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

Functional magnetic resonance imaging (fMRI) has emerged as a non-invasive technique that has proven effective in exploring the functional structure of the brain [1]. By tracking blood flow changes, fMRI provides a valuable way for healthcare professionals, neuroscientists, psychologists, data analysts, and others to understand and interpret brain activity.

To enhance computer-aided fMRI analysis, we develop a Python-based and cross-platform toolbox (called ACTION). ACTION offers a user-friendly tool to streamline the fMRI processing workflow and boost efficiency. It features four function modules: fMRI data augmentation, brain network construction, brain network feature extraction, and artificial intelligence model construction. Specifically, *fMRI data augmentation* module augments fMRI data from both blood-oxygen-level-dependent (BOLD) signal and graph levels. The *brain network construction* module incorporates several popular methods for network construction, and it also supports network visualization. The *brain network feature extraction* module facilitates the extraction of both node-level and graph-level network features. The *artificial intelligence model construction* module includes the construction of both conventional machine learning models and advanced deep learning models. This module integrates several federated learning strategies to facilitate multi-site fMRI studies. Moreover, ACTION allows users to test their self-designed algorithms through scripting, which greatly enhances its utility and extensibility.

B. Feasible Combinations of Four Function Modules

As shown in Fig. S1, we provide a diagram to guide users on feasible combinations of existing function modules, which are detailed in the following.

- The "FMRI Data Augmentation" module contains two functions, *i.e.*, "BOLD Signal Augmentation" and "Graph Augmentation". The input data of "BOLD Signal Augmentation" and "Graph Augmentation" are fMRI time series and graph-based data, respectively.
- The "Brain Network Construction" module takes fMRI time series as inputs and outputs graph-based data.

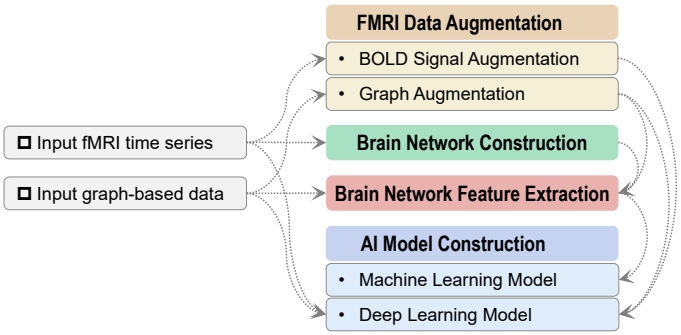

Fig. S1. Illustration of feasible combinations of four proposed function modules.

- For the "Brain Network Feature Extraction" module, its inputs can be graph-based data, augmented graphs (from "Graph Augmentation" module), and graphs generated by "Brain Network Construction" module.
- In the "AI Model Construction" module, there are two functions, *i.e.*, "Machine Learning Model" and "Deep Learning Model". (1) The input for the "Machine Learning Model" can be sourced from outputs of the "Brain Network Feature Extraction" module. (2) The input for the "Deep Learning Model" module can be fMRI time series or augmented time series.
- Several graph learning methods (*i.e.*, GAT, GIN, GCN, BrainNetCNN) integrated into the "Deep Learning Model" module can also utilize graph-based data or augmented graphs as inputs.

C. Software Version

The ACTION is coded in Python 3.8 and currently operates smoothly on Python 3.7 and higher versions (with the latest version being Python 3.12). The dependencies required to ensure the proper operation of ACTION will be detailed in Section II-B. We will continue to monitor and update the manual for future releases of Python versions.

D. License

the tool's author for permission. While ACTION is provided with the hope of being beneficial, it comes with no guarantees, including no implied warranties of being sellable or suitable for any specific use. Please refer to the GNU General Public License (https://www.gnu.org/licenses/licenses.html#GPL) for further information.

II. Installation Instructions

A. Operation System Requirements

The ACTION software can operate smoothly on the following operating systems:

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

The ACTION has been verified to function properly on the aforementioned operating systems. While the toolbox may operate effectively on other systems, we cannot fully guarantee its feature completeness or stability. Hence, we recommend users to use ACTION on the tested operating systems for the best experience.

