Aperture Neuro

Doshvargar F, Cravo F, Shearer H, Noble S. Mass univariate aggregation methods for machine learning in neuroscience. *Aperture Neuro*. 2026;6. doi:10.52294/001c.165448

Articles Describing Code

Mass univariate aggregation methods for machine learning in neuroscience

Fatemeh Doshvargar, BSc[1,2a], Fabricio Cravo, PhD[2,3], Hallee Shearer, MS[2,3], Stephanie Noble, PhD[1,2,3,4]

[1] Department of Bioengineering, Northeastern University, Boston, MA, USA, [2] Institute for Cognitive and Brain Health, Northeastern University, Boston, MA, USA, [3] Department of Psychology, Northeastern University, Boston, MA, USA, [4] Department of Radiology & Biomedical Imaging, Yale University, New Haven, CT, USA

Keywords: functional MRI, interpretable machine learning, connectome-based predictive modeling, polygenic risk score, scikit-learn, open-source software

https://doi.org/10.52294/001c.165448

Aperture Neuro

Vol. 6, 2026

Machine learning is a ubiquitous part of the modern neuroimaging toolkit, particularly for research aimed towards precision medicine goals of improving individual-level diagnosis and treatment. However, the high dimensionality of neuroimaging data poses significant challenges for constructing interpretable predictive models. Several established methods, such as Connectome-based Predictive Modeling (CPM), Polyneuro Risk Scores (PNRS, inspired by Polygenic Risk Scores), and Polyconnectomic Scoring (PCS), offer an interpretable approach, which we term "Mass Univariate Aggregation" (MUA). MUA approaches evaluate each feature independently and then use a linear combination of weighted features to derive predictions, providing directly interpretable weights for individual features. Despite the existence and widespread usage of various MUA approaches in neuroimaging, tools for their application are still fragmented, and there does not yet exist an open-access unified tool for implementing, evaluating, or comparing these models within a standardized machine learning workflow. Here, we present a unified, flexible, and accessible configurable pipeline that can be used for implementing CPM, PNRS, PCS, and many new MUA configurations facilitated by user-specified parameters. Built in Python as an add-on for scikit-learn, our configurable pipeline enables researchers to leverage the functionality and standards provided by a widely used open-access machine learning tool. We validated the configurable pipeline by replicating the outcomes achieved by existing CPM and PNRS implementations, utilizing resting-state functional connectivity data from the Human Connectome Project to predict fluid intelligence (n = 1067). We further validated the pipeline's PCS implementation, confirming PCS computation with external connectome summary statistics (CSS) matrices using the same data, and CSS derivation using simulated data. Although designed to fill a gap in neuroimaging, our open-source, configurable pipeline provides a standardized platform for applying the MUA methods to any machine learning setting that features high-dimensional data.

INTRODUCTION

Machine learning serves as a crucial tool in neuroscience for enhancing our understanding of the human brain. The application of machine learning to neuroimaging data enables researchers to extract meaningful information from brain data and build predictive models that capture individual differences in brain-behavior relationships.[1,2] However, the high dimensionality of neuroimaging data, where the number of features often vastly outnumbers observations, presents significant analytical challenges.[1,3] Multiple machine learning methods that involve mass univariate regression followed by aggregation, which we term Mass Univariate Aggregation (MUA), have been introduced and demonstrated to be effective in high-dimensional data. Notably, because these methods evaluate each feature inde-

a **Corresponding Author:**
Fatemeh Doshvargar, BSc (fatemehdoshvargar1380@gmail.com)
Department of Bioengineering, Northeastern University
Institute for Cognitive and Brain Health, Northeastern University
Boston, MA, USA

pendently and yield an explicit, quantifiable measure of its contribution to the final score, they constitute a form of interpretable predictive modeling that enables direct identification of the brain connections driving a given prediction.[2,4]

MUA methods represent a class of approaches that perform individual statistical tests on each feature separately (mass univariate), then combine weighted features into a unified predictive model (aggregation).[5-11] This philosophy underlies several successful methods across different fields. As an example, Polygenic Risk Scores (PRS), originally developed in genetics, aggregate the effects of individual genetic variants to predict complex traits and disease risk.[10-12] More recently, this approach was adapted from genetics to neuroimaging, introducing Polyconnectomic Scoring (PCS),[8,9] which generates an individual-level score reflecting the brain's alignment with disorder- or behavior-related connectivity patterns. PCS computes this score using a previously derived connectome–symptom association matrix (referred to as the connectome summary statistics (CSS) matrix), which encodes the association between each brain connection and a specific disorder or behavioral phenotype, enabling assessment of new individuals' connectivity patterns. Similarly, the Polyneuro Risk Scores (PNRS)[7] utilizes a weighted aggregation of brain features to assess the generalizability of brain-behavior associations across independent samples; these aggregate scores can also be used in a final regression step, as demonstrated in recent applications of PNRS to ADHD symptom prediction.[13] Additionally, Connectome-based Predictive Modeling (CPM)[5,6] is the most widely used MUA method in neuroimaging, employing binary feature selection based on statistical association, followed by sign-based aggregation of selected brain features and a final regression for behavioral prediction.

Although the core of the MUA algorithms is similar, their implementation remains fragmented across different programming languages and software platforms. CPM[6] is primarily implemented in MATLAB with code from Yale,[14] and a Python tutorial is also available from Finn,[15] while PNRS[7] is implemented in MATLAB with code from the Developmental Cognition and Neuroimaging (DCAN) lab at the University of Minnesota.[16] PCS,[8,9] meanwhile, is available in MATLAB, Python, and R through the Dutch Connectome Lab.[17] Moreover, the standard implementation for calculating PRS in genetics is provided by PRSice-2,[10,11] which is a C++ based software.

Here, we present a unified, open-source, and configurable pipeline that consolidates established mass univariate methods into a single platform compatible with modern machine learning workflows in Python. We validated the framework by reproducing results from previously published pipelines using the Human Connectome Project (HCP) dataset.[18] Built within the scikit-learn architecture,[19] this fully parametrized pipeline enables researchers to apply MUA approaches, including CPM,[6] PNRS,[7] and PCS[8,9] with support for both externally and internally derived CSS matrices, while leveraging the comprehensive scikit-learn ecosystem. Users can access the full range of

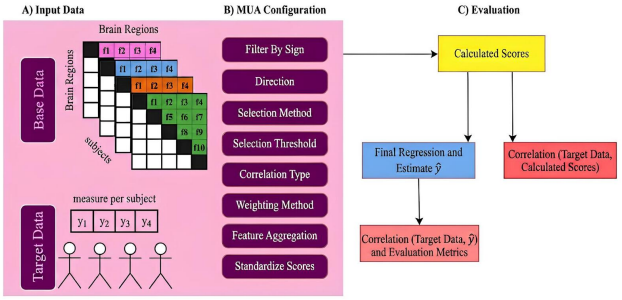

Figure 1. Overview of the unified configurable pipeline inputs and outputs for applying mass univariate aggregation methods. **(A)** Input data for the pipeline includes a matrix of features (n subjects × m predictors; e.g., edges from functional connectivity matrices) and a vector of observations (n subjects; e.g., behavioral measures). **(B)** The key parameters of the MUA class that are used for setting up the algorithm (for full descriptions of each parameter's options, see Table 1). **(C)** The calculated scores can be evaluated in two ways: directly, by correlating scores with behavioral measures (cor(y, S)), or by adding a regression step to produce predictions and evaluating with correlation (cor(y, ŷ)) and diverse evaluation metrics (e.g., mean squared error).

cross-validation strategies, evaluation metrics, hyperparameter optimization tools, various regression methods, and pipeline variations that enable developers to extend and customize the framework according to their specific research needs.

By providing a unified platform for mass univariate aggregation methods, we aim to accelerate methodological development, reduce programming errors for researchers, and enhance reproducibility in brain-behavior prediction research.

SECTION 1. A CONFIGURABLE PIPELINE FOR MASS UNIVARIATE AGGREGATION METHODS

UNIFIED CONFIGURABLE PIPELINE OVERVIEW

We developed a unified, configurable pipeline that integrates established methods, such as CPM[6] and PNRS,[7] alongside novel analytical approaches through a single interface (Figure 1). The pipeline is built around two core classes—FeatureVectorizer and MUA—both fully compatible with the scikit-learn ecosystem. These classes inherit from BaseEstimator and TransformerMixin, implementing the standard transformer interface (fit and transform) to ensure interoperability with Python's machine learning infrastructure.

This modular architecture enables researchers to use standard scikit-learn features, including automated hyperparameter optimization via GridSearchCV and RandomizedSearchCV. The pipeline is model-agnostic and can incorporate any scikit-learn regression method. By adjusting parameters within this framework, researchers can repro-

Table 1. Core parameters of the MUA class. Each parameter controls a specific aspect of the feature selection, weighting, and aggregation workflow, enabling flexible configuration of mass univariate aggregation methods within the configurable pipeline.

Parameter	Value	Description
filter_by_sign	True	Separate positive/negative features
	False	Keep all features together
direction	'difference'	Positive network score minus negative network score as a single predictor
	'positive'	Positive network score only
	'negative'	Negative network score only
	ignored	When filter_by_sign=False
selection_method	'all'	Use all features
	'pvalue'	Select features with $p < \alpha$
	'top_k'	Select the top k features by absolute correlation
selection_threshold	float (0, 1)	'pvalue' method: p-value threshold
	positive integer	'top_k' method: number of features
	ignored	For 'all' method
weighting_method	'binary'	±1 based on correlation sign
	'correlation'	The strength of the correlation
	'squared_correlation'	r^2 preserving sign
	'regression'	Feature-specific regression coefficients
	'external'	Use externally provided edge weights (e.g., CSS)
external_weights	array-like (n_features,)	External edge-wise weights applied when weighting_method='external'
correlation_type	'pearson'	Linear correlation
	'spearman'	Rank-based correlation
feature_aggregation	'sum'	The sum of the weighted features
	'mean'	The mean of the weighted features
standardize_scores	True	Z-score normalize final scores
	False	Keep raw scores

duce existing methods, create hybrid approaches, or develop new strategies while leveraging scikit-learn's comprehensive suite of cross-validation protocols (e.g., KFold, GroupKFold) and evaluation metrics (e.g., mean squared error, Pearson correlation).

