ACTION: Augmentation and Computation Toolbox for Brain Network Analysis with Functional MRI – *Manual*

In what follows, we introduce the workflow and usage of the Augmentation and Computation Toolbox for braIn netwOrk aNalysis (ACTION) with functional magnetic resonance imaging (MRI). The software and open-source code of ACTION can be found online (https://mingxia.web.unc.edu/action/).

I. Introduction

A. Purpose and Scope

Resting-state functional magnetic resonance imaging (rs-fMRI) has emerged as a non-invasive technique that has proven highly effective and simple in exploring the functional structure within the brain [1]. It has become one of the most popular techniques in current neuroscience research. The blood-oxygen-level-dependent (BOLD) signals derived from fMRI track changes in blood flow, providing an excellent way for healthcare professionals, neuroscientists, psychologists, data analysts, and others to understand and interpret brain activity over time. However, original BOLD signals collected from devices contain significant noise, which greatly weakens the expressive power of models trained directly on them. Therefore, preprocessing steps such as data augmentation, establishing functional connectivity networks (FCNs), and feature extraction are essential.

Performing these steps manually one by one is very time-consuming, tedious, and prone to errors. However, there is still a lack of user-friendly toolboxes for comprehensive analysis of fMRI data. Based on existing augmentation algorithms for fMRI data, brain network construction methods, and artificial intelligence (AI) based fMRI analysis models, we have developed this ACTION software using Python for processing and analyzing rs-fMRI data. The ACTION will provide functions such as fMRI data augmentation, brain network construction and visualization, extraction of brain network features, and intelligent analysis of brain networks based on AI models. Through a graphics user interface, the ACTION aims to provide users with comprehensive, convenient, and easy-to-use fMRI data analysis services, helping users simplify the processing pipeline and improve work efficiency.

B. Software Version

The ACTION is written in Python 3.8 and currently runs smoothly on Python 3.7 and higher versions (the current highest version being Python 3.12). The dependencies required to ensure the normal operation of ACTION will be detailed in Section II-B. For future releases of new Python versions, we will continue to monitor and update the manual accordingly.

C. License

II. Installation Instructions

A. Operation System Requirements

The ACTION software is designed to run smoothly on the following operating systems:

- Windows 10 version 22H2
- Windows 11 version 22H2
- macOS Monterey version 12.3.1
- Ubuntu version 22.04

While ACTION has been tested on the aforementioned operating systems and functions properly, due to limitations in testing coverage, we cannot guarantee its performance on all operating systems. Therefore, although the toolbox may also function properly on other operating systems, we cannot assure the completeness and stability of its features. It is recommended that users utilize this toolbox on the tested operating systems for the optimal experience.

The minimum hardware specifications required to ensure the proper functioning of ACTION have not been explicitly determined. It is advised that users employ relatively newer computer devices to ensure optimal performance and stability.

B. Dependencies

To ensure the toolbox runs successfully, users will need to install the following dependencies on Python versions not lower than 3.7:

- pyqt5 >= 5.15.6
- numpy >= 1.19.5
- networkx >= 2.5.1
- dill >= 0.3.4
- scipy >= 1.5.4
- scikit-learn >= 0.24.2

- xgboost >= 1.5.2
- torch >= 1.10.2
- matplotlib <= 3.5.0 for Python versions 3.7 – 3.10

III. Getting Started

A. User Interface

Upon successfully installing and opening the ACTION software, the welcome interface is displayed as shown in Fig. S1. As shown in Fig. S1, the welcome interface intuitively describes the functions of the ACTION software and the design purposes of each function. At the top of the page is the logo and full name of the toolbox. Below the logo, there are four buttons representing the four function modules of the toolbox. Brief introductions of the design purposes of these function modules are provided in the middle. At the bottom of the page, clicking the "MANUAL" button will display the manual, and clicking the "EXIT" button will exit the software.

