Scaling up Machine Learning Models for fMRI Brain Encoding

Sana Ahmadi

A Thesis
In The Department of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Doctor of Philosophy
Concordia University
Montréal, Québec, Canada

January 2025

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Sana Ahmadi**

Entitled: **Scaling up Machine Learning Models for fMRI Brain Encoding**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

Complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair: Dr. Luis Amador Jimenez

_____ External Examiner: Dr. Cyril Pernet

_____ Examiner: Dr. Charalambos Poullis

_____ Examiner Dr. Yiming Xiao

_____ Examiner: Dr. Nizar Bouguila

_____ Supervisor: Dr. Tristan Glatard

_____ Co-supervisor: Dr. Pierre Lune Bellec

Approved by:

_____

Dr. Leila Kosseim, Graduate Program Director

January 22, 2025 _____

Dr. Mourad Debbabi, Dean Gina Cody School of Engineering and Computer Science

Abstract

Scaling up Machine Learning Models for fMRI Brain Encoding
Sana Ahmadi, Ph.D.
Concordia University, 2024

This thesis investigates techniques for optimizing brain encoding models, emphasizing computational efficiency and the scalability of both data and models within the framework of large-scale functional magnetic resonance imaging (fMRI) datasets. Brain encoding aims to predict neural responses to complex stimuli, such as video frames, by utilizing latent feature representations from artificial neural networks. The first study explores the acceleration of ridge regression, a widely used predictive model in brain encoding, particularly when applied to large fMRI datasets like the CNeuroMod Friends dataset. By implementing a novel batch-parallelization strategy using Dask, we achieved significant computational speedups of up to 33× with 8 compute nodes and 32 threads compared to a single-threaded scikit-learn.

The second study investigates how dataset size and model scaling affect brain encoding performance using vision Transformers. To do so, the VideoGPT model was trained end-to-end to extract spatiotemporal features from the Shinobi video game dataset with varying sample sizes (10K, 100K, 1M, and 6M) and model size (number of training parameters). Ridge regression is then used to predict brain activity based on fMRI data and the extracted features from video games. Our results show that larger datasets lead to significantly improved encoding accuracy, with the 6M-sample dataset producing the highest Pearson correlation coefficients across subjects. Additionally, while increasing hidden layer dimensions in the transformer model greatly enhances performance, the number of attention heads appears to have a minimal effect. These findings emphasize the importance of data scaling for improving brain encoding, offering practical insights for optimizing neural network architectures in the context of large-scale stimuli data.
This research advances the field of computationally efficient brain encoding, which is crucial for enhancing both computational speed and accuracy. These advancements are essential not only for improving our understanding of brain function but also for enabling scalable machine learning models on high-dimensional data and sophisticated stimuli, including applications in neuroprosthetics and clinical neuroscience.

Acknowledgment

First and foremost, I would like to express my gratitude to my wonderful research supervisors, Prof.Glatard and Prof.Bellec, for their invaluable support, guidance, and encouragement over the past years. Without their assistance and dedicated involvement in every step throughout the process, the success of this thesis would not be possible. I would also like to thank my colleagues in the CNeuroMod team, especially François Paugam, and my wonderful labmates in the Big Data lab. Finally, my deepest appreciation goes to my family for their unconditional love, encouragement throughout my studies.

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 Gériatrie Montréal at CIUSSS du Centre-Sud-de-l'île-de-Montréal and University of Montreal. The Courtois NeuroMod team is based at "Centre de Recherche de l'Institut Universitaire de Gériatrie 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).

Dedication

To my family

Contribution of Authors

This thesis is manuscript-based and includes two papers for which I am the first author. These papers, co-authored with Prof. Tristan Glatard, Prof. Pierre Lune Bellec, and François Paugam, are detailed below, along with the contributions of each author:

Scaling up ridge regression for brain encoding in a massive individual fMRI dataset:
I was responsible for training models to extract features from stimuli, generating multi-resolution fMRI data, setting up the Dask distributed system, and conducting high-performance experiments (including multithreading and distributed training) for RidgeCV. I also drafted the manuscript and designed the figures. Marie St-Laurent helped me to analyzing brain encoding by creating the maps related to capturing both low-level and high-level visual cognitive functions. Prof. Glatard and Prof. Bellec provided input on the experiments, reviewed the results, and edited the manuscript. This paper is under review by GigaScience Journal.

Compute optimal vision transformers for brain encoding:
I was responsible for training machine learning models to extract features from dynamic vision data, performing scaling-up experiments, drafting the manuscript, and designing the figures. François Paugam helped with with training models and pairing of the fMRI and frames dataset. Prof. Glatard and Prof. Bellec contributed to the experimental design, discussed the results, and edited the manuscript. The pre-print and submission of this manuscript is currently in progress.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The human brain is a hierarchical computing system comprising billions of neurons that function as computational units. Cognitive neuroscience seeks to understand the intricate workings of brain function, with functional magnetic resonance imaging (fMRI) emerging as a pivotal tool in this research. A significant challenge within this field is to delineate how complex stimuli perceived through various senses, particularly vision, are represented across different brain regions. Artificial neural networks (ANNs), including deep convolutional networks and transformers, have proven to be effective in learning representations that align with neural data. These architectures, characterized by a vast number of parameters, necessitate substantial datasets for effective training.

A common methodology involves utilizing pre-trained networks, followed by the training of a linear readout layer to predict brain activity from the model features. More recently, there has been a shift towards end-to-end training of ANNs using the same stimuli presented to human subjects. However, training these sophisticated brain encoding models poses significant computational challenges, particularly with large-scale fMRI datasets.

In this thesis, we aim to establish best practices for scaling up brain encoding models using one of the largest fMRI datasets currently available, specifically the CNeuroMod dataset. Our research focuses on three critical aspects: data scaling, model scaling, and computational resources. In terms of computational resources, we investigate parallelization techniques such as multi-threading and distributed training for ridge regression, one of the most well-known approaches for brain encoding. Furthermore, we explore the impact of the varying dataset and vision transformer model sizes on the performance of brain encoding models, emphasizing the importance of leveraging large datasets to enhance encoding accuracy.

The overall aim of this thesis is to explore effective strategies for scaling brain encoding models in terms of data, model, and computational resources. By doing so, the aim is to improve both the speed and accuracy of these models, making it feasible to use them in neuroscience applications. This research not only contributes to our understanding of brain function by revealing how the brain encodes complex stimuli but also serves as a guide for practitioners in the field of cognitive neuroscience. It highlights the importance of scalable solutions in brain encoding research and sets a foundation for future innovations in using computational models to decode brain activity from neuroimaging data.

1.1 Functional MRI

There are many methods for brain imaging, with the most commonly used including functional Magnetic Resonance Imaging (fMRI), electroencephalography (EEG), Magnetoencephalography (MEG), and functional Near Infrared Spectroscopy (fNIRS). This thesis focuses on fMRI, which is the technique with the best spatial resolution amongst those listed above. The fMRI activity is measured in tens of thousands of 3D voxels covering the grey matter of the brain, where neuronal bodies are located. Each voxel represents a small cube in space of approximately 2-3 mm size along each dimension. The value at each voxel directly reflects the relative concentration in oxygenated hemoglobyn, which is itself coupled with neuronal activity through metabolism [56]. As fMRI is a space-time acquisition a series of 3D images are recorded in a given session, separated by a sampling time called TR, typically of 1-3 seconds [81, 18].

1.2 Brain encoding and decoding

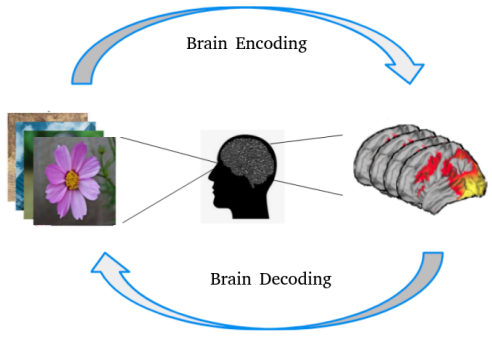


Figure 1.1: Brain encoding and decoding

Thanks to its high spatial resolution, the fMRI data helps cognitive neuroscience researchers to understand which brain networks underpin cognitive processes with more details. There are two common questions in research at the intersection of cognitive and computational neuroscience: (1) determine how the activity inside a specific region can inform us on the behavior or perception of a subject, (2) describe how the stimuli feature space is represented inside a specific region of brain. Brain encoding and decoding analyses are designed to help researchers address these two questions, respectively. An encoding model aims to predict the brain responses in a given area using features extracted from the stimuli. Conversely, a decoding model aims to predict the stimuli features based on measured brain activity patterns [12]. Figure 1.1 illustrates the process of brain encoding and decoding tasks, which are mirror of each other. An example of a brain decoding model would be a situation where a subject is presented with a large number of images depicting objects from different categories, and the model is a classifier where the voxel values are the input of the classifier and the category of an image stimulus is the target of prediction. Conversely, the inputs of a brain encoding model are the features extracted from an image (either raw pixel values, and representation of these values by an artificial neural networks) and the brain response is predicted. A large

number of computational methods have been proposed for brain encoding and decoding in the literature, using stimuli from various domains such as audition [27], language [46], emotion [75], and either static (images) or dynamic (videos) image stimuli [32]. This study focuses on brain encoding models, and state-of-the-art approaches commonly rely on two types of techniques: deep ANNs trained with gradient descent, and ridge regression to map latent representations of the ANNs to brain activity.