IMPLEMENTATION DETAILS

1. *FEATUREVECTORIZER CLASS*

The FeatureVectorizer is designed to standardize symmetric graph input data for passing through the MUA class. When handling 3D connectivity input—represented as n subjects × m regions × m regions—it efficiently vectorizes the data by extracting only the off-diagonal upper triangular elements. If the input is already in a 2D feature format, the vectorizer leaves the data unmodified. Furthermore, the class supports inverse transformation, enabling the reconstruction of full symmetric matrices from feature vectors for subsequent usage. By streamlining these data transformations, our configurable pipeline can be used to apply MUA methods on neuroimaging data (connectivity matrices) and any other feature-outcome data.

2. *MUA CLASS*

The MUA class is our main Python class that extends scikit-learn's BaseEstimator and TransformerMixin classes to ensure compatibility with standard machine learning pipelines; the MUA class operates through three steps:

I. Feature Selection: Determines which features are included based on their association with behavior (e.g., p-value thresholding, top-k selection, or all features).

II. Feature Weighting: Specifies how selected features are weighted (e.g., binary, correlation-based, or regression-derived weights).

III. Feature Aggregation: Defines how weighted features are combined into summary scores (e.g., sum or mean aggregation, with optional sign-based partitioning into positive and negative networks).

The MUA class employs a set of configurable parameters (Table 1) organized across the computational steps described above that enable flexible implementation of established methods while facilitating exploration of novel approaches.

The output of the MUA class is a 2D NumPy array of shape ($n_{samples}$, 1). When filter_by_sign is set to true, the class separately computes positive and negative network strengths and combines them according to the direction parameter — by default, as a difference score (aggregated positive network strength minus aggregated negative network strength), as popularized by the traditional CPM framework.[6] Additionally, when filter_by_sign is set to false, all selected edges are processed together without sign-based partitioning and collapsed into a single weighted score, facilitating the implementation of the PCS[8,9] and PNRS[7] methods.

3. *PREDICTION STRATEGY*

For methods such as PNRS,[7] the aggregated score is used without a final regression step to produce a score that is correlated with, but not necessarily an actual prediction of, the outcome variable. As such, researchers typically use these scores as a new variable and observe their correlations with outcomes of interest, rather than using them as estimates of the outcome variable itself. Additionally, some approaches, such as CPM,[6] include an additional step in which a final regression step is applied to the aggregated features to obtain a predicted value of the outcome variable. In both cases, after configuring the MUA class parameters, users can integrate any scikit-learn-compatible regression method directly into the pipeline via the 'regressor' step to estimate a final regression model using the aggregated features. The following section demonstrates the practical application of this approach (for further details, see *CPM Configuration via The Configurable Pipeline, PNRS Configuration via The Configurable Pipeline, and Using PNRS as a Predictor with a Final Regression Step*):

```
The_pipeline = Pipeline([

('vectorize', FeatureVectorizer()),

('mua', MUA(

...,

...,

)),

('regressor', LinearRegression()) # Linear regression

])
```

While LinearRegression() is used here as an example, the pipeline supports the integration of any alternative scikit-learn regression method, including custom implementations that follow the scikit-learn estimator interface.

4. *CROSS-VALIDATION STRATEGY*

For configurations that include a final regression step (e.g., CPM[6,14]), predictions and performance estimates can be obtained using scikit-learn's standard cross-validation utilities. Our pipeline supports any scikit-learn-compatible cross-validation splitter, including standard splitters (e.g., KFold) and group-aware splitters (e.g., GroupKFold). Note that the original CPM protocol employs k-fold cross-validation, which we demonstrate in Section 2 at the *CPM Configuration via The Configurable Pipeline* and use in Section 3 in our validation runs at the *CPM Validation*, but more generally, it is recommended to consider group-aware splitters whenever observations are non-independent — for example, in datasets with familial relationships such as HCP[18] — to prevent data leakage and inflated performance estimates.[20,21] Users can specify the dependency structure through the groups argument (e.g., family ID), as demonstrated below (for further details and a full example, see *Illustrative Application: Family-Aware Cross-Validation*):

```
# Load the family relationship data

family_data = ...

gkf = GroupKFold(n_splits=10)

scores = cross_val_score(

the_pipeline, brain_data, behavior,

cv=gkf, groups=family_data

)

predictions = cross_val_predict(

the_pipeline, brain_data, behavior,

cv=gkf, groups=family_data

)
```

While GroupKFold is used here as an example, users can substitute any scikit-learn cross-validation splitter that accommodates dependency structure, selecting the strategy best suited to their data and research design.

SECTION 2. COMMON NEUROIMAGING ALGORITHMS AND THEIR PIPELINE CONFIGURATIONS

CONNECTOME-BASED PREDICTIVE MODELING (CPM)

CPM[6] identifies connectivity patterns that predict individual differences in behavior through sign-based aggregation (Figure 2).

The method consists of the following steps:

1. **Feature Selection.** Let X be the connectivity matrix with n subjects and m features, where x_{ij} represents the connectivity value for subject j and feature i. Let y be the behavioral measure vector for all subjects. For each connectivity feature i (where i = 1, ..., m), compute the Pearson correlation coefficient r_i between feature vector x_i and behavioral measure y, along with its corresponding p-value p_i (Figure 2A). Then, given a significance threshold α, select features based on statistical significance:

$$S = \{i : p_i < \alpha\}$$

where S is the set of indices for selected features.

2. **Sign-Based Aggregation.** Partition the selected features S into two subsets based on their correlation sign (Figure 2B):

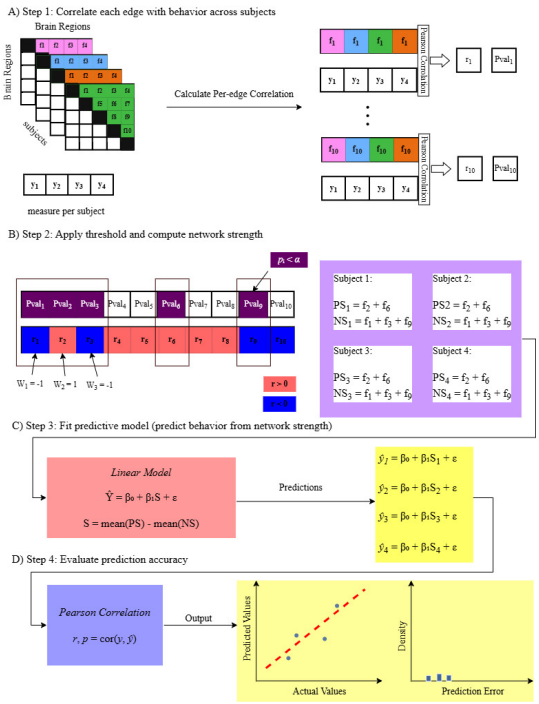

Figure 2. Connectome-based Predictive Modeling (CPM) algorithm workflow. **(A)** For each subject (represented by a distinct color), a functional connectivity matrix is extracted, where each element represents the Pearson correlation between pairs of brain regions. Each feature (connectivity edge) is then correlated with the behavioral measure across subjects, yielding a correlation coefficient (r) and associated p-value per edge. **(B)** Edges are partitioned into positive (red) and negative (blue) networks according to the sign of their correlation with behavior. Edges are selected by applying a significance threshold ($p < \alpha$). For each subject, a single summary score is computed as the difference between the mean connectivity of the positively and negatively correlated edges: S = mean(positive edges) − mean(negative edges). **(C)** A linear model uses the summary score to predict behavior ($\hat{Y} = \beta_0 + \beta_1 S + \varepsilon$), generating individual predictions for each subject. **(D)** Model performance is assessed by correlating predicted and observed behavioral values. The scatter plot depicts prediction accuracy, with the dashed red line indicating the regression fit. The accompanying distribution of prediction errors illustrates variability in model performance across subjects.

$$S_+ = \{i \in S : r_i > 0\}$$
$$S_- = \{i \in S : r_i < 0\}$$

This binary weighting approach assigns equal importance to all selected features regardless of their correlation magnitude.

3. **Behavioral Prediction.** Fit a linear model using a single summary score as the predictor, computed by taking the difference between the mean connectivity of positively and negatively correlated features (Figure 2C):

$$S_j = mean(X[S_+, j]) - mean(X[S_-, j])$$
$$\hat{y}_j = \beta_0 + \beta_1 S_j$$

where β_0 is the intercept and β_1 is the regression coefficient.

4. **Evaluation.** Assess the prediction accuracy by computing the Pearson correlation coefficient r and its associated p-value p (Figure 2D):

$$r, p = cor(y, \hat{y})$$

where y is the vector of actual behavioral values and $\hat{y}$ is the vector of predicted values.

