Reinforcement Learning in Neuroimaging

Bhuvan Chandra Sarakam[1]

[1]Mount Vernon Nazarene University, Ohio

Publication Date: 2025/07/14

Abstract: Support learning (RL) offers a promising methodology for breaking down complex neuroimaging information and up- grading how brain function can be understood through adaptive algorithms. This paper explores the integration of reinforcement learning (RL) methods within neuroimaging frameworks, demon- strating how RL can be used to model and interpret high- dimensional datasets, such as functional MRI. By leveraging sci-kit-learn's machine learning tools, potential applications of RL in neuroimaging are illustrated, including the classification and prediction of neural responses to stimuli. The discoveries propose that RL could be instrumental in recognizing designs and directing neuroimaging research, progressing customized clinical methodologies in mental and neurological wellbeing.

Keywords: *Artificial Intelligence, Python.*

How to Cite: Bhuvan Chandra Sarakam; (2025). Reinforcement Learning in Neuroimaging. *International Journal of Innovative Science and Research Technology,* (RISEM–2025), 123-132. https://doi.org/10.38124/ijisrt/25jun172

I. INTRODUCTION

Interest in applying quantifiable artificial intelligence (AI) techniques to neuroimaging data analysis has grown dramat- ically in recent years. Neuroimaging offers a unique window into brain function, yet its inherent complexity and high dimensionality pose significant challenges for data interpre- tation. Traditional analysis methods often fall short in captur- ing subtle patterns of neural activity, thereby motivating the adoption of advanced machine learning approaches.

AI provides robust methods for extracting complex, high- dimensional patterns from data. However, the development of AI tools for neuroimaging is often led by computer scientists whose expertise in neuroscience may be limited. This divergence in domain knowledge can lead to approaches that are technically sophisticated but not fully aligned with neuroscientific questions. To bridge this gap, this paper demon- strates how a widely used, general-purpose AI toolkit, *scikit- learn*, can be effectively applied to neuroimaging. This toolkit not only offers state-of-the-art algorithms but also maintains code simplicity and accessibility, making it suitable for both computational experts and neuroscientists.

The focus of this work is on the programming and method- ological aspects of neuroimaging analysis using AI. The choice of *scikit-learn* is driven by its broad adoption in the AI com- munity, its extensive ecosystem of complementary packages, and its seamless integration with Python-based neuroimaging tools. For a comprehensive introduction to AI techniques in functional magnetic resonance imaging (fMRI) analysis, [1] provides an excellent reference.

This paper investigates several applications of statistical learn- ing methods designed to address common neuroimaging chal- lenges, including:

- *Data Preprocessing:*
  Techniques to clean and prepare noisy fMRI data for analysis.
- *Model Selection:*
  Strategies for choosing and optimizing machine learning models to suit the unique characteristics of neuroimaging datasets.
- *Performance Evaluation:*
  Metrics and methods for as- sessing the interpretability and predictive power of the models.

Moreover, the discussion extends to the internal mechanics of various learning techniques, providing insights into their suitability for neuroimaging applications. The contributions of this paper are twofold: it offers a practical demonstration of applying scikit-learn to neuroimaging data and presents guidelines for adapting general-purpose AI techniques to the specific challenges encountered in neuroscience research.

Overall, by facilitating the integration of advanced AI methods with domain-specific preprocessing, this work aims to enhance the interpretability and robustness of neuroimaging analyses, ultimately contributing to a deeper understanding of brain function.

II. TOOLS AND FRAMEWORKS FOR NEUROIMAGING ANALYSIS

A. Essential Python Libraries for Neuroimaging

Python has rapidly become a cornerstone in the neuroimaging data analysis landscape, largely due to its well-developed sci- entific computing stack. Several specialized libraries have been designed to facilitate neuroimaging research, enabling efficient data manipulation, visualization, and statistical analysis. Key libraries utilized in this study include:

➢ *NumPy:*

NumPy is a core Python library for numerical computing. It provides the ndarray type for handling multi-dimensional arrays efficiently, supporting key oper- ations like matrix multiplication and broadcasting. Many scientific libraries, including *scikit-learn*, use NumPy arrays as their main data structure.

➢ *SciPy:*

Built on NumPy, SciPy offers advanced math- ematical functions for linear algebra, optimization, and signal processing. By interfacing with optimized com- piled libraries such as BLAS, Arpack, and MKL, SciPy ensures high-performance computations, which are vital for neuroimaging analyses.

➢ *Matplotlib:*

A widely used visualization library, Mat- plotlib facilitates the creation of publication-quality plots. It is seamlessly integrated with the scientific Python ecosystem and is extensively used to visualize neu- roimaging data, from time-series plots to complex brain activation maps [2].

➢ *Nibabel:*

This library provides robust tools for reading and writing various neuroimaging file formats, such as NIfTI and Analyze. It simplifies the handling of vol- umetric brain imaging data, allowing for efficient data manipulation and extraction.

➢ *Nilearn:*

Built on top of *scikit-learn*, Nilearn simpli- fies the application of machine learning techniques to neuroimaging datasets. It provides tools for statistical analysis, feature extraction, and visualization, making it particularly useful for functional MRI (fMRI) studies.

B. Scikit-learn and Its Role in Neuroimaging AI Applications

Scikit-learn is a widely used, general-purpose machine learn- ing library in Python, offering efficient implementations of a variety of machine learning algorithms. Due to its intuitive API and extensive documentation, *scikit-learn* is highly accessible to researchers in neuroscience who may not have extensive expertise in artificial intelligence. Some key advantages of *scikit-learn* include:

➢ *Extbfversatility:*

The library supports a broad range of su- pervised and unsupervised learning techniques, including regression, classification, clustering, and dimensionality reduction.

➢ *Extbfinteroperability*:

As part of the broader Python scien- tific ecosystem, *scikit-learn* integrates well with libraries like NumPy, SciPy, and Pandas.

➢ *Extbfease of Use:*

Unlike deep learning frameworks that often require extensive tuning, *scikit-learn* is designed for rapid prototyping, making it suitable for exploratory analysis in neuroimaging.

