# PHOTONAI-Graph - A Python Toolbox for Graph Machine Learning

**Jan Ernsting*[1, 2, 3], Vincent Holstein*[2]✉, Nils R. Winter[2], Kelvin Sarink[2], Ramona Leenings[2, 3], Marius Gruber[2, 4], Jonathan, Repple[2, 4], Benjamin Risse[1, 3], Udo Dannlowski[2], and Tim Hahn[2]**

[1]Institute for Geoinformatics, University of Münster, Germany
[2]Institute for Translational Psychiatry, University of Münster, Germany
[3]Faculty of Mathematics and Computer Science, University of Münster, Germany
[4]Department of Psychiatry, Psychosomatics and Psychotherapy, University of Frankfurt, Germany
*These authors contributed equally: Jan Ernsting, Vincent Holstein

Graph data is an omnipresent way to represent information in machine learning. Especially, in neuroscience research, data from Diffusion-Tensor Imaging (DTI) and functional Magnetic Resonance Imaging (fMRI) is commonly represented as graphs. Exploiting the graph structure of these modalities using graph-specific machine learning applications is currently hampered by the lack of easy-to-use software. PHOTONAI Graph aims to close the gap between domain experts of machine learning, graph experts and neuroscientists. Leveraging the rapid machine learning model development features of the Python machine learning API PHOTONAI, PHOTONAI Graph enables the design, optimization, and evaluation of reliable graph machine learning models for practitioners. As such, it provides easy access to custom graph machine learning pipelines including , hyperparameter optimization and algorithm evaluation ensuring reproducibility and valid performance estimates. Integrating established algorithms such as graph neural networks, graph embeddings and graph kernels, it allows researchers without significant coding experience to build and optimize complex graph machine learning models within a few lines of code. We showcase the versatility of this toolbox by building pipelines for both resting–state fMRI and DTI data in the hope that it will increase the adoption of graph-specific machine learning algorithms in neuroscience research.

**Graph Machine Learning | Network Neuroscience | Graph Neural Networks | Auto-ML**

**Correspondence:** ***v_hols01@uni-muenster.de***

## Introduction

Graph data is ubiquitous throughout biomedical research and can be found in many different fields. In neuroscience, graph representations are of particular interest as the neuronal connections within the brain are a naturally occurring graph structure. These graphs arise both on the microscopic level in cellular connection networks and the macroscopic level such as higher-order brain circuits. Functional connectivity and diffusion tensor imaging (DTI) are the two most commonly used modalities for studying these circuits in vivo. The two modalities allow for the construction of brain connectivity graphs, which can then be studied using graph theoretical approaches. Different established toolboxes allow neuroscientists to apply classical graph theoretical approaches using statistical analysis. However, multivariate graph analyses such as graph machine learning pipelines are mostly limited to a domain expert group, limiting the accessibility for many neuroscientists.

In classical graph analysis graph properties are calculated, sometimes referred to as graph measures, which are then analyzed using statistical analysis tools such as general linear models (GLMs). This approach has been increasingly adopted in neuroimaging, which is in part due to the availability of toolboxes, that support these analyses [1, 2]. These toolboxes are usually geared towards neuroimaging often specializing in one type of connectivity modality. One popular toolbox is Brain Connectivity Tools which offers a graphical user interface and implements the most important graph analysis methods based on graph measures [3]. Other frequently used toolboxes for analyzing brain connectivity graphs using graph measures include but are not limited to, Network-based Statistics, eConnectome, CONN, GAT, GTG, BASCO, GRETNA or BRAPH [4–11].

These neuroimaging-specific toolboxes support basic graph analysis, but not graph machine learning analyses. As one of the first toolboxes, the GraphVar 2.0 toolbox expands this framework to the area of machine learning by allowing users to perform machine learning on extracted measures [12]. This represents a novel approach towards creating low-dimensional graph representations that capture important information about the inherent graph structure which are then used for machine learning. However, this represents only a small area of the field of graph machine learning which has exponentially grown in recent years.

The approach of creating lower-dimensional graph representations is at the core of the field of graph machine learning. In this new machine learning subfield the main directions of research are graph embeddings, graph kernels and graph neural networks [13]. Graph embeddings project to a lower dimensional representation that leverages graph information and makes it accessible in Euclidean space [14]. Graph kernels are functions that either extract feature representations or calculate similarity measures between graphs, which can be leveraged using kernel methods [15]. Significant overlap exists between the two fields of research. Lastly, there are graph neural networks, which are a form of neural networks that leverage graph information, by performing neural message passing. Here information flows between nodes and is updated using neural network functions [16, 17].

In neuroimaging, most research uses graph measures to predict outcomes of interest. Recently researchers have begun to use graph machine learning in neuroimaging and even built graph neural networks specifically targeting brain connectivity matrices [18, 19]. These approaches however are still much less common than classic graph analysis, partly because they require require significant technical knowledge.

While the neuroimaging community has developed packages for graph measure analysis on brain-derived graphs, the graph machine learning community has developed specialized packages for graph machine learning. However, these require advanced coding experience and usually cover only one particular (sub-)field. Examples include *gem* for graph embeddings, *grakel* for graph kernels and deep graph library (*dgl*) for graph neural networks [20–23]. All of these toolboxes require programming expertise and technical knowledge for their usage, which bars many scientists without such background from routinely using these for their analyses. This causes an accessibility gap for many neuroscience researchers, which is not addressed by any of these toolboxes.

Therefore there is currently no toolbox that implements a wide array of graph machine algorithms in an accessible manner. Existing toolboxes in neuroimaging focus on classical graph analysis or graph measure-derived machine learning, while graph machine learning toolboxes focus on one type of algorithm and require significant computational expertise. Among the computational challenges are the incompatibility of different graph libraries, varying data structures and error-proneness of such conversions. To address this gap we developed our graph machine learning toolbox called *"PHOTONAI Graph"*.

The integration of this toolbox into the *PHOTONAI* package provides increased accessibility of complex algorithms via pre-defined keywords, extended pipeline functionality such as stacking operations, a high degree of automation with variable hyperparameter optimization and cross-validation strategies, easy visualization and model sharing options ([24]). This provides neuroscientists with access to best practices in machine learning and rapid prototyping capabilities, which will strongly improve model building without the need to learn and code a large body of graph algorithms.

## Methods & Software

*PHOTONAI-Graph* is a *scikit-learn*-compatible *Python* package that is an extension to the PHOTONAI toolbox [24], which provides a high degree of automation of the repetitive steps during model development. As an extension to the *PHOTONAI* framework, it uses the functionality of the *PHOTONAI* toolbox in the context of graph machine learning. This allows an out-of-the-box usage of graph algorithms with fully automated hyperparameter optimization, model evaluation and algorithm selection procedures. This integration also enables seamless access to *PHOTONAI's* convenience features such as result visualization and model sharing and integrates within *PHOTONAI's* structured and easy-to-learn syntax, condensing machine learning analysis to a few lines of code.

The *PHOTONAI-Graph* toolbox consists of multiple modules which correspond to different types of data transformations, learning algorithms, and utilities. In the following sections, we explain each of the different toolbox modules, and our testing strategy to ensure functionality, public access, documentation and software dependencies.

**Organisation.** The toolbox is designed to adapt to various types of graph data, with a focus on connectivity data. It is modality-agnostic however, as graph machine learning pipelines can be built for other types of graph data, as long as it is imported in a compatible format. For various types of graph data, the toolbox supports the development of whole graph classification or regression pipelines.

Pipeline development is supported by 8 main modules: *Graph Constructors*, *Graph Measures*, *Graph Kernels*, *Graph Embeddings*, *Graph Neural Networks*, *Controllability*, *Graph Conversions* and *Graph Utilities*. These modules cover key areas of current graph machine learning research and allow for the use of state-of-the-art graph machine learning in a few lines of code.