B. Dependencies

To successfully run the toolbox, users should install the following dependencies on Python versions 3.7 or higher:

- 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 successful installation and launch of the ACTION software, users are greeted with the welcome interface, as shown in Fig. S2. The interface clearly outlines the functions of the ACTION software and the intended purposes of each function. The toolbox's logo and its full name are displayed at the top of the page. Below the logo, there are four buttons, each representing a function module of the toolbox. In the middle of the page, brief descriptions of the intended purposes for these function modules are provided. At the bottom of the page, users can access the manual by clicking the "MANUAL" button, or exit the software by clicking the "EXIT" button.

B. Quick Start Guide

The interface for each function is organized into three main parts, shown in Fig. S3. The first part is used for data loading, where users input the data to be processed via the address bar labeled "Load *". The second part is the main functional area, featuring algorithms that carry out specific tasks. It also

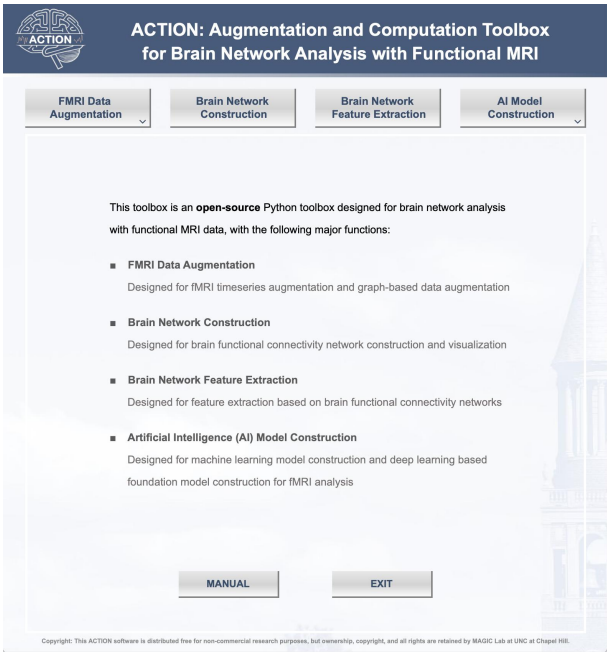

Fig. S2. Welcome page of the ACTION software.

contains a location to specify a storage path for the obtained results labeled "Save Directory." Some certain interfaces also include result displays, such as the brain network construction module. The third part contains the interface control buttons, including "Help", "Clear", and "Quit".

For the "Load *" and "Save Directory" sections, users can click the "Browse" buttons to open dialog boxes for loading and saving data, respectively. Clicking the "Run *" or "Create *" buttons starts the process. Clicking the "Help" button displays assistance information for the current interface, the "Clear" button clears contents of all text boxes in the current interface, and the "Quit" button returns users to the welcome interface. To help users quickly become acquainted with the workflow of the toolbox, we provide a step-by-step demonstration using the "BOLD Signal Augmentation" module as an example (see Fig. S3).

- **Step 1**: Click the "FMRI Data Augmentation" button and select "BOLD Signal Augmentation" from the drop-down menu to access the corresponding interface.
- **Step 2**: Click the "Browse" button next to "Load Data" to open a file selection dialog. Select the original fMRI data that needs processing.
- **Step 3**: In the "Select Algorithm" section, choose the desired algorithm and adjust the parameter values within a reasonable range. For example, you might select the "Upsampling" algorithm and set the ratio to 0.7.
- **Step 4**: Click the "Browse" button next to "Save Directory" to specify where the augmented data should be saved. If no directory is selected, the results will be saved in the "results" folder by default.
- **Step 5**: Click the "Create Data" button to start augmenting the original fMRI BOLD signals, and the result will automatically be saved in the chosen directory.