1.3 Artificial neural networks as a model of brain representations

Among different types of ANN architectures, transformer based models [73] lead to state-of-the-art performance in diverse applications, such as computer vision and natural language processing. Transformers transform a given token embedding into a new representation that incorporates a relevant context in video or language. The attention mechanism is key to the transformer architecture. Attention is implemented by computing a weighted sum of the embedding of a collection of vision or language tokens where the weights represent the importance of each token with respect to the current position. In the last few years, efficient contextual representations of stimuli have been used to predict brain responses. The results of brain encoding using transformers [53, 51, 19, 36] show improved prediction accuracy when compared to CNN and RNN models [77, 63, 58, 65]. In other words, the attention mechanism seem to produce representations of the stimuli that better align with brain activity. One of the key challenge in brain encoding with transformers is that this type of ANN typically involves a massive amount of parameters which translate in large computations to train brain encoding models. Furthermore, over the past few years, the quantity and quality of fMRI datasets have increased rapidly. The size of fMRI datasets increases along multiple dimensions, such as increasing the number of subjects, scanning hours for each subject, spatio-temporal resolution [4, 48], as well as the diversity of measures collected concurrently with fMRI data (such as eye tracking and biosignals, amongst others). Such deep datasets open the possibility of training ANNs directly on the stimuli used in a neuroimaging experiment, rather than reuse pretrained networks trained by the artificial intelligence research community. Consequently, there is an urgent need to develop scalable approaches for end-to-end brain encoding models, both in terms of computation time and memory requirements.

1.4 Ridge regression

After extracting latent space representations of a stimulus using an ANN, a ridge regression model [23] is trained using pairs of {brain response, stimuli feature} to predict brain activity. Compare to other linear regression models ridge provides better generalization to unseen data through regularization of coefficient estimates [23, 37]. Even though ridge regression utilizes some optimized linear algebra computations, this model still has an intensive computational cost in large-scale brain encoding tasks. Hence, the training process of this linear brain

encoding layer still requires scaling up techniques due to intensive matrix computations over the whole dataset [37].

1.5 Scaling up brain encoding

1.5.1 Increasing the size of fMRI datasets

The traditional fMRI datasets (pairs of fMRI and stimuli) have three weaknesses: covering limited feature space of visual stimuli, poor spatio-temporal resolution, and low Signal-to-Noise Ratio (SNR). These defects lead to poor modeling of brain mechanism and functions. Recently, researchers in cognitive neuroscience have focused on a collective comprehensive dataset to yield significant progress toward a next generation of brain encoding model. Two main trends of the comprehensive datasets are increasing the number of the subjects and expanding the stimuli feature space. Fig. 1.2 shows these two trends where x-axis and y-axis are the number of fMRI scanning hours and the number of the subjects respectively. It is worth to mention that the number of scanning hours indicates the variation of the stimuli and the conditions during the experiments. In Fig. 1.2, the traditional fMRI datasets are illustrated with gray box where the number of the subjects is above 20 and the scanning time is less than 1 hour. In contrast, the top right corner of the figure illustrates an ideal comprehensive dataset, featuring both a large number of subjects and a large number of hours of data per subject. A brain encoding model in that regime requires generalization between subjects and experimental conditions. However, due to the resource limitations (e.g. scanning time for each subject) datasets are able to grow in one dimension rather than both directions that restrict the generalization power. It means that there is a trade-off between the number of subjects and the scanning hours.

Finding the correlation between the huge feature space of natural images and brain activity is complicated. Therefore, understanding the brain hierarchical visual system requires an intensive stimuli set of outdoor and indoor images. For instance, it is possible that the voxels that are correlated with the yellow color were incorrectly correlated with the fruit banana. It is a common issue in previous brain encoding works because of the lack of comprehensive training dataset.

In computer vision studies, researchers have achieved a statistical structure model with human-level accuracy because of existing intensive datasets such as ImageNet [15], COCO [40]. The gap between Neuroscience and Computer vision refers to needing large scale data in the brain encoding and decoding tasks. Datasets BOOlD 5000 [11], Natural Scenes Dataset (NSD)[4] and CNeuroMod dataset [68] provide long fMRI scanning time for a few subjects. The motivation behind these works is that next-generation brain encoding models must be trained using extensive stimuli space. Training purely individual brain models also by-pass the challenges of modelling inter-individual variations in brain organization.

The CNeuroMod research group [68] has released the largest fMRI dataset for individual brain modeling currently available. The large-scale CNeuromod dataset provides a great opportunity to train complex DL-based brain encoding models. For some information about CNeuroMod tasks visit the CNeuroMod web page

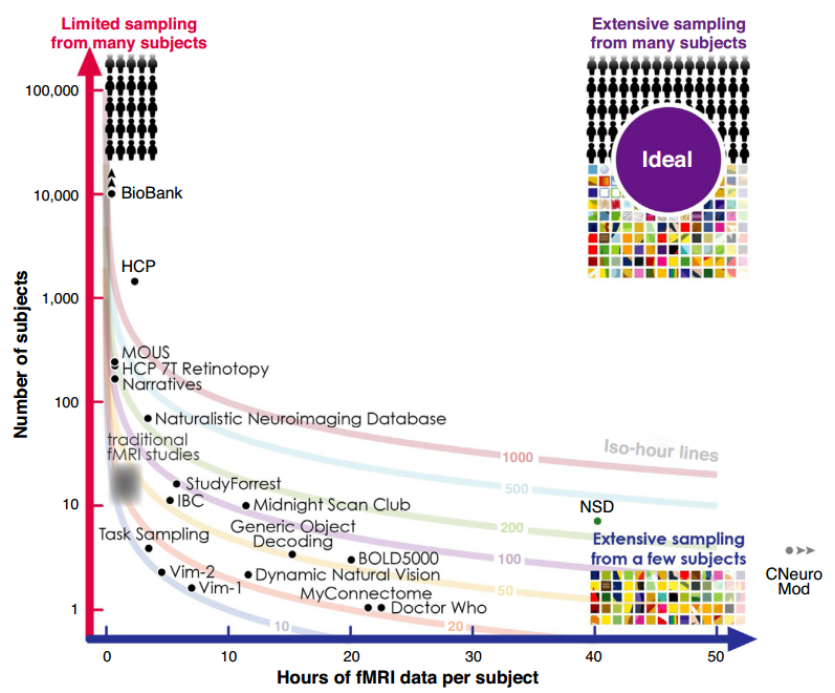


Figure 1.2: fMRI datasets are expanding along two key dimensions: number of the subjects and and number of scanning hours per subject

1.5.2 Increasing model complexity

In addition to increasing the size of datasets, some strategies can be used to improve the quality of brain models, which also increase the computational and memory overhead in brain encoding tasks. In the following, we present some of these strategies.

In most brain encoding studies, the analysis was restricted to voxels in a limited number of ROIs, for example in the visual system. This neglects fMRI signals outside of these specific ROIs, which may be missing significant part of the information. A whole brain encoding approach can uncover a convergence with ANN representations without needing specific assumptions about the location of certain functions in brain activity, and this is particularly important for complex cognitive processes engaging extensive parts of the brain, such as language.

Another strategy is to train brain encoding with full time series instead of the evoked signal. Encoding models can be trained using signal averaged across multiple trials of presentations of a stimuli, in order to increase the signal-to-noise ratio. This approach is common for example when working with presentation of fixed natural images [4]. But it may not be easily applicable to dynamic stimuli such as movies. In this case, brain encoding models can be trained directly from all time samples collected in an fMRI experiment, which greatly increases the number of data points available as target for training.

Furthermore, recently, transformer models with a huge number of parameters trained on massive amounts of data have shown impressive results in NLP and computer vision tasks. The outstanding extracted features (spatial and dynamic) encourage neuroscience researchers to leverage these models in brain encoding studies, such as the ones reviewed in the previous

section. Figure 1.3 illustrates the dramatic growth in the number of parameters in NLP models during the last 5 years: Bert-large (0.3B), GPT-2 (1.5B), Megatron-LM (8.3B), and T5 (11B). One of the key challenges in extracting features through Transformers is thus the large number of parameters to train a brain encoding model due to the large number of features. Training such brain encoding models also requires large amount of brain data. Then, using large-scale transformers in brain encoding tasks due to the limited available memory in the current AI hardware technologies, as out-of-memory issues can clearly occur during the training process with a single GPU or TPU.

Finally, neuroscience researchers have typically relied on pretrained networks developed by AI researchers, reusing the weights of these networks without modification. However, with the increasing availability of large-scale neuroimaging datasets such as NSD [4] and CNeuroMod [68], it is now possible to train moderately sized Transformers directly on the stimuli presented to human subjects. This end-to-end training approach can be advantageous for certain neuroscience experiments, where controlling both the nature and volume of data available to artificial neural networks is crucial. Consequently, neuroscientists must adopt computational tools from the AI community and address the same computational bottlenecks encountered when training large-scale Transformers.

In summary, recent trends in brain encoding studies highlight the growing necessity for researchers to leverage scaling techniques to train large artificial models, including Transformers. This raises the central question of this thesis: how can we overcome the limitations of large-scale brain encoding and train large models more efficiently?