Neuroscientific graph data, such as connectivity matrices, can be transformed using *Graph Constructors*, to ensure adequate graph representation via adjacency matrices. These can be further transformed with graph-specific dimensionality reduction techniques using *Graph Kernels*, *Graph Embeddings*, *Graph Measures* or *Controllability* functions. The transformed data can then be passed to an estimator. Alternatively, graph properties can directly be estimated using a *Graph Neural Network*.

Conversions between different graph formats are handled by the *Graph Conversions* module, while the *Graph Utilities* module supplies different utilities such as drawing functions. Each module will be described in brief below.

**Graph Constructors.** The Graph construction module implements graph construction techniques that transform connectivity matrices into adjacency matrices to reduce noise and complexity, speed up computation and increase the signal-to-noise ratio. This transformation is especially important when working with highly dense data (i.e. fully connected graphs) such as resting-state connectivity.

Implemented techniques include thresholding based on cutoff values, the percentile rank, a window of cutoff values or percentile ranks, kNN-based adjacency formation, spatial proximity-based adjacency formation and random walks on kNN graphs. These transformations are applied individually to each graph. It is also possible to use the *PopulationAveragingTransform* class to build an average graph across each fold as described by Ktena et al. [25].

Users can also decide to apply different encodings for node features, such as a one-hot encoding, similar to indicator variable encodings or they can use the unfiltered connectivity matrix as a feature matrix. This way the feature matrix, which can later be used to define node and edge features, contains all the original information before any transformation was performed.

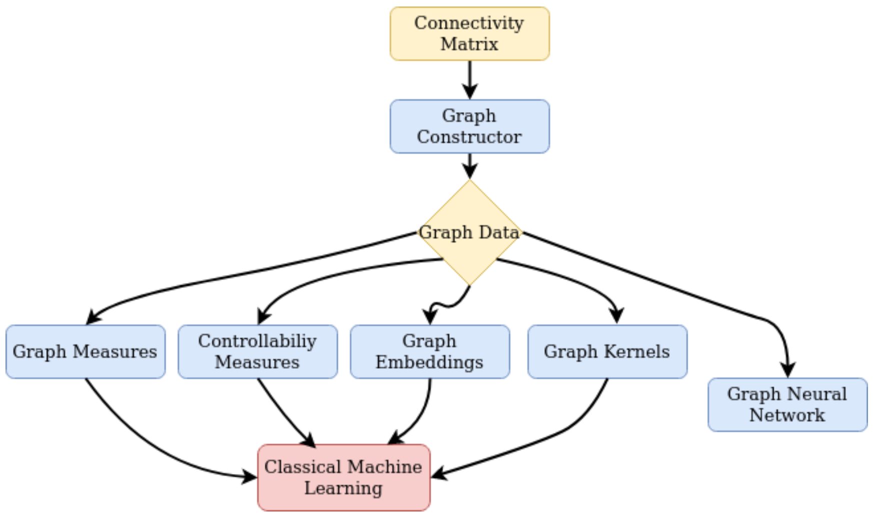


**Fig. 1.** Modules of PHOTONAI-Graph. The PHOTONAI-Graph package features multiple modules for transforming and predicting from graph data. Starting from connectivity matrices, the graph constructor module allows for graph construction with a range of established and novel thresholding and edge selection strategies. The derived graph data can be used by either transforming it with graph measures, network controllability measures, graph embeddings or graph kernels to acquire graph property-preserving representations that can then be used in concordance with classical machine learning estimators. Alternatively, users can directly use the graph data by feeding it to a graph neural network.

**Graph Measures.** The Graph Measure Transform module implements the *NetworkxMeasureTransform* and *IgraphMeasureTransform* class which calculate graph measures based on the *networkx* or *igraph python* package and concatenates them into a feature vector [26, 27]. Whole graph, node and edge measures can be combined to provide a low-dimensional feature representation of the graph. These feature vectors can then be analyzed using machine learning estimators. The choice of desired graph measures is a hyperparameter during model development.

To reduce runtime these calculations can be parallelized based on *tqdm* [28]. The *GraphMeasureTransform* class can also be used to extract graph measures into a CSV file for further statistical analysis outside of graph machine learning pipelines.

**Graph Embeddings.** The graph embedding module implements graph embeddings, based on a modified version of the *gem python* package [21]. These include the Asymmetric Transitivity Preserving Graph Embedding, Laplacian Eigenmaps and a locally linear embedding. As graph embeddings are not calculated between graphs but are a direct low-dimensional representation of the graph, they do not require the use of kernel methods.

To ensure the correct functioning of the *gem* software and avoid dependency conflicts we have developed a modified version of the *gem python* package which is publicly available (https://github.com/jernsting/nxt_gem) for this project.

**Graph Kernels.** The Graph Kernel module implements Graph Kernels based on the *grakel python* package, allowing for the usage of kernels currently available in *grakel* [22]. All available *grakel* kernels can be optimized with respect to their hyperparameters via the *GrakelTransformer* class. These kernels can then be used in a pipeline in combination with kernel methods such as support vector machines.

To ensure the correct conversion of input graph data into the required *grakel* format as part of a machine learning pipeline the module also contains the *GrakelAdapter* class, a transformer that handles these conversions. This adapter also allows the user to select different feature construction options if the graphs are converted from connectivity matrices.

**Graph Neural Networks.** The graph neural networks module currently implements six different Graph Neural Network (GNN) architectures, based on the deep graph library (*dgl*) *python* package and the *pytorch* library [23, 29]. These GNNs allow the user to use different architectures for both whole graph classification and regression tasks. As GNNs combine both graph representation learning and feature estimation, they do not require an additional estimator in the pipeline.

The GNN module consists of a *dgl_base* class that handles training and conversion steps shared between the different architectures, a set of graph neural network-specific utilities and architecture wrappers for graph convolutional networks, simple graph convolutional networks and graph attention networks. Each GNN architecture allows for wide-ranging hyperparameter optimization including layer size, depth, learning rate and the number of training epochs.

**Controllability.** The controllability module implements the modal and average controllability on graphs which can be used as a low-dimensional representation of the graph. These function are adapted from *nctpy* ([30–32] (https://github.com/BassettLab/nctpy). These low-dimensional representations can then be combined with estimators for regression or classification tasks. They can also be used outside of graph pipelines to calculate the modal and average controllability for further statistical analysis.

**Graph Conversions.** Graph Conversions are a collection of different conversion functions that handle conversions between different graph formats, namely: *networkx* graphs, *numpy* arrays, *scipy* sparse arrays, *dgl* graphs and *grakel* graphs. This facilitates the data flow between the available modules. It also implements the *check_dgl* function that ensures that incoming data is converted into the *dgl* format for graph neural networks.

**Graph Utilities.** The graph utilities module is a collection of helper functions that allow for the plotting of graphs in different formats, checking certain properties on graphs, fisher- and z-transformations and the generation of random data. The plotting functions allow the user to visually inspect the input data, both for connectivity matrices and *networkx* graphs.

**Toolbox documentation and installation.** The entire software is made publicly accessible as a repository on GitHub (https://github.com/wwu-mmll/photonai_graph) and is published under an MIT License. To facilitate public use and allow easier use by researchers without a computer science background, we created a documentation website hosted on GitHub Pages (https://wwu-mmll.github.io/photonai_graph/). Here we give a detailed description of the different classes and functions, along with their parameters and arguments. Current installation instructions can also be found on the website or in the GitHub repository.
To ensure the correct functioning of our code the software was unit tested with >90% coverage. Additionally, scenario tests ensure the correct integration of the different modules.