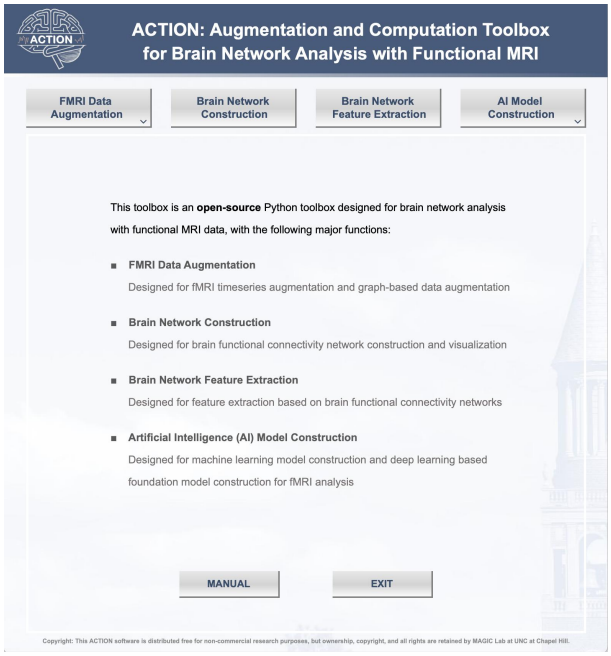

Fig. S1. Welcome page of the ACTION software.

B. Quick Start Guide

In each functional interface, in addition to displaying process diagrams for some interfaces, the entire page can be divided into three main parts, as illustrated in Fig. S2. The first part is used to load the data that needs to be processed (*i.e.*, the address bar corresponding to "Load *"). The second part is the main functional area of the interface, which contains algorithms for implementing the corresponding functions of the interface and the storage path of the results (*i.e.*, the address bar corresponding to "Save Directory"). Some interfaces also have displays for the results, such as brain network construction module. The third part consists of control buttons for the interface, including the "Help", "Clear", and "Quit" buttons.

For the "Load *" and "Save Directory" sections, the corresponding "Browse" buttons will pop up dialog boxes for loading and saving data, respectively. "Run *" or "Create *" are buttons to start the procedure. Click the "Help" button in the bottom right corner to show the corresponding interface's assistance information, the "Clear" button to clear all the text boxes' contents in the current interface, and the "Quit" button to return to the welcome interface. To help users quickly familiarize themselves with the workflow of the toolbox, we will provide a step-by-step demonstration using the "BOLD Signal Augmentation" module as an example (see Fig. S2).

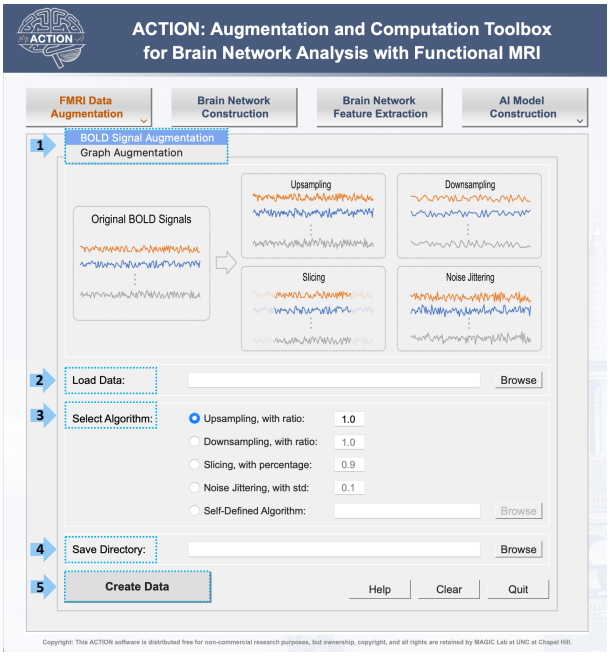

Fig. S2. Blood-oxygen-level-dependent (BOLD) signal augmentation module.

- **Step 1**: Click the "FMRI Data Augmentation" button and select "BOLD Signal Augmentation" from the drop-down menu to enter the corresponding interface. The flowchart in the interface visually expresses the BOLD signal augmentation algorithms provided in the toolbox.
- **Step 2**: Click the "Browse" button corresponding to "Load Data" and a file selection dialogue will be popped up. Select the original data to be processed.
- **Step 3**: Select the desired algorithm in the "Select Algorithm" section, and modify parameter values as needed within a reasonable range. For instance, selecting "Upsampling" algorithm and setting the ratio to 0.7.
- **Step 4**: Click the "Browse" button corresponding to "Save Directory" to select the save directory for augmented data. If no directory is selected, the results will be automatically saved in the "results" folder by default.
- **Step 5**: Click the "Create Data" button to start augmenting the original BOLD signal data, and the result will automatically be saved at the selected save directory.

IV. Modules and Functions

The ACTION software includes four function modules: (1) FMRI data augmentation, (2) brain network construction, (3) brain network feature extraction, and (4) artificial intelligence model construction. The first module contains two submodules: BOLD signal augmentation and graph augmentation.

The last module comprises machine learning model construction and deep learning-based foundation model construction.