CPM CONFIGURATION VIA THE CONFIGURABLE PIPELINE

The traditional CPM[6] algorithm, following the Yale MATLAB implementation,[14] can be implemented within our configurable pipeline using the RobustRegression() wrapper — a custom scikit-learn-compatible class available in our repository — with the following configuration:

```
cpm_pipeline = Pipeline([

('vectorize', FeatureVectorizer()),

('mua', MUA(

filter_by_sign=True,

direction='difference',

selection_method='pvalue',

selection_threshold=0.05,

weighting_method='binary',

feature_aggregation='mean',

)),

('regressor', RobustRegression()) # Robust regression

])

# Cross-validation

cpm_scores = cross_val_score(cpm_pipeline, brain_data,
behavior, cv=10)

cpm_predictions = cross_val_predict(cpm_pipeline,
brain_data, behavior, cv=10)

print(f"CPM R² (10-fold CV): {cpm_scores.mean():.3f} ±
{cpm_scores.std():.3f}")

# Evaluation

cpm_r, cpm_p = pearsonr(behavior, cpm_predictions)

mae = mean_absolute_error(behavior, cpm_predictions)

rmse = np.sqrt(mean_squared_error(behavior,
cpm_predictions))

r2 = r2_score(behavior, cpm_predictions)
```

The key parameters in the MUA class that define CPM[6] are:

- `filter_by_sign=True` creates separate positive and negative network scores
- `direction='difference'` computes a single summary score as (positive network score – negative network score)

- `selection_method='pvalue'` with threshold=0.05 implements statistical feature selection
- `weighting_method='binary'` assigns equal weights to all selected features (connectivity edge)
- `feature_aggregation='mean'` computes (mean(positive network score) – mean(negative network score))

The final prediction step uses a linear model, consistent with the CPM protocol description.[6] Users can specify their preferred regression method via the 'regressor' step in the pipeline. For users who wish to use robust regression as their final prediction step, following the Yale MATLAB implementation,[14] our repository provides a custom scikit-learn-compatible wrapper, called RobustRegression(), that closely replicates MATLAB's robustfit (for further details, see CPM Validation and Limitations). Alternatively, users who prefer ordinary least squares, as in the Python CPM tutorial by Finn,[15] can use scikit-learn's LinearRegression(). More broadly, our pipeline supports any scikit-learn-compatible regression method, and users can also implement custom scikit-learn-compatible wrappers for additional regression approaches.

POLYNEURO RISK SCORES (PNRS)

PNRS[7] assesses brain-behavior associations through weighted aggregation, where each feature (connectivity edge) contributes proportionally to its univariate association strength (Figure 3).

The method consists of the following steps:

1. **Feature Selection.** Let X be the connectivity matrix with n subjects and m features, where x_{ij} represents the connectivity value for subject j and feature i. Let y be the behavioral measure vector for all subjects (Figure 3A).

All m connectivity features are retained and for each feature i (where i = 1, ..., m), the univariate regression coefficient β_i will be computed:

$$y = \beta_i x_i$$

where x_i is the vector of connectivity values for feature i across all subjects, and the coefficient is computed as:

$$\beta_i = (x_i^T y) / (x_i^T x_i)$$

2. **Score Aggregation.** For each subject j, compute a single aggregated score S_j by summing all features weighted by their regression coefficients (Figure 3B):

$$S_j = \sum_{i=1}^{m} \beta_i x_{ij}$$

This creates a composite score that reflects the cumulative effect of all brain features.

3. **Evaluation.** Assess the strength of the brain-behavior association by computing the Pearson correlation coefficient r and its associated p-value p between the aggregated scores and the behavioral measure (Figure 3C):

$$r, p = cor(y, S)$$

where r quantifies the strength of the linear association between the brain-wide connectivity pattern and the behavioral measure.

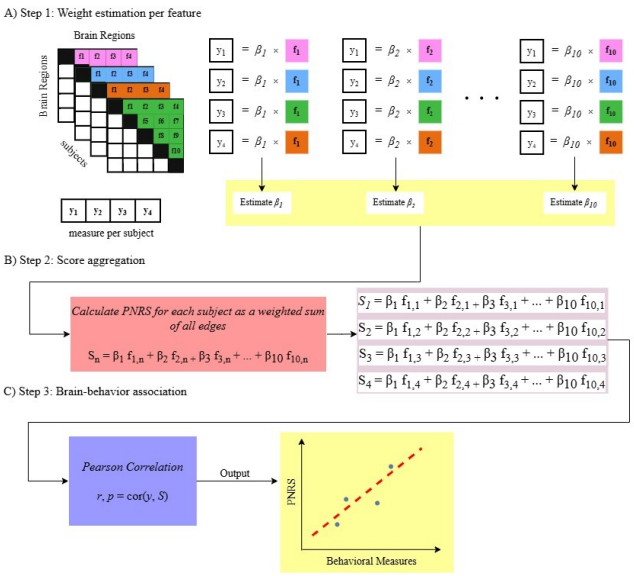

Figure 3. Polyneuro Risk Scores (PNRS) methodology. **(A)** Functional connectivity matrices are extracted for each subject (represented by a distinct color). Each matrix contains connectivity values for all brain region pairs. For each connectivity edge (feature), a univariate regression coefficient (β weight) is estimated across subjects using the model $y = \beta_i x_i$, where $\beta_i = (x_i^T y) / (x_i^T x_i)$, yielding one β weight per edge (shown here for features f_1 through f_{10}). **(B)** For each subject j, the PNRS is computed as the weighted sum of all connectivity edges, where each edge is multiplied by its corresponding β weight ($S_j = \beta_1 f_{1,j} + \beta_2 f_{2,j} + \ldots + \beta_{10} f_{10,j}$). **(C)** The aggregated PNRS scores are correlated with the observed behavioral measures. The correlation coefficient r and its associated p-value quantify the strength of the brain-behavior association captured by the score.

PNRS CONFIGURATION VIA THE CONFIGURABLE PIPELINE

The PNRS[7] algorithm, used for in-sample validation against the BWAS MATLAB toolbox[16] can be implemented within our configurable pipeline using the following configuration. This configuration uses all connectivity features without selection, creating a comprehensive brain-wide score.

```
pnrs_pipeline = Pipeline([

('vectorize', FeatureVectorizer()),

('mua', MUA(

filter_by_sign=False,

selection_method='all',

weighting_method='regression',

feature_aggregation='sum',

))

])

pnrs_scores = pnrs_pipeline.fit_transform(brain_data,
```

```
behavior)

# Evaluation

pnrs_r, pnrs_p = pearsonr(behavior,
pnrs_scores.flatten())
```

The key parameters in the MUA class that define PNRS[7] are:

- `filter_by_sign=False` processes all features (connectivity edges) together without sign-based partitioning into positive and negative networks.
- `selection_method='all'` includes every feature without statistical thresholding
- `weighting_method='regression'` assigns beta weights from univariate regressions ($\beta_i = (x_i^T y) / (x_i^T x_i)$)
- `feature_aggregation='sum'` computes the score as a weighted sum of all features

USING PNRS AS A PREDICTOR WITH A FINAL REGRESSION STEP

The PNRS[7] pipeline can also be adapted to accommodate a final regression step.[13] We are unable to validate this exact version of the pipeline since it has not been previously released to our knowledge; however, the individual components — the PNRS computation and the final regression step — are each independently validated through the original PNRS and the CPM[6] pipeline. This configuration is as follows:

```
pnrs_pipeline = Pipeline([

('vectorize', FeatureVectorizer()),

('mua', MUA(

filter_by_sign=False,

selection_method='all',

weighting_method='regression',

feature_aggregation='sum',

)),

('regressor', LinearRegression()) # Linear regression

])

# Cross-validation

pnrs_scores = cross_val_score(pnrs_pipeline,
brain_data, behavior, cv=10)

pnrs_predictions = cross_val_predict(pnrs_pipeline,
brain_data, behavior, cv=10)

print(f"PNRS R² (10-fold CV): {pnrs_scores.mean():.3f}
± {pnrs_scores.std():.3f}")

# Evaluation

pnrs_r, pnrs_p = pearsonr(behavior, pnrs_predictions)

mae = mean_absolute_error(behavior, pnrs_predictions)

rmse = np.sqrt(mean_squared_error(behavior,
pnrs_predictions))

r2 = r2_score(behavior, pnrs_predictions)
```

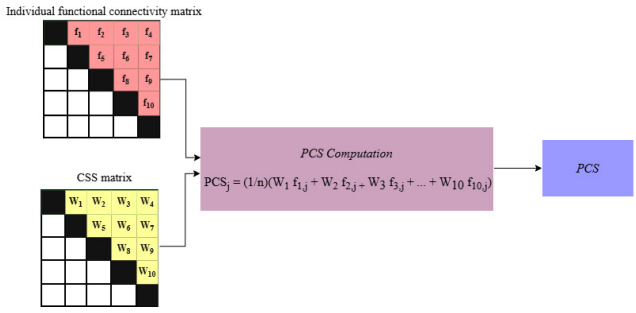

Figure 4. Polyconnectomic Scoring (PCS) methodology. An individual's functional connectivity matrix (top left, pink) contains connectivity values (f_1 through f_{10}) and a connectome summary statistic (CSS) matrix (bottom left, yellow) contains pre-computed edge-wise weights (W_1 through W_{10}) derived from a large discovery dataset, where each weight quantifies the association between a given connectivity edge and the disorder or behavioral measure of interest. The PCS is computed as the weighted mean of the individual's connectivity values multiplied by their corresponding CSS weights: $PCS_j = \frac{1}{10}(W_1 f_{1,j} + W_2 f_{2,j} + W_3 f_{3,j} + \ldots + W_{10} f_{10,j})$. The resulting PCS (right) provides a single scalar index reflecting the degree to which the individual's connectivity pattern resembles the disorder-associated connectivity profile.

The key parameters in the MUA class that define PNRS[7] with final regression are as above with the exception of the additional parameter 'regressor' set to LinearRegression().

POLYCONNECTOMIC SCORING (PCS)

PCS[8,9] measures how similar an individual's brain functional connectivity pattern is to the connectivity pattern associated with a specific disorder or behavior (Figure 4). To compute this score, researchers first estimate CSS using large discovery datasets. These statistics quantify the association between each functional connection and the disorder or behavior (specifically, regression coefficients for scale variables, analogous to the regression-derived weights used in the PNRS[7] framework to assess brain-behavior associations, and Cohen's *d* for group contrasts). The resulting CSS values form a matrix of edge-wise weights representing the disorder- or behavior-related connectivity pattern (Figure 4, CSS matrix). The PCS for a new individual (Figure 4, individual functional connectivity matrix) is then computed by aggregating their functional connectivity values weighted by the corresponding CSS values, producing a score that reflects how strongly the individual's connectome resembles the disorder- or behavior-associated connectivity pattern.

$$PCS_j = \frac{1}{n} \sum_{e=1}^{n} FC_{j,e} \times CSS_e$$

where $FC_{j,e}$ denotes the functional connectivity value of edge *e* for subject *j*, CSS_e represents the connectome summary statistic for edge *e*, and *n* is the total number of connections included in the score.