**Analysis Data.** For all analysis demonstrations, we used two publicly available imaging datasets. 10Kin1Day and the Human Connectome Project Young Adult (HCP-YA) dataset [33, 34].
The 10KIn1Day dataset contains >8000 participants processed with the Cammoun Desikan-Killiany atlas parcellation containing 128 regions resulting in 16384 edges [33]. For each participant five measures of connection strength are available as edge weights: 1) Number of streamlines (NOS), 2) average fractional anisotropy (FA), 3) average mean diffusivity (MD), 4) average length of reconstructed streamlines and 5) streamline density. We used all 8163 participants (4339 male, 3824 female) for which sex information and DTI matrices with number of streamlines was available in our analysis. Further information can be found in van den Heuvel et al, 2019 [].
The HCP-YA dataset is a dataset of >1000 young adults with resting-state fMRI (rs-fMRI) and preprocessed parcellations available [34]. Resting-state fMRI scans were preprocessed running a spatial ICA with 15, 25, 50, 100, 200 and 300 numbers of components from FSL's MELODIC tool. Network matrices were derived using the FSLNets toolbox on the node time series. We selected all preprocessed matrices with 50 components and available sex or strength information for our analysis resulting in 1003 individuals being included in sex prediction and 1002 individuals being included in our grip strength regression pipeline. The sex analysis contained 469 males and 534 females. Grip strength was measured by the NIH Toolbox Grip Strength Test using dynamometry and is provided as part of the HCP-YA data release. It is measured in pounds and adjusted for age with a sample range of 45.41 to 154.59.
All analysis scripts are available on GitHub (https://github.com/wwu-mmll/photonai_graph_usecases/) in a dedicated repository with data handling scripts for convenient handling of the datasets.

## Results

To demonstrate the versatility of our toolbox, we conducted three different analyses on two publicly available datasets to show both the versatility of the toolbox and potential use cases for it. It is important to note that for each of these demonstrations, we aim not at constructing the best available model, but rather showcase the versatility of our toolbox and the potential analyses that could be conducted with it.

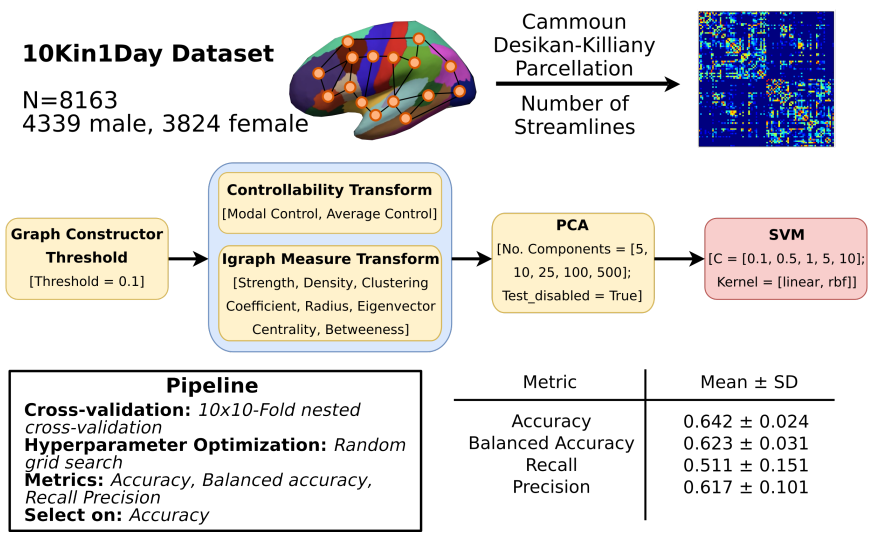
**Predicting Sex from DTI.** First, we build a sex classification model on the 10KIn1Day dataset to show the application of the toolbox to DTI data. The pipeline used a threshold constructor with a fixed threshold of 0.1, a stack of controllability transformation with average and modal controllability and a Support Vector Classifier as a predictor. Optimizable hyperparameters were the number of components in the PCA, the C parameters and the kernel of SVM. The C parameter was allowed to vary between 0.1, 1, 2.5 or 10. The potential kernels were either linear or rbf. We used 10x10 K-fold nested cross-validation with random grid search and 50 configurations for hyperparameter selection selecting configurations based on accuracy. We also calculated balanced accuracy, recall and precision.

| Metric | Test Mean ± Standard Deviation |
|---|---|
| Accuracy | 0.642 ± 0.024 |
| Balanced Accuracy | 0.623 ± 0.031 |
| Precision | 0.617 ± 0.101 |
| Recall | 0.511 ± 0.151 |

**Table 1.** Predictive performance for sex classification of the best pipeline trained on the 10KIn1Day DTI dataset.

The overall best pipeline configuration was able to classify sex with an accuracy of 0.64, balanced accuracy of 0.62, a recall of 0.51 and a precision of 0.61. It used 100 PCA components, a C parameter of 1 and an rbf kernel.

**A. Predicting Sex from resting-state fMRI.** Next, we build a sex classification model from the HCP-YA dataset to show an application to fMRI data. The pipeline consisted of a percentage constructor with a threshold of either 50, 75 or 90 and a Graph Convolutional Neural Net Classifier with 1 to 3 hidden convolutional layers, hidden dimensions of 32, 64, 128 or 256 and training epochs of either 50, 100, 250 or 500. We used a 10x10-fold nested cross-validation for hyperparameter optimization with the best configuration being selected based on accuracy. We used a Bayesian hyperparameter optimization strategy provided by *sk_opt* with the lower


10Kin1Day Dataset
N=8163
4339 male, 3824 female
Cammoun Desikan-Killiany Parcellation
Number of Streamlines
Graph Constructor Threshold
[Threshold = 0.1]
Controllability Transform
[Modal Control, Average Control]
Igraph Measure Transform
[Strength, Density, Clustering Coefficient, Radius, Eigenvector Centrality, Betweeness]
PCA
[No. Components = [5, 10, 25, 100, 500]; Test_disabled = True]
SVM
[C = [0.1, 0.5, 1, 5, 10]; Kernel = [linear, rbf]]

**Pipeline**
**Cross-validation:** *10x10-Fold nested cross-validation*
**Hyperparameter Optimization:** *Random grid search*
**Metrics:** *Accuracy, Balanced accuracy, Recall Precision*
**Select on:** *Accuracy*

| Metric | Mean ± SD |
|---|---|
| Accuracy | 0.642 ± 0.024 |
| Balanced Accuracy | 0.623 ± 0.031 |
| Recall | 0.511 ± 0.151 |
| Precision | 0.617 ± 0.101 |

**Fig. 2.** Predicting sex from DTI. We built a sex classification model from the 10KIn1Day dataset (N=8163; 4339 male, 3824) using connectivity matrices with the number of streamlines derived from the Cammoun Desikan Killiany parcellation to showcase a pipeline for DTI derived connectivity. The pipeline consisted of threshold graph constructor, a stack of a controllability and igraph measure transform, a PCA and an SVM as an estimator. The best model was selected using a 10x10 K-Fold nested cross-validation with the best model selected on accuracy. The best pipeline configuration classified sex with an accuracy of 0.64, a balanced accuracy of 0.62, recall of 0.51 and a precision of 0.61. The model used 100 PCA components, a C parameter of 1 and an rbf kernel.

confidence bound as the acquisition strategy and a quasirandom Sobol vector point generator with 15 initial points and 25 configurations. We again calculated balanced accuracy, recall and precision.
The overall best pipeline configuration was able to classify sex with an accuracy of 0.74, balanced accuracy of 0.74, precision of 0.72 and recall of 0.74. The selected percentage threshold was 90, while the Graph Convolutional Classifier with the best performance used 50 training epochs, 2 hidden layers, and a hidden layer size of 256.

| Metric | Test Mean ± Standard Deviation |
|---|---|
| Accuracy | 0.749 ± 0.026 |
| Balanced Accuracy | 0.747 ± 0.026 |
| Precision | 0.727 ± 0.035 |
| Recall | 0.740 ± 0.053 |

**Table 2.** Predictive performance for sex classification of the best pipeline configuration trained on the HCP-YA dataset.

