

Scaling up ridge regression for brain encoding in a massive individual fMRI dataset

Sana Ahmadi^{1a,b,c}, Pierre Bellec^{2b,c}, Tristan Glatard^{3a}

^a*Department of Computer Science and Software Engineering, Concordia University, Montreal, QC, Canada*

^b*Université de Montréal, Montréal, QC, Canada*

^c*Centre de Recherche de L'Institut Universitaire de Gériatrie de Montréal, Montréal, Canada*

Abstract

Brain encoding with neuroimaging data is an established analysis aimed at predicting human brain activity directly from complex stimuli features such as movie frames. Typically, these features are the latent space representation from an artificial neural network, and the stimuli are image, audio or text inputs. Ridge regression is a popular prediction model for brain encoding due to its good out-of-sample generalization performance. However, training a ridge regression model can be highly time-consuming when dealing with large-scale deep functional magnetic resonance imaging (fMRI) datasets that include many space-time samples of brain activity. This paper evaluates different parallelization techniques to reduce the training time of brain encoding with ridge regression on the CNeuroMod Friends dataset, one of the largest deep fMRI resource currently available. With multi-threading, our results show that the Intel Math Kernel Library (MKL) significantly outperforms the OpenBLAS library, being 1.9 times faster using 32 threads on a single machine. Yet, the performance benefits of multi-threading are limited, and reached a plateau after 8 threads in our main experiment. We then evaluated the Dask multi-CPU implementation of ridge regression readily available in scikit-learn (MultiOutput), and we proposed a new “batch” version of Dask parallelization, motivated by a time complexity analysis. With this Batch-MultiOutput approach, batches of brain targets are processed in parallel across multiple machines, and multi-threading is applied concurrently to further accelerate computation within a batch. In line with our theoretical analysis,

¹Corresponding author, sana.ahmadi@mail.concordia.ca

²pierre.bellec@criugm.qc.ca

³tristan.glatard@concordia.ca

MultiOutput parallelization was found to be impractical, i.e., slower than multi-threading on a single machine. In contrast, the Batch-MultiOutput regression scaled well across compute nodes and threads, providing speed-ups of up to $33\times$ with 8 compute nodes and 32 threads compared to a single-threaded scikit-learn execution. Batch parallelization using Dask thus emerges as a scalable approach for brain encoding with ridge regression on high-performance computing systems using scikit-learn and large fMRI datasets. These conclusions likely apply as well to many other applications featuring ridge regression with a large number of targets.

Keywords: Brain Encoding, Ridge Regression, BLAS Libraries, Multi-threading , Dask distributed system

1. Introduction

The human brain is a computing system with billions of neurons as computing units. Cognitive neuroscience aims to discover functional principles of brain organization by leveraging large-scale neuroimaging data. One of the key methods used for this purpose is brain encoding [1], in which a model predicts brain responses directly from rich stimuli such as natural images or videos, using the internal representations of an artificial neural network as a feature space for prediction.

Among the regression methods used in brain encoding to predict brain activity, ridge regression [2] has become popular and well-accepted [3, 4, 5, 6, 7, 8, 9, 10, 11, 12] due to its two key features: 1) ridge regression tends to be generalizable to new stimuli and avoids overfitting, and 2) efficient implementations of ridge regression are available [13] which are less computationally intensive than other approaches.

For brain encoding of visual tasks, ridge regression is often applied to the activations produced by various neural networks architectures in response to visual stimuli, such as convolutional neural networks (CNN) and transformers [14, 15, 16, 17, 18, 19, 20]. For instance, in [12], the authors compared the activation of CNN units with brain response to a dynamic visual stimulus (movie frames) and found that these representations were able to accurately predict fMRI data collected with human subjects watching movies.

Even though linear algebra optimizations exist for ridge regression [13], the training process still requires compute-intensive matrix computations over the whole dataset. This computational cost is especially substantial for brain encoding models trained separately for each spatial measurement sample (voxel), as the num-

ber of voxels can range from tens to hundreds of thousands in a full brain fMRI acquisition with high spatial resolution. Thus, full brain encoding using ridge regression remains a challenge, even with modern computational resources.

Furthermore, the computational requirements of ridge regression are exacerbated by the need to train brain encoding tasks on large datasets. Indeed, finding the correlation between the huge feature space of natural images and brain activity requires to explore a large space of visual stimuli [21]. Over the past few years, the quantity and quality of fMRI datasets have increased rapidly in terms of the number of human subjects, the number of scanning hours available for each subject, as well as spatio-temporal resolution. In particular, datasets such as BOLD5000 [22], Natural Scenes Dataset (NSD)[23] provide so-called deep fMRI datasets, with long scanning time for a few subjects and an extensive stimuli space to properly estimate the generalization of brain encoding to different types of stimuli, e.g. images from many different categories. Training purely individual brain models also by-pass the challenges of modelling inter-individual variations in brain organization, which is substantial [24]. As a consequence of increased spatial resolution and volume of time samples available for a single subject with the advance of simultaneous multislice fMRI REF, there is thus an urgent need to understand the efficiency of various implementations of ridge regression for brain encoding with large fMRI datasets.

The CNeuroMod research group [25] has released the largest fMRI dataset for individual brain modeling currently available, featuring up to 200 hours of fMRI data per subject (N=6). The CNeuromod dataset provides an opportunity to train complex brain encoding models based on artificial neural networks, but also raises substantially the computational costs of brain encoding. This work investigates several parallelization techniques for ridge regression, using the CNeuroMod Friends dataset to predict brain activity from video stimuli. We focus on a standard brain encoding pipeline using an established pretrained network (VGG16), and we used the scikit-learn library [26] for brain encoding, that provides efficient implementations of various machine-learning models, including ridge regression.

We benchmarked the efficiency of different types of parallelization, namely multi-threading (multiple cores on a single CPU) and multi-processing (distributing computations across multiple CPUs in a high performance computing environment). For multi-threading, scikit-learn can leverage the BLAS (Basic Linear Algebra Subprograms) specification for linear algebra implemented using the open-source OpenBLAS library [27] or the proprietary Intel oneAPI Math Kernel Library (MKL) [28]. Both of these linear algebra libraries support multi-threading on a single CPU. Moreover, scikit-learn models rely on the [Joblib library](#) to in-

terface with various parallelization backends including Dask [29], which can be used to distribute computations across multiple compute nodes. Specifically, we utilized scikit-learn’s MultiOutput regressor, which by default trains individual ridge regression models for each target variable (here, each location in the brain) independently. The MultiOutput however comes with substantial overhead, as it introduces many redundant computations across brain targets. To reduce the amount of redundant computations happening with MultiOutput ridge regression, we also modified MultiOutput to train a series of model on batches of brain targets, using one compute node per batch and multi-threading execution within each batch. We conducted a theoretical complexity analysis to motivate the choice of this approach. We also repeated our benchmark for both MultiOutput and the batch MultiOutput by assessing the efficiency of parallelization with varying number of threads per node and the number of compute nodes.

Taken together, this study will provide concrete guidelines for practitioners who want to run brain encoding efficiently with ridge regression and large fMRI datasets, using high-performance computing infrastructure and CPUs.

2. Materials and Methods

2.1. fMRI dataset

We used the 2020-alpha2 release of the Friends fMRI dataset collected by the Courtois project on neuronal modeling, CNeuroMod [30]. Some of the text in this section is adapted from the Courtois NeuroMod technical documentation (<https://docs.cneuromod.ca>).

2.1.1. Friends TV show stimuli

Participants watched three seasons of the Friends TV show while their brain activity was recorded using fMRI. Each episode was divided into two segments (a/b) to provide shorter scanning runs and allow participants to take a break. There was a slight overlap between the end of each video segment and the beginning of the next video segment to provide an opportunity for participants to catch up with the story line.

2.1.2. Participants

The Friends dataset includes fMRI time series collected on six participants in good general health, 3 women (sub-03, sub-04, and sub-06) and 3 men (sub-01, sub-02, and sub-05). Three of the participants reported being native francophone speakers (sub-01, sub-02, and sub-04), one as being a native anglophone (sub-06), and two as bilingual native speakers (sub-03 and sub-05). All subjects had a good comprehension of English, which was used in the sound track of the Friends videos. All subjects also provided written informed consent to participate in

this study, which was approved by the local research ethics review board (under project number CER VN 18-19-22) of the CIUSSS du Centre-Sud-de-l'Île-de-Montréal, Montréal, Canada.