PCS CONFIGURATION VIA THE CONFIGURABLE PIPELINE

Because of the shared computational backbone of MUA methods, our pipeline naturally accommodates PCS[8,9] analyses: users supply a pre-computed CSS vector as external weights, and PCS is produced within the same analytical workflow used for other MUA approaches. The pipeline accepts any array-like structure as CSS (e.g., NumPy array, Python list, or Pandas Series), provided as a one-dimensional vector of edge-wise weights whose length equals the number of unique edges in the upper triangle of the connectivity matrix. When the CSS is available as a full regions × regions matrix, the upper triangle can be extracted via np.triu_indices(n, k=1) to produce the required vector format for the pipeline. FeatureVectorizer is included in the pipeline to apply the same extraction to the input connectivity matrices, ensuring consistent edge ordering between the CSS and the data:

```
# Load pre-computed CSS matrix from discovery study
CSS_matrix = ...

# Extract upper triangle to match vectorized FC edges

CSS = CSS_matrix[np.triu_indices(n_nodes, k=1)]

pcs_pipeline = Pipeline([

('vectorize', FeatureVectorizer()),

('mua', MUA(

filter_by_sign=False,

selection_method='all',

weighting_method='external',

external_weights=CSS,

feature_aggregation='mean',

))

])

pcs_scores = pcs_pipeline.fit_transform(brain_data,
behavior)

pcs_results = pcs_scores.flatten()
```

The key parameters in the MUA class that define PCS[8,9] are:

- `filter_by_sign=False` processes all features (connectivity edges) together without sign-based partitioning into positive and negative networks.
- `selection_method='all'` includes every feature, consistent with the application of a full CSS weight map.
- `weighting_method='external'` with external_weights applies an externally derived CSS containing edge-wise weights associated with a specific disorder or behavioral phenotype
- `feature_aggregation='mean'` computes the PCS value as the average weighted connectivity across all features

COMPUTING CSS WITHIN THE PIPELINE

Beyond applying pre-computed CSS, users can use our pipeline to derive CSS from a discovery sample. For scale variables, CSS can be extracted as univariate regression coefficients (β-weights). For group contrasts, CSS is expressed as Cohen's *d*.[8,9] Our pipeline supports this by first computing edge-wise correlations from the discovery sample and then converting them to Cohen's *d* using established formulas derived either directly or through the t-statistic.[22-24] Our implementation uses the t-statistic-based conversion, as this enabled validation against the effectsize R package,[25,26] which implements the same approach; to our knowledge, no existing study implements the direct correlation-to-Cohen's *d* conversion formula. Alternatively, users can compute Cohen's *d* externally using dedicated tools in Python (e.g., the Pingouin[27] Python library), which compute Cohen's *d* from the means and standard deviations of the two groups.[28] All three approaches — direct computation from group statistics, and the two correlation-to-Cohen's *d* conversions — are presented in Appendix A.

Note that the pipeline's built-in correlation-to-Cohen's *d* conversion computes unadjusted effect sizes for two-group contrasts; however, CSS matrices from prior PCS studies have sometimes been derived using Cohen's *d* corrected for covariates such as age, sex, and in-scanner motion.[8,9] For datasets requiring confound correction, multi-group contrasts, or other custom CSS for PCS computations, users can derive weights externally and supply them via the external_weights parameter.

The pipeline computes and stores CSS internally as a one-dimensional vector. To save the resulting CSS as a full regions × regions matrix, users can reconstruct it using FeatureVectorizer.inverse_transform.

1. SCALE VARIABLES

For scale variables (e.g., fluid intelligence), CSS is computed as univariate regression coefficients.[8,9] Users can derive these weights from a discovery sample and apply them as external weights to compute PCS in a validation sample. As there is no publicly available CSS implementation from prior studies to validate against, we were unable to validate this CSS derivation step directly. Since the CSS values for scale variables are mathematically identical to the regression coefficients used in PNRS[7] (both computed as $\beta_i = (x_i^T y) / (x_i^T x_i)$), our validated PNRS implementation confirms the correctness of this step.

```
# Step 1: Derive CSS (β-weights) from discovery
sample

css_pipeline = Pipeline([

('vectorize', FeatureVectorizer()),

('mua', MUA(

filter_by_sign=False,

selection_method='all',

weighting_method='regression',
```

```
 feature_aggregation='sum',

 ))

 ])

 css_pipeline.fit(brain_data_discovery,
behavior_discovery)

 CSS = css_pipeline.named_steps['mua'].edge_weights_

 # Save CSS as a symmetric matrix CSV

 vectorizer = css_pipeline.named_steps['vectorize']

 CSS_matrix = vectorizer.inverse_transform(

 CSS.reshape(1, -1))[0]

 pd.DataFrame(

 CSS_matrix,

 index=region_labels, columns= region_labels

 ).to_csv('css_beta_weights.csv')

 # Step 2: Apply CSS to compute PCS in validation
sample

 pcs_pipeline = Pipeline([

 ('vectorize', FeatureVectorizer()),

 ('mua', MUA(

 filter_by_sign=False,

 selection_method='all',

 weighting_method='external',

 external_weights=CSS,

 feature_aggregation='mean',

 ))

 ])

 pcs_scores =
pcs_pipeline.fit_transform(brain_data_validation,
behavior_validation)

 pcs_results = pcs_scores.flatten()
```

2. *GROUP CONTRASTS*

For two-group contrasts (e.g., patients vs. controls), CSS is expressed as Cohen's *d*.[8,9] Users can use our configurable pipeline to derive edge-wise correlations between connectivity values and binary group labels from a discovery sample and convert them to Cohen's *d* .[22-24] When binary group labels are provided (e.g., 0 = controls, 1 = patients), the resulting Pearson correlations are mathematically equivalent to point-biserial correlations, which can be directly converted to Cohen's *d* without additional configuration.

$$d_e = \frac{r_e}{\sqrt{(1 - r_e^2)}} \times \sqrt{(\frac{(n_1 + n_2 - 2)}{n_1} + \frac{(n_1 + n_2 - 2)}{n_2})}$$

where r_e is the correlation for edge *e*, and n_1 and n_2 are the sample sizes of the two groups. This equation is derived through the t-statistic,[23,24] which is also used in the effect-size R package.[25,26] Full derivations are provided in Appendix A.

```
 # Step 1: Derive correlations from discovery sample

 css_pipeline = Pipeline([

 ('vectorize', FeatureVectorizer()),

 ('mua', MUA(

 filter_by_sign=False,

 selection_method='all',

 weighting_method='correlation',

 feature_aggregation='sum',

 ))

 ])

 css_pipeline.fit(brain_data_discovery,
group_labels_discovery)

 r = css_pipeline.named_steps['mua'].correlations_

 # Convert correlation to Cohen's d (general formula)

 n1 = np.sum(group_labels_discovery == 0) # e.g.,
controls

 n2 = np.sum(group_labels_discovery == 1) # e.g.,
patients

 # Correlation-to-Cohen's d conversion

 CSS_cohen_d = (r / np.sqrt(1 - r**2)) * np.sqrt(

 ((n1 + n2 - 2) / n1) + ((n1 + n2 - 2) / n2)

 )

 # Save CSS as a symmetric matrix CSV

 vectorizer = css_pipeline.named_steps['vectorize']

 CSS_cohen_d_matrix = vectorizer.inverse_transform(

 CSS_cohen_d.reshape(1, -1))[0]

 pd.DataFrame(

 CSS_cohen_d_matrix,

 index=region_labels, columns= region_labels

 ).to_csv('css_cohen_d.csv')

 # Step 2: Apply CSS to compute PCS in validation
sample

 pcs_pipeline = Pipeline([

 ('vectorize', FeatureVectorizer()),

 ('mua', MUA(

 filter_by_sign=False,

 selection_method='all',

 weighting_method='external',

 external_weights=CSS_cohen_d,

 feature_aggregation='mean',
```

```
))

])

 pcs_scores =
pcs_pipeline.fit_transform(brain_data_validation,
group_labels_validation)

 pcs_results = pcs_scores.flatten()
```

SECTION 3. VALIDATION

To ensure the accuracy of our pipeline, we compared the results of our pipeline configurations for CPM[6] and PNRS[7] against their established MATLAB implementations[14,16] using the HCP data.[18] The CPM protocol describes the final prediction step as a linear model, which has been implemented as robust regression in the Yale MATLAB code and as ordinary least squares in the Python CPM tutorial by Finn.[15] Using identical fold assignments, we validated our pipeline against the Yale MATLAB implementation in two ways: (1) using our custom scikit-learn-compatible robust regression wrapper to closely replicate MATLAB's robustfit, achieving near-perfect numerical equivalence on the HCP data; and (2) using scikit-learn's LinearRegression, which, as expected in a large-sample single-predictor setting, showed high agreement, with residual differences reflecting the methodological distinction between ordinary least squares and robust regression rather than any pipeline error (for further details, see CPM Validation and Limitations). For PNRS, the pipeline achieved exact numerical equivalence. For PCS,[8,9] we validated two aspects of our pipeline. (1) PCS computation: To ensure that our pipeline correctly computes PCS, we compared its output against results obtained directly from the PCS formula. For this, we used the HCP data, consistent with the CPM and PNRS validations, alongside a simulated CSS matrix. (2) CSS computation for group contrasts (correlation-to-Cohen's *d* conversion): To verify the pipeline's correlation-to-Cohen's *d* conversion,[22-24] we validated our results against the effect-size R package,[25,26] which implements the same r-to-d conversion derived from the t-statistic,[23,24] and independently against the Pingouin Python library, which uses the means and standard deviations of the two groups.[27] Both the PCS[8,9] computation and the CSS derivation approach matched exactly with their ground-truth values, confirming the correctness of our pipeline's PCS computation and the correlation-to-Cohen's *d* conversion via the t-statistic.[23,24]

Complete validation scripts and visualization utilities for generating publication-quality figures of prediction accuracy and error distributions are available in the repository.

Additionally, to provide guidance on computational feasibility, we report single-run runtimes for each pipeline configuration, measuring only the core pipeline computation — excluding data loading and visualization — on a single CPU core (Intel Core i7, 10th generation, 8GB RAM; Table 2).