**Predicting grip strength from resting-state fMRI.** Lastly, we build a strength prediction model to showcase the building of a regression model using the HCP-YA dataset to showcase the application of our toolbox to a regression problem.
We constructed a pipeline consisting of a threshold constructor with a threshold between 0 and 1, and a Graph Attention Neural Net Classifier with 1 to 3 hidden convolutional layers, hidden dimensions of 32, 64, 128 or 256 and training epochs of either 50, 100, 250 or 500. We used 10x10-fold nested cross-validation for hyperparameter optimization with the best configuration being selected based on Pearson correlation. We also calculated mean absolute error and explained variance. For hyperparameter optimization, we used the same *sk_opt* strategy as previously in the resting-state fMRI sex prediction.
The overall best pipeline configuration achieved a Pearson correlation of 0.36, a mean absolute error of 16.58 and a variance explained of 0.135. The threshold value for the constructor was 0.877, and the Graph Convolutional Net Regressor used 250 training epochs with 3 hidden layers and a hidden layer size of 32.

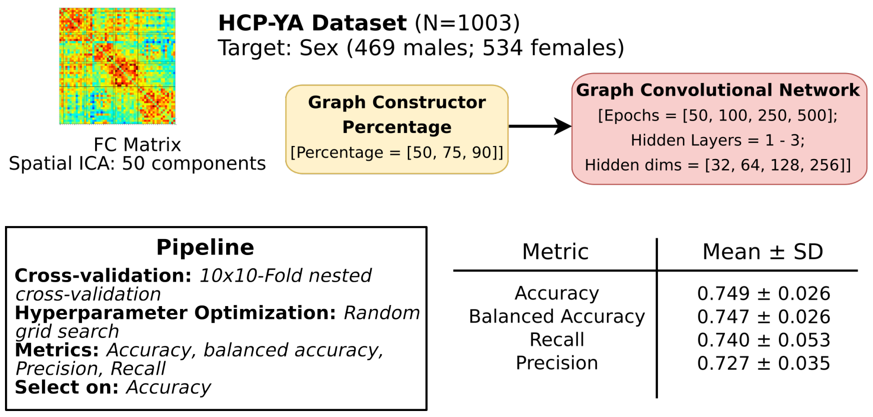
| Metric | Test Mean ± Standard Deviation |
|---|---|
| Pearson Correlation | 0.361 ± 0.068 |
| Mean Absolute Error | 16.584 ± 0.814 |
| Variance Explained | 0.135 ± 0.049 |

**Table 3.** Predictive performance for grip strength prediction of the best pipeline configuration trained on the HCP-YA dataset.

## Discussion

The previous sections showcase how our toolbox can be applied to different types of imaging-derived connectivity data to perform graph machine learning within a few lines of code. Having established the utility of the tool, we discuss the utility and value of this toolbox within the framework of existing toolboxes for different types of graph analyses.

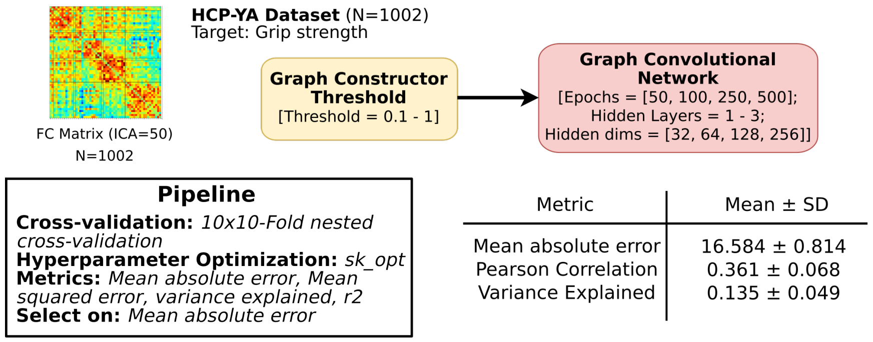
**A new toolbox for graph machine learning.** Graph machine learning is an important subfield of machine learning, due to the common occurrence of graph data [35]. Yet, despite their ubiquitous occurrence, machine learning that leverages the information by graph structures has only recently come into focus. While packages for specific types of graph machine learning exist (e.g. gem for graph embeddings), no package makes it possible to use these various approaches in an easy end-to-end framework. This is where *PHOTONAI-Graph* fills a gap.
Through integration with *PHOTONAI* the *PHOTONAI-Graph* toolbox allows scientists to set up graph complicated machine learning pipelines with minimal coding expertise, providing access to automated pipeline selection, best practices such as nested cross-validation, advanced hyperparam-


HCP-YA Dataset (N=1003)
Target: Sex (469 males; 534 females)
FC Matrix
Spatial ICA: 50 components
Graph Constructor Percentage
[Percentage = [50, 75, 90]]
Graph Convolutional Network
[Epochs = [50, 100, 250, 500];
Hidden Layers = 1 - 3;
Hidden dims = [32, 64, 128, 256]]

**Pipeline**
**Cross-validation:** *10x10-Fold nested cross-validation*
**Hyperparameter Optimization:** *Random grid search*
**Metrics:** *Accuracy, balanced accuracy, Precision, Recall*
**Select on:** *Accuracy*

| Metric | Mean ± SD |
|---|---|
| Accuracy | 0.749 ± 0.026 |
| Balanced Accuracy | 0.747 ± 0.026 |
| Recall | 0.740 ± 0.053 |
| Precision | 0.727 ± 0.035 |

**Fig. 3.** Predicting sex from fMRI. We build a sex classification model on the HCP-YA dataset (N=1003; 469 males, 534 females) using functional connectivity matrices based on an spatial ICA with 50 components to showcase an application with functional connectivity data. The pipeline contained a percentage constructor and a Graph Convolutional Neural Net Classifier. The best model was selected based on accuracy within a 10x10 K-fold nested cross-validation using a bayesian hyperparameter optimization strategy. The best model classified sex with an accuracy of 0.74, balanced accuracy of 0.74, precision of 0.72 and recall of 0.74 with percentage threshold of 90 and a Graph Convolutional Classifier with 50 training epochs, 2 hidden layers, and a hidden layer size of 256.


HCP-YA Dataset (N=1002)
Target: Grip strength
FC Matrix (ICA=50)
N=1002
Graph Constructor Threshold
[Threshold = 0.1 - 1]
Graph Convolutional Network
[Epochs = [50, 100, 250, 500];
Hidden Layers = 1 - 3;
Hidden dims = [32, 64, 128, 256]]

**Pipeline**
**Cross-validation:** *10x10-Fold nested cross-validation*
**Hyperparameter Optimization:** *sk_opt*
**Metrics:** *Mean absolute error, Mean squared error, variance explained, r2*
**Select on:** *Mean absolute error*

| Metric | Mean ± SD |
|---|---|
| Mean absolute error | 16.584 ± 0.814 |
| Pearson Correlation | 0.361 ± 0.068 |
| Variance Explained | 0.135 ± 0.049 |

**Fig. 4.** Predicting grip strength from fMRI. We built a grip strength regression model on the HCP-YA dataset (N=1002) using functional connectivity matrices derived using a spatial ICA with 50 components to showcase an application to regression data. The pipeline contained a threshold constructor and a Graph Convolutional Network Classifier. The best model was selected on mean absolute error with a bayesian hyperparameter optimization strategy within a 10x10 K-Fold cross-validation. The best model predicted grip strength with a mean absolute error of 15.58, a pearson correlation of 0.36 and a variance explained of 0.13 using a threshold value of 0.877 and a Graph Convolutional Net Regressor with 250 training epochs, hidden layers and a hidden layer size of 32.

eter optimization strategies and model visualization. This allows researchers without strong coding expertise to use graph machine learning with correct algorithm evaluation. Furthermore, it significantly decreases development time, allowing for rapid prototyping while safeguarding against conceptual errors such as information leakage between, training, validation and test folds.