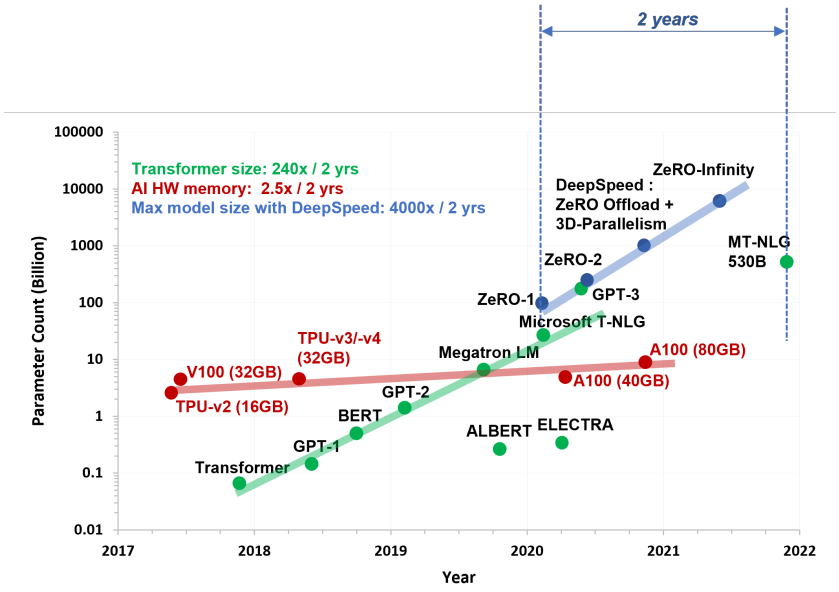


Figure 1.3: Landscape of large-scale models and hardware capabilities. This figure was extracted from Microsoft Blog.

1.6 Outline of the thesis

In this chapter, we reviewed key concepts in brain encoding, emphasizing the significance of scalability in brain representation analysis. We addressed the critical question of why enhancing scalability is essential for accurate and effective brain encoding, considering the growing complexity of neural data and the demands of modern machine learning approaches. In Chapter 2, we provide a detailed review of state-of-the-art brain encoding methods, with particular focus on transformer models and ridge regression techniques, both applicable to a wide range of stimuli. Additionally, Chapter 2 explores high-performance computational techniques, such as parallelism and mixed precision, which enhance the efficiency of model training and deployment. The challenges of scaling brain encoding models are tackled through two primary projects. The first project, discussed in Chapter 3, focuses on scaling up the ridge regression model by optimizing computational resources and distributed training.

The second project, presented in Chapter 4, centers around scaling the VideoGPT model, specifically addressing challenges related to both the size of the dataset and the complexity of the model. This includes strategies for handling larger datasets, increasing model capacity, and optimizing training for more efficient spatio-temporal feature extraction from video data. In Chapter 5, which discusses conclusions and future work, we reflect on the implications of these advancements for the field of neuroscience and outline future directions for further scaling brain encoding approaches. Together, these projects represent significant steps toward making brain encoding models more scalable, efficient, and practical for real-world applications.

Chapter 2

Related works

The study of brain encoding aims to map brain activity to cognitive processes using machine learning models. This chapter delves into the various approaches used in this field, with a particular focus on ridge regression and transformer-based architectures. Additionally, key techniques for scaling neural network training are reviewed, highlighting their importance in enhancing performance for complex brain encoding tasks.

2.1 Brain encoding with ridge regression and Transformer models

2.1.1 Vision transformers

Colin Conwell and colleague [13] discuss how computer vision models can help us understand how the human visual system works. In this paper, the authors analyze a large-scale benchmarking based on 85 modern deep-learning models to see how differences in model architecture (ConvNets, MLP-Mixers, and Transformers) and training tasks (2D, 3D, Semantic, and Geometric) contribute to the prediction of brain activity. The benchmarking results show that even when the architecture of the models was very different, they still performed similarly. Importantly, they found that even though convolutional neural networks are based on operations believed to mimic early layers of the human visual cortex, transformer architectures performed as well without featuring such inductive biases on the model representations. They also found that models trained to categorize images (semantic tasks) were more efficient at predicting brain activity.

2.1.2 Language transformers

Pasquiou and colleagues [53] compared several types of NLP models in terms of their ability to encode brain activity. The NLP models included word embedding such as GloVe, recurrent architecture (LSTM) and transformer-based architectures (GPT-2 and BERT). Figure 2.1 illustrates the proposed brain encoding model. Subjects were presented with "The Little Prince" audio storybook while their fMRI brain activity was recorded. NLP models were trained using different text corpus (a curated corpus matching the audiobook, and a

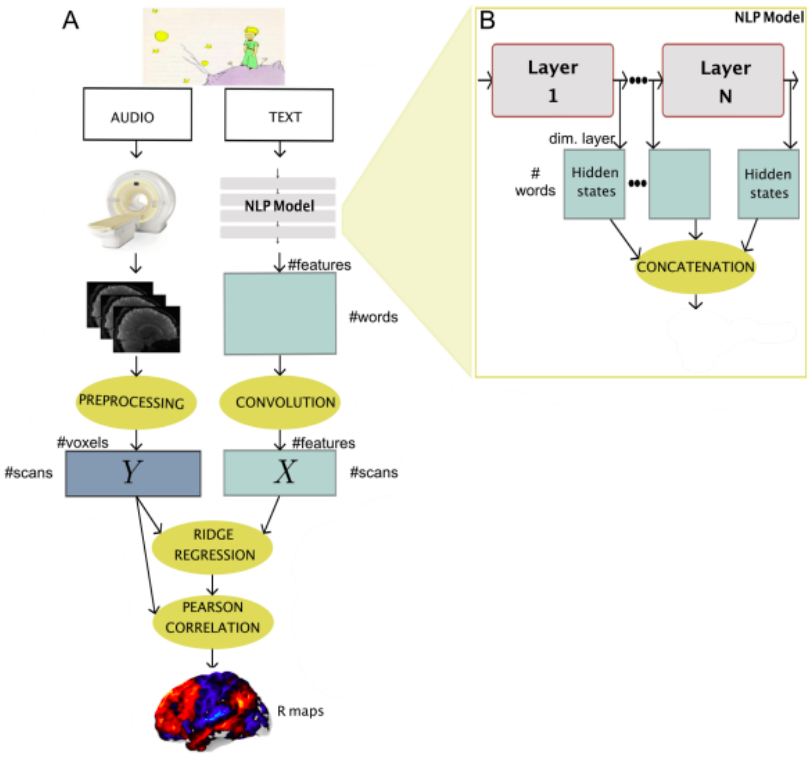


Figure 2.1: Brain encoding using NLP models while subjects were presented with an audiobook of the story "The Little Prince". Figure taken from [53].

non-curated corpus from Wikipedia). Pairs of fMRI brain data and corresponding layer activations of NLP models were considered as the training data of the Ridge-regression model. Finally, brain maps were computed based on correlation coefficients between ridge model predictions and actual fMRI data. The authors investigated the impact of several factors of NLP models on the brain encoding results, such as model architecture, model perplexity, and type of training data. The perplexity of an NLP model is a statistical measure that indicates how accurately a model can predict either the next word in a sentence or masked words within a sentence, which is a common type of self-supervised learning task in NLP. The results of this paper show that the untrained versions of these models are able to predict a significant amount of signal in the brain by capturing identical words present in the stimuli text. Training these NLP models leads to improving the prediction of brain encoding independently of the model's architecture (Figure 2.2). Models that integrate contextual information, such as LSTMs predict brain activity more accurately than static models such as GloVe. The perplexity measure was not found to be an efficient predictor of the quality of brain encoding. And finally, the training data have a large impact on brain encoding results. The authors compared a carefully curated text corpus with a Wikipedia corpus, and noted that the non-curated Wikipedia corpus led to poorer brain encoding.

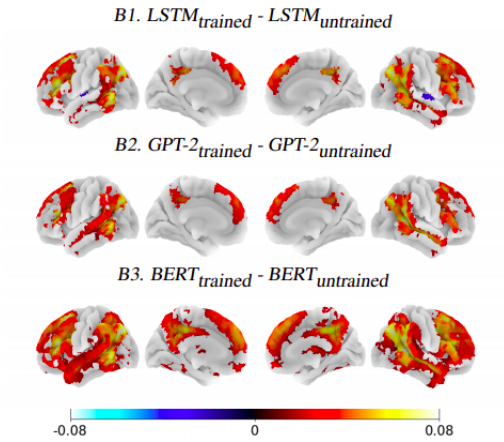


Figure 2.2: Impacts of trained and untrained three types of NLP models on brain encoding. There is a significant overlap in brain areas identified with LSTM, GPT2 and BERT in the brain encoding task [53].

2.1.3 Layer-wise encoding with language transformers

In [19], Goldstein and colleagues proposed a GPT-2 layerwise brain encoding model. The proposed approach is a standard ridge regression, quite similar to the previous study. In this work for each word in the story and from each layer of GPT-2, a contextual embedding was extracted. The dimensions of this embedding was reduced to 50 through a principal component analysis. Pairs of {extracted embedding, brain activity} were presented as training data for a ridge regression. After this training step, the ridge regression model was able to predict brain activity using word embeddings as inputs, in a held-out test set. To evaluate the performance of the proposed model, a Pearson correlation (r) was derived between the predicted brain activity values and the actual value.

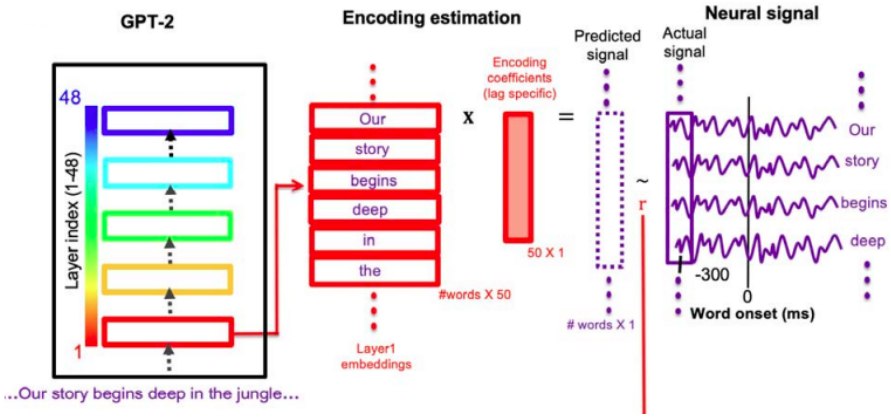


Figure 2.3: GPT2 layer-wise brain encoding [19]

This work confirmed the lack of correlation between Transformer model accuracy and brain encoding efficiency. Figure 2.4 shows the performance of brain encoding for all hidden layer

(1-48) individually. This result is based on mean brain encoding performance across all brain regions for each individual layer.