DATASET

We validated these pipelines using resting-state fMRI data obtained from the HCP.[18] The dataset comprised 1067 subjects. Functional connectivity matrices were derived by preprocessing the data using the Yale preprocessing and functional connectivity estimation pipelines (cf. Noble et al.[29]) to obtain resting-state functional connectivity data as the Pearson correlations between time series for each region of the Shen268 atlas,[30] yielding 268 × 268 symmetric matrices per subject. Following the original CPM,[6] fluid intelligence scores served as the behavioral prediction target.

CPM VALIDATION

We benchmarked our CPM implementation against the original Yale code[6,14] using HCP[18] connectivity data (n = 1067) with a significance threshold of 0.05 and identical 10-fold cross-validation fold assignments to isolate algorithmic differences. As the original MATLAB implementation uses robust regression (robustfit) for the final prediction step, we created a scikit-learn-compatible wrapper available in the repository that uses statsmodels' RLM[31] with Tukey's bisquare weighting to closely match this behavior, as no existing Python package exactly replicates MATLAB's robustfit.

Figure 5 presents the results using our robust regression wrapper: the original MATLAB implementation (panels A–B; r = 0.223, p < 0.001, max |diff| = 1.30e+01, mean |diff| = 3.93e+00) and our configurable pipeline (panels C–D; r = 0.223, p < 0.001, max |diff| = 1.30e+01, mean |diff| = 3.93e+00) showed equivalent prediction performance. Panels E–F directly compare the predicted values, confirming near-perfect numerical agreement (r = 1.000, p < 0.001, max |diff| = 1.33e-03, mean |diff| = 5.28e-04), with negligible residual differences attributable to implementation-level differences between MATLAB's robustfit and our wrapper (see Limitations). Using the same data, we additionally used ordinary least squares (scikit-learn's LinearRegression) in place of the robust regression wrapper, consistent with the ordinary least squares approach used in the Python CPM tutorial by Finn[15] (Figure 6). The MATLAB implementation (panels A–B; r = 0.223, p < 0.001) and our pipeline (panels C–D; r = 0.224, p < 0.001, max |diff| = 1.28e+01, mean |diff| = 3.95e+00) showed comparable prediction performance, with agreement between the two sets of predictions (panels E–F; r = 1.000, p < 0.001, max |diff| = 3.56e-01, mean |diff| = 1.62e-01). The larger residual differences compared to Figure 5 reflect the expected discrepancy between ordinary least squares and robust regression rather than any pipeline error. Together, these results validate the ability of our pipeline to reliably reproduce the standard CPM algorithm, including its characteristic sign-based aggregation into a single summary score followed by a final regression step.

While the CPM protocol describes the final prediction step as a linear model without specifying ordinary least squares or robust regression, the Yale MATLAB implementation uses robust regression and the Python tutorial by Finn uses ordinary least squares.[6,14,15] As both approaches

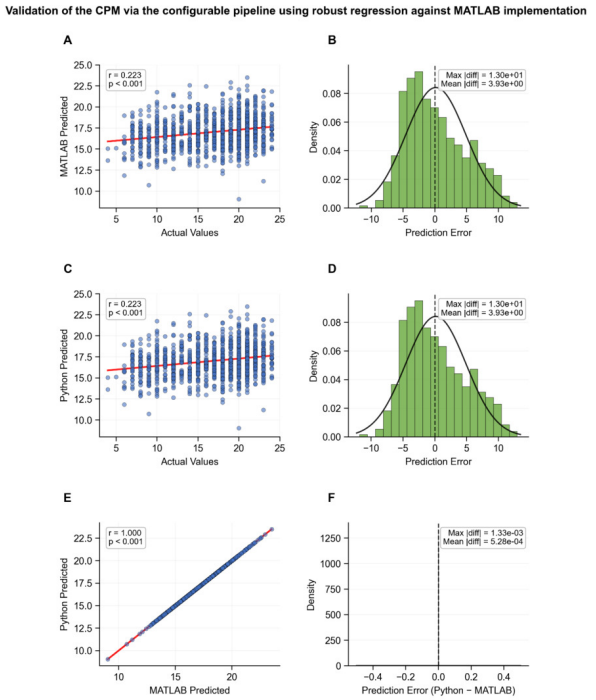

Figure 5. Validation of the CPM implementation via the configurable pipeline using robust regression against the original MATLAB implementation. Both implementations used 10-fold cross-validation with identical fold assignments on HCP connectivity data (n = 1067), predicting fluid intelligence from resting-state functional connectivity. The pipeline used a scikit-learn-compatible wrapper of statsmodels' RLM with Tukey's bisquare weighting to match MATLAB's robustfit. **(A)** MATLAB-predicted versus actual values (r = 0.223, p < 0.001). **(B)** Distribution of prediction errors for the MATLAB implementation, showing maximum (1.30e+01) and mean (3.93e+00) absolute differences. **(C)** Pipeline-predicted versus actual values (r = 0.223, p < 0.001). **(D)** Distribution of prediction errors for the configurable pipeline, showing maximum (1.30e+01) and mean (3.93e+00) absolute differences. **(E)** Direct comparison of predicted values between the two implementations (r = 1.000, p < 0.001). **(F)** Distribution of absolute differences between implementations, showing maximum (1.33e-03) and mean (5.28e-04) absolute differences.

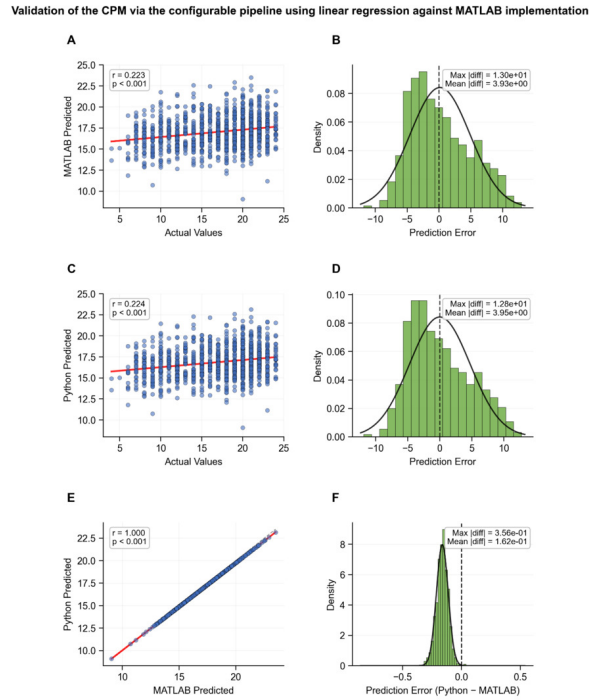

Figure 6. Validation of the CPM implementation via the configurable pipeline using linear regression against the original MATLAB implementation. Both implementations used 10-fold cross-validation with identical fold assignments on HCP connectivity data (n = 1067), predicting fluid intelligence from resting-state functional connectivity. The pipeline used ordinary least squares (LinearRegression) while the MATLAB implementation used robust regression (robustfit). **(A)** MATLAB-predicted versus actual values (r = 0.223, p < 0.001). **(B)** Distribution of prediction errors for the MATLAB implementation, showing maximum (1.30e+01) and mean (3.93e+00) absolute differences. **(C)** Pipeline-predicted versus actual values (r = 0.224, p < 0.001). **(D)** Distribution of prediction errors for the configurable pipeline, showing maximum (1.28e+01) and mean (3.95e+00) absolute differences. **(E)** Direct comparison of predicted values between the two implementations (r = 1.000, p < 0.001). **(F)** Distribution of absolute differences between implementations, showing maximum (3.56e-01) and mean (1.62e-01) absolute differences.

yield comparable predictions in this single-predictor setting (Figures 5–6), users may select either method based on their specific needs; robust regression is more resistant to the influence of outliers, while ordinary least squares assumes homoscedastic, normally distributed residuals. Both are supported by the pipeline: ordinary least squares via scikit-learn's LinearRegression, and robust regression via the custom scikit-learn compatible wrapper provided in the repository.

PNRS VALIDATION

We validated our PNRS implementation against the DCAN Labs BWAS toolbox.[7,16] Using identical in-sample validation on HCP[18] connectivity data (n = 1067), both implementations yielded identical correlation between aggregated scores and behavior. Figure 7 presents the results from the original MATLAB implementation (panel A; r = 0.152, p < 0.001) and our configurable pipeline (panel B; r = 0.152, p < 0.001), showing identical performance. Panels C–D directly compare the PNRS scores of the two implementations, confirming exact numerical equivalence (r = 1.000, p < 0.001, max |diff| = 3.67e-09, mean |diff| = 7.89e-10). These results validate the ability of our pipeline to reproduce the PNRS methodology through appropriate parameter configuration.

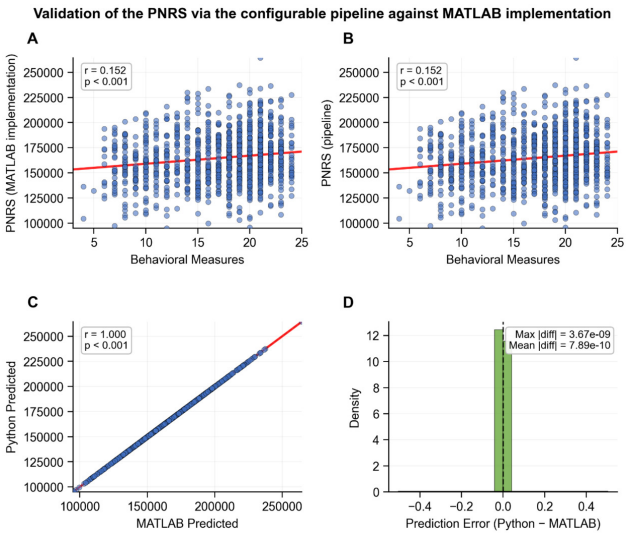

Figure 7. Validation of the PNRS implementation via the configurable pipeline against the original MATLAB implementation. Both implementations used identical in-sample validation on HCP connectivity data (n = 1067), computing PNRS scores from resting-state functional connectivity. **(A)** MATLAB PNRS scores versus actual behavioral values (r = 0.152, p < 0.001). **(B)** Pipeline PNRS scores versus actual behavioral values (r = 0.152, p < 0.001). **(C)** Direct comparison of PNRS scores between the two implementations (r = 1.000, p < 0.001). **(D)** Distribution of absolute differences between implementations, showing maximum (3.67e-09) and mean (7.89e-10) absolute differences, confirming numerical equivalence at machine precision.