**Software for graph machine learning.** Major fields in graph machine learning are graph embeddings, graph kernels and graph neural networks [13, 36]. Network controllability and graph analysis are not part of classical graph machine learning but can be used to reduce dimensionality while preserving graph information. Different packages exist for these approaches, many of which have been integrated into the *PHOTONAI-Graph* toolbox.

The gem package is a *python* package that implements graph embeddings. It covers static graph embeddings and dynamic graph embeddings are currently under development [21]. It is to our knowledge the only package that specifically covers graph embeddings in *python*. By integrating *gem* into the *PHOTONAI-Graph* toolbox, we provide users with easy access to graph embedding techniques and make these embeddings available for *scikit-learn*-based machine learning pipelines.

For graph kernels, the *grakel* package has been developed as a *python* package that implements 18 different graph kernels as *scikit-learn*-compatible transformer classes [22]. While the *scikit-learn*-based API *grakel* kernels can easily be used in machine learning pipelines, *grakel* requires a specific *grakel* graph object, which requires additional coding for format conversion. Integrating *grakel* into our toolbox allows for automatic graph conversion and hyperparameter optimization as part of machine learning pipelines, which lowers the entry bar for graph kernel-based analyses. Alternative packages for graph kernels such as *graphkernels* [37] or *graphkit-learn* [38] were not selected for integration as *grakel* provides the largest amount of available graph kernels with an active community and regular package maintenance.

For graph neural networks different packages exist, mostly written in *python*. Among the most widely used packages are *stellargraph*, *spektral*, *Pytorch Geometric*, *graphnets* and *dgl* [23, 39–41]. All of these packages provide frameworks for the creation of graph neural networks, with different neural network dependencies. Of these, *dgl* provides the highest amount of flexibility as it is backend-agnostic and supports *PyTorch*, *TensorFlow* and *MxNet* [23, 29, 42, 43]. It is regularly updated and supported by an active developer community. *dgl* models in *PHOTONAI-Graph* use a *pytorch* backend. As graph neural networks rapidly develop, *dgl* allows for an easy switch to another backend in the future.

Next to graph machine learning, classical graph analysis has gained increasing popularity in the area of neuroimaging and this is reflected by a range of toolboxes that offer the extraction of different graph measures from DTI and rs-fMRI data. Notable examples include GAT, Brain Connectivity Toolbox, GRETNA, Dynamic Graph Metrics and NeuroPycon [3, 10, 44–46]. All these toolboxes implement the extraction of graph measures from brain connectivity networks and often provide full pipelines for brain connectivity analyses. These toolboxes however are focused on brain connectivity and are not written to be applied to other forms of graph data. For this reason, we chose the *networkx* and *igraph python* package for our graph *GraphMeasureTransform* module which is agnostic to the data source as long as graphs can be transformed into the *networkx* or *igraph* graph format. This allows us to adapt the toolbox to other types of graph data, outside of neuroimaging. Providing both graph measure transformations, users are able to choose between *networkx* for breadth and *igraph* for speed when needed.

Network control theory has recently been introduced to brain connectivity networks [30]. As network controllability measures relate to psychiatric disease we have included two established measures of brain controllability in our controllability module [47–49]. Here we use our own implementation of average and modal controllability based on Tang et al [50]. As research in this area expands and network controllability toolboxes emerge, we plan to include these as part of *PHOTONAI-Graph*.

**Software for graph machine learning in neuroimaging.** In the area of neuroimaging, a python package that incorporates existing neuroimaging toolboxes into a unified framework is the NeuroPyCon package [46]. NeuropyCon combines various toolboxes that allow connectivity-based analyses and incorporates them into a single python framework, which is an extension of NiPype [51]. It includes shareable parameters to facilitate reproduction in neuroimaging pipelines and can work with source data from fMRI, MEG and EEG. It specializes in graph analysis using the tools of existing toolboxes like CONN or Network-Based Statistics (NBS) but does not cover graph machine learning. As our toolbox does not focus on deriving graph measures but on graph machine learning it does not stand in competition with existing graph analysis toolboxes for neuroimaging.

A toolbox that specializes in graph-specific machine learning is *GraphVar 2.0*. It is a neuroimaging-specific toolbox, that addresses machine learning based on graphs using graph measures. It is based on MATLAB and is extended to SPM, to allow easy integration with existing machine learning tools. Like *PHOTONAI-Graph*, it allows for nested cross-validation, hyperparameter optimization, model selection and evaluation [12]. It does not support model sharing however and machine learning methods are limited to extracting graph measures in order to perform classical machine learning analyses. Furthermore, it is not agnostic to the type of input data, as it only supports brain connectivity analyses. Here *PHOTONAI-Graph* widely expands the space of usable algorithms, covering state-of-the-art graph machine learning techniques from multiple subfields.

**Limitations and future developments.** One key limitation of *PHOTONAI-Graph* is that despite strongly increasing accessibility of state-of-the-art graph machine learning techniques it still requires a certain degree of technical knowledge for preprocessing and preparation of the input data. It is not constructed as a front-to-end pipeline tool, which means that technical knowledge is still required for the construction of connectivity matrices. This can be done using established tools such as CONN, NeuroPycon or CATO which sometimes offer GUI support [6, 46, 52].

It also requires basic familiarity with *numpy* array notation, for input data to have the required form: Individuals x nodes x nodes x matrix types (adjacency, features). GUI support is currently not available and basic coding skills are required for setting up pipelines. We plan to introduce *PHOTONAI-Graph* to the PHOTONAI Wizard (https://photon-ai.com/wizard) in the near future to allow users to construct *PHOTONAI-Graph* pipelines using a GUI. Another limitation is the focus on brain connectivity matrices. While it is possible to perform other types of graph analyses with *PHOTONAI-Graph*, the input data requires conversion into an appropriate format. This requires a higher degree of technical knowledge. For future updates, we plan to expand the number of supported data structures, expand import and conversion functions, include sparse support and develop an internal data structure that can accommodate large and sparse graph structures.

Furthermore, we plan to include support node classification and link prediction pipelines in the future. These are two vital areas of graph machine learning, which can also be applied to brain connectivity graphs. While most applications of graph machine learning to brain connectivity focus on whole graph prediction, the application of methods to brain graphs could enable researchers to answer new questions about brain connectivity. We plan to introduce these changes along with sparse support for giant single graphs.

**Conclusion.** Our toolbox incorporates various graph machine learning packages and makes their algorithms available via a *scikit-learn*-compatible API that could be used with the *PHOTONAI* package or outside of it. It offers a unified framework for graph machine learning, without the need for manually converting graph data from one format to the other. This eases model building and hyperparameter optimization, allowing researchers from different biomedical research fields to use graph machine learning for their own analyses. Furthermore, the toolbox delivers a set of functions that can be specifically used with connectivity matrices, such as the graph constructor classes. They include established and novel approaches geared toward machine learning on neuroimaging data [53]. Through integration with the *PHOTONAI* environment, it also offers the functionality provided by the *PHOTONAI* toolbox, like model sharing, pipeline evaluation and model interpretation.

In conclusion we present a novel toolbox that allows neuroimaging researchers to construct graph machine learning pipelines within a few lines of code. With this toolbox, researchers can now easily apply graph-based machine learning methods to their data and extract valuable insights that were previously difficult to access. We hope this will further the utilization of graph machine learning in neuroimaging.