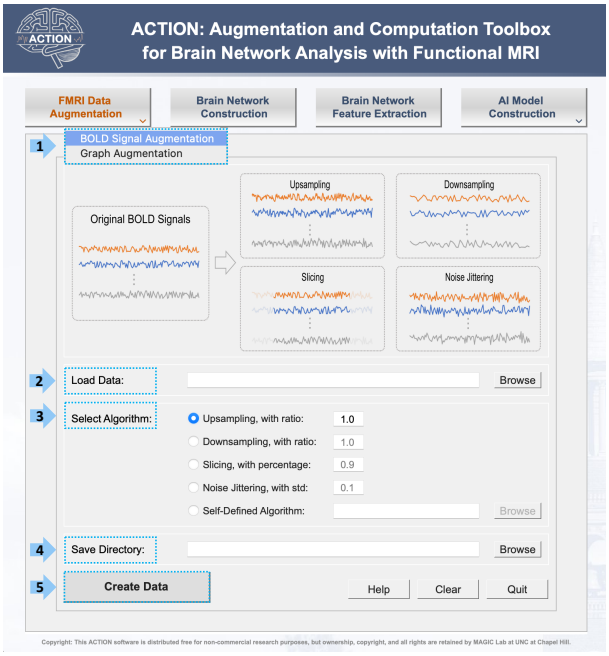

Fig. S3. Blood-oxygen-level-dependent (BOLD) signal augmentation module.

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: First, users need to upload the original BOLD signals 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 number of time points. If the input does not meet this requirement, an error message "Input shape error, please check your input dimensions" will appear, along with information about the correct dimension. Next, users should select a desired augmentation algorithm from the available options. Currently, the ACTION software includes four methods for BOLD signal augmentation, detailed as follows.

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

Users can choose any of the above four methods and adjust the corresponding parameters within a reasonable range, or the default parameters will be applied. Afterward, specify the location for saving the augmented data and click "Create Data" to initiate BOLD signal augmentation. The result is named as "save_data + algorithm". Details can be found in Fig. S3.

2) Graph Augmentation: First, users need to upload the original adjacency matrix and corresponding untreated node attribute matrix in the specified 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 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. Users need to choose from one of the six graph-based augmentation algorithms provided, with details introduced as follows.

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

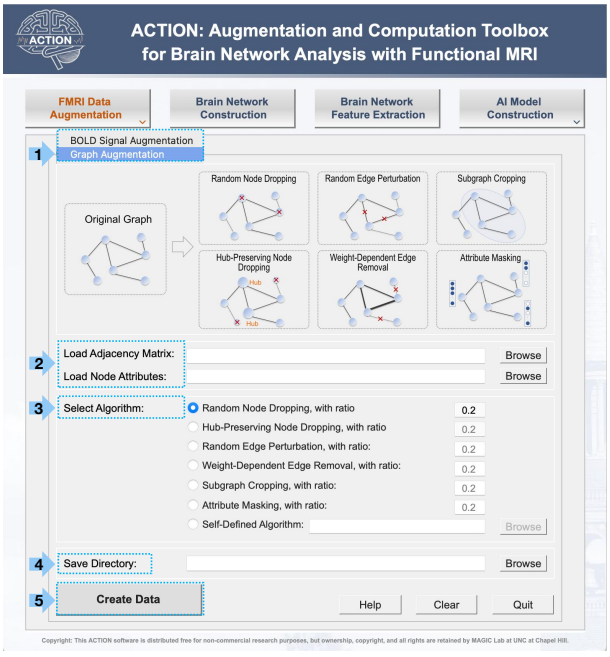

Fig. S4. Brain network/graph augmentation module.

After selecting a method, users can adjust the corresponding parameters, or the default parameters will be applied. Next, users need to specify the save location for the augmented results and click "Create Data" to initiate the graph augmentation process. The file name for the saved augmented adjacency

matrix and node attribute matrix are formatted as "save_adjmat + algorithm" (if exists) and "save_nodeatt + algorithm" (if exists), respectively. Details can be found in Fig. S4.

B. Brain Network Construction

This module allows users to construct and visualize brain networks from BOLD signals. Users need to upload a 3-dimensional matrix in the shape of (S, N, T) as a *.npy* file. 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** (PrC): 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, which is non-negative with a default value of 1.
- **Low-rank Representation** (LR): Key parameter is the low-rank regularization parameter, which is non-negative with a default value of 1.