➢ *Extbfmodular Structure:*

The library’s design allows users to build custom machine learning pipelines, optimiz- ing preprocessing and model selection for specific neu- roimaging tasks.

Although other machine learning frameworks, such as Ten- sorFlow and PyTorch, offer deep learning capabilities, they often require more computational resources and complex tun- ing. Furthermore, alternative AI packages like Weka [3] and PyMVPA [4] cater to specific niches but may not offer the same level of flexibility and accessibility as *scikit-learn.*

C. Core Concepts in Scikit-learn

In *scikit-learn*, data is structured as 2D arrays (matrices), where rows represent individual samples and columns rep- resent features. This uniform structure allows for flexibility in applying various algorithms to neuroimaging datasets. The primary components of *scikit-learn* are:

➢ *Estimators:*

These objects implement the fit method, allowing models to learn from data. Examples include classifiers (e.g., Support Vector Machines) and regressors (e.g., Ridge Regression).

➢ *Predictors:*

A subset of estimators that implement the predict method, enabling them to make predictions on new data. Classification and regression models fall into this category.

➢ *Transformers:*

These objects implement the transform method to preprocess data, such as standardization (e.g., StandardScaler), feature selection, or dimensionality reduction (e.g., PCA). If a transformation is invertible, the inverse_transform method is also provided.

D. Ensuring Robust Model Evaluation

A major challenge in neuroimaging machine learning is over- fitting, where a model excels on training data but performs poorly on new data. Cross-validation is used to help prevent this issue:

➢ *K-Fold Cross-Validation:*

The dataset is partitioned into k subsets, with each subset serving as a validation set while the remaining $k-1$ subsets are used for train- ing. The process is repeated k times, and the average performance score provides a robust estimate of model generalization.

➢ *Grid Search and Hyperparameter Tuning:*

The GridSearchCV class automates the selection of op- timal hyperparameters by evaluating multiple parameter combinations using cross-validation. This paper utilizes grid search to fine-tune the regularization coefficient in the classification of neuroimaging data (see Section V).

Through the modular approach of *scikit-learn*, researchers can systematically preprocess neuroimaging data, apply various learning techniques, and rigorously evaluate model perfor- mance. The following sections delve into specific applications of these methods in neuroimaging analysis.

III. DATA READINESS: FROM X-RAY VOLUMES TO AN INFORMATION MATRIX

Prior to applying measurable learning strategies to neuroimag- ing information, it is fundamental to perform standard prepro- cessing steps. For utilitarian attractive reverberation imaging (fMRI), regular preprocessing steps incorporate movement revision, cut timing remedy, co-enlistment with a physical picture, and standardization to a standard layout, like the Montreal Neurological Foundation (MNI) space. Normal pro- gramming bundles utilized for these errands are SPM [5] and FSL, with a Python interface accessible through the nipype library [6]. In this segment,The process of shaping preprocessed neuroimaging data into a format suitable for input into scikit-learn is outlined. The primary goal is to create a data matrix, denoted as X, and optionally, a target variable y for prediction.

A. Spatial Resampling

Neuroimaging information are regularly put away in NIfTI de- sign as four-layered information (3D spatial aspects with time series for each voxel). These information likewise incorporate a relative change framework, which relates voxel records to world directions. While working with different subjects, the information for every individual is enlisted to a typical layout (e.g., MNI or Talairach), adjusting them to a common relative change during preprocessing.

The relative grid can catch anisotropy in the information, where the distance between two voxels may differ depending upon the course. This spatial data is significant for calcula- tions that use the spatial design of the information, like the searchlight examination method.

To perform picture resampling and change the spatial goal of the information, the scipy.ndimage.affine_transform capability can be used. Resampling includes an introduction, which might adjust the information, and, subsequently, ought to be done carefully. Downsampling is a typical procedure to diminish the information size for handling, with normal goals being 2mm or 3mm. Be that as it may, headways in MR material science are prompting higher spatial goal checks. The relative grid can likewise encode scaling factors for each aspect.

B. Signal Cleaning

Neuroimaging data are inherently noisy due to the complex and indirect nature of data acquisition. A low signal-to-noise ratio (SNR) can obscure meaningful brain activity, thereby hindering subsequent analysis. To address this, several prepro- cessing steps are employed to clean the signal by removing trends and artifacts. The key steps in signal cleaning include:

Detrending: Detrending involves the removal of sys- tematic linear (or polynomial) trends from the time series of each voxel. Since the absolute intensity of a voxel is less informative than its temporal variation, detrending helps isolate the relevant fluctuations in brain activity. In prac- tice, detrending is commonly implemented using SciPy’s scipy.signal.detrend function, which subtracts the best-fit line from the data. This step is crucial to eliminate scanner drifts and other slow fluctuations that could bias subsequent analyses.

Normalization: Normalization scales the time series data such that the variance of each voxel is standardized, typi- cally to 1. This process is essential because many machine learning algorithms assume that all features contribute equally; unnormalized data can lead to certain voxels dominating the analysis due to their larger variance. By normalizing the data, each voxel is placed on an equal footing, ensuring that the algorithm’s performance is not skewed by differences in intensity ranges.

Frequency Filtering: Physiological noise (e.g., heart rate, respiration) and scanner-related artifacts often introduce un- wanted high-frequency and low-frequency signals in the data. Frequency filtering targets these components by removing frequencies outside the band of interest. Techniques such as the Fourier Transform (via scipy.fftpack.fft) and Butterworth filters (using scipy.signal.butter) can be applied to selectively retain the frequency components that are most likely to reflect neural activity, while discarding the rest.

C. Transformation from 4D Images to a 2D Array: Masking

Neuroimaging datasets are typically acquired as 4D images, with three spatial dimensions and one temporal (or trial-based) dimension. However, most machine learning algorithms, such as those in scikit-learn, require data to be organized in a 2D matrix format, where each row represents a sample and each column a feature.