**ACKNOWLEDGEMENTS**

Data were provided in part by the Human Connectome Project, MGH-USC Consortium (Principal Investigators: Bruce R. Rosen, Arthur W. Toga and Van Wedeen; U01MH093765) funded by the NIH Blueprint Initiative for Neuroscience Research grant; the National Institutes of Health grant P41EB015896; and the Instrumentation Grants S10RR023043, 1S10RR023401, 1S10RR019307.

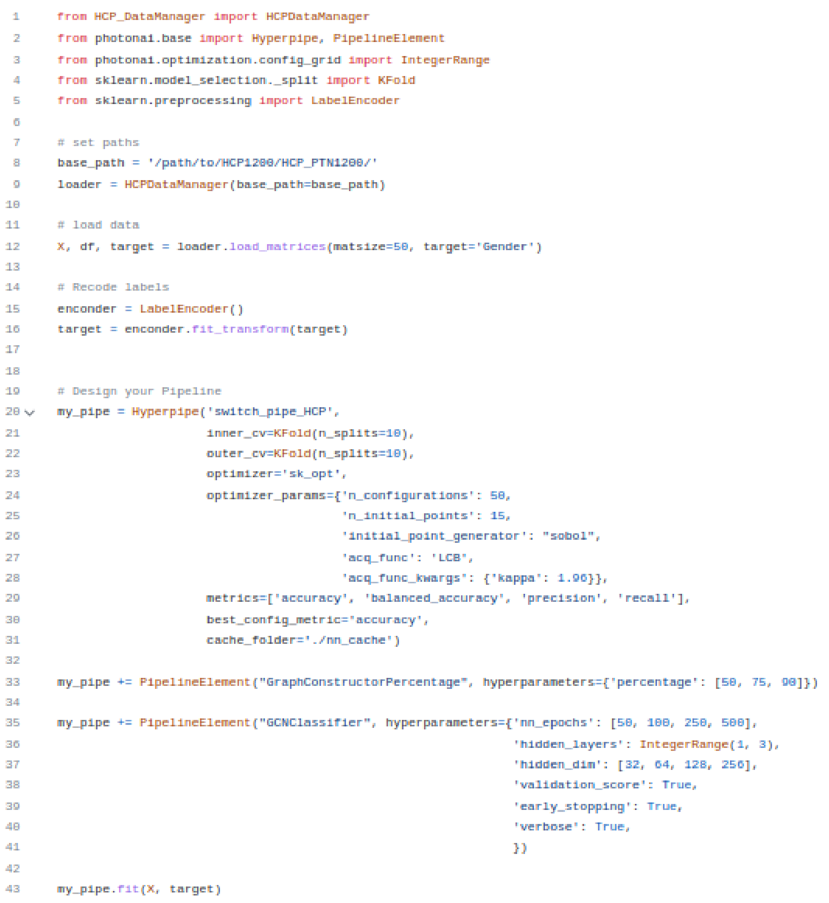
## Bibliography

1. Ed Bullmore and Olaf Sporns. Complex brain networks: Graph theoretical analysis of structural and functional systems, 2009. ISSN 1471003X.
2. Alex Fornito, Andrew Zalesky, and Michael Breakspear. Graph analysis of the human connectome: Promise, progress, and pitfalls. *NeuroImage*, 80, 2013. ISSN 10538119. doi: 10.1016/j.neuroimage.2013.04.087.
3. Mikail Rubinov and Olaf Sporns. Complex network measures of brain connectivity: Uses and interpretations. *NeuroImage*, 52(3), 2010. ISSN 10538119. doi: 10.1016/j.neuroimage.2009.10.003.
4. Andrew Zalesky, Alex Fornito, and Edward T. Bullmore. Network-based statistic: Identifying differences in brain networks. *NeuroImage*, 53(4), 2010. ISSN 10538119. doi: 10.1016/j.neuroimage.2010.06.041.
5. Bin He, Yakang Dai, Laura Astolfi, Fabio Babiloni, Han Yuan, and Lin Yang. EConnectome: A MATLAB toolbox for mapping and imaging of brain functional connectivity. *Journal of Neuroscience Methods*, 195(2), 2011. ISSN 01650270. doi: 10.1016/j.jneumeth.2010.11.015.
6. Susan Whitfield-Gabrieli and Alfonso Nieto-Castanon. Conn: A Functional Connectivity Toolbox for Correlated and Anticorrelated Brain Networks. *Brain Connectivity*, 2(3), 2012. ISSN 21580022. doi: 10.1089/brain.2012.0073.
7. S. M.Hadi Hosseini, Fumiko Hoeft, and Shelli R. Kesler. Gat: A graph-theoretical analysis toolbox for analyzing between-group differences in large-scale structural and functional brain networks. *PLoS ONE*, 7(7), 2012. ISSN 19326203. doi: 10.1371/journal.pone.0040709.
8. Jeffrey M. Spielberg, Regina E. McGlinchey, William P. Milberg, and David H. Salat. Brain network disturbance related to posttraumatic stress and traumatic brain injury in veterans. *Biological Psychiatry*, 78(3), 2015. ISSN 18732402. doi: 10.1016/j.biopsych.2015.02.013.
9. Martin Göttlich, Frederike Beyer, and Ulrike M. Krämer. Basco: A toolbox for task-related functional connectivity. *Frontiers in Systems Neuroscience*, 9(September), 2015. ISSN 16625137. doi: 10.3389/fnsys.2015.00126.
10. Jinhui Wang, Xindi Wang, Mingrui Xia, Xuhong Liao, Alan Evans, and Yong He. GRETNA: A graph theoretical network analysis toolbox for imaging connectomics. *Frontiers in Human Neuroscience*, 9(JUNE), 2015. ISSN 16625161. doi: 10.3389/fnhum.2015.00386.
11. Mite Mijalkov, Ehsan Kakaei, Joana B. Pereira, Eric Westman, and Giovanni Volpe. BRAPH: A graph theory software for the analysis of brain connectivity. *PLoS ONE*, 12(8), 2017. ISSN 19326203. doi: 10.1371/journal.pone.0178798.
12. L. Waller, A. Brovkin, L. Dorfschmidt, D. Bzdok, H. Walter, and J. D. Kruschwitz. GraphVar 2.0: A user-friendly toolbox for machine learning on functional connectivity measures. *Journal of Neuroscience Methods*, 308:21–33, 10 2018. ISSN 1872678X. doi: 10.1016/j.jneumeth.2018.07.001.
13. William L Hamilton. Graph Representation Learning. Technical Report 3, 2020.
14. Hongyun Cai, Vincent W. Zheng, and Kevin Chen Chuan Chang. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9), 2018. ISSN 15582191. doi: 10.1109/TKDE.2018.2807452.
15. Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels, 2020. ISSN 23648228.
16. Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep Learning on Graphs: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020. ISSN 1041-4347. doi: 10.1109/tkde.2020.2981333.
17. Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020. ISSN 2162-237X. doi: 10.1109/tnnls.2020.2978386.
18. Byung Hoon Kim and Jong Chul Ye. Understanding Graph Isomorphism Network for rs-fMRI Functional Connectivity Analysis. *Frontiers in Neuroscience*, 14, 2020. ISSN 1662453X. doi: 10.3389/fnins.2020.00630.
19. Tzu An Song, Samadrita Roy Chowdhury, Fan Yang, Heidi Jacobs, Georges El Fakhri, Quanzheng Li, Keith Johnson, and Joyita Dutta. Graph convolutional neural networks for Alzheimer's disease classification. In *Proceedings - International Symposium on Biomedical Imaging*, volume 2019-April, 2019. doi: 10.1109/ISBI.2019.8759531.
20. Palash Goyal and Emilio Ferrara. Graph Embedding Techniques, Applications, and Performance: A Survey. 5 2017. doi: 10.1016/j.knosys.2018.03.022.
21. Palash Goyal and Emilio Ferrara. GEM: A Python package for graph embedding methods. *Journal of Open Source Software*, 3(29), 2018. ISSN 2475-9066. doi: 10.21105/joss.00876.
22. Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. Grakel: A graph kernel library in python. *Journal of Machine Learning Research*, 21, 2020. ISSN 15337928.
23. Da Zheng, Minjie Wang, Quan Gan, Zheng Zhang, and Geroge Karypis. Scalable Graph Neural Networks with Deep Graph Library. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020. doi: 10.1145/3394486.3406712.
24. Ramona Leenings, Nils Ralf Winter, Lucas Plagwitz, Vincent Holstein, Jan Ernsting, Jakob Steenweg, Julian Gebker, Kelvin Sarink, Daniel Emden, Dominik Grotegerd, Nils Opel, Benjamin Risse, Xiaoyi Jiang, Udo Dannlowski, and Tim Hahn. PHOTON – A Python API for Rapid Machine Learning Model Development. 2 2020.