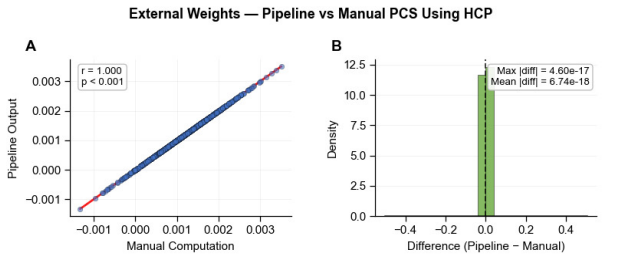

Figure 8. Validation of PCS computation with external CSS weights via the configurable pipeline against manual computation. Both approaches used HCP connectivity data (n = 1067) and a simulated CSS matrix. **(A)** Correspondence between pipeline PCS scores and manually computed PCS scores (r = 1.000, p < 0.001). **(B)** Distribution of absolute differences between pipeline and manual PCS scores, showing maximum (4.60e-17) and mean (6.74e-18) absolute differences, confirming numerical equivalence.

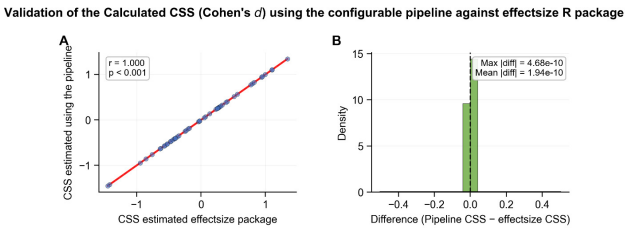

Figure 9. Validation of CSS derivation (Cohen's *d*) for group contrasts via the configurable pipeline against the effectsize R package. Both approaches used simulated data with two groups (n_1 = 120, n_2 = 80). **(A)** Correspondence between pipeline-derived and effectsize-derived Cohen's *d* values (r = 1.000, p < 0.001). **(B)** Distribution of absolute differences between the two implementations, showing maximum (4.68e-10) and mean (1.94e-10) absolute differences.

PCS VALIDATION

PCS COMPUTATION USING AN EXTERNAL CSS MATRIX

We validated the PCS[8,9] implementation using the external weights workflow with HCP connectivity data[18] and a simulated CSS matrix. Figure 8 presents the correspondence between the pipeline's PCS scores and those computed directly from the PCS formula (panel A; r = 1.000, p < 0.001, max |diff| = 4.60e-17, mean |diff| = 6.74e-18), confirming exact numerical equivalence (panel B). These results validate the ability of our pipeline to correctly compute PCS using externally derived CSS weights.

CSS DERIVATION FOR GROUP CONTRASTS

We validated the pipeline's correlation-to-Cohen's *d* conversion via the t-statistic derivation[23,24] using two independent approaches. First, we compared the derived CSS values against those computed by the effectsize R package,[25,26] which implements the same r-to-d conversion through the t-statistic. Second, we compared against the Pingouin[27] Python library, which computes Cohen's *d* directly from group means and pooled standard deviations, providing an independent validation of the conversion pathway itself. Using simulated data with two groups (n_1 = 120, n_2 = 80), Figure 9 presents the correspondence between the pipeline's CSS values and those from the effectsize R package (panel A; r = 1.000, p < 0.001; panel B; max |diff| = 4.68e-10, mean |diff| = 1.94e-10). Figure 10 presents the correspondence between the pipeline's CSS values and those from the Pingouin library (panel A; r = 1.000, p < 0.001; panel B; max |diff| = 4.98e-10, mean |diff| = 2.48e-10), confirming that the conversion pathway produces Cohen's *d* values consistent with direct computation from group statistics. Together, these results validate our pipeline's correlation-to-Cohen's *d* conversion for group contrast analyses.

RUNTIME BENCHMARKS

Table 2 reports runtime benchmarks for each pipeline configuration described in the validation section, measuring only the core pipeline computation from input data to final results, excluding data loading and visualization. All

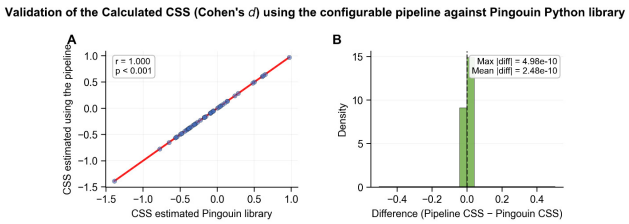

Figure 10. Validation of CSS derivation (Cohen's *d*) for group contrasts via the configurable pipeline against the Pingouin Python library. Both approaches used simulated data with two groups (n_1 = 120, n_2 = 80). **(A)** Correspondence between pipeline-derived and Pingouin-derived Cohen's *d* values (r = 1.000, p < 0.001). **(B)** Distribution of absolute differences between the two implementations, showing maximum (4.98e-10) and mean (2.48e-10) absolute differences.

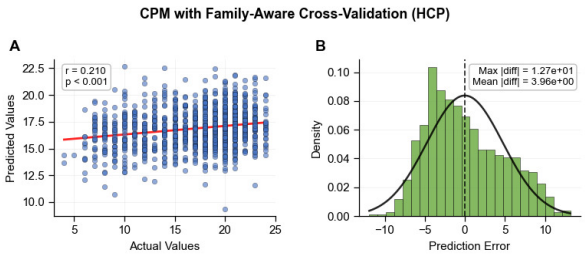

Figure 11. Illustrative application of CPM with family-aware cross-validation on HCP data. Applied to HCP connectivity data (n = 1067) using GroupKFold with HCP family IDs, predicting fluid intelligence from resting-state functional connectivity. **(A)** Pipeline-predicted versus actual values (r = 0.210, p < 0.001). **(B)** Distribution of prediction errors, showing maximum (1.27e+01) and mean (3.96e+00) absolute differences

benchmarks were obtained on an Intel Core i7 (10th generation) with 8GB RAM.

Table 2. Runtime benchmarks for each pipeline configuration.

Pipeline Configuration	Computation Mode	Runtime (s)
CPM using robust regression	10-fold cross-validation	23.64
CPM using linear regression	10-fold cross-validation	29.72
PNRS	In-sample	3.73
PCS computation using external CSS	In-sample	1.40
CSS derivation (Cohen's *d*)	In-sample	0.02

ILLUSTRATIVE APPLICATION: FAMILY-AWARE CROSS-VALIDATION

Beyond the validation analyses above, we applied the canonical CPM configuration[6,14] to the HCP data[18] using GroupKFold with each subject's HCP family ID as the grouping variable, to demonstrate how to use our configurable pipeline for this purpose. Results are shown in Figure 11 (panel A: r = 0.210, p < 0.001; panel B: max |diff| = 1.27e+01, mean |diff| = 3.96e+00). This example illustrates how researchers can integrate family-aware cross-validation into the pipeline for datasets with non-independent observations such as HCP.[20,21]

DISCUSSION

The primary contribution of this work is the technical consolidation of MUA methods into a single, standardized, and configurable pipeline. Built within the scikit-learn ecosystem in Python, this unified framework brings together two widely used techniques — CPM[6] and PNRS[7] — which we have validated against their original MATLAB implementations,[14,16] alongside PCS[8,9,17] with support for both externally derived and internally computed CSS. Existing implementations of these methods remain fragmented across MATLAB, Python, and R, with no unified framework bringing them together. Furthermore, our pipeline addresses a critical accessibility gap: while the original implementations of CPM and PNRS were developed in MATLAB, which is not open-source, our Python implementation offers a freely accessible alternative, enabling researchers to apply these validated techniques within a modern machine learning framework.

Although different MUA methods share a common computational core, they differ in their specific goals, which our pipeline accommodates through configurable parameters. PNRS[7] is intended to produce a score that captures the brain-behavior association by evaluating how well the score correlates with the outcome, while CPM[6] adds a final regression step to produce actual behavioral predictions, as also demonstrated in recent PNRS applications.[13] Our unified framework supports both: users can work with scores directly or add a final regression step via the 'regressor' step in the pipeline, using any scikit-learn-compatible regression method, including the custom robust regression wrapper provided in our repository. PCS[8,9] uses pre-computed CSS matrices to quantify the degree to which an individual's brain connectivity aligns with disorder- or behavior-related connectivity patterns. As CSS matrices are typically derived from large discovery datasets, users can apply pre-computed CSS from prior studies to assess individual-level connectivity patterns. Our pipeline supports this approach through the external weights option. In addition, users can also derive CSS from a discovery sample within the pipeline itself — using regression-derived β-weights for scale variables or correlation-to-Cohen's *d* conversion for two-group contrasts (e.g., patients vs. controls) — supporting the PCS workflow within a single framework.

We validated CPM[6] and PNRS[7] against their original MATLAB implementations,[14,16] using HCP data.[18] For CPM, using identical fold assignments, we validated the

pipeline with both our robust regression wrapper and ordinary least squares, confirming that the pipeline correctly reproduces the CPM algorithm with either regression method as the final prediction step. For PNRS, the pipeline achieved exact numerical equivalence with the MATLAB implementation. For PCS,[8,9] we validated the external weights workflow against the PCS formula using HCP data, and verified the CSS derivation workflow for two-group contrasts (correlation-to-Cohen's *d* conversion) using simulated data against two independent implementations: the effectsize R package[25,26] and Python Pingouin library.[27] CSS derivation for scale variables was verified through mathematical equivalence with the validated PNRS regression coefficients. In all cases, the pipeline's accuracy and correctness were confirmed. By providing comprehensive documentation, validation scripts, and a configurable interface, we aim to minimize implementation errors and enhance reproducibility in neuroimaging research.