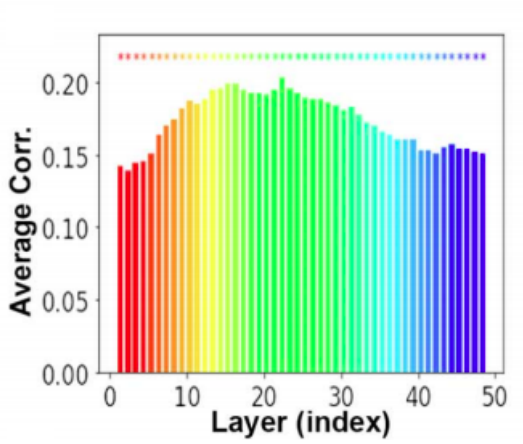


Figure 2.4: Brain encoding analysis for each individual hidden GPT-2 layer (1-48) [19]

A similar result was derived in [36], where fMRI data were recorded while subjects listened to naturalistic spoken stories. In the brain encoding task, multiple types of features are used to predict brain activity. These features are extracted through classical linguistic models, static models and Transformer model (BERT-base). Pairs of extracted features and fMRI data are presented as training data to a banded ridge regression [37]. This work broke down the computations of transformers into individual layers to gain more information about linguistic computations in the human brain. Through layer-wise brain encoding, for each region the preferred layer was determined regarding to the predict correlation values. Figure 2.5 shows the preferred embedding and transformations layers for each region of the brain. Most brain regions prefer the last embedding layers while the transformations layers discover a hierarchy of brain maps across layers.

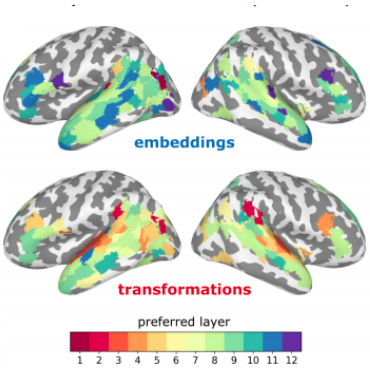


Figure 2.5: Brain maps based on BERT layer preferences. [36].

2.1.4 Multimodal transformers

Oota and coll. [51] used multimodal (visual and text) transformers to extract features from rich stimuli. The authors extracted activations of Transformers such as ViT, VisualBERT,

and LXMERT and used these activations as inputs to the ridge model to predict the brain responses to the same stimuli as those presented to the Transformers. Figure. 2.6 illustrate the proposed brain encoding approach in this work. The brain encoding results were evaluated by computing Pearson's correlation between actual time courses of brain activities and predicted time courses generated by the ridge regression. The main contributions of the Lanq-vision study were twofold. First, the authors developed a multimodal brain encoding mixing visual and text inputs instead of separate models of each input type separately, as was done for example by [13] for vision and [10] for language. This multimodal model may capture more variance in human brain activity as the brain constantly receives information about the environment across multiple modalities. Second, the authors mapped a hiercharchical organization in brain processing by testing different brain encoding models, starting from activations with different layers of the Transformer models. The study applied brain encoding to two large established datasets, BOLD5000 [11] and Pereira [55] . Results showed that multimodal Transformers significantly outperformed single-modality models such as convolutional neural networks. Furthermore, the results illustrated that specific regions such as LPTG, LMTG, LIFG, and STS were better encoded using multimodal models. In the following we provide more details about this brain regions: 1) LPTG is a brain region for attention, memory, and space-time integration and located in the parietal and temporal lobes, 2) LMTG is located in temporal lobe associated with language, meaning, and visual perception, 3) LIFG is located frontal lobe for speech production and language comprehension and 4) STS is located in temporal lobe with a role in social perception and communication cues.

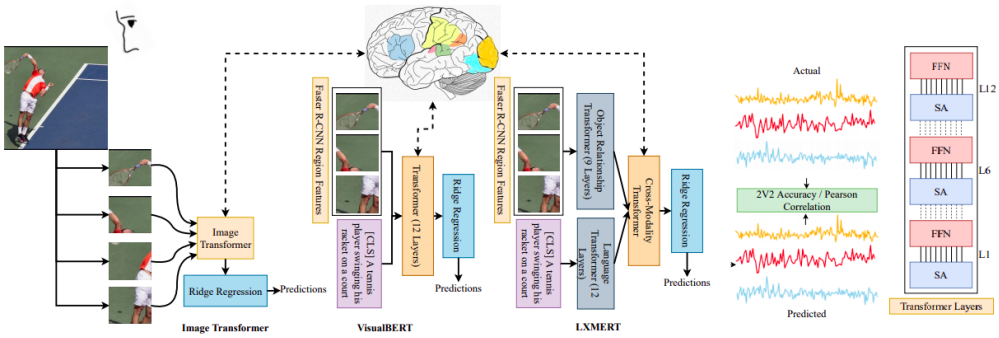


Figure 2.6: Mimicking the human brain behavior using multi-modal vision and languages Transformers. The co-attentive stimuli features was considered as input of ridge regression [51]

2.2 Scaling up techniques

The main techniques to scale-up the training of artificial neural networks include parallelism-based approaches and memory-based approaches. This section gives an overview of the main current techniques in each category.

2.2.1 Parallelism-based approach to ANN training

In the DL training process, the computations involve many matrix multiplications which leads to opportunities for parallelization. In this section, we review the main parallelism approaches to scale up DL models and we investigate their strengths and weaknesses with respect to DL-based brain encoding. We consider that the training is parallelized over a set of workers that may represent cores of a CPU or GPU and are potentially distributed over multiple computing nodes.

2.2.1.1 Data parallelism

In data parallelism, each worker loads a copy of the DL model and trains it using a non-overlapping block of data. Two approaches exist to implement data parallelism: the centralized approach and the decentralized approach. In the centralized approach, each worker computes the gradient of the loss function on its block of data and sends it to a central module. The central module receives gradients from all workers, computes the gradient average, updates the model using back-propagation, and sends the updated model to the workers. In the decentralized approach, the workers communicate with each other to update their parameters, which can lead to substantial communication overheads in the default "all to all", fully-connected communication strategy. The ring-allreduce strategy reduces the overhead significantly [44]. In this approach, each worker communicates only with two other workers: each worker sends data to its left neighbor, and receives data from its right neighbor. Both the centralized and the decentralized approach introduce substantial synchronization overheads. Indeed, the model needs to be updated periodically, for instance at the end of every batch, which requires all the workers to pause until the updated model is available.

2.2.1.2 Model parallelism

In model parallelism, the layers of the DL model are distributed among the workers. The training data is presented to the worker holding the input layer. In the forward pass, each worker computes the activation of its assigned layer(s) and send it to the worker that holds the next layer. Conversely, the backpropagation starts with the worker holding the output layer, the gradients are computed in each worker, and the results are propagated toward the worker holding the input layer [44]. This approach results in substantial synchronization overheads as layers cannot be concurrently evaluated. The main advantage of model parallelism is to distribute memory contraints among compute nodes.

2.2.1.3 Pipeline parallelism

Pipeline parallelism is a combination of data parallelism and model parallelism. In this technique, both data and model are divided between workers. Each worker trains a specific set of layers of the DL model using a non-overlapping part of the dataset. Pipeline parallelism improves scalability compared to data and model parallelism by reducing synchronization overheads. Training different parts of the model with multi-batches leads to increasing the utilization of cores significantly. For instance, Fig.2.7 shows a pipeline parallelism to train a DL model with multiple-batches across four cores. In this figure, terms $F_{n,k}$ and $B_{n,k}$

represent the computations of kth batch size on nth core in the forward and backward pass respectively. It is worth mentioning that, in this process, $B_{k,n}$ is dependent on both $F_{k,n}$ and $B_{K,n+1}$ [28].
However, pipeline parallelism comes at an increased communication cost since layers have to repeatedly transmit activations during the forward pass and gradients during the backward pass, which are constant in size. Overall, the trade-off between reduced synchronization delays and increased communication costs is usually beneficial to pipeline parallelism [44].

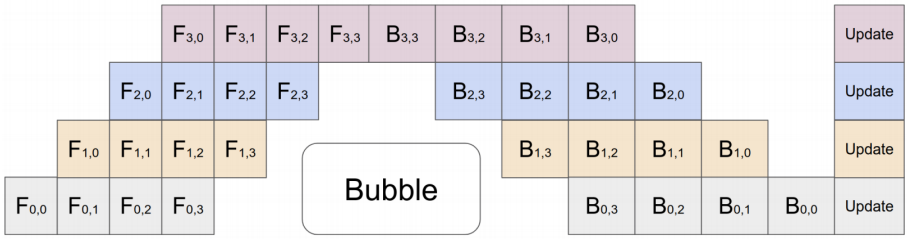


Figure 2.7: Forward and backward steps in the pipeline parallelism for four cores. This figure was extracted from [28]

2.2.2 Memory-based scaling up of ANN training

In the forward pass, the layer activations are computed and stored in memory to update the parameters during the backpropagation phase. Layer activations represent a substantial amount of memory. For instance, training ResNet50 on the ImageNet dataset requires about 40GB of memory [21]. Mixed-precision approaches have been used to reduce memory consumption and communication overheads, leading to important reductions of the training time [45].
The main idea of mixed precision is to use 16-bit floating-point formats instead of 32-bit formats. Three advantages of this method are: reducing the memory usage, reducing the communication delays, and accelerating floating-point computations. During the training process, parameters, activations and gradient values are stored in memory with a 16-bit format. Then, all the computations in the forward and backward pass are performed with 16 bits. To preserve accuracy, it is necessary to maintain a copy of parameters in FP32 format. Consequently, after each iteration, parameters are updated from their values expressed in FP32 format. The NVIDIA research group reported that training a large scale natural language model (RNN-NLP) using mixed precision approach is 4.2 times faster than FP32 on the same platform [57]. In addition, the Facebook research group showed that an RNN model can converge 5 times faster than baseline when using mixed precision [52].
Representing the gradient values in FP16 creates an issue called Gradient Underflow. When gradient values are too small (negative exponents), they are represented as zeros in the FP16 format while a large range of FP16 remain unused. In [45], the authors proposed a solution to address this problem by shifting the gradient values through a scaling factor of 8, 32, 64, or 128, or tuned automatically during the training process. In the automatic approach, the scaling factor is initialized with a large number, for instance 128. During the

training process, the scaling factor is decreased when gradient values are Inf or NaN, and it is increased otherwise.

