive predictions to the total number of actual positives. Precision, or the ratio of correctly predicted positive observations to the total predicted positive observations, is calculated as:

$$\text{Precision}_{\text{macro}} = \frac{1}{N}\sum_{i=1}^{N}\frac{TP_i}{TP_i + FP_i}$$

Recall, also known as sensitivity, is the ratio of correctly predicted positive observations to all observations in actual class:

$$\text{Recall}_{\text{macro}} = \frac{1}{N}\sum_{i=1}^{N}\frac{TP_i}{TP_i + FN_i}$$

We also reported the F1 score, which is the harmonic mean of precision and recall, providing a single metric that balances both the concerns of precision and recall in one number. This score is particularly useful when seeking a balance between the model's performance regarding false positives and false negatives and is defined as:

$$\text{F1 Score}_{\text{macro}} = 2 \times \frac{\text{Precision}_{\text{macro}} \times \text{Recall}_{\text{macro}}}{\text{Precision}_{\text{macro}} + \text{Recall}_{\text{macro}}}$$

Additionally, the confusion matrix was presented, offering a detailed breakdown of the model's predictions across the different classes, showing the true positives, true negatives, false positives, and false negatives.

NePSTA graph-neural network architecture

The NePSTA prediction network consists of a Graph Isomorphism Network (GIN) backbone and multiple multilayer perceptron (MLP) prediction heads which were chosen based on the defined prediction tasks. Inputs of NePSTA contain the local spatial graphical structures of gene expression values derived from the 3-hop neighborhood of Visium spots. We used the pytorch geometic library and defined each spot as node and edges were defined as the direct neighbors of each individual spot. The following node definitions were implemented in the input object:

Node Features:

Expression data: The node features are encapsulated by an $N\ x\ G$ matrix, where N denotes the number of nodes in a given subgraph and G represents the set of genes. Specifically, these features are derived from the log-scaled, normalized expression values of the top 5000 most variably expressed genes within the cohort. Due to the zero-inflated distribution commonly observed in spatially resolved gene expression datasets, non-expressed genes (those with zero counts) within a subgraph are masked to avoid skewing the analysis with non-informative features.

Copy number alterations: Copy-number alterations per node were extracted in a similar way then explained in the expression data. For each subgraph, a $N\ x\ C$ matrix containing the C vector or chromosomal alterations arranged along the genomic coordinates, N denotes the number of nodes in a given subgraph.

Histological annotations: For each subgraph, a $N\ x\ A$ matrix containing the A with hot-one encoded histological classification of each node, N denotes the number of nodes in a given subgraph.

H&E image: To encode the H&E images of each spot (node) we utilized a Convolutional Neural Network (CNN) designed to encode a (256×256) image into an N-length vector, the encoding process is achieved through a sequence of convolutional and pooling layers. These layers work to downsample the image and extract increasingly complex features. The process can be described mathematically as follows: Let I be the input image with dimensions $256 \times 256 \times C$, where C represents the number of color channels $(C = 3)$ in RGB. The encoder consists of L layers, and each layer l can be described by the function:

$$F_l = P\big(\phi(W_l * F_{l-1} + b_l)\big)$$

Where F_l is the feature map after the l-th layer, W_l and b_l are the weights and biases of the convolutional filters at layer l, $*$ denotes the convolution operation, ϕ is the ReLU activation

function, defined as $\phi(z) = \max(0, z)$, P represents a pooling operation like max pooling, which reduces the spatial dimensions of the feature maps. After the final layer L, the feature map F_L is flattened and passed through a fully connected layer to produce the encoded vector V : $[V = \text{FC}(F_L)]$ The fully connected layer, denoted by FC, transforms the flattened feature map into the N -length vector, which serves as a dense representation of the original image.

Edge Features:

Edges within the graph symbolize the connections among nodes, with each node having up to six neighbors to reflect the spatial arrangement. To ensure robust analysis, subgraphs containing fewer than 15 nodes are not considered, maintaining a minimum complexity in the network structure. Additionally, we incorporate self-loops, where each node is connected to itself, as part of the graph's edge features. These self-loops are critical as they allow each node to retain its original feature information throughout the message-passing phases of the graph neural network during the forward pass.

This structured representation of node and edge features is fundamental to our spatial transcriptomics analysis, allowing for the intricate modeling of cellular environments based on gene expression profiles and spatial relationships.