In most cases, the constructed brain network is dense. Our toolbox offers the function to sparsify the constructed brain network through thresholding (found under the "Optional" settings), catering to users who prefer a sparser network. By default, this option is set to "None", which means no sparsification is applied. The toolbox offers two methods for sparsification:

- **Binarization**: Key parameter is the threshold T, which retains the top $T\%$ of edges by assigning them a weight of 1 and sets all other edges to 0. The default T is 30.
- **Sparsity**: Key parameter is the threshold K, which retains the top $K\%$ of edges with their original weights and sets all other edges to 0. The default K is 30.

After selecting the brain network construction algorithm and a sparsification method, users can click the "Run & Save" button to save the constructed brain network in the specified directory with the file name "algorithm + Adjacency_Matrix + sparsity method". To visualize the constructed brain network for each subject, users can enter the subject index (default is the first subject) and click the "Visualization" button. This action will display the visualization results on the right side of the interface. Two visualization options are provided: Adjacency Matrix and Network Topology. For Adjacency Matrix, each row and column corresponds to a brain region, and each entry denotes the functional connections between regions. For Network Topology, users can directly visualize the connectivity patterns formed by nodes and edges. Details can be found in Fig. S5.

C. Brain Network Feature Extraction

First, users need to upload the constructed brain network, which should be saved as a *.npy* file containing a 3-dimensional matrix with a shape of (S, N, N). The ACTION

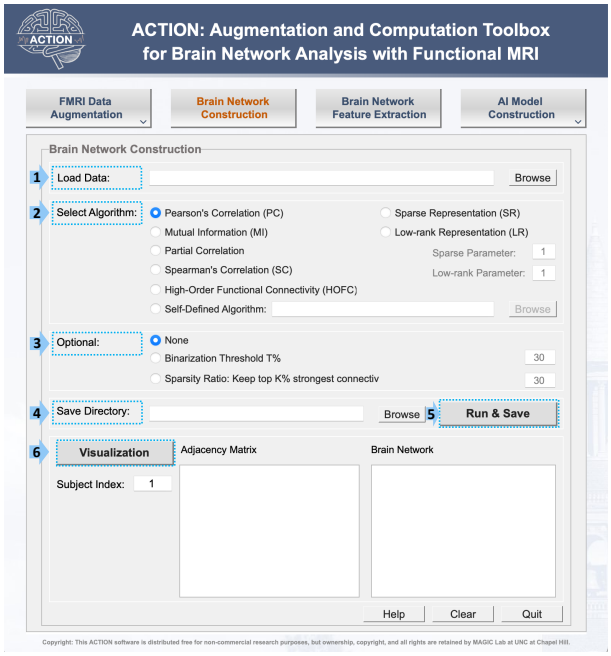

Fig. S5. Brain network construction module.

software enables feature extraction from brain networks at both node level and graph level, and the corresponding functions are available through the "Node-based" and "Graph-based" sections. Moreover, users can consider edge weights during feature extraction by selecting the "Weighted Graph" option. If this option is selected, the network is treated as a weighted graph, or it is treated as an unweighted/binary graph.

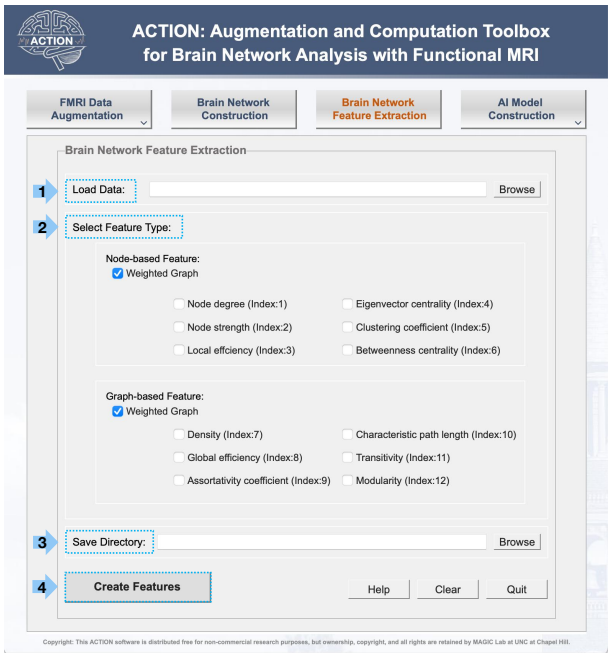

Fig. S6. Brain network feature extraction module.