A. FMRI Data Augmentation

1) BOLD Signal Augmentation: Firstly, users need to load the original BOLD signal data by clicking the "Browse" button. The data should be a *.npy* file containing a 3-dimensional matrix with a shape of (S, N, T), where S is the number of subjects, N represents the number of nodes, and T denotes the time points. If not satisfied, an error message "Input shape error, please check your input dimensions" will pop up, along with information about the correct dimension. Then, users should select a desired augmentation algorithm. Currently, the ACTION software includes four methods for BOLD signal augmentation, detailed as follows.

- **Upsampling**: Key parameter is the upsampling rate (between 0 and 1, default is 1.0).
- **Downsampling**: Key parameter is the downsampling rate (between 0 and 1, default is 1.0).
- **Slicing**: Key parameter is the slice percentage (between 0 and 1, default is 0.9).
- **Noise Jittering**: Key parameter is the standard deviation of Gaussian noise (default is 0.1).

Users can choose any one of the above four methods and adjust the corresponding parameter values within a reasonable range as needed, otherwise, default parameters will be used. Afterward, specify the saving address for the augmented data and click "Create Data" to initiate the BOLD signal augmentation process. The result is named as "save_data + algorithm". Details can be found in Fig. S2.

2) Graph Augmentation: First of all, users need to load the original adjacency matrix and corresponding untreated node attribute matrix in the correct position. The adjacency matrix should be a *.npy* file containing a 3-dimensional matrix with a shape of (S, N, N) and the node attribute data also should be a *.npy* file containing a 3-dimensional matrix with a shape of (S, N, D), where D denotes the dimension of node attributes. As same as BOLD signal augmentation, users should select one of the provided six algorithms for graph-based augmentation, with details introduced as follows.

- **Random Node Dropping**: Key parameter is the random node dropping rate (between 0 and 1, default is 0.2).
- **Hub-Preserving Node Dropping**: Key parameter is the hub-preserving node dropping rate (between 0 and 1, default value is 0.2).
- **Random Edge Perturbation**: Key parameter is the random edge perturbation rate (between 0 and 1, default setting is 0.2).
- **Weight-Dependent Edge Removal**: Key parameter is the weight-dependent edge removal rate (between 0 and 1, default setting is 0.2).
- **Subgraph Cropping**: Key parameter is the subgraph cropping rate (between 0 and 1, default is 0.2).
- **Attribute Masking**: Key parameter is the attribute masking rate (between 0 and 1, default is 0.2).

After users choose a method, they can adjust corresponding parameter values within a reasonable range as needed, otherwise, default parameters will be used. Then, users need to specify the saving address for augmented results and click "Create Data" to start the graph augmentation process. The file name of saved augmented adjacency matrix and augmented node attribute matrix are set to "save_adjmat + algorithm" (if exists) and "save_nodeatt + algorithm" (if exists), respectively. Details can be found in Fig. S3.

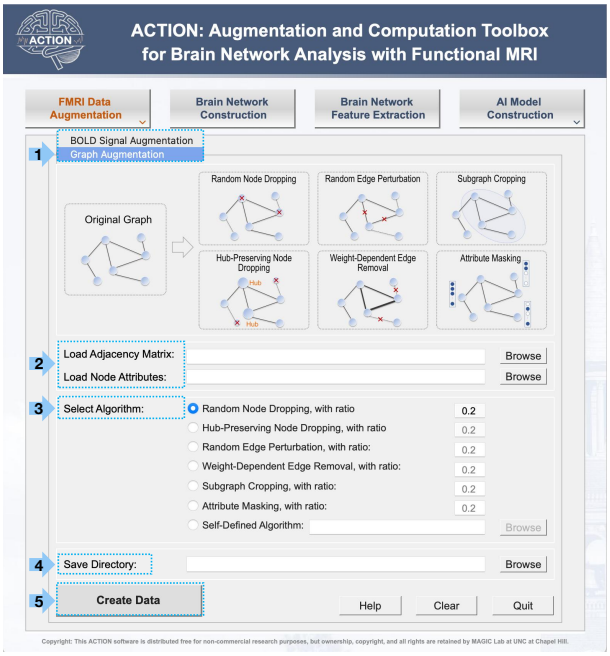

Fig. S3. Brain network/graph augmentation module.