➢ *Brain Masking:*

To convert 4D images into a 2D matrix while retaining only the informative regions, a *brain mask* is applied. A brain mask is a binary image that identifies voxels within the brain, effectively excluding areas that contain only noise or non-brain tissue. This masking process reduces the dimensionality of the data by focusing the analysis on voxels that are expected to contain relevant signal.

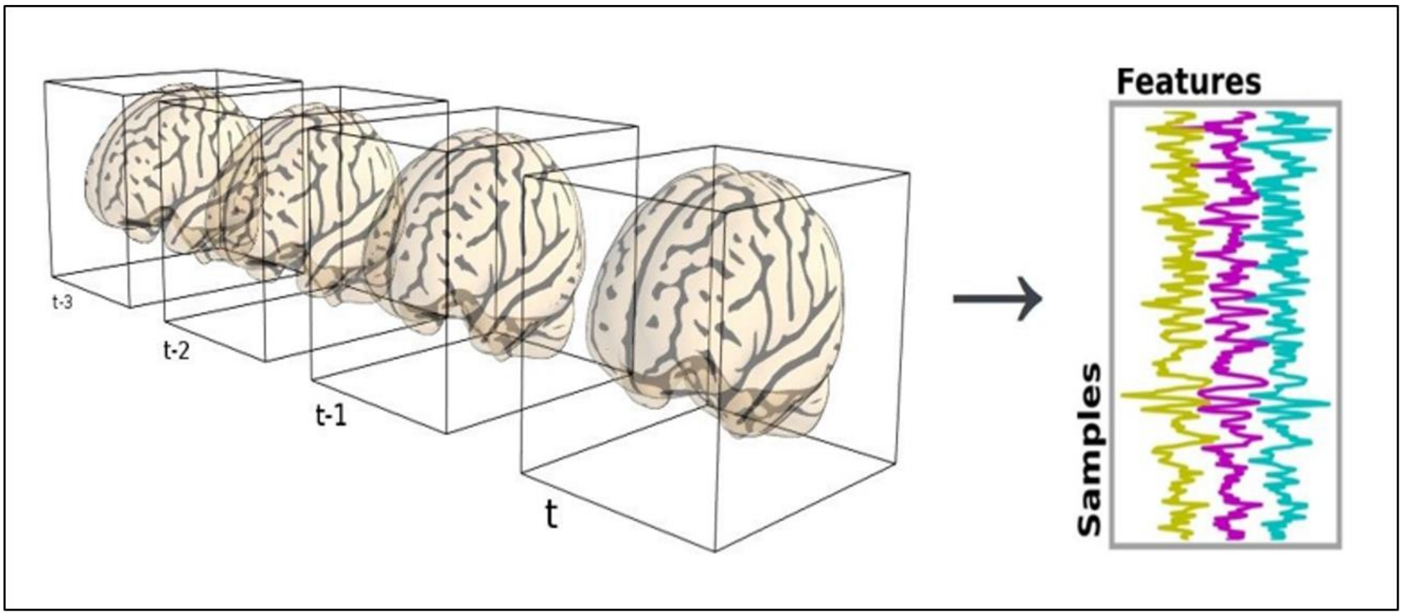

Fig 1 Conversion of Brain Scans into a 2D Data Matrix. Uninformative Voxels are Removed using a Brain Mask.

Applying the Mask: After the brain mask is generated or provided, it is applied to the functional data using NumPy’s advanced indexing with boolean arrays. The resulting 2D matrix, conventionally denoted as X, contains only the time series data of the voxels within the mask. The following code snippets illustrate this process:

Listing 1: Loading Mask and Functional Data import nibabel as nib import numpy as np # Load the brain mask and functional (fMRI) data mask = nib.load(’mask.nii’).get_fdata() func_data = nib.load(’epi.nii’).get_fdata() # Convert the mask to a boolean array mask = mask.astype(bool) Once the data are loaded, the mask is applied to extract the relevant voxel time series:

Listing 2: Applying Mask and Extracting Data # Apply the brain mask: each row corresponds to a time point, each column to a voxel X = func_data[mask].T # Optionally, restore the data structure by mapping the masked data back to the original volume unmasked_data = np.zeros_like(func_data) unmasked_data[mask] = X.T

D. *Data Visualization*

Visualization plays a critical role in neuroimaging by pro- viding intuitive insights into both anatomical and functional aspects of the brain. Commonly, Regions of Interest (ROIs) are visualized on axial slices, with activation maps overlaid on an anatomical background.

E. *Activation Map Overlay*

An activation map highlights regions of the brain that exhibit significant activity changes. To visualize such maps:

- An anatomical (structural) image is loaded to serve as a background.
- An activation map is generated, either from statistical tests or by thresholding.
- The activation map is overlaid on the anatomical image to identify areas of significant activation

The code below demonstrates how to overlay an activation map onto an anatomical slice using Matplotlib. In this exam- ple, a synthetic activation map is created by thresholding the anatomical image to isolate high-intensity voxels.

Listing 3: Visualization of the Activation Map import matplotlib.pyplot as plt import numpy as np import nibabel as nib # Load the anatomical background image bg_img = nib.load(’bg.nii.gz’) bg = bg_img.get_fdata() # Create a synthetic activation map by thresholding values above 6000 activation_map = bg.copy() activation_map[activation_map < 6000] = 0 # Display the anatomical background on a selected axial slice plt.imshow(bg[..., 10].T, origin=’lower’, cmap=’gray ’) # Overlay the activation map using a hot colormap for visual contrast masked_activation = np.ma.masked_equal(activation_map, 0) plt.imshow(masked_activation[..., 10].T, origin=’ lower’, interpolation=’nearest’, cmap=’hot’) # Remove axes for a cleaner display and render the plot plt.axis(’off’) plt.show()

F. *Advanced Visualization Techniques*

Beyond the basic overlay, several enhancements can improve visualization:

➢ *Anatomical Templates:*
Incorporate standard brain tem- plates to provide spatial context.

➢ *Colormap Customization:*
Experiment with different colormaps to maximize contrast and readability.