2.3 Conclusion

In this chapter, we explored a range of innovative approaches in the field of brain encoding using ridge regression and transformer models. The reviewed literature highlights the significant role that both vision and language transformers play in understanding and predicting brain activity. Importantly, the research indicates that the training tasks and data quality significantly influence the brain encoding outcomes, with models that integrate contextual information proving to be more effective. These findings pave the way for further investigation into optimizing model architectures and training data selection for improved brain activity prediction.

Moreover, this chapter also delved into scaling techniques essential for enhancing the training of artificial neural networks, particularly in the context of brain encoding. By examining parallelism-based approaches, including data and model parallelism, we gained insights into how these methods can effectively distribute computational workloads while addressing the inherent synchronization and communication challenges. Techniques like pipeline parallelism emerged as promising strategies to balance computational efficiency with communication costs, showcasing the importance of optimizing training frameworks for the next generation of deep learning models.

Overall, the integration of transformer architectures and effective scaling techniques holds great potential for advancing our understanding of neural mechanisms underlying human cognition and behavior.

Chapter 3

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

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 work 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 resources 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, 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× 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.

3.1 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 [50], 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 [23] has become popular and well-accepted [76, 53, 13, 19, 51, 36, 38, 35, 30, 77] 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 [37] 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 [63, 33, 34, 26, 7, 58, 65]. For instance, in [77], 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 [37], 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 number 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 [49]. 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 [11], Natural Scenes Dataset (NSD)[4] 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 [61]. As a consequence of increased spatial resolution and volume of time samples available for a single subject with the advance of simultaneous multislice fMRI [6], 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 [68] 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 [54] 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 [78] or the proprietary Intel oneAPI Math Kernel Library (MKL) [74]. Both of these linear algebra libraries support multi-threading on a single CPU. Moreover, scikit-learn models rely on the Joblib library to interface with various parallelization backends including Dask [62], which can be used to distribute computations across multiple compute nodes. Specifically, We utilized scikit-learn's MultiOutput regressor (MOR), 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 introduced Batch MultiOutput (B-MOR) to train a series of models 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.

3.2 Materials and Methods

3.2.1 fMRI dataset

We used the 2020-alpha2 release of the Friends fMRI dataset collected by the Courtois project on neuronal modeling, CNeuroMod [9]. Some of the text in this section is adapted from the Courtois NeuroMod technical documentation (`https://docs.cneuromod.ca`).

3.2.1.1 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.

3.2.1.2 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.

3.2.1.3 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 [64, 79] developed at the University of Minnesota, as part of the Human Connectome (HCP) Project [72]. 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.

3.2.1.4 Preprocessing

All fMRI data were preprocessed using the fMRIprep pipeline version 20.2.3 [17]. 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` tool[1] (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).

3.2.1.5 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

[1] https://github.com/simexp/load_confounds

Table 3.1: Brain datasets summary: number ($t \times s$) of time x space samples and size (in MB or GB) of fMRI time series in three resolutions. MOR and B-MOR indicate the Multioutput Regression and Batch Multioutput Regression, respectively.

Resolution	Subject	t	s	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

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 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 [1] 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) [70]. 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). As the parcels are based on a group template, and the cneuromod fMRI is high spatial resolution with 2 mm isotropic voxels, there are sometimes marked discrepancies between the location of individual gray matter and parcels of the MIST atlas, even after non-linear wrapping in the MNI template space. This issue, common in group parcellation analyses, is mitigated through spatial smoothing. We thus implemented spatial smoothing for the preprocessing of fMRI data, as is standard practice for generating time series for group parcels. Although this may not be an optimal strategy, our aim here was not to push the boundary of quality in brain encoding, but rather demonstrate the scalability of a standard brain encoding workflow.

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. Our primary motivation was to use the same dimensions across subjects to harmonize the amount of spatial smoothing, in order to make brain encoding scores more comparable. Overlapping group regions with individual gray matter segmentation is also non-trivial, and we wanted to apply a standard, straightforward preprocessing strategy. 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 resampled 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 3.1 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 (t time sample) and targets (s 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 3.1, 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.

3.2.2 Brain encoding

Figure 3.1 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 3.1 to test the scaling efficiency of different implementations of the ridge regression.

3.2.2.1 VGG16 artificial vision network

In this work, we used the approach of [77, 14], and applied a VGG16 model [66] 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 [16], and the weights of the models were retrieved through TensorFlow. This

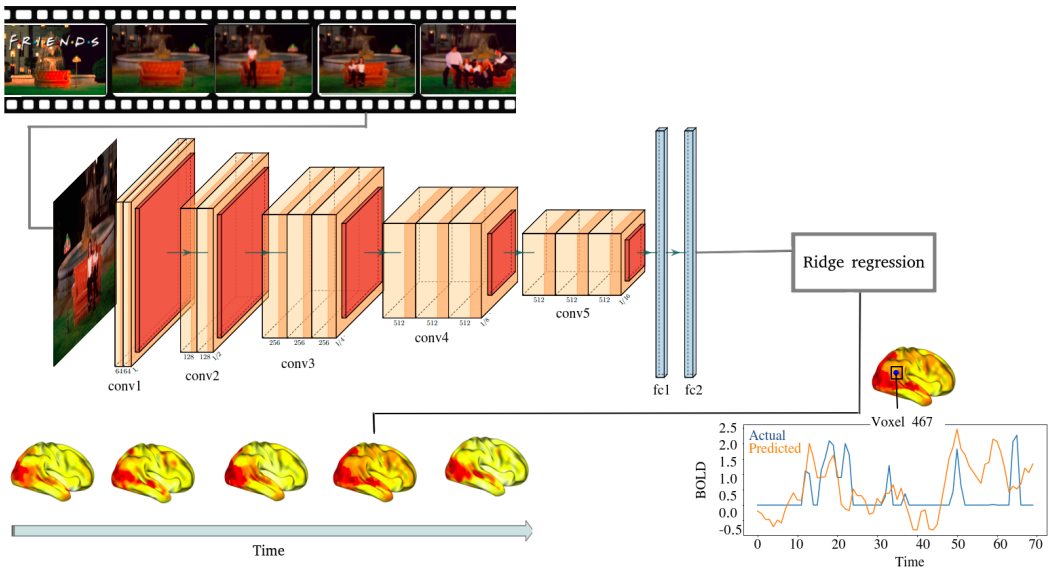


Figure 3.1: The two main steps of brain encoding: Extracting features from movie frames using VGG16 pretrained model and predicting brain response using ridge regression.

model achieved 92.7% top-5 test accuracy for image-object classification. The VGG16 architecture [67] is a widely-used convolutional neural network (CNN) known for its simplicity and effectiveness in image classification tasks [14]. The network comprises 16 layers, including 13 convolutional layers and 3 fully connected layers. The convolutional 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 3.8 (based on the TensorFlow summary of the model), making it suitable for various computer vision applications.

3.2.2.2 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 x 1.49 = 5.96s 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. We concatenated VGG16 features across four TRs, starting with the TR preceding the image frame of the video, resulting into a single feature vector of length $p = 16384$. Our feature space thus approximately covers the interval [-7.5 sec, -1.5 sec] (the sequence TR=1.49, close to 1.5). This is the time window which achieves strong hemodynamic effects, at least using canonical models of the hemodynamic response function. We also explored the impact of the size of the time window on brain encoding accuracy (results not shown) and found that a window length of 4 TRs maximized the quality of brain encoding. 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).

3.2.2.3 Ridge regression

Ridge regression was first proposed by Hoerl and Kennard [23] as a generalization of ordinary least-square regression. In comparison with ordinary least squares regression, ridge regression provides better generalization to unseen data through regularization of coefficient estimates, particularly in the presence of a large number of predictor variables. Ridge regression is expressed as the following optimization problem solving for regression coefficients b^* independently at each spatial location:

$$b^* = \underset{b \in \mathbb{R}^P}{\arg\min} \left(\|Y - Xb\|_2^2 + \lambda \|b\|_2^2 \right), \tag{3.1}$$

where $X \in \mathbb{R}^{t \times p}$ is the matrix of stimuli features with t time samples and p features, $\|.\|_2$ is the ℓ^2 norm of a vector, and $Y \in \mathbb{R}^{t \times s}$ is the target matrix 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 [37].

3.2.2.4 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. 3.1 based on the training set only. Table 3.2 presents the memory size and number of ridge training parameters with three levels of resolution across six subjects.

Table 3.2: Number of training parameters (rounded to closest M) and size of weight matrices in three resolutions for brain encoding. MOR and B-MOR indicate the Multioutput Regression and Batch Multioutput Regression, respectively.