2.1.3. Magnetic resonance imaging

Magnetic resonance imaging (MRI) was collected using a 3T Siemens Prisma Fit scanner and a 64-channel head/neck coil, located at the Unit for Functional Neuroimaging (UNF) of the Research Centre of the Montreal Geriatric Institute (CRIUGM), Montréal, Canada. Functional MRI data were collected using an accelerated simultaneous multi-slice, gradient echo-planar imaging sequence [31, 32] developed at the University of Minnesota, as part of the Human Connectome (HCP) Project [33]. The fMRI sequence used the following parameters: slice acceleration factor = 4, TR = 1.49s, TE = 37 ms, flip angle = 52 degrees, 2 mm isotropic spatial resolution, 60 slices, acquisition matrix 96x96. The structural data was acquired using a T1-weighted MPRAGE 3D sagittal and the following parameters: duration 6:38 min, TR = 2.4 s, TE = 2.2 ms, flip angle = 8 deg, voxel size = 0.8 mm isotropic, R=2 acceleration. For more information on the sequences used or information on data acquisition (including fMRI setup), visit the [CNeuroMod technical documentation](#) page.

2.1.4. Preprocessing

All fMRI data were preprocessed using the fMRIPrep pipeline version 20.2.3 [34]. We applied a volume-based spatial normalization to standard space (MNI152 NLin2009cAsym). Furthermore, a denoising strategy was applied to regress out the following basic confounds: (1) a 24-degrees of freedom expansion of the motion parameters, (2) a basis of slow time drifts (slower than 0.01 Hz). This step was implemented with the Nilearn maskers (see below) and the `load_confounds` tool⁴ (option `Params24`). A spatial smoothing with a 8 mm field-width-at-half-maximum and a Gaussian kernel was also applied with Nilearn prior to time series extraction. For each fMRI run, time series were also normalized to zero mean and unit variance (over time, for each voxel independently).

2.1.5. Multiresolution time series extraction

Functional MRI data takes the form of a 3D+t array, where the 3D spatial dimensions encode for different spatial locations on a regular 3D sampling grid (with 2 mm isotropic voxels for this dataset) within the field of view of acquisition, and the time axis (t) encodes brain samples recorded at different times, again on a regular sampling grid (with the time interval TR=1.49s for this dataset). It is common practice to translate this 3D+t array into a 2D array, where the

⁴https://github.com/simexp/load_confounds

Table 1: Brain datasets summary: number ($n \times t$) of time x space samples and size (in MB or GB) of fMRI time series in three resolutions.

Resolution	Subject	n	t	Size (float64)
Parcels	sub-0(1-6)	69,202	444	244 MB
ROI	sub-0(1-6)	"	6,728	2.6 GB
Whole-Brain	sub-01	"	264,805	138 GB
	sub-02	"	266,126	142 GB
	sub-03	"	261,880	136 GB
	sub-04	"	266,391	142 GB
	sub-05	"	263,574	138 GB
	sub-06	"	281,532	148 GB
Whole-Brain (B-MOR)	sub-01	10,000	264,805	21 GB
	sub-02	"	266,126	21.2 GB
	sub-03	"	261,880	20.8 GB
	sub-04	"	266,391	21.2 GB
	sub-05	"	263,574	21 GB
	sub-06	"	281,532	21.8 GB
Whole brain (MOR)	sub-0(1-6)	1,000	2,000	16 MB

first dimension encodes time, and the second dimension encodes space. There are multiple ways to perform this translation, which corresponds to different spatial resolution choices for the analysis. In this work, we used the so-called maskers of the Nilearn library [35] to perform this operation, and we considered three common spatial resolutions to investigate the scalability of different implementations of ridge regression. These approaches vary markedly in the size of the resulting spatial dimension: parcel-wise, ROI-wise, and whole brain. These three resolutions are further described below:

1. **Parcels:** The preprocessed BOLD time series were averaged across all voxels in each parcel of a parcellation atlas, using the `NiftiLabelsMasker` masker from Nilearn. We used the Multiresolution Intrinsic Segmentation Template (MIST) [36]. MIST provides a hierarchical decomposition of functional brain networks in nine levels (7 to 444), and we used here the largest available resolution (444 brain parcels).
2. **ROI:** In this approach, a binary mask of the visual network was extracted from MIST at resolution 7. Voxel-wise time series were extracted for all voxels present in this mask, using the `NiftiMasker` masker from Nilearn. Note that the location of the mask was based on non-linear registration only, and did not use subject-specific segmentation of the grey matter. The exact same number of voxels (6728) was thus present in the mask for all subjects' data, after realignment in stereotaxic space.

3. **Whole-Brain:** In this approach, the brain mask generated by the fMRIprep pipeline based on the structural scans of each participant was re-sampled at the resolution of the fMRI data. This mask included both grey matter, white matter, cerebro-spinal fluid but excluded all tissues surrounding the brain. Voxel-wise time series of all voxels included in the mask were extracted, again using the `NiftiMasker` masker from Nilearn. As the brain mask was subject specific, the number of voxels in the mask varied slightly across subjects.

Table 1 presents the shape of the brain data array Y with these three levels of resolution and six subjects, where the number of rows and columns indicate the number of volumes (n time sample) and targets (t spatial targets) respectively. The temporal dimension is identical for all three approaches, while the spatial dimension of ROI is one order of magnitude larger than Parcels, and the spatial dimension of Whole-Brain is three orders of magnitude larger than Parcels. We also introduce two truncated versions of the whole-brain resolution, marked as MOR and B-MOR in Table 1, which represent subsets of the dataset. We truncated the number of time samples and brain target from the whole-brain data, to accommodate memory requirements in the benchmark infrastructure. In this table, memory sizes are presented in the float64 format used in Scikit-learn for ridge regression.

2.2. Brain encoding

Figure 1 recapitulates the two main steps of brain encoding: extracting features from movie frames through a pretrained artificial network (here VGG16) and predicting brain response using a regularized linear regression model, called ridge. The ridge regression is trained through pairs of prediction targets (fMRI data Y) and dynamic visual stimuli features (predictors X), and experiments are implemented at several levels of resolution, see Table 1 to test the scaling efficiency of different implementations of the ridge regression.

2.2.1. VGG16 artificial vision network

In this work, we used the approach of [12, 37], and applied a VGG16 model [38] pretrained for image classification to extract visual features from the movie frames. The VGG16 model was trained on a dataset of over 2 million images belonging to 1000 classes from the ImageNet database [39], and the weights of the models were retrieved through [TensorFlow](#). This model achieved 92.7% top-5 test accuracy for image-object classification. The VGG16 architecture [40] is a widely-used convolutional neural network (CNN) known for its simplicity and effectiveness in image classification tasks [37]. The network comprises 16 layers, including 13 convolutional layers and 3 fully connected layers. The convolutional

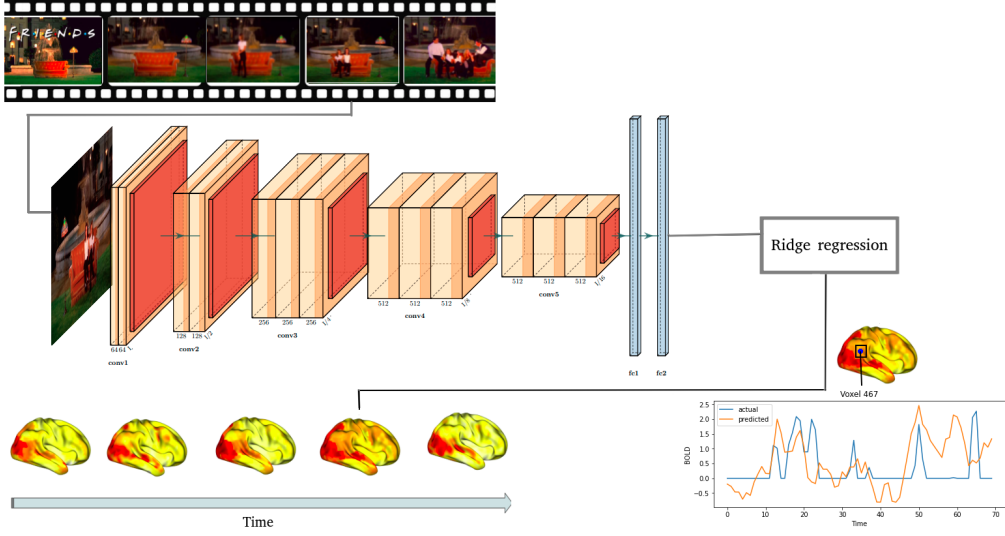


Figure 1: The two main steps of brain encoding: Extracting features from movie frames using VGG16 pretrained model and predicting brain response using ridge regression.

layers use small 3x3 filters with a stride of 1 and employ rectified linear unit (ReLU) activation functions. Max-pooling layers with 2x2 filters are applied for spatial down-sampling. The architecture is characterized by a large number of trainable parameters, summarized in Appendix 7.1 (based on the TensorFlow summary of the model), making it suitable for various computer vision applications.