B. Brain Network Construction

With this module, the toolbox provides users with the function to construct and visualize brain networks. Therefore, users need to load a 3-dimensional matrix of shape (S, N, T) with file type *.npy*. After loading the data, users can select a brain network construction method from the "Select Algorithm" menu. There are seven built-in options:

- **Pearson's Correlation** (PC): No parameters.
- **Mutual Information** (MI): No parameters.
- **Partial Correlation**: No parameters.
- **Spearman's Correlation** (SC): No parameters.
- **High-Order Functional Connectivity** (HOFC): No parameters.
- **Sparse Representation** (SR): Key parameter is the sparse regularization parameter. The value range is non-negative and the default is 1.
- **Low-rank Representation** (LR): Key parameter is the low-rank regularization parameter. The value range is non-negative and the default is 1.

In most cases, the constructed brain network is dense. However, users may wish to obtain a sparse brain network. Therefore, the toolbox provides the function to sparsify the constructed brain network using thresholding, which is located under "Optional". By default, the option is set to "None", meaning no sparsification is performed. Additionally, the toolbox offers two sparsification methods:

- **Binarization**: Key parameter is the threshold T, which retains the top $T\%$ of edges, assigns a weight of 1 to these edges, and sets the rest to 0. The default value of T is set as 30.
- **Sparsity**: Key parameter is the threshold K, which retains the top $K\%$ of edges while keeping their weights unchanged and sets the rest as 0. The default value of K is set as 30.

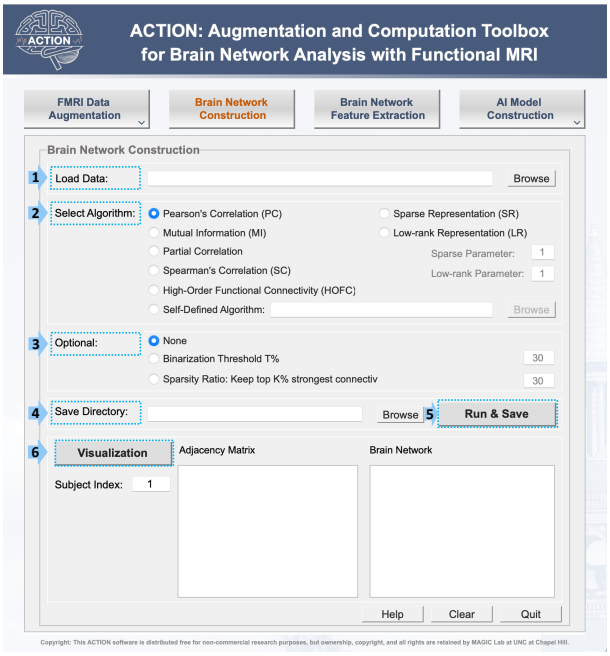

Fig. S4. Brain network construction module.

After selecting the brain network construction algorithm and sparsification method, users can click the "Run & Save" button, and the constructed brain network will be saved in the specified directory and named "algorithm + Adjacency_Matrix + sparsity method". If users want to visually inspect the brain network for each subject, they can input the subject index (default is the first subject) and click the "Visualization" button. This will display the visualization results on the right side. Two visualization options are provided: Adjacency Matrix and Brain Network. In the Adjacency Matrix, edge weights are represented by different colors, and in the Brain Network, it directly depicts nodes and edges to provide an intuitive connectivity pattern. Details can be found in Fig. S4.

C. Brain Network Feature Extraction

At the outset, users are required to upload the constructed brain network, which should be stored as a *.npy* file comprising a 3-dimensional matrix with dimensions (S, N, N). The ACTION software can extract features from brain networks at both node-level and graph-level, located respectively in the "Node-based" and "Graph-based" blocks in this module. Meanwhile, users can decide whether to consider the influence of edge weights when extracting features by selecting the "Weighted Graph" option. If selected, the graph will be treated as a weighted graph (*i.e.*, edges between nodes are assigned weights), otherwise, it will be treated as an unweighted graph (*i.e.*, edges between nodes are all assigned a weight of 1).

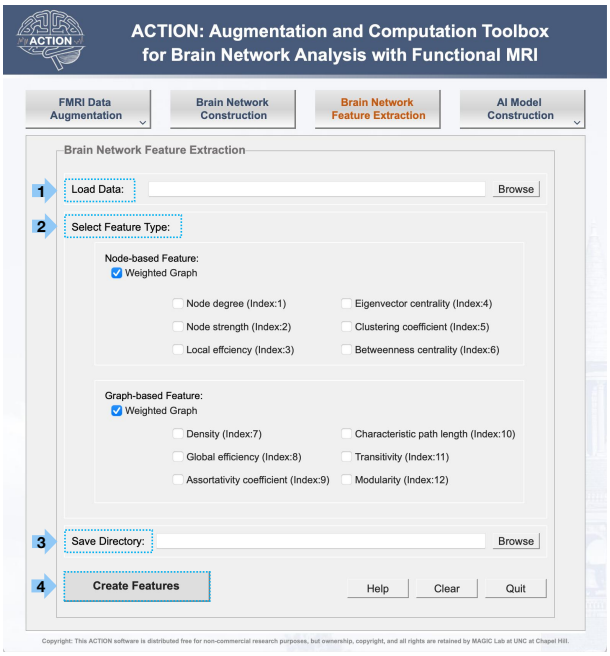

Fig. S5. Brain network feature extraction module.