➢ *Interactive Tools:*
Use interactive visualization libraries, such as NiPy's plot_map, for dynamic exploration of activation patterns.

The integration of an anatomical background with activation overlays is commonly facilitated by NumPy's masked array functionality (numpy.ma.masked_array), which handles zero-valued or non-significant data gracefully. Such visualiza- tions are critical for interpreting the spatial distribution of brain activity and for identifying regions that merit further investigation.

IV. DECODING THE PSYCHOLOGICAL PORTRAYAL OF ITEMS IN THE BRAIN

Decoding in neuroimaging refers to the process of constructing predictive models that infer cognitive or phenotypic states from brain imaging data. This framework stands in contrast to encoding, where the goal is to predict brain activity from external stimulus descriptors. Decoding techniques are pivotal for understanding how specific mental representations are instantiated in neural activity and have important applications in both basic research and clinical contexts.

A seminal demonstration of decoding is provided by Haxby et al. [7]. In that study, visual stimuli spanning eight distinct categories were presented to six subjects across 12 sessions, and the objective was to classify the stimulus category based on the corresponding fMRI data. This work established a benchmark that has since been extended and analyzed in numerous studies [8], [9], [10], [4]. For the purposes of this paper, the analysis is simplified by focusing on data from a single subject and reducing the number of stimulus categories to two (faces and houses).

In this context, a *target* variable *y* represents the stimulus category, thereby formulating the problem as a supervised classification task. This is in contrast to regression problems where *y* would assume continuous values (e.g., age or reaction times).

A. Classification with Univariate Feature Selection and Linear SVM

The high dimensionality of fMRI data (typically tens of thousands of voxels) poses significant challenges due to the *curse of dimensionality*, especially when the number of sam- ples (time points) is limited. To address this, a two-step approach is employed: univariate feature selection followed by classification using a linear Support Vector Machine (SVM).

➢ *Feature Selection::*
Each voxel is evaluated independently using statistical tests (such as an F-test) to determine its discriminative power. By retaining only the top-ranking vox- els, the dimensionality of the data is reduced, which helps mitigate overfitting and enhances the classifier's performance.

Formally, if the full set of features is denoted by F, a subset

$F_{\text{selected}} \subset F$ is chosen such that

$$F_{\text{selected}} = \{f_i \in F \ : \ p(f_i) < \alpha\},$$

where $p(f_i)$ is the p-value corresponding to voxel f_i and α is a predefined significance threshold.

➢ *Linear SVM::*
Once the feature space has been reduced, a linear SVM is applied. The SVM seeks to identify an optimal hyperplane that separates the two classes. Its decision function is given by

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b,$$

Where **w** is the weight vector, **x** is the feature vector (repre- senting the selected voxel time series), and *b* is the bias term. The linear SVM is particularly effective in high-dimensional spaces and tends to yield robust classification performance even with relatively few training samples.

A. Searchlight Analysis

Searchlight analysis is a spatially localized decoding technique that provides a fine-grained map of informational content across the brain. In this method, a small spherical region (the *searchlight*) is defined around each voxel. Within each searchlight, a classifier (typically an SVM) is trained using only the data confined to that region. The cross-validated prediction accuracy of the classifier is then assigned to the central voxel of the searchlight.

This approach allows for the detailed mapping of regions where local patterns of activity contain discriminative infor- mation about the stimulus. However, because the analysis requires training a classifier for each voxel (or overlapping set of voxels), it is computationally intensive. Nevertheless, the spatial specificity provided by searchlight analysis makes it a valuable tool for understanding the localized contributions of different brain regions to cognitive processes.

B. Results

The decoding analysis reveals that voxels with the highest classifier weights are predominantly located in brain regions known to be involved in processing the stimuli—for example, regions responsive to faces or houses. While the global SVM classifier provides an overall ranking of brain regions based on their discriminative power, the searchlight analysis yields a spatial map that details the local decoding performance.

The implementation of this decoding model is encapsulated in the script haxby_decoding.py, which is publicly avail- able on GitHub. This script encompasses all stages of the analysis, from data preprocessing and feature selection to classifier training and visualization of results. One of the key strengths of the scikit-learn toolkit is its modular design; for instance, the linear SVM classifier can be easily replaced with alternative models, such as ElasticNet or Logistic Regression, by modifying only a single line of code. Similarly, non-linear classifiers such as Gaussian Naive Bayes can be substituted with minimal changes (e.g., replacing the attribute coef_ with theta_).

Overall, the decoding methods presented here demonstrate how predictive models can elucidate the relationship between brain activity and cognitive states, providing insights that may drive further advances in neuroimaging research.

V. ENCODING CEREBRUM ACTION AND DECIPHERING VISUAL STIMULI

Previous studies have shown that brain activity in the visual cortex can predict the type of visual stimulus shown to a subject [7]. In this section, we extend this work by linking the presented visual stimulus directly to the corresponding FMRI activity. We use a set of 10×10 binary images. In the original study, random binary images were used for training,

while structured images (e.g., shapes and letters) were used for testing. For simplicity, our work focuses only on the training set, and cross-validation is used to obtain reliable performance metrics on new data.

Relationship between visual stimulus pixels and brain voxels is examined using two complementary frameworks:

- **Decoding:** Reconstructing the visual stimulus from recorded brain activity.
- **Encoding:** Predicting fMRI activity from visual stimulus descriptors.

Decoding Visual Stimuli from Brain Activity

In the decoding framework, the objective is to reconstruct the binary visual stimulus that was presented to the subject based on the corresponding brain activity. Given the binary nature of the stimuli, the problem is formulated as a classification task. However, while binary classification is well suited for discrete stimuli, it may not directly extend to cases involving continuous stimuli such as grayscale images.

In this study, I compare several models to evaluate their performance in reconstructing the stimulus:

- An ℓ_2-regularized Support Vector Machine (SVM), as previously applied in similar experiments.
- A logistic regression model, which provides a proba- bilistic interpretation of the classification decision.
- An ℓ_1-regularized SVM, which promotes sparsity in the model parameters by utilizing a squared hinge loss.