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
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 the cost function defined in Eq. 3.1), 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 s brain targets, or a common value can be selected for all brain targets based on the average performance of the model across all s brain targets. In this work, a single λ is used for all targets.

3.2.3 Ridge regression implementations

3.2.3.1 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 [37] and is implemented in the scikit-learn library.
In a multi-target regression problem, the matrix Y in Equation 3.1 is of size $t \times s$ where s is the number of spatial targets. Matrix X is still of dimension $t \times p$. The weight matrix W can be calculated as follows:

$$W = MY \tag{3.2}$$

where

$$M = (X^T X + \lambda I_p)^{-1} X^T \tag{3.3}$$

and I_p is the identity matrix of size $p \times p$. The key point is that M is independent of Y and can therefore be reused for all s targets. This strategy reduces the time complexity of multi-target ridge regression from $O(p^3 s + p^2 ts)$ to $O(p^3 + p^2 t + pts)$ [37], see Section 3.3 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:

$$X = USV^T, \tag{3.4}$$

where $U \in \mathbb{R}^{t \times p}$ 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:

$$M(\lambda) = V(S^2 + \lambda I_p)^{-1} SU^T \tag{3.5}$$

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 $O(p^3 r + tpr)$ to $O(p^2 tr)$ [37], where r is the number of tested hyper-parameter values (see Section 3.3 for details).

3.2.3.2 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 hyperthreaded 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.

3.2.3.3 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) [74] and OpenBLAS [78]. 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, see [29] for a benchmark using MKL. Note that the optimization we investigated here with MKL is only available for Intel(c) hardware. Figure 3.2 summarizes the different parallelization modes benchmarked by our experiments.

3.2.3.4 Distributed parallelism

In ridge regression, predicting one target is independent of the values of other targets due to the way the model is structured. 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 s sub-problems, each corresponding to a single spatial target. Ridge regression can now be expressed as solving s independent estimations of the weights matrix W, as illustrated in Figure 3.3. This subdivision 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. Joblib supports multiple execution backends including single-host thread-based or process-based parallelism, and distributed parallelism using the Dask [62] or Ray [47] engines. We used the Dask backend and launched its distributed scheduler that simultaneously manages the computation requests and tracks the compute node statuses.

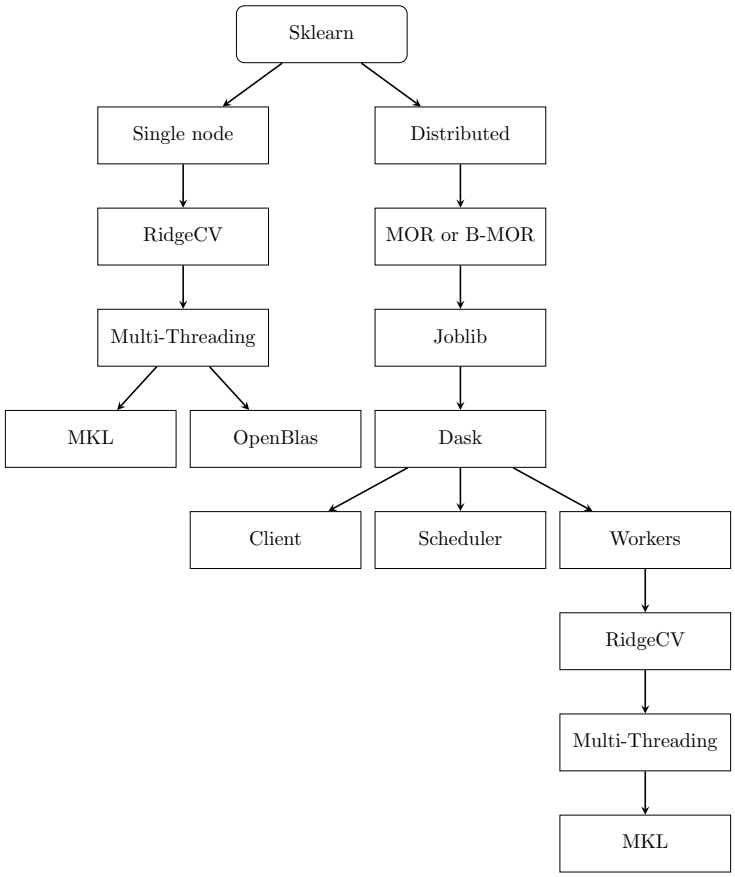


Figure 3.2: Multithreading and Distributed Parallelism in Scikit-learn's Ridge Regression. MOR and B-MOR indicate the Multioutput Regression and our proposed version of Batch Multioutput Regression, respectively

3.2.3.5 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 3.3). 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 $Y_1, \cdots, Y_n$. The algorithm uses the following helper functions:

- `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.

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

input : X—Input stimuli feature matrix
input : Y—Target matrix
input : s—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

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

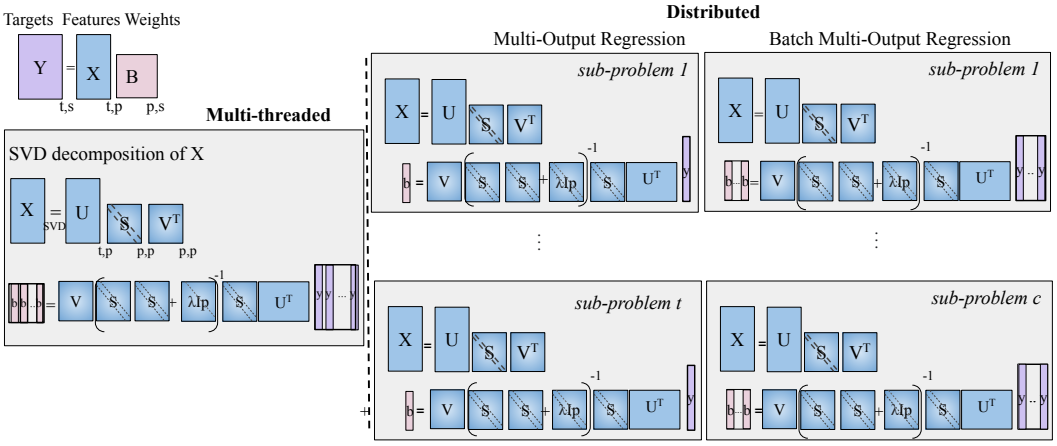


Figure 3.3: Matrix computations in Multi-threading ridgeCV, MOR, and B-MOR model fitting. Assuming $X \in \mathbb{R}^{t \times p}$, $Y \in \mathbb{R}^{t \times s}$, and $X = USV^T$, then the weight matrix $B \in \mathbb{R}^{p \times s}$ equals $B = V(S^2 + \lambda I_p)^{-1} SU^T Y$.

3.3 Complexity analysis

3.3.1 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 $XM_\lambda Y$. Matrix notations as well as their dimensions are summarized in Table 3.3. Time complexities are expressed in the number of floating-point multiplications.
The cost T_W of computing $XM_\lambda Y$ independently over r values of hyper-parameter λ is:

$$T_W = O(ptsr).$$

Regarding T_M, Equation 3.3 requires inverting a square matrix of size p — time complexity $O(p^3)$ — and multiplying the resulting inverted matrix with matrix X^T of size $p \times t$ — time complexity $O(p^2 t)$. Matrix M_λ is computed once for all the targets. With a single hyper-parameter value ($r = 1$), Equation 3.3 thus gives a complexity of $O(p^3 + p^2 t)$.
If we were to naively iterate this approach for r hyper-parameter values, the resulting time complexity would be $O(p^3 r + p^2 tr)$. However, expressing matrix M_λ from the SVD of matrix X as in Equation 3.5 reduces the time complexity of computing M to:

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

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

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:

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

Table 3.3: Matrix Sizes. t: number of time samples; p: number of features; s: number of brain targets. Other important notations include c: number of concurrent distributed executions and r: number of hyper-parameters.

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

3.3.2 Ridge regression with MOR

In the case of MOR, the matrix multiplication $M_\lambda Y$ is replaced by s multiplications of M with a vector y, which does not change the time complexity. Provided that the number of targets s 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 sT_M distributed over c concurrent processes. Overall, the computational cost of ridge regression implemented with MOR is:

$$T_{\text{MOR}} = c^{-1}(T_W + sT_M). \tag{3.6}$$

3.3.3 Ridge regression with B-MOR

B-MOR scales better than the previous approach due to the use of c sub-problems instead of s. 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 executions. The computational cost of ridge regression implemented with B-MOR is thus:

$$T_{\text{B-MOR}} = c^{-1}T_W + T_M. \tag{3.7}$$

Comparing Equations 3.6 and 3.7, we observe that the time complexity of the B-MOR implementation is much lower than for the MOR implementation, as $T_{\text{MOR}} - T_{\text{B-MOR}} = (c^{-1}s - 1)T_M$. This difference may be massive when $c \ll s$. 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.

3.4 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).