There are six algorithms to extract node-based features from brain networks, listed as follows:

- Node degree (index: 1)
- Node strength (index: 2)
- Local efficiency (index: 3)
- Eigenvector centrality (index: 4)
- Clustering coefficient (index: 5)
- Betweenness centrality (index: 6)

In addition, there are six algorithms for graph-based feature extraction, including:

- Density (index: 7)
- Global efficiency (index: 8)
- Assortativity coefficient (index: 9)
- Characteristic path length (index: 10)
- Transitivity (index: 11)
- Modularity (index: 12)

Users can select one or more algorithms from all algorithms according to their needs. The final extracted features consist of the features that are obtained from the algorithms selected from the aforementioned list, concatenated sequentially in ascending order of their indices. After selecting a saving address for the extracted features, click "Create Features" for starting feature extraction. After execution, users will find a result file named "save_feature + index" in the specified save directory, where "index" represents the index corresponding to the selected feature types. If users select node degree, modularity, and density, the corresponding result file will be named "save_feature_1_5_8". Details are given in Fig. S5.

D. Artificial Intelligence Model Construction

1) Machine Learning Model Construction: Initially, users are required to upload the sample data for classification or

regression tasks, along with their corresponding category labels. Here, the sample data consists of a 2-dimensional matrix with a shape of (S, F), where each sample is represented by a feature vector of length F. This data is stored in a *.npy* file. The corresponding labels are stored as a vector of length S, also saved as a *.npy* file. Note that in classification tasks, category labels are either {0,1} or {-1,1}.

Following that, users need to select the type of task they want to perform (*i.e.*, classification or regression). Before executing the task, users can decide whether to perform dimension reduction on the input original features. In the "Dimension Reduction Algorithm" section, the toolbox is equipped with three methods for feature dimension reduction:

- **Principal Component Analysis** (PCA) [2]: Mapping original data to a new coordinate system via linear transformation. Key parameter is the reduced dimension. If the value is set between 0 and 1, it represents the lowest cumulative contribution rate of principal component variance satisfied. If set to an integer greater than or equal to 1, it represents the number of maximum principal components selected. The default value is 1.
- **Independent Component Analysis** (ICA) [3]: Key parameter is the reduced dimension (an integer greater than or equal to 1, with a default value of 1).
- **Canonical Correlation Analysis** (CCA) [4]: Key parameter is the reduced dimension (an integer greater than or equal to 1, with a default value of 1).

Users can choose one of these algorithms, or select "None" to indicate that dimension reduction is not needed for the original data. Additionally, the toolbox offers several popular machine learning based classifiers/regressors, including:

- **Support Vector Machine** (SVM): For classification.
- **Support Vector Regression** (SVR): For regression.
- **Random Forest** (RF) [5]
- **Extreme Gradient Boosting** (XGBoost) [6]
- **K-Nearest Neighbors** (KNN) [7]

Users also need to choose the method for partitioning the data into training and validation sets. There are two data partition strategies provided in the toolbox:

- **K-fold Cross Validation** (default): Key parameter is the fold of cross-validation K (an integer greater than or equal to 2, with a default value $K = 5$).
- **Random Partition**: Key parameter is the partition ratio of the training set (between 0 and 100, default is 70).

When all the aforementioned preparations are completed and the "Run" button is clicked, the results will be displayed in tabular form in the corresponding area labeled "Result".

At the same time, the result will also be saved as a *.json* file in the selected save directory, named "save + classification/regression algorithm + cls/reg_result + reduction algorithm + partition strategy". If users perform dimension reduction, the reduced features will also be saved as a *.npy* file and named as "save + reduction algorithm + fea_reduced_dim + reduced dimension". Otherwise, this step will be ignored, and the "reduction algorithm" in the *.json* file will be replaced with "Original_feature".