NePSTA employed a three-layer GIN, and in the kth graph convolutional layer to process batches (batch size: 32) of spatial transcriptomic data. Messages are computed as follow:

$$m_{uv} = MLP(h_u)$$

which were aggregated:

$$a_v = \sum_{u \in N(v)} m_{uv}$$

in which a_v is the aggregated message for node v and $N(v)$ is the set of neighbors of node v. The embedding of node v is updated on the basis of all incoming messages to v:

$$h'_v = MLP(a_v)$$

$N(v)$ is the set of neighboring nodes of v, the self-loop edge guarantees $v \in N(v)$. The GIN layers are represented as follows: x_v defines the expression vector of node v and $N(v)$ is the set of its neighbors. The GIN convolution operation updates the feature vector of node v by aggregating features from $N(v)$ and combining them with x_v own features. The updated feature vector x'_v is computed as follows:

$$x'_v = ReLU\left(\text{BN}\left((1+\epsilon)\cdot x_v + \sum_{u \in \mathcal{N}(v)} \text{ReLU}(x_u)\right)\right)$$

we define ϵ as a learnable parameter that allows the model to weigh the importance of a node's own features versus the features of its neighbors. The sum is over the set $\mathcal{N}(v)$ representing the aggregation of messages from the neighbors. ReLU (Rectified Linear Unit) is a non-linear activation function applied element-wise. BN stands for Batch Normalization, which normalizes the feature vector to improve training stability and performance. This operation is stacked multiple times ($k =$

2) in the kth GIN to allow for deeper aggregation of neighborhood information. After each GIN convolutional layer, batch normalization and LeakyReLU activation with a negative slope of 0.2 are applied, followed by a dropout layer with a dropout rate of 0.5 for regularization. The latent space representation of the graph is obtained by passing the output of the second GIN convolutional layer through a linear transformation (self.merge) with weights initialized using the Xavier uniform method. The resulting features are merged into a latent space, and then global mean pooling is applied to create graph-level representations.

For the prediction tasks, separate multilayer perceptron (MLP) modules are employed. Each MLP consists of a linear layer, a ReLU activation, batch normalization, dropout, and a final linear layer that outputs the predictions. The MLPs are structured as follows:

$$h(x) = W_2 \cdot D \cdot B \cdot \phi(W_1 \cdot x + b_1) + b_2$$

Where: x is the latent space vector to the MLP. W_1 and W_2 are the weight matrices for the first and second linear transformations, respectively. $b_1$and b_2 are the bias vectors for the first and second linear transformations, respectively. ϕ denotes the ReLU activation function, applied element-wise, where $\phi_z = max(0, z)$, B represents the batch normalization operation applied to the activated output. D represents the dropout operation which randomly zeroes some of the elements of its input with a certain probability to prevent overfitting.

We applied different loss strategies for individual prediction heads (MLPs) which were integrated. For categorical variables we used cross-entropy loss which is calculated for each class and summed, where for each sample and class pair, it combines $log(softmax)$ operations:

$$\text{CrossEntropyLoss} = -\sum_{c=1}^{M} y_{o,c} \log\left(p_{o,c}\right)$$

For continuous variables we used the L1 norm loss or mean square error (MSE) or a combination out of both. The L1 norm is computed as:

$$L1\text{norm} = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i|$$

where N is the number of observations, y_i is the true value for the i^{th} observation, and $\hat{y}_i$ is the predicted value for the i^{th} observation. The is given by:

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Multiple MLPs were integrated by the weighted sum of losses. The weight ω was defined for N losses l or N MLPs.

$$loss = \frac{1}{N}\sum_{i=1}^{N} l_i \times \omega_i$$

In the hyperparameter optimization the ω was optimized. All networks were initialized with $\omega = 1$.

Model training and inference

The training of the model was conducted over a pre-defined number of epochs, encapsulated within a progress-tracking loop. For each epoch, the model iterated through batches of data provided by

the loader. Within each iteration, the optimizer's gradients were first reset to zero to prevent accumulation from previous forward passes. The model then performed a forward pass by processing the input data, which had been moved to the appropriate computing device, yielding latent representations and predictions for survival, neuron scores, and status outcomes. The neuron score predictions were assessed against the ground-truth neuron scores using the L1norm loss function. This loss was then propagated backward through the network to compute the gradients. Following this, an optimization step was taken, adjusting the model's weights to minimize the aggregate loss. Adam optimizer was employed to minimize corresponding losses in different tasks.