2.2.2. Extracting VGG16 features of dynamic visual stimuli

For each of the $n = 69,202$ fMRI time samples, we extracted the stimulus video frames corresponding to the 4 TRs immediately preceding each fMRI samples (equivalent to a window of $4 \times 1.49 = 5.96$ s duration). This operation was done to take into account the known delayed, convolutional nature [41] of the relationship between the visual stimulus and the hemodynamic response. Each frame was resampled to a $(224, 224, 3)$ array and fed into VGG16 to extract 4096 features from the last layer. The features of the 4 TRs were concatenated, resulting into a single feature vector of length $p = 16384$. In total, the array of features X used for brain encoding had a size of $(n = 69202, p = 16384)$ (number of time samples x number of features), or 8.5 GB (in float32 precision).

2.2.3. Ridge regression

Ridge regression was first proposed by Hoerl and Kennard [2] as a generalization of ordinary least-square regression. In comparison with ordinary least mean square regression, ridge regression provides better generalization to unseen data through regularization of coefficient estimates, in particular in the presence of a

large number of predictor variables. The ridge regression is expressed as the following optimization problem solving for regression coefficients b^* independently at each spatial location:

$$b^* = \arg \min_{b \in \mathbb{R}^p} (\|y - Xb\|_2^2 + \lambda \|b\|_2^2), \quad (1)$$

where $X \in \mathbb{R}^{n \times p}$ is the matrix of stimuli features with n time samples and p features, $\|\cdot\|_2$ is the ℓ^2 norm of a vector, and $y \in \mathbb{R}^n$ is the target vector obtained from fMRI data at a single spatial location (at either Parcels, ROI or Whole-Brain resolutions). The hyper-parameter λ is used to control the weighting of the penalty in the loss function. The best value for λ is estimated among a set of candidate values through cross-validation, as explained below. If the value of λ is too low the training process may overfit and if the value of λ is too high then the brain encoder model may underfit [13].

2.2.4. Brain encoding performance and hyper-parameter optimization

For a given subject, the samples X were split into training (90% random) and test (10% remaining) subsets. The coefficients of the ridge regression were selected through Eq. 1 based on the training set only. Table 2 presents the memory size and number of ridge training parameters with three levels of resolution across six subjects.

Table 2: Number of training parameters (rounded to closest M) and size of weight matrices in three resolutions for brain encoding

Resolution	Subject	# of training parameters	Size(float64)
Parcel	sub-0(1-6)	7 M	58 MB
ROI	sub-0(1-6)	110 M	1.2 GB
Whole brain and Whole brain (B-MOR)	sub-01	4338 M	34.6 GB
	sub-02	4360 M	34.8 GB
	sub-03	4290 M	34.2 GB
	sub-04	4364 M	34.6 GB
	sub-05	4318 M	34.6 GB
	sub-06	4612 M	34.8 GB
	sub-0(1-6)	32.7 M	262.0 MB
Whole brain (MOR)	sub-0(1-6)	32.7 M	262.0 MB

We measured the final quality of brain encoding as the Pearson’s correlation coefficient between the actual fMRI time series and the time series predicted by the ridge regression model, on the test set. A leave-one-out validation was used inside the training set to estimate the hyper-parameter value λ with optimal performance (based on cost function defined in Eq. 1), based on the grid:

$$\lambda \in \{0.1, 1, 100, 200, 300, 400, 600, 800, 900, 1000, 1200\}.$$

The choice of λ can either be made separately for each of the t brain targets, or a common value can be selected for all brain targets based on the average performance of the model across all t brain targets. In this work, a single λ is used for all targets.

2.3. Ridge regression implementations

2.3.1. Scikit-learn efficient ridge implementation

In large-scale brain encoding tasks, the computational cost of ridge regression increases linearly with the increasing number of targets. To reduce computing time when multiple targets are used, formulations of ridge regression have been proposed to mutualize computations among the targets. The formulation described below was presented in [13] and is implemented in the scikit-learn library.

In a multi-target regression problem, vector y in Equation 1 is now a matrix Y of size n by t where t is the number of spatial targets. Matrix X is still of dimension n by p . The weight matrix W can be calculated as follows:

$$W = MY \quad (2)$$

where

$$M = (X^T X + \lambda I_p)^{-1} X^T \quad (3)$$

and I_p is the identity matrix. The key point is that M is independent of Y and can therefore be reused for all t targets. This strategy reduces the time complexity of multi-target ridge regression from $O(p^3 t + p^2 n t)$ to $O(p^3 + p^2 n + p n t)$ [13], see Section 3 for details.

Scikit-learn also mutualizes computations among subsequent estimations of M for different λ values, typically encountered during hyper-parameter optimization. To do so, it relies on the SVD decomposition of X :

$$X = U S V^T, \quad (4)$$

where $U \in \mathbb{R}^{n \times p}$ and $V \in \mathbb{R}^{p \times p}$ are orthonormal matrices, and $S \in \mathbb{R}^{p \times p}$ is diagonal. Then, the matrix M can be rewritten as:

$$M(\lambda) = V(S^2 + \lambda I_p)^{-1} S U^T \quad (5)$$

Computing $(S^2 + \lambda I_p)^{-1} S$ is inexpensive as this matrix is diagonal. SVD decomposition of feature matrix X reduces complexity of computing M from $O(p^3 r + p n r)$ to $O(p^2 n r)$ [13], where r is the number of tested hyper-parameter values (see Section 3 for details).

2.3.2. Computational environment

Brain encoding experiments were run on Beluga, a high-performance computing (HPC) cluster of Canada Digital Alliance, providing researchers with a robust infrastructure for advanced scientific computations. Beluga features numerous compute nodes, high-speed interconnects, and parallel processing capabilities, visit the [Beluga technical documentation](#) page for details.

All benchmarking experiments were run on a high-performance computing cluster called “slashbin”, fully dedicated to benchmarking, without concurrent users accessing the platform during tests. This cluster was located at Concordia University Montreal, and featured 8 compute nodes. Each compute node featured an Intel®Xeon®Gold 6130 CPU @ 2.10GHz with 32 physical cores (64 hyper-threaded cores), 250 GB of RAM, Rocky Linux 8.9 with Linux kernel 4.18.0-477.10.1.el8_lustre.x86_64. Input and output data were located on a 960GB Serial-Attached SCSI (SAS) 12GBPS 512E Solid State Drive that was network mounted to each compute node using NFS v4.

2.3.3. Multi-threading parallelism

Multi-threading is a mechanism to parallelize executions on multi-core CPUs. In the case of ridge regression, multi-threading is available mainly for linear algebra routines implemented through the Basic Linear Algebra Subprograms (BLAS) specification. Two well-known BLAS libraries are Intel Math Kernel Library (MKL) [28] and OpenBLAS [27]. These BLAS libraries incorporate optimized implementations that leverage multi-threading parallelism for efficient execution. In particular, the OpenBLAS and MKL libraries enable multithreaded execution of ridge regression over the CPU cores in a single machine for a faster execution time. Figure 2 summarizes the different parallelization modes benchmarked by our experiments.

2.3.4. Distributed parallelism

In brain encoding, ridge regression independently fits a regression model on each spatial target. Therefore, ridge regression can be easily parallelized into multiple sub-models addressing different targets. For a given matrix X , scikit-learn’s MultiOutputRegressor class subdivides the set of all target values Y into t sub-problems, each corresponding to a single spatial target. Ridge regression can now be expressed into solving t independent estimations of the weights matrix W , as illustrated in Figure 3. This sub-division is repeated for each value of the regularization parameter λ . As all the targets and λ values are independent, no communication is required between the sub-problems. Scikit-learn’s MultiOutputRegressor class parallelizes the resolution of the sub-problems using a configurable number of concurrent processes c executed with the `joblib` library.

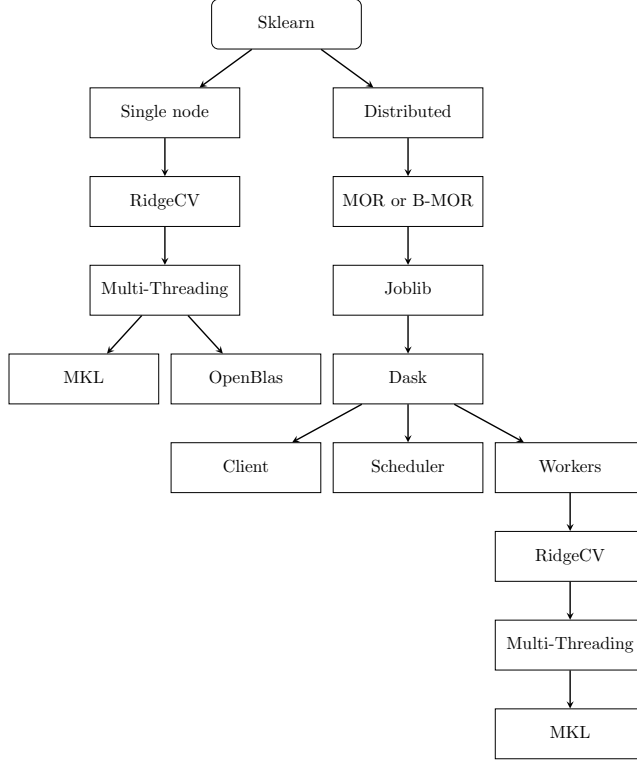


Figure 2: Mutilthreading and Distributed parallelism in scikit-learn’s ridge regression

Joblib supports multiple execution backends including single-host thread-based or process-based parallelism, and distributed parallelism using the Dask [29] or Ray [42] engines. We used the Dask backend and launched its distributed scheduler that simultaneously manages the computation requests and tracks the compute node statuses.