3.4.1 Brain encoding models successfully captured brain activity in the visual cortex

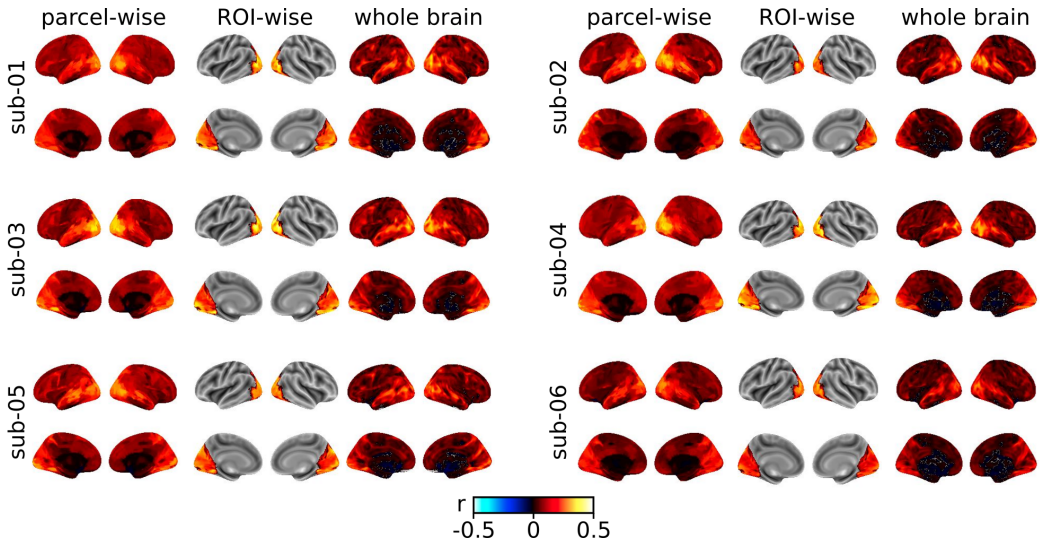


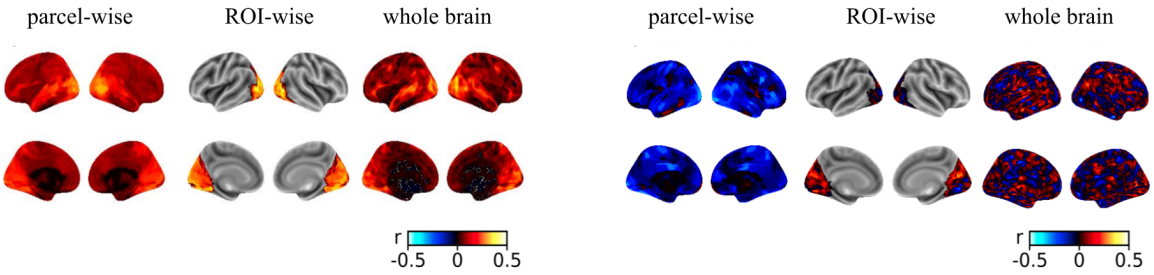
Figure 3.4: Brain encoding results, with performance based on Pearson Correlation Coefficient (r) between real and predicted time series in the friends dataset (N=6 subjects).

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 [66]. 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 3.4 shows the functional alignment between the features of the last fully connected layer (FC2) of VGG16 brain activities and brain activity for six subjects. 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. Brain encoding maps were highly consistent across subjects and resolutions of analysis. Specifically, the spatial Pearson's correlation between brain encoding maps of different subjects ranged from 0.79 to 0.87 (mean 0.83 across 15 distinct pairs of subjects), see supplementary material Figures 3.12a and Figure 3.12b.
Overall, brain encoding models successfully predicted activity in expected brain regions, for all subjects and resolutions.

3.4.2 Brain encoding was significant compared to a null distribution

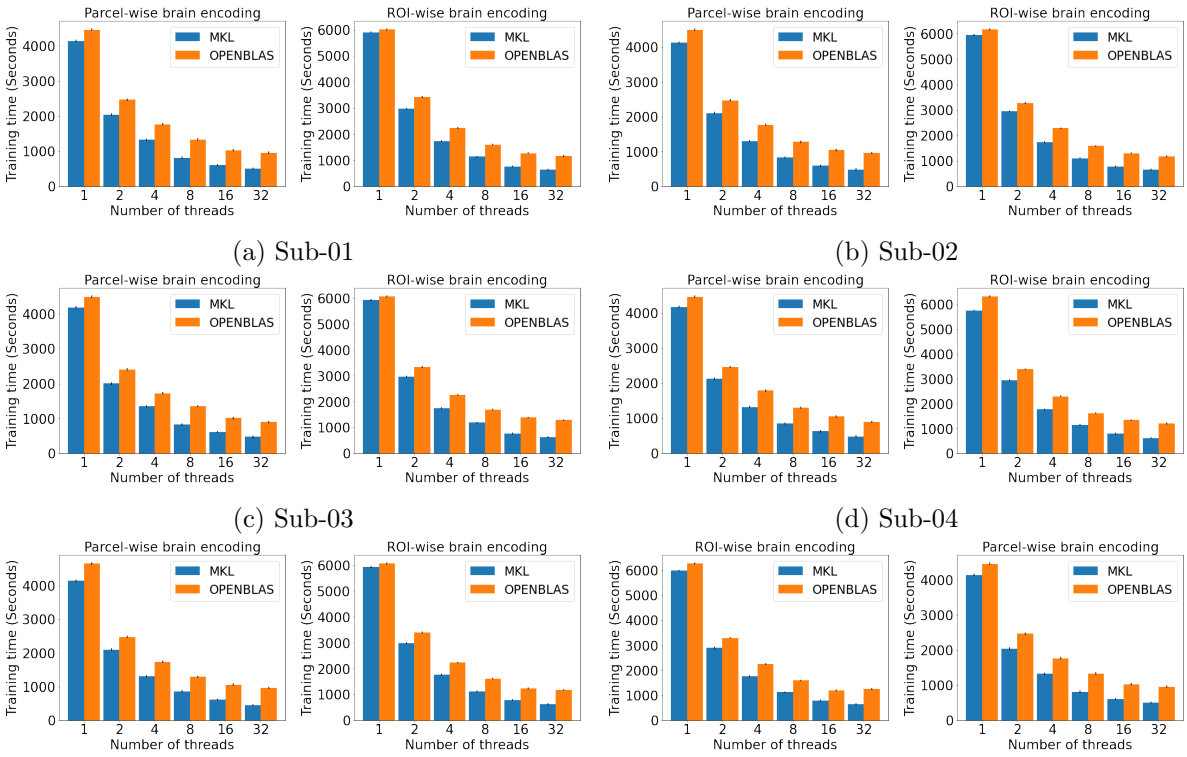
Next, we wanted to assess the significance of the brain encoding, compared to a null distribution were the movie frames used as input to the model did not correspond with brain data time series. We repeated the brain encoding procedure presented in the previous section for one subject (sub-01), after random shuffling of the image features and brain images. Figure 3.5 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. The dummy model produces predictions that are no better than random noise, resulting in a mix of positive and negative correlations. As brain activity is structured, the dummy model's inability to capture this structure can lead to negative correlations by chance. Additionally, the dummy models cause negative correlations because the predicted values systematically deviate from the true values in an opposite manner. We believe this is a reflection of over-fitting the training set, which could have been mitigated through more aggressive regularization.



(a) Actual performance

(b) Shuffled performance

Figure 3.5: 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.



(a) Sub-01 (b) Sub-02

(c) Sub-03 (d) Sub-04

(e) Sub-05 (f) Sub-06

Figure 3.6: Comparison of MKL and OpenBLAS implementations for multithreaded execution.

3.4.3 Brain encoding captures both low-level and high level visual cognitive functions

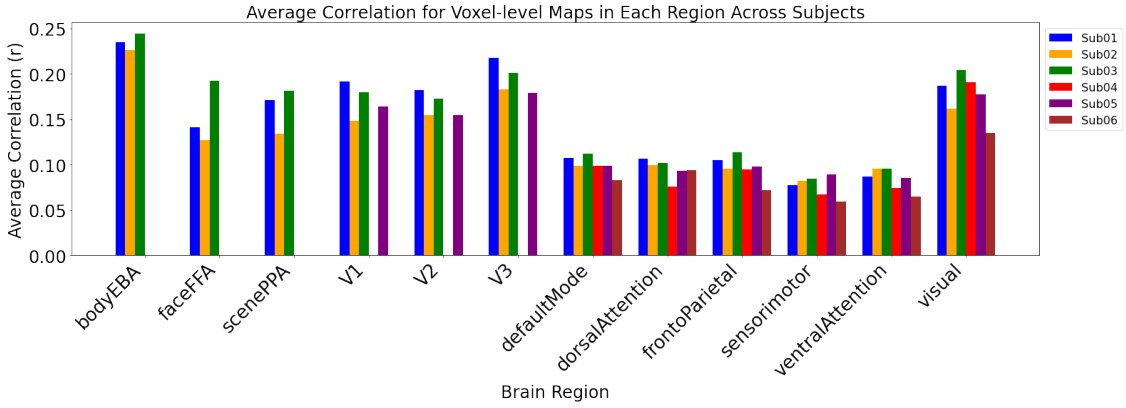
In this study, we utilized early visual ROIs derived from retinotopy and higher visual ROIs derived from fLoc tasks. For the early visual ROIs, regions of interest (ROIs) such as V1, V2, V3, V3a, V3b, VO1, VO2, hV4, LO1, LO2, TO1, and TO2 were identified for four subjects (sub-01, sub-02, sub-03, and sub-05) who completed a retinotopy task. These ROIs were derived from their population receptive fields and group priors using the Neuropythy toolbox [8], with masks available in both MNI and native (T1w) space.