Feature selection plays a critical role in the decoding pipeline due to the high dimensionality of fMRI data. Typically, a brain mask is applied to reduce the feature space to a subset of ap- proximately 40,000 voxels over 1,400 time points (samples). In such a high-dimensional setting, univariate feature selection is essential to mitigate the *curse of dimensionality*. Specifically, a statistical test (e.g., an F-test) is applied to each voxel to determine its discriminative power, and only the top features are retained for classification.

The following Python code snippet illustrates a pipeline that first performs univariate feature selection and then applies logistic regression with an ℓ_1 penalty to encourage sparsity:

Listing 4: Pipeline with Logistic Regression pipeline_LR = Pipeline([('feature_selection', SelectKBest(f_classif, 500)), ('classifier', LogisticRegression(penalty='l1', C=0.05))])

This pipeline first selects the top 500 features based on the F-test statistic and then trains a logistic regression classifier on the reduced feature set. The use of an ℓ_1 penalty helps in identifying the most relevant voxels by driving the coefficients of less informative features towards zero.

Encoding fMRI Data from Stimulus Descriptors

In contrast to decoding, the encoding framework aims to pre- dict the fMRI response given the visual stimulus descriptors. This approach quantifies the extent to which the stimulus can explain the variability in each voxel's signal. A common metric for evaluating encoding models is the predictive r^2 score, which measures the proportion of variance in the fMRI signal that is captured by the model relative to a baseline constant model.

For the encoding task, ridge regression is employed. Ridge regression, which applies an ℓ_2 penalty, is well-suited for handling multicollinearity and stabilizing estimates in high- dimensional spaces. The model is trained using cross-validation to ensure that the performance metric is robust against overfitting. The following Python code snippet demon- strates the evaluation process via cross-validation:

Listing 5: Ridge Regression Evaluation for Encoding # For each voxel, compute the predictive rˆ2 score using cross-validation scores = [] for train, test in cv: # Fit the ridge regression model on training data model.fit(X_train[train], y_train[train]) # Predict on test data pred = model.predict(X_train[test]) # Compute the rˆ2 score for the current voxel score = 1.0 - np.sum((y_train[test] - pred)**2) / \ np.sum((y_train[test] - np.mean(y_train[test]))**2) scores.append(score) mean_scores = np.mean(scores, axis=0)

Alternative regression methods, such as Lasso regression, can be used if a sparser solution is desired. However, Lasso may introduce additional computational complexity due to the need for tuning the regularization parameter.

➢ *Receptive Fields and Sparse Regression:*

Due to the retino- topic organization of early visual areas, only a small subset of stimulus pixels is typically responsible for driving the activity in each voxel of the primary visual cortex. Neighboring voxels are expected to be influenced by adjacent regions of the stimulus. To identify these localized relationships, sparse linear regression techniques are employed. One effective method is the LassoLarsCV estimator, which utilizes the Least Angle Regression (LARS) algorithm combined with cross-validation to select a sparse set of stimulus features that best explain the voxel responses. This approach effectively reveals the *receptive fields* of neurons in the visual cortex.

➢ *Results*

Figure 2 summarizes the results from both the decoding and encoding analyses. In the decoding experiment, Figures 2(a) and (c) display classifier weight maps obtained from logistic regression and SVM, respectively, focusing on voxels in V1 and adjacent retinotopic areas. Figures 2(b) and (d) present the reconstruction accuracy per pixel for the respective models. The results indicate that both classifiers yield similar per- formance, with notably higher reconstruction accuracy in the foveal region, likely due to its denser neuronal population.

In the encoding experiment, Figure 2(e) illustrates the recep- tive fields corresponding to voxels with the highest predictive scores, while Figure 2(f) shows reconstruction accuracy as a function of stimulus pixel position. The convergence of results from both encoding and decoding analyses underscores a consistent relationship between specific stimulus pixels and brain voxel responses, thereby reinforcing the validity of the methodologies.