2.3.5. Proposed distributed ridge regression: Batch Multi-Output Regression (B-MOR)

This approach reduces the amount of redundant computations by partitioning the problem into a number of sub-problems equal to the number of available compute nodes in the distributed system, denoted as c (Figure 3). This strategy preserves maximal parallelism while reducing computational overheads. Algorithm 1 describes the approach. It consists of a main parallel for loop where the target output matrix (Y) is partitioned into n sub-problems, where n is the minimum value between the number of targets and the number of compute nodes. Each sub-problem represents a batch of targets Y_1, \dots, Y_n . The algorithm uses the following helper functions:

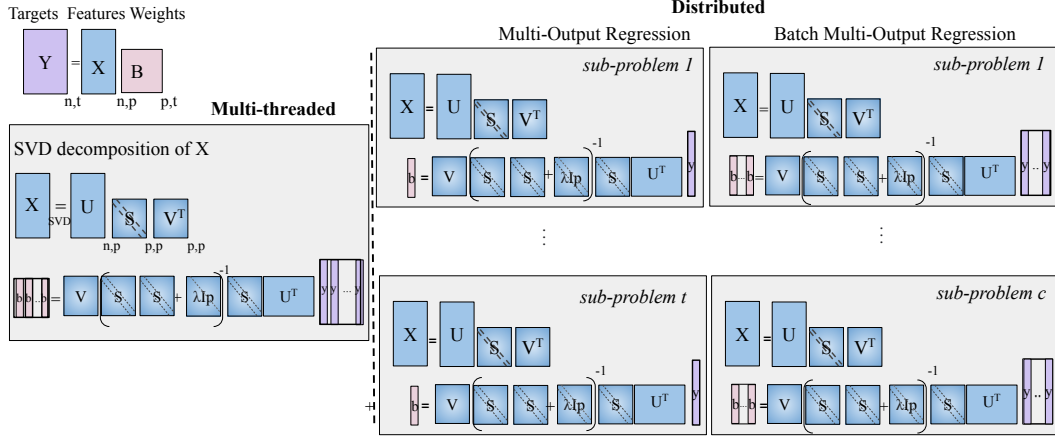


Figure 3: Matrix computations in Multi-threading ridgeCV, MOR and B-MOR model fitting. Assuming $X \in \mathbb{R}^{n \times p}$, $Y \in \mathbb{R}^{n \times t}$ and $X = USV^T$ then the weight matrix $B \in \mathbb{R}^{p \times t}$ equals to $B = V(S^2 + \lambda I_p)^{-1} S U^T Y$.

- `split` returns training and validation sets associated with a given cross-validation split.
- `svd` computes the singular-value decomposition of a matrix.
- `eval_score` computes the regression performance score from predicted and true values. Higher scores denote better performance.

3. Complexity analysis

3.1. Ridge regression with a single thread

Ridge regression for a given hyper-parameter λ is computed by:

$$\hat{Y}_\lambda = X M_\lambda Y.$$

In this section, we outline the time complexity T_M of computing M_λ as well as the time complexity T_W of computing the multiplications $X M_\lambda Y$. Matrix notations as well as their dimensions are summarized in Table 3. Time complexities are expressed in the number of floating-point multiplications.

The cost T_W of computing $X M_\lambda Y$ independently over r values of hyper-parameter λ is:

$$T_W = O(pntr).$$

Regarding T_M , Equation 3 requires inverting a square matrix of size p —time complexity $O(p^3)$ — and multiplying the resulting inverted matrix with matrix

Algorithm 1: Batch Multi-Output Regression (B-MOR)

input : X —Input stimuli feature matrix
input : Y —Target matrix
input : t —Number of targets
input : λ —Candidate hyper-parameters
input : c —Number of concurrent jobs
output: B —List of trained weight matrices for each sub-problem

```
1  $n \leftarrow \min(t, c);$   
  // Main parallel for loop  
2 parfor  $i = 0$  to  $n - 1$  do  
  // Divide the target matrix  $Y$  into  $n$  sub-problems  
3    $Y_i \leftarrow$  Sub-matrix of  $Y$  with columns  $\left[\frac{i \cdot t}{n}, \frac{(i+1) \cdot t}{n}\right];$   
4   for all cross-validation splits  $s$  do  
5      $X_{\text{train}}, X_{\text{val}}, Y_{\text{train}}, Y_{\text{val}} \leftarrow \text{split}(s, X, Y_i);$   
6      $USV^T \leftarrow \text{svd}(X_{\text{train}});$   
7     for all  $\lambda$  do  
8        $M_\lambda \leftarrow V(S^2 + \lambda I_p)^{-1} S U^T;$   
9        $\hat{Y}_{\text{val}} \leftarrow X_{\text{val}} M_\lambda Y_{\text{train}};$   
10       $\text{score}[i, s, \lambda] \leftarrow \text{eval\_score}(\hat{Y}_{\text{val}}, Y_{\text{val}});$   
  // Calculate mean score across cross-validation splits  
11  for all  $\lambda$  do  
12     $\text{mean\_score}[i, \lambda] \leftarrow \frac{1}{|s|} \sum_s \text{score}[i, s, \lambda];$   
  // Find the best hyperparameter  $\lambda$  for each sub-problem  
13   $\text{best\_}\lambda[i] \leftarrow \arg \max_\lambda \{\text{mean\_score}[i, \lambda]\};$   
14   $B[i] \leftarrow M_{\text{best\_}\lambda[i]} Y_i;$   
15 return  $B;$ 
```

X^T of size $p \times n$ —time complexity $O(p^2n)$. Matrix M_λ is computed once for all the targets. With a single hyper-parameter value ($r=1$), Equation 3 thus gives a complexity of $O(p^3 + p^2n)$.

If we were to naively iterate this approach for r hyper-parameter values, the resulting time complexity would be $O(p^3r + p^2nr)$. However, expressing matrix M_λ from the SVD of matrix X as in Equation 5 reduces the time complexity of computing M to:

$$T_M = O(p^2nr + pr)$$

Indeed, computing the SVD of X has time complexity $O(p^2n)$ since $p \leq n$, the computation of $(S^2 + \lambda I_p)^{-1} S U^T$ has time complexity $O(p)$ since S is diagonal, and the multiplication of V with $(S^2 + \lambda I_p)^{-1} S U^T$ has time complexity $O(p^2n)$.

Finally, the overall time complexity T_{ridge} of ridge regression iterated on r hyper-parameter values, including computation of M_λ and multiplication by matrix Y is:

$$T_{\text{ridge}} = T_M + T_W = O(p^2nr + pr + pntr)$$

Table 3: Matrix Sizes. n : number of time samples; p : number of ANN features; t : number of brain targets. Other important notations include c : number of concurrent distributed executions and r : number of hyper-parameters.

Matrix	Dimensions	Description
X	$n \times p$	Feature matrix
Y	$n \times t$	Brain target matrix
M_λ	$p \times n$	Resolution matrix
U	$n \times p$	Left singular matrix
S	$p \times p$	Singular values matrix
V	$p \times p$	Right singular matrix

3.2. Ridge regression with MOR

In the case of MOR, the matrix multiplication $M_\lambda Y$ is replaced by t multiplications of M with a vector y , which does not change the time complexity. Provided that the number of targets t is larger than the number of concurrent computing nodes c —which is the case in our application—, all these matrix-vector multiplications happen in parallel, and the resulting time complexity on the application critical path is $c^{-1}T_W$.

By contrast, the computation of matrix M_λ is repeated independently for each brain target, resulting in a massive overhead computation tT_M distributed over

c concurrent processes. Overall the computational cost of ridge regression implemented with MOR is:

$$T_{\text{MOR}} = c^{-1}(T_W + tT_M). \quad (6)$$

3.3. Ridge regression with B-MOR

B-MOR scales better than the previous approach due to the use of c sub-problems instead of t . The computation of the matrix multiplication costs $c^{-1}T_W$, similar to MOR. However, the overhead of recomputing matrix M_λ for each sub-problem is only cT_M , which is distributed across c concurrent execution. The computational cost of ridge implemented with B-MOR is thus:

$$T_{\text{B-MOR}} = c^{-1}T_W + T_M. \quad (7)$$

Comparing Equations 6 and 7, we observe that the time complexity of the B-MOR implementation is much lower than for the MOR implementation, as $T_{\text{MOR}} - T_{\text{B-MOR}} = (c^{-1}t - 1)T_M$. This difference may be massive when $c \ll t$. We also observe that when $c > 1$, B-MOR has lower time complexity than single-threaded ridge regression. However, the parallel efficiency of B-MOR is limited by the term T_M which is not reduced by c .