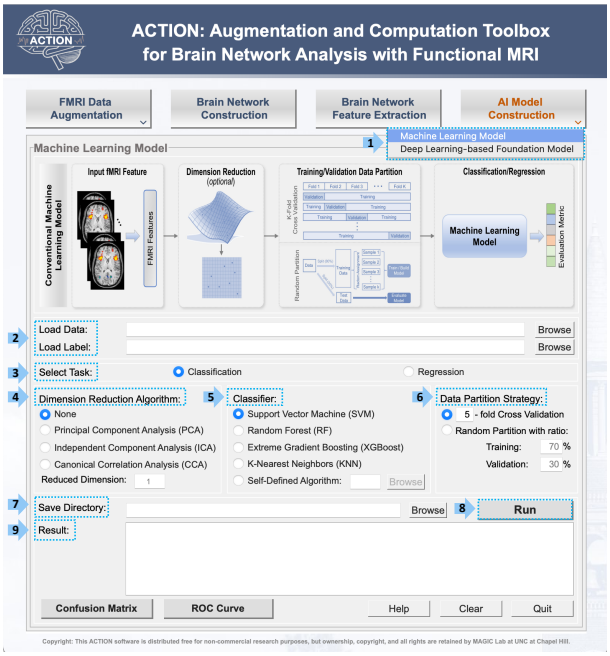

Fig. S6. Machine learning model construction module.

It is worth mentioning that, for classification tasks, the ACTION software provides a feature that plots the confusion matrix and the ROC curve for better visualization of the results. Clicking on the respective buttons will display the corresponding plots. Details can be found in Fig. S6.

2) Deep Learning-based Foundation Model Construction: In addition to the various machine learning models included in the Machine Learning Model interface, the toolbox also provides pre-trained results based on deep learning models, aiming to reduce the time spent by users on training deep learning models and improve model representation capabilities. Currently, the software includes ten deep learning models as the backbone encoder and five federated learning strategies to facilitate multi-site fMRI studies. The feature encoders of these models are pre-trained on 3,806 unlabeled fMRI scans from public cohorts in a self-supervised learning manner, while the pre-training process is presented at the top of Fig. S7. These ten deep learning models include:

- **Transformer** [8]
- **Graph Attention Network** (GAT) [9]
- **Graph Isomorphism Network** (GIN) [10]
- **Graph Convolutional Network** (GCN) [11]
- **Brain Graph Neural Network** (BrainGNN) [12]
- **Graph SAmple and aggreGatE** (GraphSAGE) [13]
- **Spatio-Temporal Graph Convolutional Network** (STGCN) [14]
- **Modularity-constrained Graph Neural Network** (MGNN) [15]
- **Brain Network Convolutional Neural Network** (BrainNetCNN) [16]
- **Spatio-Temporal Attention Graph Isomorphism Network** (STAGIN) [17]

With GCN as the default backbone model, these five federated learning methods include:

- **Federated Avgeraging** (FedAvg) [18]
- **Federated Proximal** (FedProx) [19]
- **Model-Contrastive Federated Learning** (MOON) [20]
- **Local Global Federated Avgeraging** (LGFedAvg) [21]
- **Personalized Federated Learning with Moreau Envelopes** (pFedMe) [22]

After selecting a specific backbone encoder, one can click the "Fine-Tuning Process" button to obtain the pre-trained model and the detailed procedures for model fine-tuning. After selecting a specific federated learning strategy, one can click the "Source Code Link" button to obtain the corresponding algorithm. It is worth noting that the toolbox only provides pre-trained deep learning models. In addition to using the default GCN, users can also utilize other backbone models in each of these federated learning methods.

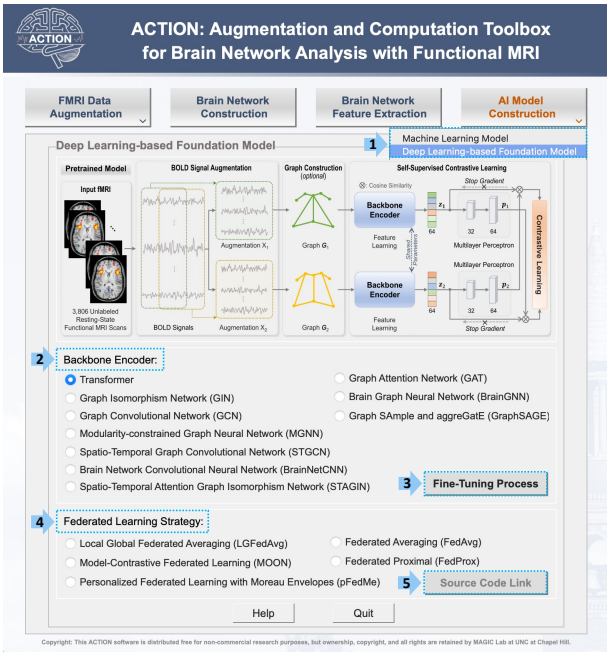

Fig. S7. Deep learning based foundation model construction module.