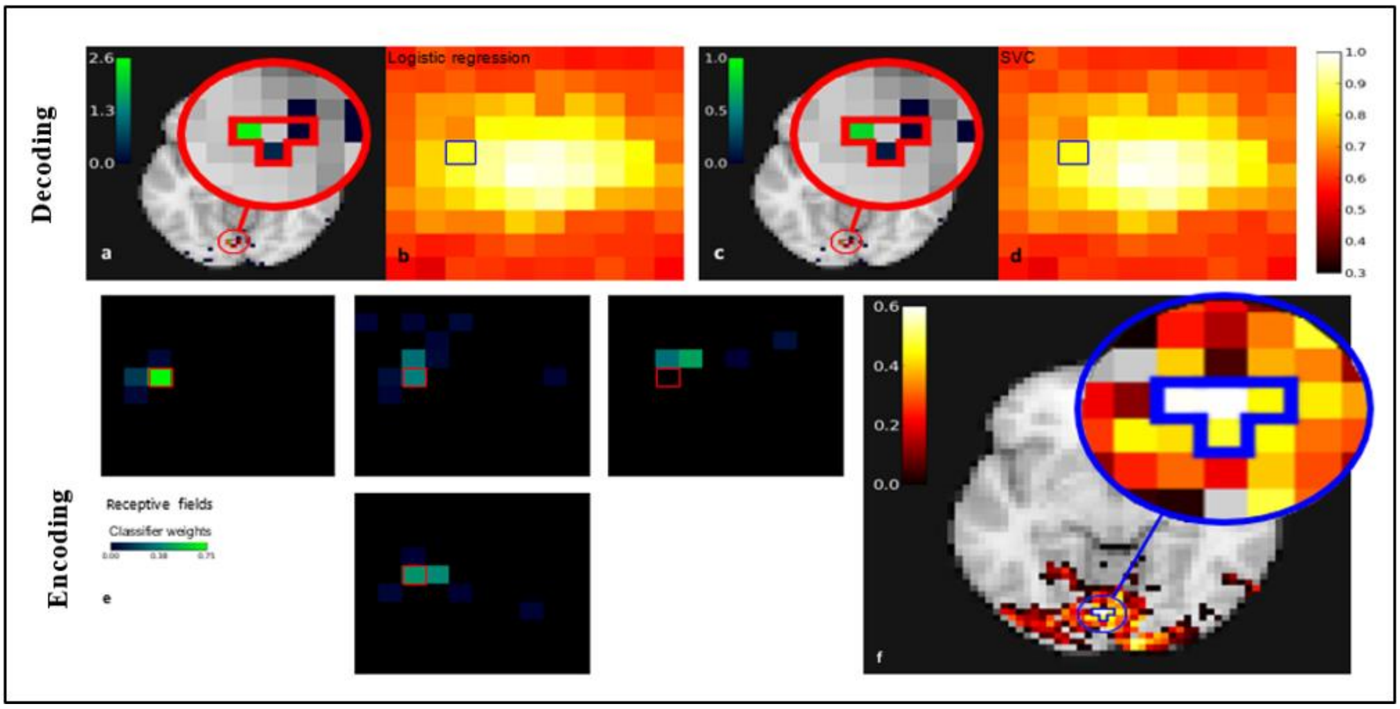

Fig 2 Miyawaki Results. Top (Decoding): (a)/(c) Classifier Weights from Logistic Regression/SVM; (b)/(d) Reconstruction Accuracy per Pixel. Bottom (Encoding): (e) Receptive fields and (f) Reconstruction Accuracy by Stimulus pixel.

The implementation of these models is encapsulated in the script haxby_decoding.py, which is available on GitHub. This script integrates all stages—from data preprocessing and feature selection to classifier training and visualiza- tion—demonstrating the versatility and modularity of the scikit-learn toolkit. Owing to this modular design, switching between different classifiers (e.g., from SVM to ElasticNet or Logistic Regression) requires minimal modifications to the code, thereby facilitating rapid experimentation and model optimization.

Overall, the integrated decoding and encoding analyses not only demonstrate the feasibility of predicting both visual stimuli and fMRI responses but also provide critical insights into the spatial and functional organization of the visual cortex. Such insights can pave the way for future applications in cognitive neuroscience and brain-computer interfacing.

VI. RESTING-STATE AND PRACTICAL AVAILABILITY ANALYSIS

Indeed, even without outside conduct or clinical factors, in- vestigating the construction of mind cues can give critical experiences. Truth be told, [11] showed that cerebrum en- actment displays intelligible spatial examples during resting states. These associated voxel actuations structure practical organizations that line up with notable undertaking related networks.

Biomarkers found through prescient demonstrating of resting- state fMRI could be especially valuable in situations where subjects can't perform explicit undertakings. In this review,A dataset comprising resting-state fMRI data from both control subjects and ADHD (Attention Deficit Hyperactivity Disor- der) patients is examined. These subjects were studied with minimal specific tasks to capture their brain activity at rest.

Resting-state fMRI data is considered unlabeled, as brain activity cannot be directly associated with a particular outcome variable. This type of problem is classified as unsupervised learning in machine learning. To extract meaningful networks or regions, techniques that group similar voxels based on their time series are employed. A widely used technique in neuroimaging for this purpose is Independent Component Analysis (ICA), which serves as the focus of the first model. Additionally, the use of clustering methods to identify nearly homogeneous regions will be demonstrated.

A. Independent Part Investigation (ICA) to Concentrate Net- works

ICA is an outwardly hindered source segment methodology that hopes to crumble a multivariate sign into free parts by growing their non-Gaussianity. A model delineation of ICA, where ICA is used to confine covering voices from various speakers using beneficiaries put around a room.

➢ *ICA in Neuroimaging:*

ICA is seen as the standard system for isolating organizations from resting-state fMRI data. A couple of systems have been made to combine ICA re- sults across various subjects. For example, [12] proposed a dimensionality decline approach (using PCA), followed by association of time series, which is the strategy displayed in this model. Then again, use dimensionality decline got together with authorized relationship assessment to add up to data from different subjects. The FSL suite consolidates Melodic [13], which utilizes an association procedure, but it isn't distinct in this work.

➢ *Application:*

Prior to applying ICA, the information goes through preprocessing steps, including focusing and detrend- ing of the time series. This guarantees that straight patterns are not caught by ICA. The FastICA calculation is then applied to the subsequent time series. Utilizing the scikit-learn library, ICA is direct to carry out because of the transformer idea. The information network is rendered for spatial ICA, where voxels are considered as irregular factors and the time focuses are treated as fixed. The subsequent parts address different sign designs, for example, commotion and resting-state orga- nizations. For the investigation,The focus is on extracting only 10 components, which represent the primary signal patterns.

➢ *Results:*

Figure 3 illustrates a comparison between a basic concat-ICA approach implemented in this work and more advanced multi-subject ICA methods. Although both tech- niques were executed using scikit-learn, the detailed imple- mentation of CanICA is not provided here. For demonstration purposes, the default mode network—a prominent resting- state network—is highlighted. Initial observations suggest that both CanICA and Melodic's concat-ICA produce results with reduced noise and yield comparable outcomes, even though definitive conclusions cannot be drawn from a single example. Scikit-learn gives a few other framework disintegration strate- gies under the 'sklearn.decomposition' module. One option in contrast to ICA is word reference realizing, which applies a ℓ_1 regularization to the separated parts. This outcomes in sparser and more minimized parts than those got with ICA, which commonly address full-mind action and require thresholding for significant understanding.