25. Sofia Ira Ktena, Sarah Parisot, Enzo Ferrante, Martin Rajchl, Matthew Lee, Ben Glocker, and Daniel Rueckert. Distance metric learning using graph convolutional networks: Application to functional brain networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10433 LNCS, 2017. doi: 10.1007/978-3-319-66182-7{_}54.
26. A A Hagberg, D A Schult, and P J Swart. Exploring network structure, dynamics, and function using NetworkX. In *7th Python in Science Conference (SciPy 2008)*, 2008.
27. Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal Complex Systems*, Complex Sy(1695), 2006.
28. Casper O. da Costa-Luis. tqdm: A Fast, Extensible Progress Meter for Python and CLI. *Journal of Open Source Software*, 4(37), 2019. doi: 10.21105/joss.01277.
29. Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.
30. Shi Gu, Fabio Pasqualetti, Matthew Cieslak, Qawi K. Telesford, Alfred B. Yu, Ari E. Kahn, John D. Medaglia, Jean M. Vettel, Michael B. Miller, Scott T. Grafton, and Danielle S. Bassett. Controllability of structural brain networks. *Nature Communications*, 6, 2015. ISSN 20411723. doi: 10.1038/ncomms9414.
31. Shi Gu, Richard F. Betzel, Marcelo G. Mattar, Matthew Cieslak, Philip R. Delio, Scott T. Grafton, Fabio Pasqualetti, and Danielle S. Bassett. Optimal trajectories of brain state transitions. *NeuroImage*, 148, 2017. ISSN 10959572. doi: 10.1016/j.neuroimage.2017.01.003.
32. Teresa M. Karrer, Jason Z. Kim, Jennifer Stiso, Ari E. Kahn, Fabio Pasqualetti, Ute Habel, and Danielle S. Bassett. A practical guide to methodological considerations in the controllability of structural brain networks. *Journal of Neural Engineering*, 17(2), 2020. ISSN 17412552. doi: 10.1088/1741-2552/ab6e8b.
33. Martijn P. Van Den Heuvel, Lianne H. Scholtens, Hannelore K. Van Der Burgh, Federica Agosta, Clara Alloza, Celso Arango, Bonnie Auyeung, Simon Baron-Cohen, Silvia Basaia, Manon J.N.L. Benders, Frauke Beyer, Linda Booij, Kees P.J. Braun, Geraldo Busatto Filho, Wiepke Cahn, Dara M. Cannon, Tiffany M. Chaim-Avancini, Sandra S.M. Chan, Eric Y.H. Chen, Benedicto Crespo-Facorro, Eveline A. Crone, Udo Dannlowski, Sonja M.C. De Zwarte, Bruno Dietsche, Gary Donohoe, Stefan Du Plessis, Sarah Durston, Covadonga M. Díaz-Caneja, Ana M. Díaz-Zuluaga, Robin Emsley, Massimo Filippi, Thomas Frodl, Martin Gorges, Beata Graff, Dominik Grotegerd, Dariusz Gąsecki, Julie M. Hall, Laurena Holleran, Rosemary Holt, Helene J. Hopman, Andreas Jansen, Joost Janssen, Krzysztof Jodzio, Lutz Jäncke, Vasiliy G. Kaleda, Jan Kassubek, Shahrzad Kharabian Masouleh, Tilo Kircher, Martijn G.J.C. Koevoets, Vladimir S. Kostic, Axel Krug, Stephen M. Lawrie, Irina S. Lebedeva, Edwin H.M. Lee, Tristram A. Lett, Simon J.G. Lewis, Franziskus Liem, Michael V. Lombardo, Carlos Lopez-Jaramillo, Daniel S. Margulies, Sebastian Markett, Paulo Marques, Ignacio Martínez-Zalacaín, Colm McDonald, Andrew M. McIntosh, Genevieve McPhilemy, Susanne L. Meinert, José M. Menchón, Christian Montag, Pedro S. Moreira, Pedro Morgado, David O. Mothersill, Susan Mérillat, Hans Peter Müller, Leila Nabulsi, Pablo Najt, Krzysztof Narkiewicz, Patrycja Naumczyk, Bob Oranje, Victor Ortiz Garcia De la Foz, Jiska S. Peper, Julian A. Pineda, Paul E. Rasser, Ronny Redlich, Jonathan Repple, Martin Reuter, Pedro G.P. Rosa, Amber N.V. Ruigrok, Agnieszka Sabisz, Ulrich Schall, Soraya Seedat, Mauricio H. Serpa, Stavros Skouras, Carles Soriano-Mas, Nuno Sousa, Edyta Szurowska, Alexander S. Tomyshev, Diana Tordesillas-Gutierrez, Sofie L. Valk, Leonard H. Van Den Berg, Theo G.M. Van Erp, Neeltje E.M. Van Haren, Judith M.C. Van Leeuwen, Arno Villringer, Christiaan H. Vinkers, Christian Vollmar, Lea Waller, Henrik Walter, Heather C. Whalley, Marta Witkowska, A. Veronica Witte, Marcus V. Zanetti, Rui Zhang, and Siemon C. De Lange. 10kin1day: A bottom-up neuroimaging initiative, 2019. ISSN 16642295.
34. David C. Van Essen, Stephen M. Smith, Deanna M. Barch, Timothy E.J. Behrens, Essa Yacoub, and Kamil Ugurbil. The WU-Minn Human Connectome Project: An overview. *NeuroImage*, 80, 2013. ISSN 10538119. doi: 10.1016/j.neuroimage.2013.05.041.
35. Georgios A. Pavlopoulos, Maria Secrier, Charalampos N. Moschopoulos, Theodoros G. Soldatos, Sophia Kossida, Jan Aerts, Reinhard Schneider, and Pantelis G. Bagos. Using graph theory to analyze biological networks, 2011. ISSN 17560381.
36. William L Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. Technical report, 2017.
37. Mahito Sugiyama, M. Elisabetta Ghisu, Felipe Llinares-López, and Karsten Borgwardt. Graphkernels: R and Python packages for graph comparison. *Bioinformatics*, 34(3), 2018. ISSN 14602059. doi: 10.1093/bioinformatics/btx602.
38. Linlin Jia, Benoit Gaüzère, and Paul Honeine. graphkit-learn: A Python library for graph kernels based on linear patterns. *Pattern Recognition Letters*, 143, 2021. ISSN 01678655. doi: 10.1016/j.patrec.2021.01.003.
39. Daniele Grattarola and Cesare Alippi. Graph Neural Networks in TensorFlow and Keras with Spektral. 6 2020.
40. Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric. 3 2019.
41. Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess,

Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. 6 2018.

42. Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, 2016.
43. Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. 12 2015.
44. Einar A. Høgestøl, Tobias Kaufmann, Gro O. Nygaard, Mona K. Beyer, Piotr Sowa, Jan E. Nordvik, Knut Kolskår, Geneviève Richard, Ole A. Andreassen, Hanne F. Harbo, and Lars T. Westlye. Cross-sectional and longitudinal MRI brain scans reveal accelerated brain aging in multiple sclerosis. *Frontiers in Neurology*, 10(APR), 2019. ISSN 16642295. doi: 10.3389/fneur.2019.00450.
45. Ann E. Sizemore and Danielle S. Bassett. Dynamic graph metrics: Tutorial, toolbox, and tale, 10 2018. ISSN 10959572.
46. David Meunier, Annalisa Pascarella, Dmitrii Altukhov, Mainak Jas, Etienne Combrisson, Tarek Lajnef, Daphné Bertrand-Dubois, Vanessa Hadid, Golnoush Alamian, Jordan Alves, Fanny Barlaam, Anne Lise Saive, Arthur Dehgan, and Karim Jerbi. NeuroPycon: An open-source python toolbox for fast multi-modal and reproducible brain connectivity pipelines. *NeuroImage*, 219, 2020. ISSN 10959572. doi: 10.1016/j.neuroimage.2020.117020.
47. Tim Hahn, Hamidreza Jamalabadi, Daniel Emden, Janik Goltermann, Jan Ernsting, Nils R Winter, Lukas Fisch, Ramona Leenings, Kelvin Sarink, Vincent Holstein, Marius Gruber, Dominik Grotegerd, Susanne Meinert, Katharina Dohm, Elisabeth J Leehr, Maike Richter, Lisa Sindermann, Verena Enneking, Hannah Lemke, Stephanie Witt, Marcella Rietschel, Katharina Brosch, Julia-Katharina Pfarr, Tina Meller, Kai Gustav Ringwald, Simon Schmitt, Frederike Stein, Igor Nenadic, Tilo Kircher, Bertram Müller-Myhsok, Till FM Andlauer, Jonathan Repple, Udo Dannlowski, and Nils Opel. A Network Control Theory Approach to Longitudinal Symptom Dynamics in Major Depressive Disorder. Technical report, 2021.
48. Tim Hahn, Hamidreza Jamalabadi, Erfan Nozari, Nils R Winter, Jan Ernsting, Marius Gruber, Marco J Mauritz, Pascal Grumbach, Lukas Fisch, Ramona Leenings, Kelvin Sarink, Julian Blanke, Leon Kleine Vennekate, Daniel Emden, Nils Opel, Dominik Grotegerd, Verena Enneking, Susanne Meinert, Tiana Borgers, Melissa Klug, Elisabeth J Leehr, Katharina Dohm, Walter Heindel, Joachim Gross, Udo Dannlowski, Ronny Redlich, and Jonathan Repple. Towards a network control theory of electroconvulsive therapy response. *PNAS Nexus*, 2(2), 2023. doi: 10.1093/pnasnexus/pgad032.
49. Tim Hahn, Nils R. Winter, Jan Ernsting, Marius Gruber, Marco J. Mauritz, Lukas Fisch, Ramona Leenings, Kelvin Sarink, Julian Blanke, Vincent Holstein, Daniel Emden, Marie Beisemann, Nils Opel, Dominik Grotegerd, Susanne Meinert, Walter Heindel, Stephanie Witt, Marcella Rietschel, Markus M. Nöthen, Andreas J. Forstner, Tilo Kircher, Igor Nenadic, Andreas Jansen, Bertram Müller-Myhsok, Till F.M. Andlauer, Martin Walter, Martijn P. van den Heuvel, Hamidreza Jamalabadi, Udo Dannlowski, and Jonathan Repple. Genetic, individual, and familial risk correlates of brain network controllability in major depressive disorder. *Molecular Psychiatry*, 28(3), 2023. ISSN 14765578. doi: 10.1038/s41380-022-01936-6.
50. Evelyn Tang, Chad Giusti, Graham L. Baum, Shi Gu, Eli Pollock, Ari E. Kahn, David R. Roalf, Tyler M. Moore, Kosha Ruparel, Ruben C. Gur, Raquel E. Gur, Theodore D. Satterthwaite, and Danielle S. Bassett. Developmental increases in white matter network controllability support a growing diversity of brain dynamics. *Nature Communications*, 8(1), 2017. ISSN 20411723. doi: 10.1038/s41467-017-01254-4.
51. Krzysztof Gorgolewski, Christopher D. Burns, Cindee Madison, Dav Clark, Yaroslav O. Halchenko, Michael L. Waskom, and Satrajit S. Ghosh. Nipype: A flexible, lightweight and extensible neuroimaging data processing framework in Python. *Frontiers in Neuroinformatics*, 5, 2011. ISSN 16625196. doi: 10.3389/fninf.2011.00013.
52. Siemon C. de Lange, Koen Helwegen, and Martijn P. van den Heuvel. Structural and functional connectivity reconstruction with CATO - A Connectivity Analysis TOolbox. *NeuroImage*, 273:120108, 6 2023. ISSN 10538119. doi: 10.1016/j.neuroimage.2023.120108.
53. Guixiang Ma, Dipanjan Sengupta, Nesreen K. Ahmed, Michael W. Cole, Philip S. Yu, Theodore L. Willke, and Nicholas B. Turk-Browne. Deep graph similarity learning for brain data analysis. In *International Conference on Information and Knowledge Management, Proceedings*, 2019. doi: 10.1145/3357384.3357815.

# Supplementary Material



```
1     from HCP_DataManager import HCPDataManager
2     from photonai.base import Hyperpipe, PipelineElement
3     from photonai.optimization.config_grid import IntegerRange
4     from sklearn.model_selection._split import KFold
5     from sklearn.preprocessing import LabelEncoder
6
7     # set paths
8     base_path = '/path/to/HCP1200/HCP_PTN1200/'
9     loader = HCPDataManager(base_path=base_path)
10
11    # load data
12    X, df, target = loader.load_matrices(matsize=50, target='Gender')
13
14    # Recode labels
15    enconder = LabelEncoder()
16    target = enconder.fit_transform(target)
17
18
19    # Design your Pipeline
20    my_pipe = Hyperpipe('switch_pipe_HCP',
21                        inner_cv=KFold(n_splits=10),
22                        outer_cv=KFold(n_splits=10),
23                        optimizer='sk_opt',
24                        optimizer_params={'n_configurations': 50,
25                                          'n_initial_points': 15,
26                                          'initial_point_generator': "sobol",
27                                          'acq_func': 'LCB',
28                                          'acq_func_kwargs': {'kappa': 1.96}},
29                        metrics=['accuracy', 'balanced_accuracy', 'precision', 'recall'],
30                        best_config_metric='accuracy',
31                        cache_folder='./nn_cache')
32
33    my_pipe += PipelineElement("GraphConstructorPercentage", hyperparameters={'percentage': [50, 75, 90]})
34
35    my_pipe += PipelineElement("GCNClassifier", hyperparameters={'nn_epochs': [50, 100, 250, 500],
36                                                                 'hidden_layers': IntegerRange(1, 3),
37                                                                 'hidden_dim': [32, 64, 128, 256],
38                                                                 'validation_score': True,
39                                                                 'early_stopping': True,
40                                                                 'verbose': True,
41                                                                 })
42
43    my_pipe.fit(X, target)
```

**Fig. 5.** Prediction Script. The HCP-YA Sex Prediction pipeline can be built in 43 lines of code. Line 19-43 define and fit the pipeline, meaning that the entire pipeline optimization strategy, optimizer parameters, validation metrics, Graph Constructor, Graph Neural Network and pipeline fitting can be defined within 24 lines of code. This significantly decreases development time and increases accessibility.