Six algorithms are provided for extracting 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, six algorithms for graph-based feature extraction are included, listed as:

- 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 the provided list according to their needs. The final extracted features consist of the features from the selected algorithms, concatenated in the order of their indices.

To save the extracted features, users should specify a saving location and then click "Create Features" to initiate the feature extraction process. Upon completion, the results will be saved in a file named "save_feature + index" in the specified directory, where "index" represents the indices of the selected features. For example, if node degree, modularity, and density are selected, the file will be named "save_feature_1_5_8". Details are given in Fig. S6.

D. Artificial Intelligence Model Construction

1) Machine Learning Model Construction: First, users need to upload the sample data for classification or regression tasks, along with the corresponding labels. The sample data should be a 2-dimensional matrix with a shape of (S, F), where each row represents a feature vector of length F, stored in a *.npy* file. The labels are a vector of length S, also stored in a *.npy* file. Note that for classification tasks, labels should be either $\{0,1\}$ or $\{-1,1\}$. After uploading the data, users should specify whether they perform a classification or a regression task. Prior to starting the task, users can decide whether to perform dimension reduction on the input features. The "Dimension Reduction Algorithm" section of the toolbox offers three methods for reducing feature dimensions:

- **Principal Component Analysis** (PCA) [2]: 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 as an integer equal to or greater than 1, it denotes the number of principal components to retain. The default value is 1.
- **Univariate Feature Selection** (UFS), which chooses features that have the strongest relationship with the target variable based on ANOVA.
- **T-test**, which selects features based on their p-values, where a low p-value indicates that the corresponding feature is statistically significant in distinguishing between the two groups.

Users can choose one of these algorithms for dimension reduction, or select "None" if no dimension reduction is required. Moreover, our 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) [3]: For both classification and regression.
- **Extreme Gradient Boosting**(XGBoost) [4]: For both classification and regression.
- **K-Nearest Neighbors** (KNN) [5]: For both classification and regression.

Users also need to choose a method for dividing the data into training and validation sets. The toolbox offers two strategies for data partitioning:

- **K-fold Cross Validation**: Key parameter is the number of folds for cross-validation, K, which is an integer greater than or equal to 2. The default value is $K = 5$.
- **Random Partition**: Key parameter is the partition ratio of the training set, which ranges from 0 to 100, with a default of 70.

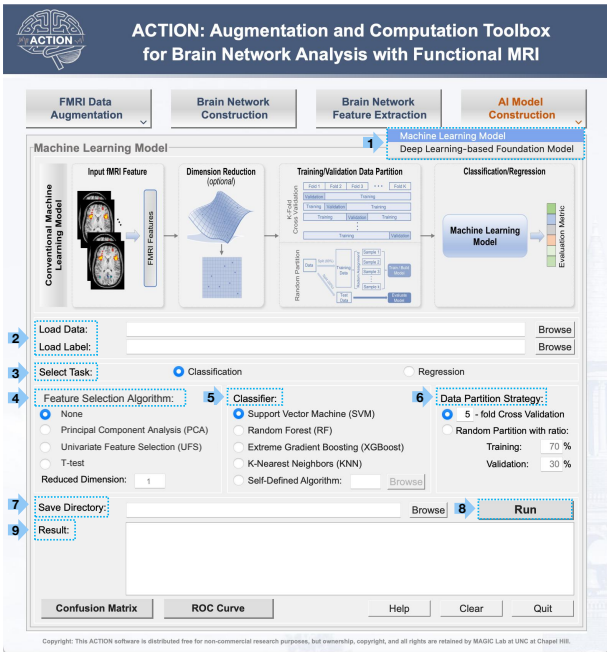

Fig. S7. Machine learning model construction module.