MUA-style approaches are preferable to fully multivariate machine learning methods when interpretability and understanding the independent contribution of individual features are of primary interest. In MUA methods, features are selected univariately, and in contrast to fully multivariate models, the model does not learn how to weight features relative to each other. As a result, each feature's contribution is independently quantifiable and the selected set of connections participating in the model is directly identifiable. This interpretability represents a central advantage of MUA methods, as it enables researchers to determine which specific connections contribute to brain–behavior relationships or play a role in a given clinical symptom.[2,32]

However, aggregation-based approaches may be suboptimal in several scenarios: when the relevant brain–behavior relationship depends on interactions between features that cannot be captured by independent evaluation[33]; when selected features have unequal predictive importance but are combined with equal contributions[2]; and when the signal is subtle and distributed, as stringent multiple comparison corrections required by univariate selection may lead to loss of statistical power and information.[33,34] In such cases, fully multivariate methods capable of modelling higher-order dependencies between features may offer improved predictive accuracy and better generalization to independent datasets. However, in fully multivariate models, individual feature weights do not directly reflect the underlying neural sources of an effect due to feature correlations and suppressor variables, making interpretation considerably more challenging.[32,35,36]

Therefore, the choice between these approaches should be guided by the specific research question and the relative priority placed on interpretability versus predictive performance. MUA approaches are better suited for establishing transparent, reproducible baselines with direct neurobiological interpretability, while fully multivariate methods may be preferred when maximizing prediction accuracy is the primary objective.

RECOMMENDATIONS & FUTURE WORK

For optimal results using our configurable pipeline, we recommend following standard machine learning best practices: (1) ensuring adequate sample sizes based on recent neuroimaging guidelines[1,2]; (2) running the "Preprocessing" script available in our repository, which helps to remove faulty subjects; (3) carefully selecting method parameters aligned with research objectives; (4) using group-aware cross-validation whenever observations are non-independent — including related subjects, multi-site data, and repeated measures — to prevent data leakage and inflated performance estimates[20,21]; and (5) comprehensively reporting all methodological details.

While our current implementation reliably reproduces established MUA methods, including CPM[6] and PNRS,[7] the parameterized configurable pipeline opens exciting possibilities for methodological innovation. Future work will systematically characterize performance across the full parameter space, exploring hybrid configurations such as: (1) combinations of weighting methods (e.g., correlation-based weights for feature selection with regression weights for aggregation); (2) ensemble approaches that combine predictions from multiple parameter configurations; (3) formal runtime benchmarks and memory profiling on large-scale connectome datasets to provide practical guidance.

CONCLUSION

Our pipeline significantly simplifies the reporting process; users can achieve full methodological transparency by directly documenting the specific input parameters passed to the function. By utilizing these built-in parameters to define the MUA class, standardization, and cross-validation strategy, researchers can seamlessly provide the documentation necessary for exact reproducibility and cross-study comparisons.[2] By providing this unified platform with comprehensive parameter control, which facilitates application of validated MUA methods and systematic exploration of the MUA design space using standardized machine learning toolkits, this pipeline is intended to provide a more usable, standardized foundation for advancing our understanding of brain-behavior relationships.

LIMITATIONS

We note that the HCP dataset[18] includes twins and siblings, which may introduce non-independence among observations. To prevent data leakage when applying these methods to related-family data, users should use family-aware cross-validation strategies (e.g., scikit-learn's GroupKFold where family ID is the grouping variable; see the *Cross-Validation Strategy* subsection and the *Illustrative Application: Family-Aware Cross-Validation* subsection for examples using our configurable pipeline).[20,21] In our validation, we replicated the original studies' cross-validation schemes (which did not account for family structure) to ensure direct numerical equivalence; however, we strongly recommend

family-aware splitting for independent applications. The magnitude of the effect of family-aware cross-validation on prediction performance varies across datasets, outcomes, and modeling choices. Prior work suggests that family-related leakage has relatively minor effects compared to other forms of leakage, though these effects become more pronounced as the proportion of multi-member families increases; users should therefore evaluate this consideration in the context of their own data and pipeline.

For the CPM validation, the Yale MATLAB implementation uses robust regression (robustfit) for the final prediction step.[6,14] As no existing Python package exactly replicates MATLAB's robustfit, we created a scikit-learn-compatible wrapper using statsmodels' Robust Linear Model (RLM) with Tukey's bisquare weighting function (tuning constant = 4.685).[31] Both implementations use iteratively reweighted least squares (IRLS) with median absolute deviation (MAD)-based scale estimation and share the same default tuning constant. The numerical differences (max $|\Delta\beta|$ = 1.33e-03; Figure 5) between the two implementations remain, attributable to the absence of leverage-corrected residuals in statsmodels' scale estimation, differences in scale updating procedures across iterations, and convergence criteria. Specifically, MATLAB's robustfit adjusts residuals by the hat matrix diagonal before computing the MAD scale, whereas statsmodels' RLM computes the scale from unadjusted residuals. The magnitude of these differences may vary depending on sample characteristics such as sample size and the presence of outliers; however, here, these differences are negligible relative to the effect sizes reported and do not affect the interpretation of results.

For PCS validation, while real CSS matrices are available,[8,9,17] we lacked access to CSS data that matched our specific parcellation scheme (Shen268 atlas[30]). Therefore, we validated the PCS computation using a simulated CSS matrix and confirmed that our pipeline's output matched the direct PCS formula with perfect numerical agreement (r = 1.000). Users can apply their own externally derived or internally computed CSS matrices using the same workflow demonstrated in this paper. Additionally, the pipeline's built-in correlation-to-Cohen's *d* conversion computes unadjusted effect sizes for two-group contrasts; however, CSS matrices from prior PCS studies were sometimes derived using Cohen's *d* corrected for covariates such as age, sex, and in-scanner motion.[8,9] For datasets requiring confound correction, multi-group contrasts, or other custom CSS for PCS computations, users can derive weights externally and supply them via the external_weights parameter.

DATA AVAILABILITY

All raw data used in the present study have been made publicly available by HCP in accordance with data use and access regulations set by the HCP.[18] The Northeastern University and Yale University Human Research Protection Programs approved secondary analyses of these datasets. Human Connectome Project (HCP) data are available through the HCP repository (https://www.humanconnectome.org/study/hcp-young-adult). Users must agree to data use terms before accessing ConnectomeDB; details are provided at https://www.humanconnectome.org/study/hcp-young-adult/data-use-terms.

CODE AVAILABILITY

All code used for running statistical analyses has been made publicly available on GitHub at https://github.com/neuroprismlab/_MUA_Pipeline. The BioImage Suite command line software used for processing is freely available at (https://bioimagesuiteweb.github.io/webapp/index.html).

ACKNOWLEDGEMENTS

This work was supported by funding from the National Institute of Mental Health (R00 MH130894 to S.N). Data were provided by the Human Connectome Project, WU-Minn Consortium (Principal Investigators: David Van Essen and Kamil Ugurbil; 1U54MH091657) funded by the 16 NIH Institutes and Centers that support the NIH Blueprint for Neuroscience Research; and by the McDonnell Center for Systems Neuroscience at Washington University.

CONTRIBUTION STATEMENT

S.N. conceived of the study. F.D. wrote all code and performed all analyses and interpretation under the guidance of S.N. F.C. provided additional guidance on code implementation, and H.S. assisted in final packaging of the pipeline for release. F.D. wrote the manuscript with input from all co-authors.

Submitted: February 06, 2026 CDT. Revised: April 02, 2026 CDT; May 19, 2026 CDT. Accepted: May 25, 2026 CDT. Published: August 31, 2026 CDT.

REFERENCES

1. Marek S, Tervo-Clemmens B, Calabro FJ, et al. Reproducible brain-wide association studies require thousands of individuals. *Nature*. 2022;603(7902):654-660. doi:10.1038/s41586-022-04492-9

2. Scheinost D, Noble S, Horien C, et al. Ten simple rules for predictive modeling of individual differences in neuroimaging. *NeuroImage*. 2019;193:35-45. doi:10.1016/j.neuroimage.2019.02.057

3. Varoquaux G, Thirion B. How machine learning is shaping cognitive neuroimaging. *GigaScience*. 2014;3(1):28. doi:10.1186/2047-217X-3-28

4. Pervaiz U, Vidaurre D, Woolrich MW, Smith SM. Optimising network modelling methods for fMRI. *NeuroImage*. 2020;211:116604. doi:10.1016/j.neuroimage.2020.116604

5. Finn ES, Shen X, Scheinost D, et al. Functional connectome fingerprinting: identifying individuals using patterns of brain connectivity. *Nat Neurosci*. 2015;18(11):1664-1671. doi:10.1038/nn.4135

6. Shen X, Finn ES, Scheinost D, et al. Using connectome-based predictive modeling to predict individual behavior from brain connectivity. *Nat Protoc*. 2017;12(3):506-518. doi:10.1038/nprot.2016.178

7. Byington N, Grimsrud G, Mooney MA, et al. Polyneuro risk scores capture widely distributed connectivity patterns of cognition. *Dev Cogn Neurosci*. 2023;60:101231. doi:10.1016/j.dcn.2023.101231

8. Libedinsky I, Helwegen K, Guerrero Simón L, et al. Quantifying brain connectivity signatures by means of polyconnectomic scoring. *bioRxiv*. Published online September 27, 2023. doi:10.1101/2023.09.26.559327

9. Libedinsky I, Helwegen K, Boonstra J, et al. Polyconnectomic Scoring of Functional Connectivity Patterns Across Eight Neuropsychiatric and Three Neurodegenerative Disorders. *Biol Psychiatry*. 2025;97(11):1045-1058. doi:10.1016/j.biopsych.2024.10.007

10. Choi SW, O'Reilly PF. PRSice-2: Polygenic Risk Score software for biobank-scale data. *GigaScience*. 2019;8(7):giz082. doi:10.1093/gigascience/giz082

11. Choi SW, Mak TSH, O'Reilly PF. Tutorial: a guide to performing polygenic risk score analyses. *Nat Protoc*. 2020;15(9):2759-2772. doi:10.1038/s41596-020-0353-1