4. Results

We report on a series of benchmark experiments for brain encoding, using scikit-learn’s multithreaded, MOR and B-MOR implementations of ridge regression with hyper-parameter optimization across 11 values of λ . The benchmarks were applied to the Friends CNeuroMod dataset (N=6 subjects) at multiple spatial resolutions to investigate the scalability of different implementations of ridge regression (parcel-wise, ROI-wise, and truncated versions of whole-brain voxel-wise time series).

4.1. Brain encoding models successfully captured brain activity in the visual cortex

We first aimed to validate that our brain encoding model performed in line with recent studies, at the individual level and for different resolutions of target brain data.

We extracted features of dynamic visual stimuli, by selecting a subset of images in the video, and feeding these images independently into an established vision pretrained network called VGG16 [38]. We used 3 seasons of Friends TV show, where 10% of data was set aside as the test data. Brain encoding was implemented using ridge regression, and the best value of hyper-parameter λ was selected through cross-validation.

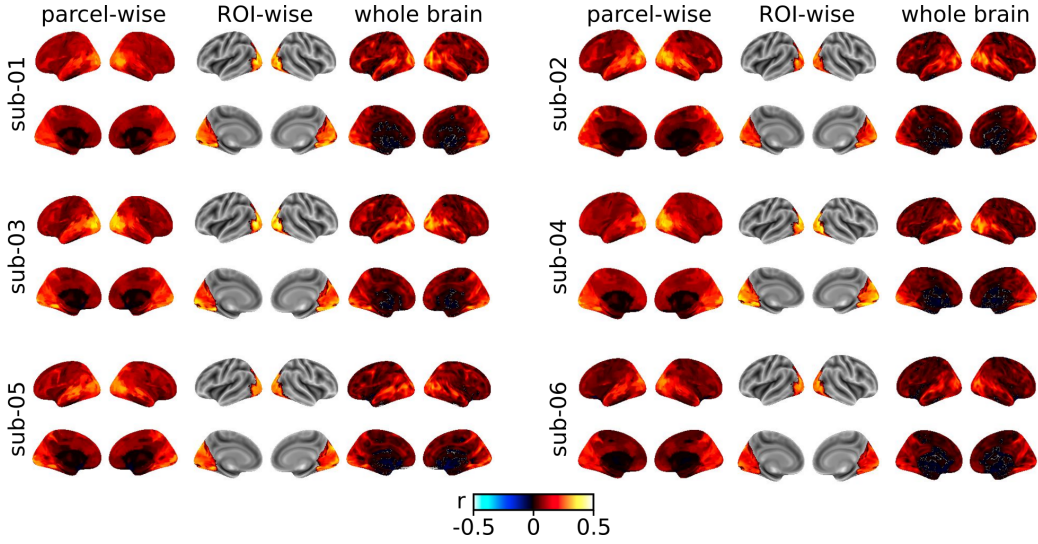


Figure 4: Brain encoding results, with performance based on Pearson Correlation Coefficient (r) between real and predicted time series in the friends dataset ($N=6$ subjects).

Figure 4 shows the functional alignment between the features of the last fully connected layer (FC2) of VGG16 brain activities and brain activity for six subjects. Brain encoding maps were highly consistent across subjects and resolutions of analysis. In all cases, a moderate correlation, up to 0.5, was observed in the visual cortex between the real fMRI time series and the time series predicted by the brain encoding model. For the analysis that included the full brain, moderate accuracy for brain encoding was observed in other brain areas as well, such as temporal cortices involved in high level visual processing as well as language. Analysis at the voxel level brought more anatomical details but were overall consistent with brain encoding maps at the parcel level.

Overall, brain encoding models successfully predicted activity in expected brain regions, for all subjects and resolutions.

4.2. Brain encoding was significant compared to a null distribution

Next, we wanted to assess the significance of the brain encoding, compared to a null distribution where the movie frames used as input to the model did not correspond with brain data time series. We repeated the brain encoding procedure presented in the previous section for one subject (sub-01), after random shuffling of the image features and brain images. Figure 5 compares the original brain encoding results (panel a) with brain encoding based on shuffled features (panel b). While the original brain encoding results reached moderate accuracy, up to 0.5 correlation between real and predicted brain activity, the performance using shuffled features was dramatically lower. The correlation values were typically an

order of magnitude smaller, less than 0.05. The quality of brain encoding using original image frames thus appeared as significant compared to a null distribution where image features were randomly shuffled.

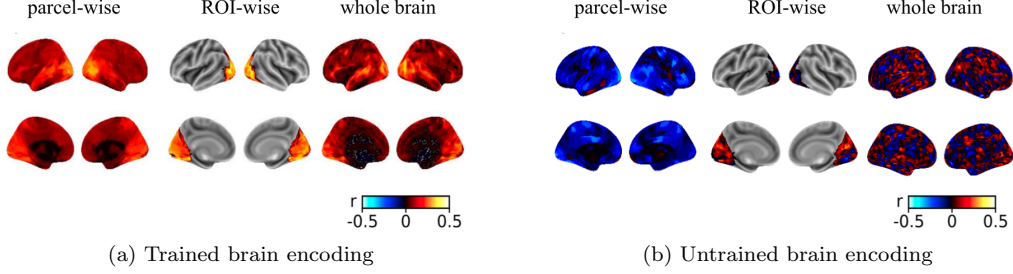


Figure 5: Brain encoding predictions for a single individual (sub-01) in two cases. Panel **a**: corresponding pairs of {fMRI time series and stimuli} were presented to the ridge regression models. Panel **b**: random permutations of fMRI time series and stimuli data were presented to the ridge regression model.

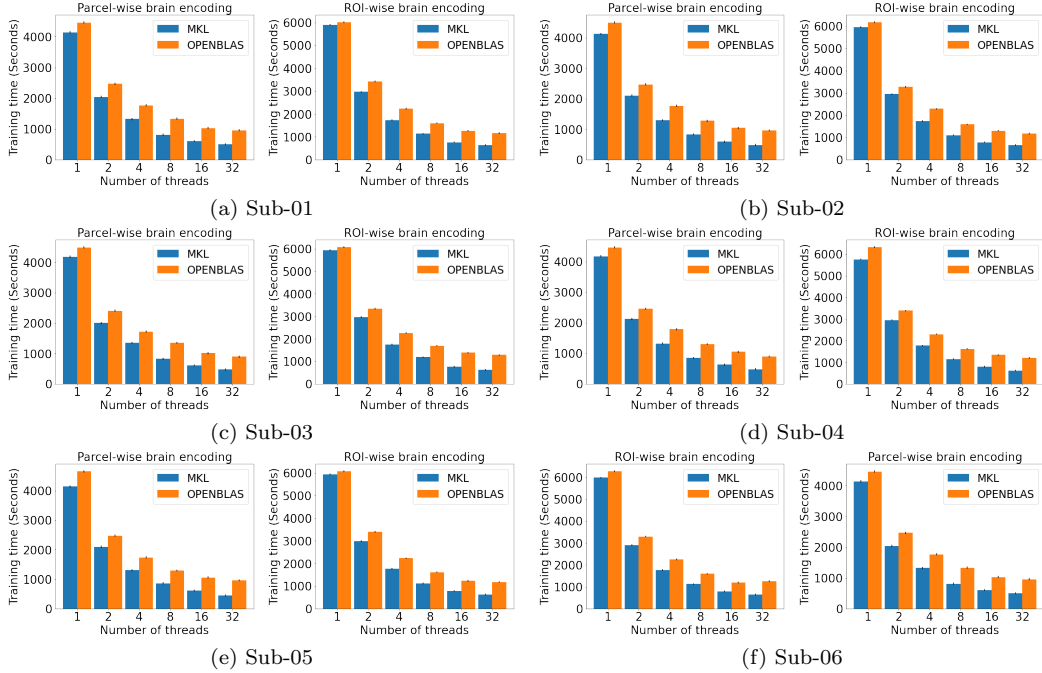


Figure 6: Comparison of MKL and OpenBLAS implementations for multithreaded execution.