After completing all the necessary preparations and clicking the "Run" button, the results will be displayed in a table in the designated "Result" area. The result will also be saved as a *.json* file in the selected directory, named "save + classification/regression algorithm + cls/reg_result + reduction algorithm + partition strategy". If dimension reduction is performed, the reduced features will be saved as a *.npy* file and named "save + reduction algorithm + fea_reduced_dim + reduced dimension". It is worth mentioning that, for classification tasks, the ACTION software includes functionality to plot the confusion matrix and the ROC curve, providing a more visual interpretation of the results. These plots can be accessed by clicking the respective buttons. Details can be found in Fig. S7.

2) Deep Learning Model Construction: The toolbox also supports the construction of deep learning models. It features ten deep learning models as the backbone encoders and in-

cludes 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, and the pre-training process is presented at the top of Fig. S8.

The ten deep learning models include:

- **Transformer** [6]
- **Graph Attention Network** (GAT) [7]
- **Graph Isomorphism Network** (GIN) [8]
- **Graph Convolutional Network** (GCN) [9]
- **Brain Graph Neural Network** (BrainGNN) [10]
- **Graph SAmple and aggreGatE** (GraphSAGE) [11]
- **Spatio-Temporal Graph Convolutional Network** (STGCN) [12]
- **Modularity-constrained Graph Neural Network** (MGNN) [13]
- **Brain Network Convolutional Neural Network** (BrainNetCNN) [14]
- **Spatio-Temporal Attention Graph Isomorphism Network** (STAGIN) [15]

The five federated learning methods are as follows (with GCN as the default backbone model):

- **Federated Averaging** (FedAvg) [16]
- **Federated Proximal** (FedProx) [17]
- **Model-Contrastive Federated Learning** (MOON) [18]
- **Local Global Federated Averaging** (LGFedAvg) [19]
- **Personalized Federated Learning with Moreau Envelopes** (pFedMe) [20]

After selecting a specific backbone encoder, users can click the "Fine-Tuning Process" button to access the pre-trained model and the detailed instructions for model fine-tuning. After selecting a federated learning strategy, users can click the "Source Code Link" button to access the corresponding algorithm. It is worth noting that the toolbox provides only pre-trained deep learning models, and users can fine-tune these pre-trained models to their specific downstream tasks. Moreover, for each federated learning strategy, the default backbone is the GCN while other models can also be employed.

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 allows users to integrate their custom algorithms into various modules, including the "FMRI Data Augmentation", "Brain Network Construction", and "Machine Learning Model" modules. Each of these modules features a "Self-Defined Algorithm" option. Users can select this option and use the "Browse" button to upload their custom algorithms. Once uploaded, these custom algorithms can be executed in the same manner as the toolbox's built-in algorithms.

To ensure compatibility with the data processing needs, users should format their custom algorithms as functions in Python and package them into a *.pkl* file using the "dill" library. Note that users need to use the "dill.settings['recurse'] = True" command to ensure that all dependencies in the function are successfully packaged and that these dependencies are properly installed on the operational device. The packaging process is depicted in Figure S9. Details on the specific forms of custom algorithms for each module are provided as follows.