V. Advanced Features

In practical applications, users may need to use their own developed algorithms instead of the ones provided in the toolbox. To meet this demand, the ACTION software supports users to upload their custom algorithms to different modules of the toolbox to accomplish various tasks. On the "FMRI Data Augmentation", "Brain Network Construction", and "Machine Learning Model" modules, there is a "Self-Defined Algorithm" option available. After selecting this option, users can click the "Browse" button on the right to upload their custom algorithms, and then execute the corresponding functionality just like the provided algorithms in the toolbox.

To adapt to the data processing requirements, users should modify their custom algorithms into a function using the Python language and package them into a *.pkl* file via "dill". Note that users need to use the "dill.settings['recurse'] = True" command to ensure that all dependencies in the function are successfully packaged and ensure that these dependencies are correctly installed on the running device. Figure S8 shows the packaging process. The specific forms of custom algorithms in each module are introduced in the following.

```python
# Step 1. Import dependencies.
import numpy as np
import dill

# Step 2. Ensure all dependencies are packaged.
dill.settings['recurse'] = True

# Step 3. Define the function.
def Self_Defined_Algorithm(input_1, input_2):
    output_1 = input_1 + np.zeros_like(input_1)
    output_2 = input_2 + np.zeros_like(input_2)
    return output_1, output_2

# Step 4. Package into a .pkl file.
file = '/Users/Toolbox/Desktop/alg.pkl'
with open(file, 'wb') as f:
    dill.dump(Self_Defined_Algorithm, f)
```

alg.pkl

Fig. S8. Illustration of packaging process for self-defined algorithms.

1) BOLD Signal Augmentation: With this module, users are allowed to use their own custom BOLD signal augmentation functions with two ordered inputs:

- **X**: The original BOLD signal, which is a 3-dimensional matrix of shape (S, N, T) and T denotes the time points.
- **param**: All hyperparameters in the function (assigned).

Therefore, users can define the BOLD signal augmentation algorithm *BOLDAug* as X′ = *BOLDAug*(X, param), where X′ is the augmented BOLD signal. After successfully importing the algorithm, clicking the "Create Data" button will save the result in the specified directory, named "save_custom_augmented_result".

2) Graph Augmentation: Users can use custom algorithms to augment the graph, which should have three ordered inputs:

- **X_adj**: The original adjacency matrix, which is a 3-dimensional matrix of shape (S, N, N).
- **X_att**: The original node attributes matrix, which is a 3-dimensional matrix of shape (S, N, D).
- **param**: All hyperparameters in the function (assigned).

Therefore, the form of the graph augmentation algorithm *GraphAug* should be X′_adj, X′_att = *GraphAug*(X_adj, X_att, param), where X′_adj and X′_att are the augmented adjacency matrix and node attributes matrix, respectively. Sometimes, users may only need to augment either nodes or edges while keeping the other unchanged. In this case, "None" should be used as a placeholder for the missing output. For example, if the user only needs to augment the edges, the function should be changed to X′_adj, None = *GraphAug*(X_adj, X_att, param). After successfully importing the algorithm, clicking the "Create Data" button will generate the augmented adjacency matrix named "save_adjmat_custom" (if exists) and the augmented node attributes matrix named "save_nodeatt_custom" (if exists) in the specified directory.

3) Brain Network Construction: With this module, users can use their own developed algorithms to construct brain networks, and perform sparsification through thresholding, and visualization. The algorithm should have two ordered inputs:

- **X**: The original BOLD signal, which is a 3-dimensional matrix of shape (S, N, T).

- **param**: All hyperparameters in the function (assigned).

Additionally, it requires an output (*i.e.*, the constructed brain network) X′ of shape (S, N, N), so the final form of the function *BrainNet* is X′ = *BrainNet*(X, param). After successfully importing the algorithm, clicking the "Run & Save" button will save the brain network constructed by the custom algorithm (named "Custom_Adjacency_Matrix") in the specified storage directory. Similar to using built-in algorithms, if the user clicks the "Visualization" button, the visualization results of the specified subject can be shown.