4.3. Multithreaded execution with Intel MKL provided significant speedup compared to OpenBLAS

After establishing the quality of our multiresolution brain encoding benchmark, we proceeded to compare the performance and scalability of ridge regression using

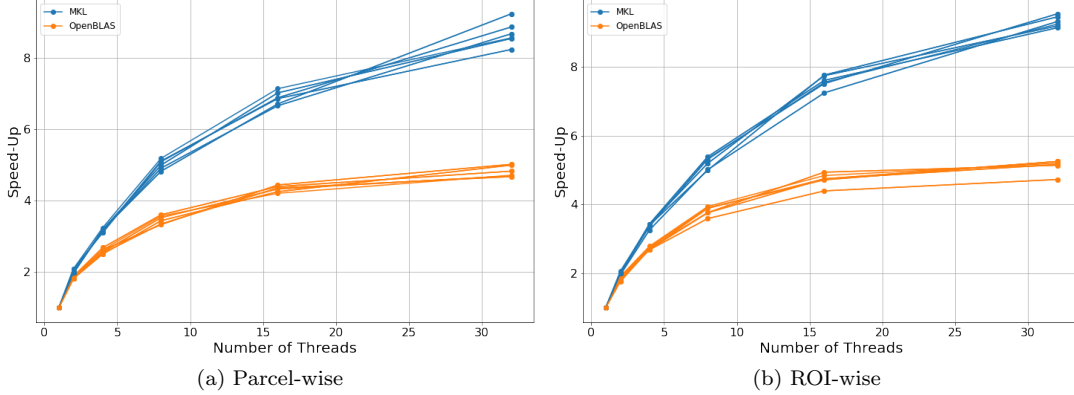


Figure 7: Speed-up rates of ROI-wise ridgeCV execution time for MKL and OpenBLAS across different number of threads. Each line on the plot corresponds to a specific subject. For each subject, the solid lines represent MKL, and the dashed lines represent OpenBLAS.

scikit-learn on a 32-core compute node, and comparing the libraries underlying multithreaded parallelization, i.e. MKL and OpenBLAS. Figure 6 illustrates the comparison between OpenBLAS and MKL multi-threading for two different resolutions (parcel-wise and ROI-wise), six subjects, and varying numbers of parallel threads. The experiments with whole-brain resolution could not be completed due to out-of-memory limitation with our benchmark system. The results consistently demonstrated that the MKL library outperformed the OpenBLAS library for all subjects and thread configurations. Specifically, when using 32 threads, the MKL library exhibited a speedup factor of 1.90 and 1.98 compared OpenBLAS for parcel-wise and ROI, respectively, on average across all subjects. This indicates a substantial improvement in processing time with the MKL library compared to OpenBLAS.

4.4. Speed-up of multi-threading quickly reached a plateau with an increasing number of threads

We also observed a sharp decrease in the efficiency of parallelization with an increasing number of threads. Figure 7 represents the speed-up performance of two libraries MKL and OpenBLAS for parcel-wise and ROI-wise brain encoding across different numbers of threads. The parallelization speed-up is calculated as follows:

$$SU = \frac{T_R}{T_P}$$

where SU is the speed-up, T_R is the execution time with 1 thread, and T_P is the execution time with 2, 4, 8, 16, or 32 threads. The speed-up measures how effectively the parallel resources are being used.

A consistent observation across subjects was that, as the number of threads increased, the parallel efficiency decreased, as expected according to Amdahl’s law. In other words, as more threads were employed, the incremental improvement in speed-up rate was reduced. These diminishing returns in speed-up highlight the need for careful selection of thread count to balance computational resources with performance.

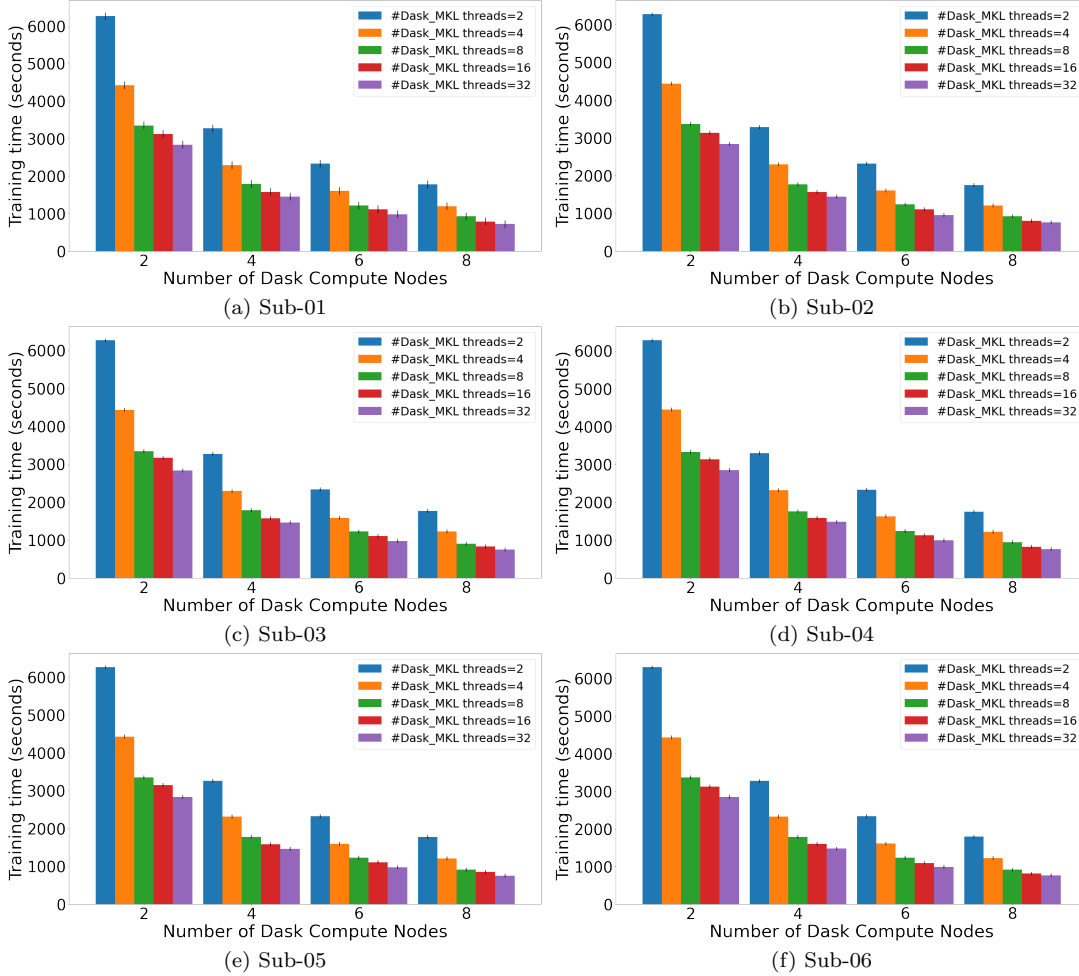


Figure 8: MultiOutput ridgeCV training time for 6 subjects using whole brain (MOR) data described in Table 1. The MOR implementation scales across threads and Dask compute nodes, however, it has a massive overhead: multi-threaded scikit-learn implementation with a single compute node and 32 threads takes approximately 1s.

4.5. MultiOutput ridge regression scales across compute nodes and threads, but is much slower than multi-threading with RidgeCV

In this next experiment, we implemented scikit-learn’s original MultiOutput ridge regression (MOR) within a Dask-based distributed parallelism setting for brain

encoding tasks. We chose to focus on whole-brain resolution for this experiment, as lower resolutions failed to show the advantage of parallelization with MOR. However, whole-brain resolution was too slow to run as is with MultiOutput ridge regression. Therefore, we created a custom truncated version of whole-brain resolution, called whole-brain (MOR), as seen in Table 1. Figure 8 reports on the parallelization of MultiOutput across six subjects using this dataset, where the training process was distributed across multiple compute nodes and threads.

Figure 8 shows a substantial reduction in training time with an increasing number of threads and compute nodes, for all subjects, which illustrates the good scalability of MOR parallelization. However, compute time was massively increased compared to the multi-threaded scikit-learn implementation on a single compute node. For example, using 8 compute nodes and 32 threads, compute time with MOR is in the order of 1000s, whereas the multi-threaded scikit-learn implementation with a single compute node and 32 threads takes approximately 1s. This overhead directly results from the increase in time complexity reflected in Equation 6.

4.6. Batch multi-output regression leads to efficient speed-up across multiple compute nodes and threads

In the next experiment, we benchmarked our B-MOR implementation of ridgeCV, that divides the brain targets into batches, and runs scikit-learn’s multi-threaded RidgeCV on each batch independently with different compute nodes. Figure 9 shows that, as the number of threads and compute nodes increased, substantial speed-up in training time was achieved compared to scikit-learn’s multithreaded implementation (labelled “RidgeCV” in the figure), which demonstrates the practical value of the B-MOR implementation. To quantify this observation, we computed the distributed speed-up ratio as follows:

$$\text{DSU} = \frac{T_R}{T_P}$$

where T_R indicates the execution time of scikit-learn original ridge regression on a single compute node and 1 thread, and T_P indicates computation time with B-MOR for a given number of compute nodes and threads. Overall, the distributed speed-up ratio increased as the number of threads or compute nodes increased (Figure 10). The training time for B-MOR was approximately 30 – 33 times less than the original scikit-learn ridge regression with 1 thread and 1 compute node. As it was the case with the multi-threaded implementation, the DSU reached a plateau beyond a certain number of compute nodes and threads, with diminishing performance returns as parallelization overheads and the time spent in unparallelized code sections started to outweigh parallelization benefits.