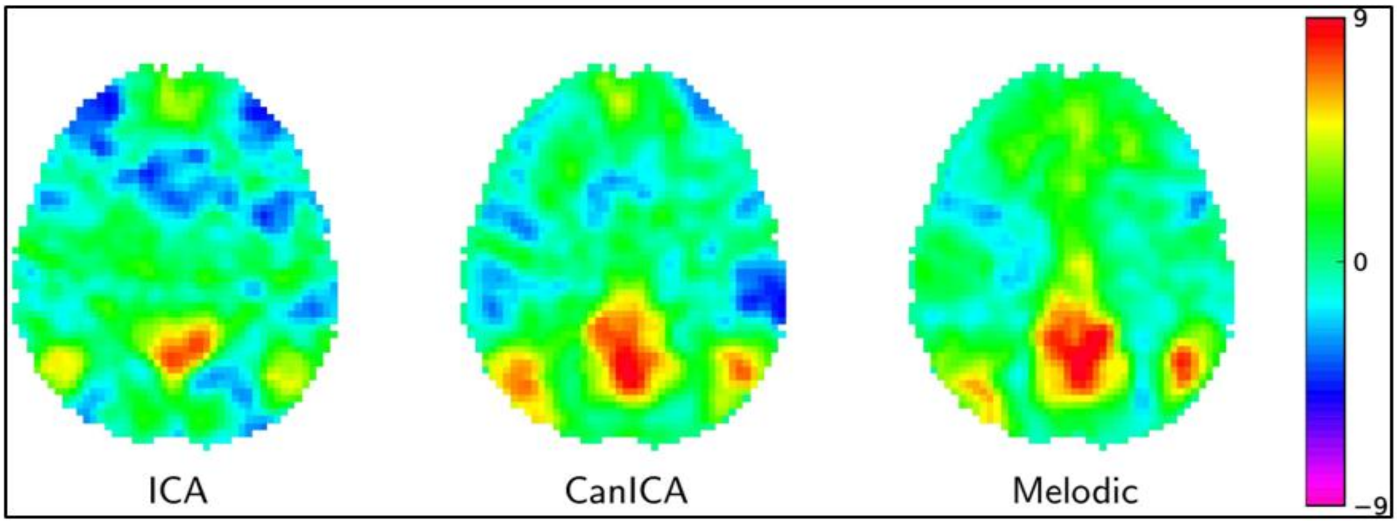

Fig 3 Default Mode Network Components Obtained by three Methods: (left) simple Concat-ICA, (Middle) CanICA (Nilearn), and (right) Melodic's Concat-ICA. Data Were nor- Malized for Visualization.

B. *Learning Practically Homogeneous Locales with Cluster- ing*

From an AI viewpoint, bunching strategies bunch tests into groups by expanding a comparability measure between the examples inside each bunch. When applied to voxels from a useful mind picture, this action can be founded on utilitarian comparability, bringing about bunches of voxels that address practically homogeneous locales.

➢ *Approaches:*

Different grouping approaches exist, each with its own assets and impediments. Many require the quan- tity of groups to be foreordained, a decision that relies upon the application. A bigger number of groups gives a better portrayal of the information, saving a greater amount of the first sign, yet additionally increments model intricacy. Some bunching techniques can integrate spatial data, delivering spatially ad- joining groups, otherwise called packages. In this segment, Two basic and quick clustering approaches are depicted.

➢ *Ward Clustering:*

utilizes a base up various leveled ap- proach where voxels are logically gathered into groups. In scikit-learn, primary data can be presented through a network chart, which compels the converges to just adjoining voxels, prompting coterminous packages.

➢ *K-Means Clustering:*

partitions the data into clusters by assigning each voxel to the nearest centroid, effectively group- ing similar data points. However, because K-Means does not inherently enforce spatial coherence, spatial smoothing is often applied prior to clustering to improve the meaningfulness of the results.

To apply clustering algorithms effectively, the data must first be prepared using standard preprocessing steps to create an appropriate data matrix. Since both Ward clustering and K- Means rely on second-order statistical properties, applying PCA for dimensionality reduction while preserving these statistics can enhance performance. It is important to recognize that clustering methods are designed to group samples—in this case, the aim is to cluster voxels. Consequently, when the data matrix is organized as (time points × voxels), it must be reshaped accordingly before being used with scikit- learn’s clustering estimators. While scikit-learn offers the WardAgglomeration transformer for feature agglomera- tion via Ward clustering, a direct equivalent for K-Means is not provided.

➢ *Results:*

The clustering outcomes are presented in Figure 4. Although clustering highlights large-scale brain structures, such as the calcarine sulcus (see Figure 4a), it does not necessarily yield sharply defined anatomical regions. Instead, clustering serves as a dimensionality reduction tool that groups similar voxels, providing a coarse overview of the data struc- ture. K-Means, which does not enforce spatial contiguity, may produce numerous small clusters. In contrast, Ward clustering, by imposing spatial constraints, naturally yields more con- tiguous regions. Being a bottom-up method, Ward clustering generally performs better when a high number of clusters is considered. Although scikit-learn provides several clustering techniques, selecting the optimal one for fMRI time-series analysis necessitates a detailed understanding of the specific application.