4) Machine Learning Model: In addition to providing four machine learning models, the ACTION software allows users to use custom models for classification or regression tasks. The custom model is still required to be presented in the form of a function, with four ordered inputs:

- **Y_tr**: The labels of the training set, which is a vector of length S. If the model performs a classification task, the labels should only consist of 0 and 1 or -1 and 1.
- **X_va**: The feature matrix of the validation samples, which is a 2-dimensional matrix of shape (S_{va}, F), where S_{va} is the number of samples in the validation set, and F is the dimension of the features of the validation/training samples.
- **X_tr**: The feature matrix of the training samples, which is a 2-dimensional matrix of shape (S_{tr}, F), where S_{tr} is the number of samples in the training set.
- **param**: All hyperparameters in the function (assigned).

In practical use, users only need to ensure that the dimensions of each input correspond, without worrying about the specific shapes, because the ACTION software provides data partitioning functionality. Additionally, if it is a classification model, it should have two ordered outputs:

- **Y_va**: The predicted values of the validation samples by the model, which is a vector of length S_{va}. If the model performs a classification task, it should have the same type of category labels as Y_tr.
- **Prob**: The probabilities that the model predicts the validation samples as positive class, which is also a vector of length S_{va}.

If it is a regression model, there is only Y_va as output. So, the custom classification model *Cls* and regression model *Reg* are defined as Y_va, Prob = *Cls*(Y_tr, X_va, X_tr, param) and Y_va = *Reg*(Y_tr, X_va, X_tr, param), respectively.

After successfully importing the model, specify the data partition strategy, and then click the "Run" button to run the custom model. The experimental results obtained by the custom model will be shown below after running, and a *.json* file named "save_custom + task + dimension reduction method + partition strategy" will be saved in the specified directory. For classification tasks, users can click the "Confusion Matrix" button and the "ROC Curve" button to view the confusion matrix and ROC curve, respectively.

REFERENCES

[1] M. Khosla, K. Jamison, G. H. Ngo, A. Kuceyeski, and M. R. Sabuncu, "Machine learning in resting-state fmri analysis," *Magnetic resonance imaging*, vol. 64, pp. 101–121, 2019.

[2] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[3] A. Hyvärinen and E. Oja, "A fast fixed-point algorithm for independent component analysis," *Neural Computation*, vol. 9, no. 7, pp. 1483–1492, 1997.

[4] H. Hotelling, "Relations between two sets of variates," in *Breakthroughs in statistics: Methodology and distribution*. Springer, 1992, pp. 162–190.

[5] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[6] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[7] E. Fix, *Discriminatory analysis: Nonparametric discrimination, consistency properties*. USAF school of Aviation Medicine, 1985, vol. 1.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[9] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[10] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[11] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[12] X. Li, Y. Zhou, N. Dvornek, M. Zhang, S. Gao, J. Zhuang, D. Scheinost, L. H. Staib, P. Ventola, and J. S. Duncan, "Braingnn: Interpretable brain graph neural network for fmri analysis," *Medical Image Analysis*, vol. 74, p. 102233, 2021.

[13] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[14] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," *arXiv preprint arXiv:1709.04875*, 2017.

[15] Q. Wang, M. Wu, Y. Fang, W. Wang, L. Qiao, and M. Liu, "Modularity-constrained dynamic representation learning for interpretable brain disorder analysis with functional mri," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2023, pp. 46–56.

[16] J. Kawahara, C. J. Brown, S. P. Miller, B. G. Booth, V. Chau, R. E. Grunau, J. G. Zwicker, and G. Hamarneh, "Brainnetcnn: Convolutional neural networks for brain networks; towards predicting neurodevelopment," *NeuroImage*, vol. 146, pp. 1038–1049, 2017.

[17] B.-H. Kim, J. C. Ye, and J.-J. Kim, "Learning dynamic graph representation of brain connectome with spatio-temporal attention," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4314–4327, 2021.

[18] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[19] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

[20] X. Li, Y. Gu, N. Dvornek, L. H. Staib, P. Ventola, and J. S. Duncan, "Multi-site fmri analysis using privacy-preserving federated learning and domain adaptation: Abide results," *Medical Image Analysis*, vol. 65, p. 101765, 2020.

[21] P. P. Liang, T. Liu, L. Ziyin, N. B. Allen, R. P. Auerbach, D. Brent, R. Salakhutdinov, and L.-P. Morency, "Think locally, act globally: Federated learning with local and global representations," *arXiv preprint arXiv:2001.01523*, 2020.

[22] C. T Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with moreau envelopes," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 394–21 405, 2020.