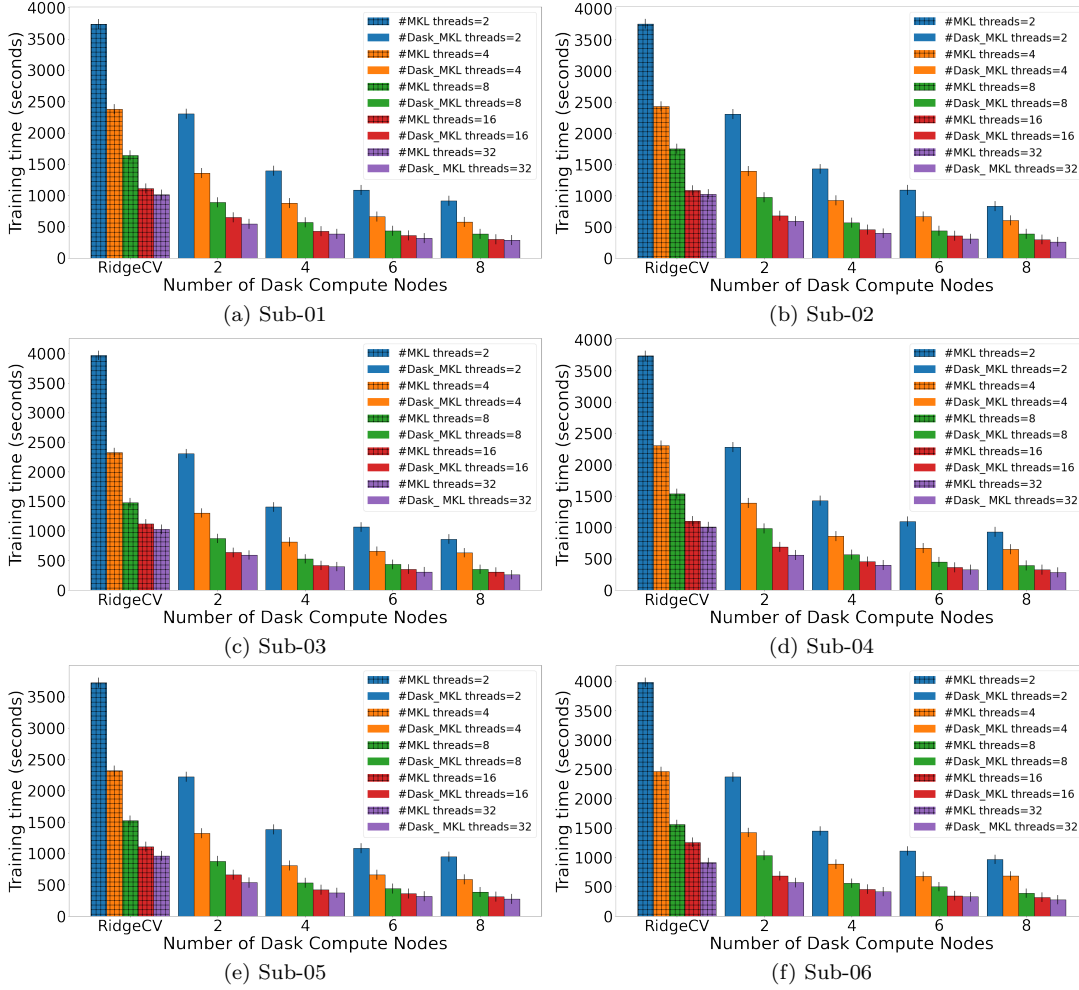


Figure 9: B-MOR ridgeCV training time 6 subjects with whole brain (B-MOR) data described in Table 1. B-MOR scales across compute nodes and threads, and provides substantial speed-up compared to scikit-learn’s multi-threaded implementation (labelled as “RidgeCV”).

5. Conclusion

In this paper, we evaluated the efficiency of different implementations of ridge regression for a specific application: brain encoding using a vision model (VGG16) during movie watching. We found that the multithreaded parallelization available in scikit-learn could be used to reduce substantially computation time, and that the BLAS implementation provided by the proprietary Intel oneAPI Math Kernel Library (MKL) substantially outperforms the open-source OpenBLAS implementation. For increased scalability, the Dask-based scikit-learn MultiOutput implementation can parallelize computations across multiple compute nodes, but this comes with massive redundancy in some of the computations, which we

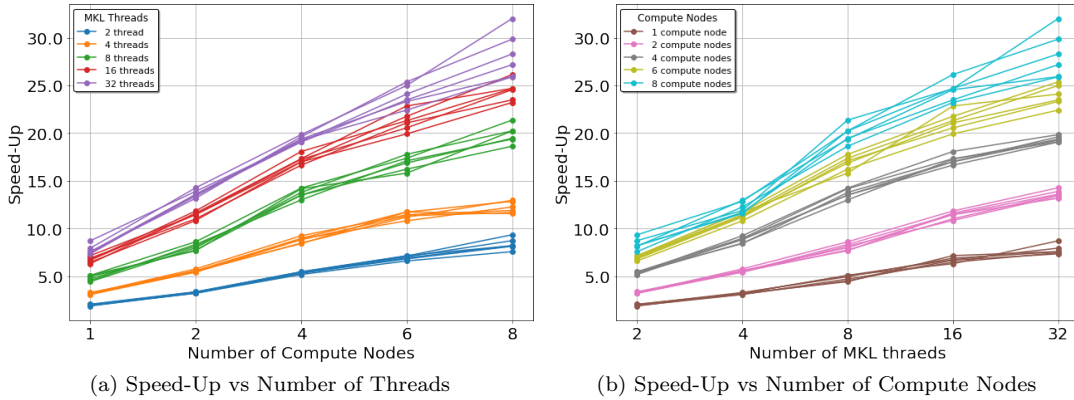


Figure 10: Speed up in B-MOR training time for truncated B-MOR data across 6 subjects with varying numbers of threads and compute nodes in the Dask distributed system.

found to be impractical when using high-resolution brain targets (tens to hundreds of thousands). Therefore, we implemented a more efficient version of the MultiOutput method (B-MOR), that parallelizes ridge regression across batches of brain targets. Our B-MOR method scales well, both in terms of the number of compute nodes and the number of threads used by nodes. This approach allowed us to generate brain encoding maps with high spatial resolution and within a reasonable time. Our method could be useful for fMRI researchers who want to process high-resolution deep datasets with high-performance computing clusters. Our conclusion likely applies to any ridge regression for data arrays with a very large number of targets (up to the order of 100k) and a large number of predictors (in the order of thousands). As our proposed method is straightforward to implement, it may become available in scikit-learn in the future.

6. Availability of code and data

The code to reproduce our experiments is available at <https://github.com/Sana3883/Scaling-up-Ridge>. The CNeuroMod dataset is available at <https://www.cneuromod.ca/gallery/datasets>.

7. Acknowledgments

The computing platform used in the experiments was obtained with funding from the Canada Foundation for Innovation. The Courtois project on neural modelling was made possible by a generous donation from the Courtois foundation, administered by the Fondation Institut G  riatrie Monr  al at CIUSSS du Centre-Sud-de-l  le-de-Monr  al and University of Montreal. The Courtois NeuroMod team is based at "Centre de Recherche de l'Institut Universitaire de G  riatrie

de Montréal”, with several other institutions involved. See the cneuromod documentation for an up-to-date list of contributors (<https://docs.cneuromod.ca>). PB is a senior fellow (chercheur boursier) from Fonds de Recherche du Québec (Santé).

References

- [1] T. Naselaris, K. Kay, S. Nishimoto, J. Gallant, Encoding and decoding in fmri. *neuroimage*, *Technometrics* 56 (2011) 400–410.
- [2] A. Hoerl, R. Kennard, Ridge regression: applications to nonorthogonal problems, *Technometrics* 12 (1970) 69–82.
- [3] L. Wehbe, A. Ramdas, R. Steorts, C. Shalizi, Regularized brain reading with shrinkage and smoothing., *The annals of applied statistics* (2015) 1997.
- [4] A. Pasquiou, Y. Lakretz, J. Hale, B. Thirion, C. Pallier, Neural language models are not born equal to fit brain data, but training helps, *arXiv preprint arXiv:2207.03380* (2022).
- [5] C. Conwell, J. Prince, A. G., T. Konkle, Large-scale benchmarking of diverse artificial vision models in prediction of 7t human neuroimaging data, *bioRxiv* (2022).
- [6] A. Goldstein, E. Ham, S. Nastase, Z. Zada, A. Dabush, B. Aubrey, M. Schain, H. Gazula, A. Feder, W. Doyle, S. Devore, Correspondence between the layered structure of deep language models and temporal structure of natural language processing in the human brain., *bioRxiv* (2022).
- [7] S. Oota, J. Arora, V. Rowtula, M. Gupta, R. Bapi, Visio-linguistic brain encoding., *preprint arXiv:2204.08261* (2022).
- [8] S. Kumar, T. Sumers, T.R.and Yamakoshi, A. Goldstein, U. Hasson, K. Norman, T. Griffiths, R. Hawkins, S. Nastase, Reconstructing the cascade of language processing in the brain using the internal computations of a transformer-based language model., *bioRxiv* (2022).
- [9] M. Lescroart, J. Gallant, Human scene-selective areas represent 3d configurations of surfaces, *Neuron* 101 (2019) 178–192.
- [10] A. Kell, D. Yamins, E. Shook, S. Norman-Haignere, J. McDermott, A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy., *Neuron* 98 (2018) 630–644.