12. Lewis CM, Vassos E. Polygenic risk scores: from research tools to clinical instruments. *Genome Med*. 2020;12(1):44. doi:10.1186/s13073-020-00742-5

13. Mooney MA, Hermosillo RJM, Feczko E, et al. Cumulative Effects of Resting-State Connectivity Across All Brain Networks Significantly Correlate with Attention-Deficit Hyperactivity Disorder Symptoms. *J Neurosci*. 2024;44(10):e1202232023. doi:10.1523/JNEUROSCI.1202-23.2023

14. YaleMRRC/CPM. Accessed February 9, 2026. https://github.com/YaleMRRC/CPM

15. esfinn/cpm_tutorial: Python-based tutorial on Connectome-based Predictive Modeling (CPM), originally prepared for Georgetown Methods Lab. Accessed February 9, 2026. https://github.com/esfinn/cpm_tutorial

16. DCAN-Labs/BWAS. August 25, 2025. Accessed February 9, 2026. https://github.com/DCAN-Labs/BWAS

17. pcs-toolbox/calculate_PCS.py at main · dutchconnectomelab/pcs-toolbox. Accessed March 23, 2026. https://github.com/dutchconnectomelab/pcs-toolbox/blob/main/calculate_PCS.py

18. Van Essen DC, Smith SM, Barch DM, Behrens TEJ, Yacoub E, Ugurbil K. The WU-Minn Human Connectome Project: An overview. *NeuroImage*. 2013;80:62-79. doi:10.1016/j.neuroimage.2013.05.041

19. Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine Learning in Python. *J Mach Learn Res*. 2011;12(85):2825-2830.

20. Winkler AM, Webster MA, Vidaurre D, Nichols TE, Smith SM. Multi-level block permutation. *Neuroimage*. 2015;123:253-268. doi:10.1016/j.neuroimage.2015.05.092

21. Rosenblatt M, Tejavibulya L, Jiang R, Noble S, Scheinost D. Data leakage inflates prediction performance in connectome-based machine learning models. *Nat Commun*. 2024;15(1):1829. doi:10.1038/s41467-024-46150-w

22. Borenstein M, Hedges LV, Higgins JPT, Rothstein HR. *Introduction to Meta-Analysis*. John Wiley & Sons; 2009. doi:10.1002/9780470743386

23. Rosnow RL, Rosenthal R, Rubin DB. Contrasts and correlations in effect-size estimation. *Psychol Sci*. 2000;11(6):446-453. doi:10.1111/1467-9280.00287

24. Pustejovsky JE. Converting from d to r to z when the design uses extreme groups, dichotomization, or experimental control. *Psychol Methods*. 2014;19(1):92-112. doi:10.1037/a0033788

25. Ben-Shachar M, Lüdecke D, Makowski D. effectsize: Estimation of Effect Size Indices and Standardized Parameters. *J Open Source Softw*. 2020;5(56):2815. doi:10.21105/joss.02815

26. Ben-Shachar MS, Makowski D, Lüdecke D, et al. effectsize: Indices of Effect Size. March 11, 2026. Accessed March 27, 2026. https://cran.r-project.org/web/packages/effectsize/index.html

27. Vallat R. Pingouin: statistics in Python. *J Open Source Softw*. 2018;3(31):1026. doi:10.21105/joss.01026

28. Cohen J. *Statistical Power Analysis for the Behavioral Sciences*. 2nd ed. L. Erlbaum Associates; 1988.

29. Noble S, Mejia AF, Zalesky A, Scheinost D. Improving power in functional magnetic resonance imaging by moving beyond cluster-level inference. *Proc Natl Acad Sci*. 2022;119(32):e2203020119. doi:10.1073/pnas.2203020119

30. Shen X, Tokoglu F, Papademetris X, Constable RT. Groupwise whole-brain parcellation from resting-state fMRI data for network node identification. *NeuroImage*. 2013;82:403-415. doi:10.1016/j.neuroimage.2013.05.081

31. Seabold S, Perktold J. Statsmodels: Econometric and statistical modeling with Python. In: *Proceedings of the 9th Python in Science Conference*. ; 2010:92-96. doi:10.25080/Majora-92bf1922-011

32. Hebart MN, Baker CI. Deconstructing multivariate decoding for the study of brain function. *NeuroImage*. 2018;180:4-18. doi:10.1016/j.neuroimage.2017.08.005

33. Habeck CG. Basics of multivariate analysis in neuroimaging data. *J Vis Exp*. 2010;(41):e1988. doi:10.3791/1988

34. Chen G. Sources of Information Waste in Neuroimaging: Mishandling Structures, Thinking Dichotomously, and Over-Reducing Data. *Aperture Neuro*. 2022;2:1-22. doi:10.52294/ApertureNeuro.2022.2.ZRJI8542

35. Habeck C, Foster NL, Perneczky R, et al. Multivariate and univariate neuroimaging biomarkers of Alzheimer's disease. *NeuroImage*. 2008;40(4):1503-1515. doi:10.1016/j.neuroimage.2008.01.056

36. Haufe S, Meinecke F, Görgen K, et al. On the interpretation of weight vectors of linear models in multivariate neuroimaging. *NeuroImage*. 2014;87:96-110. doi:10.1016/j.neuroimage.2013.10.067

APPENDIX A: DERIVING COHEN'S *D* FOR CONNECTOME SUMMARY STATISTICS (CSS) IN GROUP CONTRASTS

The following derivations apply to two-group contrasts (e.g., patients vs. controls), where a point-biserial correlation between connectivity values and binary group membership can be converted to Cohen's *d*.

This appendix presents three established approaches for correlation-to-Cohen's *d* conversion: direct computation from group statistics, and two established conversion formulas.[23,24,28]

A.1 DIRECT COMPUTATION OF COHEN'S *D*

Cohen's *d* can be computed directly from the means and standard deviations of the two groups.[28] For each connectivity edge *e*:

$$d_e = \frac{(\overline{X}_{1,e} - \overline{X}_{2,e})}{S_{pooled,e}}$$

where $\bar{X}_{1,e}$and $\bar{X}_{2,e}$are the mean connectivity values of edge e for groups 1 and 2, respectively, and $s_{\text{pooled},e}$is the pooled standard deviation:

$$S_{pooled,e} = \sqrt{\frac{(n_1 - 1)s_{1,e}^2 + (n_2 - 1)s_{2,e}^2}{n_1 + n_2 - 2}}$$

where $s_{1,e}$and $s_{2,e}$are the standard deviations of edge e for each group, and n_1 and n_2 are the respective group sizes.

A.2 CONVERSION FROM POINT-BISERIAL CORRELATION TO COHEN'S *D*

When using the pipeline's built-in correlation computation with binary group labels (e.g., 0 = controls, 1 = patients), the resulting Pearson correlations are mathematically equivalent to point-biserial correlations which can be easily converted to Cohen's *d*.

1. CONVERSION THROUGH THE T-STATISTIC[23,24]

Relationship A. The point-biserial correlation r can be obtained from a two-sample t-statistic.

$$r = \frac{t}{\sqrt{t^2 + df}}$$

where $df = n_1 + n_2 - 2$ is the within-groups degrees of freedom from an independent-samples t-test.

Relationship B. We can obtain Cohen's *d* from a two-sample t-statistic

$$d = t\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

Derivation: $r \rightarrow t \rightarrow d$

Step 1. From the first formula, solve for *t*. Square both sides:

$$r^2 = \frac{t^2}{t^2 + df}$$

Multiply both sides by $t^2 + df$ and rearrange:

$$r^2(t^2 + df) = t^2$$
$$t^2(1 - r^2) = r^2.df$$

Consequently, we have:

$$t = \frac{r\sqrt{df}}{\sqrt{1 - r^2}}$$

Step 2. Substitute into Relationship B:

$$d = t\sqrt{\frac{1}{n_1} + \frac{1}{n_2}} = \frac{r\sqrt{df}}{\sqrt{1 - r^2}} \times \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

Step 3. Substitute $df = n_1 + n_2 - 2$ and combine under a single square root:

$$d = \frac{r}{\sqrt{1 - r^2}} \times \sqrt{(n_1 + n_2 - 2)\,(\frac{1}{n_1} + \frac{1}{n_2})}$$

Step 4. Distribute $(n_1 + n_2 - 2)$ across the sum:

$$d_e = \frac{r_e}{\sqrt{(1 - r_e^2)}} \times \sqrt{(\frac{(n_1 + n_2 - 2)}{n_1} + \frac{(n_1 + n_2 - 2)}{n_2})}$$

2. DIRECT CONVERSION[22]

Borenstein et al. provide the following formula for converting Cohen's *d* to a point-biserial correlation *r*:

$$r = \frac{d}{\sqrt{(d^2 + a)}},$$

where

$$a = \frac{(n_1 + n_2)^2}{n_1 n_2}$$

$$d_e = \frac{r_e}{\sqrt{(1 - r_e^2)}} \times \frac{(n_1 + n_2)}{\sqrt{(n_1\, n_2)}}$$

Algebraic inversion. To obtain *d* from *r*, we invert the above expression:

Step 1. Square both sides:

$$r^2 = \frac{d^2}{d^2 + a}$$

Step 2. Multiply both sides by $d^2 + a$ and expand:

$$r^2\, d^2 + r^2\, a = d^2$$

Step 3. Rearrange and solve for d^2:

$$d^2\,(1 - r^2) = r^2\, a$$
$$d^2 = \frac{r^2 a}{1 - r^2}$$

Step 4. Take the square root, preserving the sign of r:

$$d = \frac{r\sqrt{a}}{\sqrt{(1 - r^2)}}$$

Substituting $a = \frac{(n_1+n_2)^2}{n_1 n_2}$ and simplifying $\sqrt{a} = \frac{n_1 + n_2}{\sqrt{n_1 n_2}}$ yields:

$$d_e = \frac{r_e}{\sqrt{(1 - r_e^2)}} \times \frac{(n_1 + n_2)}{\sqrt{(n_1\, n_2)}}$$