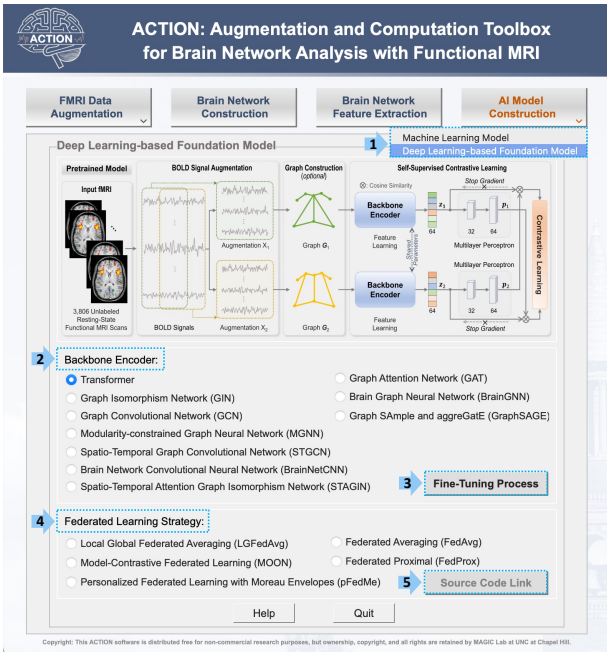

Fig. S8. Deep learning based foundation model construction module.

```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. S9. Illustration of packaging process for self-defined algorithms.

A. BOLD Signal Augmentation

With this module, users are allowed to use their own custom BOLD signal augmentation functions, which require two ordered inputs:

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

Therefore, users can define the BOLD signal augmentation algorithm *BOLDAug* as $\mathrm{X}' = \mathit{BOLDAug}(\mathrm{X}, \mathrm{param})$, where X' is the augmented BOLD signals. Once the algorithm is successfully imported, clicking the "Create Data" button will save the result in the specified directory, named "save_custom_augmented_result".

B. Graph Augmentation

Users can use custom algorithms to augment the graph, which require three ordered inputs:

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

The graph augmentation algorithm, *GraphAug*, is defined as X′_adj, X′_att = *GraphAug*(X_adj, X_att, param), where X′_adj and X′_att represent the augmented adjacency matrix and node attributes matrix, respectively. Sometimes, users may want to augment either nodes or edges, while keeping the other unchanged. In such cases, "None" should be used as a placeholder for the missing output. For example, if the user only needs to augment the edges, the function will be changed to X′_adj, None = *GraphAug*(X_adj, X_att, param). Once the algorithm is successfully imported and the "Create Data" button is clicked, the augmented adjacency matrix will be saved as "save_adjmat_custom", and the augmented node attributes matrix will be saved as "save_nodeatt_custom" in the specified directory.

C. Brain Network Construction

In this module, users can use their own developed algorithms to construct brain networks, perform sparsification through thresholding, and carry out visualization. The algorithm should have two ordered inputs:

- **X**: The original BOLD signals, 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), and the function *BrainNet* should be structured as X′ = *BrainNet*(X, param). Once the algorithm is successfully imported, clicking the "Run & Save" button will save the brain network constructed by the custom algorithm (named "Custom_Adjacency_Matrix") in the specified directory. Similar to using built-in algorithms, clicking the "Visualization" button will display the visualization results for the chosen subject.

D. Machine Learning Model

The ACTION software allows users to use custom models for classification or regression tasks. These custom models should be structured as functions with four ordered inputs:

- **Y_tr**: This is the label vector for the training set, containing S elements. For classification tasks, the labels should be either 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 represents the dimension of the features.
- **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).

If the model is designed for classification, it should have two ordered outputs:

- **Y_va**: The predicted values for the validation samples, with a length of S_{va}.
- **Prob**: The probabilities that the model predicts the validation samples as positive class, which is a vector of length S_{va}.

If the model is designed for regression, there is one output:

- **Y_va**: The predicted values for the validation samples, with a length of S_{va}.

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 loading the custom model, specify the data partition strategy, and then click the "Run" button to execute the model. The experimental results obtained from the custom model will be shown upon completion, and a *.json* file named "save_custom + task + dimension reduction method + partition strategy" will be saved in the specified directory. For classification tasks, additional functionality is provided through the "Confusion Matrix" and "ROC Curve" buttons, allowing users to visualize the respective results.

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

[4] 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.

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

[6] 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.

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

[8] B.-H. Kim and J. C. Ye, "Understanding graph isomorphism network for rs-fMRI functional connectivity analysis," *Frontiers in Neuroscience*, p. 630, 2020.

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

[10] 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.

[11] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[12] S. Gadgil, Q. Zhao, A. Pfefferbaum, E. V. Sullivan, E. Adeli, and K. M. Pohl, "Spatio-temporal graph convolution for resting-state fMRI analysis," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2020, pp. 528–538.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] Q. Li, B. He, and D. Song, "Model-contrastive federated learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 713–10 722.

[19] 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.

[20] 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.