- [11] S. Jain, A. Huth, Incorporating context into language encoding models for fmri, *Advances in neural information processing systems* (2018) 31.
- [12] H. Wen, J. Shi, Y. Zhang, K. Lu, J. Cao, Z. Liu, Neural encoding and decoding with deep learning for dynamic natural vision, *Cerebral Cortex* 28(12) (2018) 4136–4160.
- [13] T. D. la Tour, M. Eickenberg, A. O. Nunez-Elizalde, J. L. Gallant, Feature-space selection with banded ridge regression, *NeuroImage* 264 (2022) 119728.
- [14] K. Seeliger, L. Ambrogioni, Y. Güçlütürk, L. van den Bulk, U. Güçlü, M. van Gerven, End-to-end neural system identification with neural information flow., *PLOS Computational Biology* 17(2) (2021).
- [15] K. Kay, T. Naselaris, R. Prenger, J. Gallant, Identifying natural images from human brain activity, *Nature* 452 (2008) 352–355.
- [16] K. Kay, T. Naselaris, R. Prenger, J. Gallant, Reconstructing visual experiences from brain activity evoked by natural movies., *Current Biology* 21 (2011) 1641–1646.
- [17] T. Horikawa, S. Aoki, M. Tsukamoto, Y. Kamitani, Characterization of deep neural network features by decodability from human brain activity., *Scientific data* (2019) 190012.
- [18] R. Beliy, G. Gaziv, A. Hoogi, F. Strappini, T. Golan, M. Irani, From voxels to pixels and back: Self-supervision in natural-image reconstruction from fmri., In *Advances in Neural Information Processing Systems* (2019) 6517–6527.
- [19] K. Qiao, J. Chen, L. Wang, C. Zhang, L. Zeng, L. Tong, B. Yan, Category decoding of visual stimuli from human brain activity using a bidirectional recurrent neural network to simulate bidirectional information flows in human visual cortices, *Frontiers in neuroscience* (2019).
- [20] G. Shen, K. Dwivedi, K. Majima, T. Horikawa, Y. Kamitani, End-to-end deep image reconstruction from human brain activity., *Frontiers in computational neuroscience* 13 (2019) 21.
- [21] T. Naselaris, E. Allen, K. Kay, Extensive sampling for complete models of individual brains. *current opinion in behavioral sciences*, *bioRxiv* 40 (2021) 45–51.

- [22] N. Chang, J. Pyles, A. Marcus, A. Gupta, M. Tarr, E. Aminoff, Bold5000: a public fmri dataset while viewing 5000 visual images., *Scientific data* (2019) 1–18.
- [23] E. Allen, G. St-Yves, Y. Wu, J. Breedlove, L. Dowdle, B. Caron, F. Pestilli, I. Charest, J. Hutchinson, T. Naselaris, K. Kay, A massive 7t fmri dataset to bridge cognitive and computational neuroscience., *bioRxiv* (2021).
- [24] C. ratton, T. Laumann, A. Nielsen, D. Greene, E. Gordon, A. Gilmore, S. Nelson, R. Coalson, A. Snyder, B. Schlaggar, N. Dosenbach, Functional brain networks are dominated by stable group and individual factors, not cognitive or daily variation., *Neuron* (2018) 439–452.
- [25] Cneuromod, Cneuromod dataset, <https://www.cneuromod.ca/gallery/datasets/>, 2021.
- [26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, the *Journal of machine Learning research* 12 (2011) 2825–2830.
- [27] Z. Xianyi, W. Qian, Z. Yunquan, Model-driven level 3 blas performance optimization on loongson 3a processor., *IEEE 18th international conference on parallel and distributed systems* (2012) 1–18.
- [28] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, Y. Wang, Intel math kernel library. in high-performance, *Computing on the Intel® Xeon Phi™* (2014) 167–188.
- [29] M. Rocklin, Dask: Parallel computation with blocked algorithms and task scheduling, In *Proceedings of the 14th python in science conferenc* 130 (2015) 136.
- [30] J. Boyle, e. a. Pinsard, B., The courtois project on neuronal modeling - 2021 data release, Poster 2224 was presented at the 2021 Annual Meeting of the Organization for Human Brain Mapping held virtually (2021).
- [31] K. Setsompop, J. Cohen-Adad, B. Gagoski, T. Raij, A. Yendiki, B. Keil, V. Wedeen, L. Wald, Improving diffusion mri using simultaneous multi-slice echo planar imaging., *Neuroimage* 63 (2012) 569–580.
- [32] J. Xu, S. Moeller, E. Auerbach, J. Strupp, S. Smith, D. Feinberg, E. Yacoub, K. Ugurbil, Improving diffusion mri using simultaneous multi-slice echo planar imaging., *NeuroimageT* 83 (2013) 991–1001.

- [33] D. Van Essen, M. Glasser, The human connectome project: Progress and prospects. in *cerebrum: the dana forum on brain science*, Dana Foundation 63 (2016).
- [34] O. Esteban, C. Markiewicz, R. Blair, C. Moodie, A. Isik, A. Erramuzpe, J. Kent, M. Goncalves, E. DuPre, M. Snyder, H. Oya, fmriprep: a robust preprocessing pipeline for functional mri. *nature methods*, Neuroimage 16 (2019) 111–116.
- [35] A. Abraham, F. Pedregosa, M. Eickenberg, P. Gervais, A. Mueller, J. Kossai, A. Gramfort, B. Thirion, G. Varoquaux, Machine learning for neuroimaging with scikit-learn, *Frontiers in neuroinformatics* (2014) 14.
- [36] S. Urchs, J. Armoza, C. Moreau, Y. Benhajali, J. St-Aubin, P. Orban, P. Bellec, Mist: A multi-resolution parcellation of functional brain networks, *MNI Open Research* 1 (2019) 3.
- [37] C. Conwell, J. S. Prince, K. N. Kay, G. A. Alvarez, T. Konkle, What can 1.8 billion regressions tell us about the pressures shaping high-level visual representation in brains and machines?, *BioRxiv* (2022).
- [38] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv* (2014).
- [39] J. Deng, W. Dong, R. L. Socher, L. Fei-Fei, Imagenet: A large-scale hierarchical image database., *IEEE conference on computer vision and pattern recognition* (2009) 248–255.
- [40] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv* (2014).
- [41] N. Logothetis, J. Pauls, M. Augath, T. Trinath, A. Oeltermann, Neurophysiological investigation of the basis of the fmri signal, *nature* (2001) 150–157.
- [42] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, et al., Ray: A distributed framework for emerging {AI} applications, in: *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pp. 561–577.

7.1. Appendix

Table 4: Key Parameters of VGG16 (Keras Model)

Layer	Num. of Kernels	Activation Size	Size of Kernels	Parameters (M)
Input	-	224x224x3	-	-
block1_conv1	64	224x224x64	3x3	1792
block1_conv2	64	224x224x64	3x3	36928
block1_pool	-	112x112x64	2x2	-
block2_conv1	128	112x112x128	3x3	73856
block2_conv2	128	112x112x128	3x3	147584
block2_pool	-	56x56x128	2x2	-
block3_conv1	256	56x56x256	3x3	295168
block3_conv2	256	56x56x256	3x3	590080
block3_conv3	256	56x56x256	3x3	590080
block3_pool	-	28x28x256	2x2	-
block4_conv1	512	28x28x512	3x3	1180160
block4_conv2	512	28x28x512	3x3	2359808
block4_conv3	512	28x28x512	3x3	2359808
block4_pool	-	14x14x512	2x2	-
block5_conv1	512	14x14x512	3x3	2359808
block5_conv2	512	14x14x512	3x3	2359808
block5_conv3	512	14x14x512	3x3	2359808
block5_pool	-	7x7x512	2x2	-
Flatten	-	25088	-	-
FC1	-	4096	-	102764544
FC2	-	4096	-	16781312
predictions	-	1000 (output)	-	4097000
Total	-	-	-	138,357,544