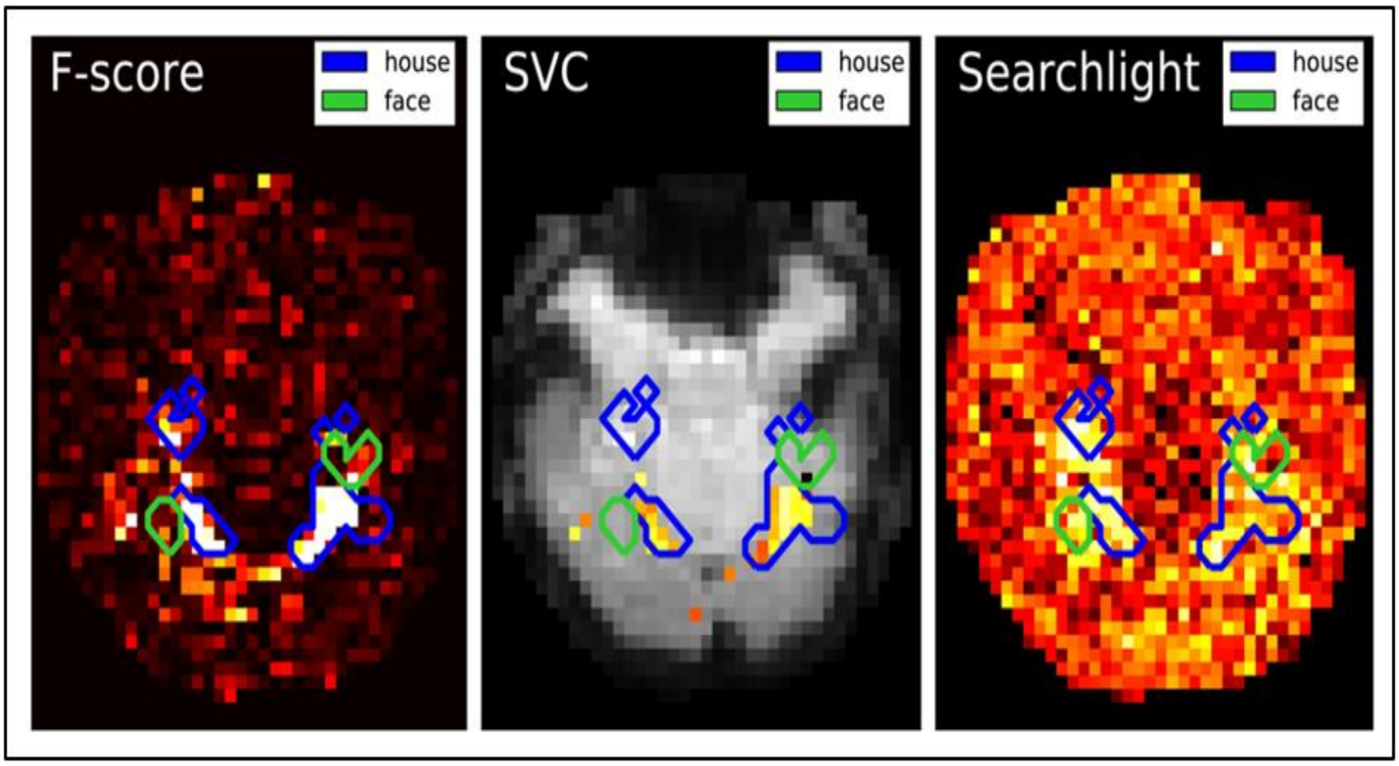

Fig 4 Brain Parcellations Obtained via Clustering. Colors are Assigned Arbitrarily.

VII. CONCLUSION

This study demonstrated the use of AI methods on fMRI data by leveraging the scikit-learn Python toolkit to address key challenges in neuroscience. Our work shows that supervised approaches, such as encoding and decoding, can effectively link brain images with external information, while unsuper- vised methods uncover natural patterns in resting-state data. Although the Python implementations are straightforward and user-friendly, challenges remain in data preprocessing, model selection, and result interpretation. Integrating scikit-learn with neuroimaging-specific libraries like Nilearn is essential for streamlining these processes.

The techniques presented here are just a small sample of the wide range of applications in neuroimaging. By combining scikit-learn's diverse algorithms with customized preprocess- ing steps, researchers can reveal new insights—such as con- nectivity patterns obtained through sparse inverse covariance estimation. This seamless integration of AI tools and neu- roimaging workflows promises to drive further breakthroughs in understanding brain function.

Future efforts should focus on refining these methods and exploring additional models and preprocessing strategies. The integration of general-purpose AI toolkits with domain-specific approaches holds great promise for advancing both scientific research and clinical applications.

REFERENCES

[1]. T. Hastie, R. Tibshirani, and J. J. H. Friedman, *The elements of statistical learning.* Springer New York, 2001, vol. 1.
[2]. J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
[3]. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.
[4]. M. Hanke, Y. O. Halchenko, P. B. Sederberg, S. J. Hanson, J. V. Haxby, and S. Pollmann, "PyMVPA: A python toolbox for multivariate pattern analysis of fMRI data," *Neuroinformatics*, vol. 7, no. 1, pp. 37–53, 2009.
[5]. K. Friston, *Statistical parametric mapping: the analysis of functional brain images*. Academic Press, 2007.
[6]. K. Gorgolewski, C. D. Burns, C. Madison, D. Clark, Y. O. Halchenko, M. L. Waskom, and S. S. Ghosh, "Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python." *Front Neuroinform*, vol. 5, 08 2011. [Online]. Available: http://dx.doi.org/10.3389/fninf.2011.00013
[7]. J. V. Haxby, I. M. Gobbini, M. L. Furey, A. Ishai, J. L. Schouten, and
[8]. P. Pietrini, "Distributed and overlapping representations of faces and objects in ventral temporal cortex," *Science*, vol. 293, no. 5539, p. 2425, 2001.
[9]. S. J. Hanson, T. Matsuka, and J. V. Haxby, "Combinatorial codes in ventral temporal lobe for object recognition: Haxby (2001) revisited: is there a "face" area?" *Neuroimage*, vol. 23, no. 1, pp. 156–166, 2004.
[10]. Detre, S. Polyn, C. Moore, V. Natu, B. Singer, J. Cohen, J. Haxby, and
[11]. K. Norman, "The multi-voxel pattern analysis (mvpa) toolbox," in *Poster presented at the Annual Meeting of the Organization for Human Brain Mapping (Florence, Italy). Available at: http://www. csbmb. princeton. edu/mvpa*, 2006.
[12]. S. J. Hanson and Y. O. Halchenko, "Brain reading using full brain support vector machines for object recognition: there is no "face" identification area," *Neural Computation*, vol. 20, no. 2, pp. 486–503, 2008.
[13]. Biswal, F. Zerrin Yetkin, V. Haughton, and J. Hyde, "Functional connectivity in the motor cortex of resting human brain using echo- planar MRI," *Magn Reson Med*, vol. 34, p. 53719, 1995.
[14]. V. D. Calhoun, T. Adali, G. D. Pearlson, and J. J. Pekar, "A method for making group inferences from fMRI data using independent component analysis." *Hum Brain Mapp*, vol. 14, p. 140, 2001.
[15]. F. Beckmann and S. M. Smith, "Probabilistic independent component analysis for functional magnetic resonance imaging," *Trans Med Im*, vol. 23, pp. 137–152, 2004.