For the higher visual ROIs, regions such as the extrastriate body area (body-EBA), fusiform face area (face-FFA), occipital face area (face-OFA), posterior superior temporal sulcus (face-pSTS), medial place area (scene-MPA), occipital place area (scene-OPA), and parahippocampal place area (scene-PPA) were identified for three subjects (sub-01, sub-02, and sub-03) who completed the fLoc task. These higher-level visual area ROIs, reflecting preferences for faces, scenes, and bodies, were identified using a combination of group priors and individual data, with binary ROI masks available in both MNI and native (T1w) space. Additionally, group parcels of regions with face, scene, and body preferences identified by the Kanwisher lab [31] were also transformed into single-subject space and are available in MNI space in a standardized format fitting all subjects

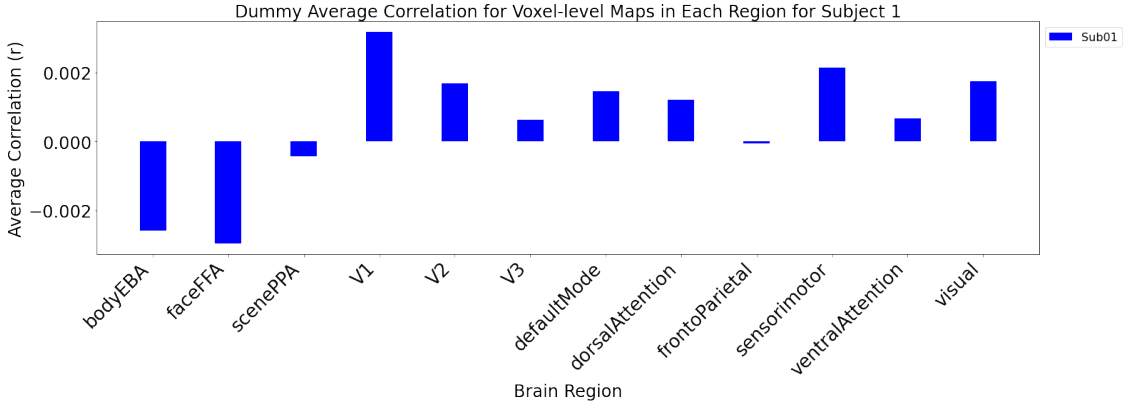
Figure 3.7a presents the average correlation for voxel-level maps in each brain region across multiple subjects (sub-0[1-6]) using a trained model. Each color represents a different sub-

ject, and the bars show significantly higher correlation coefficients compared to the untrained model. This indicates that the trained model successfully captures the underlying patterns in the data, leading to better predictions. The trained model shows consistently high correlations across various brain regions for all subjects, demonstrating its effectiveness in generalizing across different individuals. The improvement in correlations from the untrained to the trained model underscores the impact of model training on enhancing predictive performance in brain imaging data.

Figure 3.7b illustrates the average correlation for voxel-level maps in each brain region for Subject 1 using a dummy (untrained) model. The bar plot shows the average correlation coefficient (r) for various brain regions, including early visual areas (e.g., V1, V2, V3) and higher-level visual areas (e.g., bodyEBA, faceFFA, scenePPA). The correlations are generally low, indicating that the untrained model does not capture the patterns in the data effectively. This serves as a baseline, showing the initial performance before any training or optimization is applied. The low correlations across most regions highlight the need for model training to improve prediction accuracy.

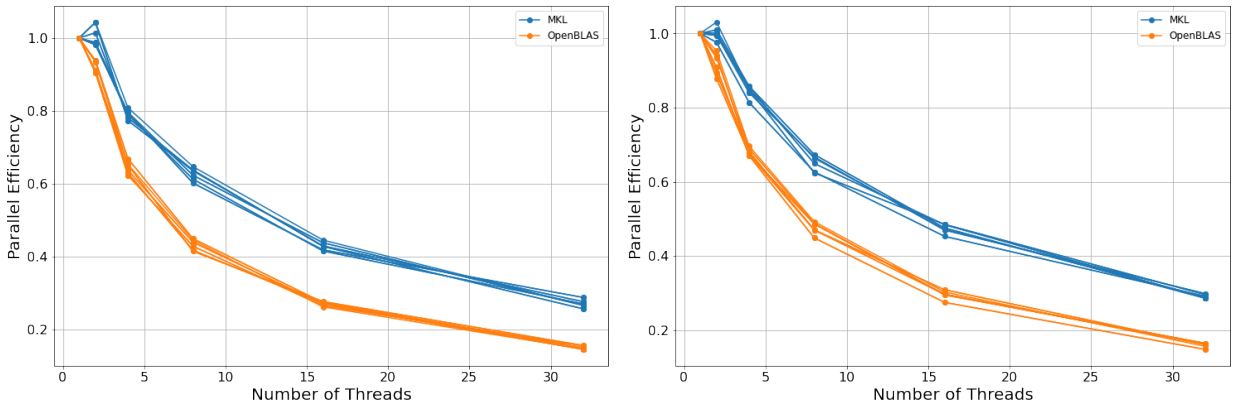


(a) Trained brain encoding



(b) Shuffled (dummy) brain encoding

Figure 3.7: Average correlation for voxel-level maps in each brain region



(a) Parcel-wise

(b) ROI-wise

Figure 3.8: Parallel efficiency of RidgeCV execution time for MKL and OpenBLAS across different numbers of threads. The plot on the right shows parcel-wise brain encoding, while the plot on the left shows ROI-wise brain encoding.

3.4.4 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 scikit-learn on a 32-core compute node, and comparing the libraries underlying multithreaded parallelization, i.e. MKL and OpenBLAS. Figure 3.6 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.

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

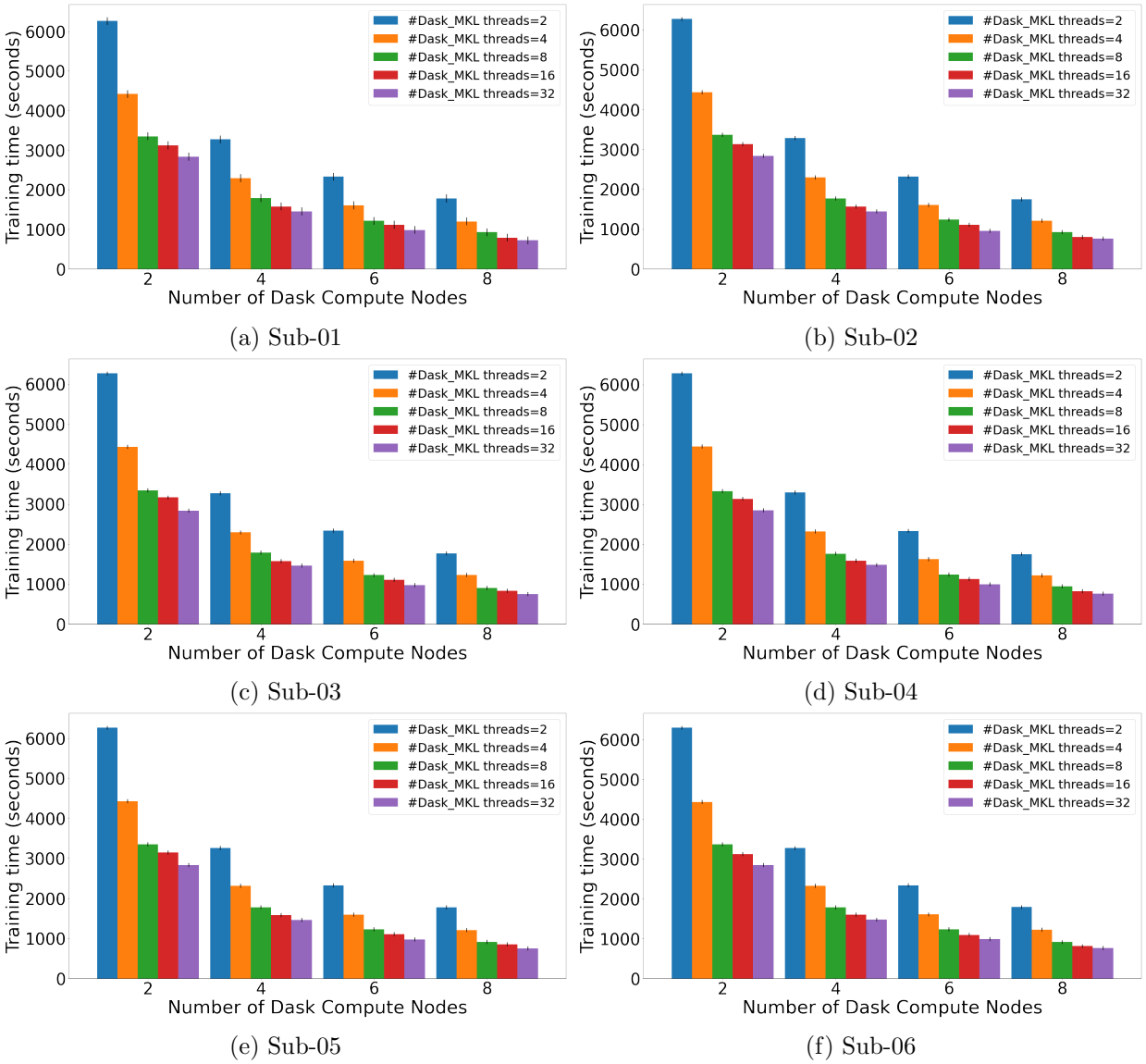
We also observed a sharp decrease in the efficiency of parallelization with an increasing number of threads. Figure 3.8 represents the parallel efficiency performance of two libraries MKL and OpenBLAS for parcel-wise and ROI-wise brain encoding across different numbers of threads. The parallelization efficiency is calculated as follows:

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

$$PE = \frac{SU}{N}$$

Where N, SU and PE represents the number of threads, speedup and parallel efficiency respectively. T_R is the execution time with 1 thread, and T_P is the execution time with $N(2, 4, 8, 16, or 32)$ threads. The parallel efficiency 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. Parallel efficiency consistently decreases below 0.4 as the number of threads exceeds 20, for both MKL and OpenBLAS. These diminishing returns in parallel efficiency highlight the need for careful selection of thread count to balance computational resources with performance.



(a) Sub-01 (b) Sub-02

(c) Sub-03 (d) Sub-04

(e) Sub-05 (f) Sub-06

Figure 3.9: MultiOutput ridgeCV training time for 6 subjects using whole brain (MOR) data described in Table 3.1. 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.

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

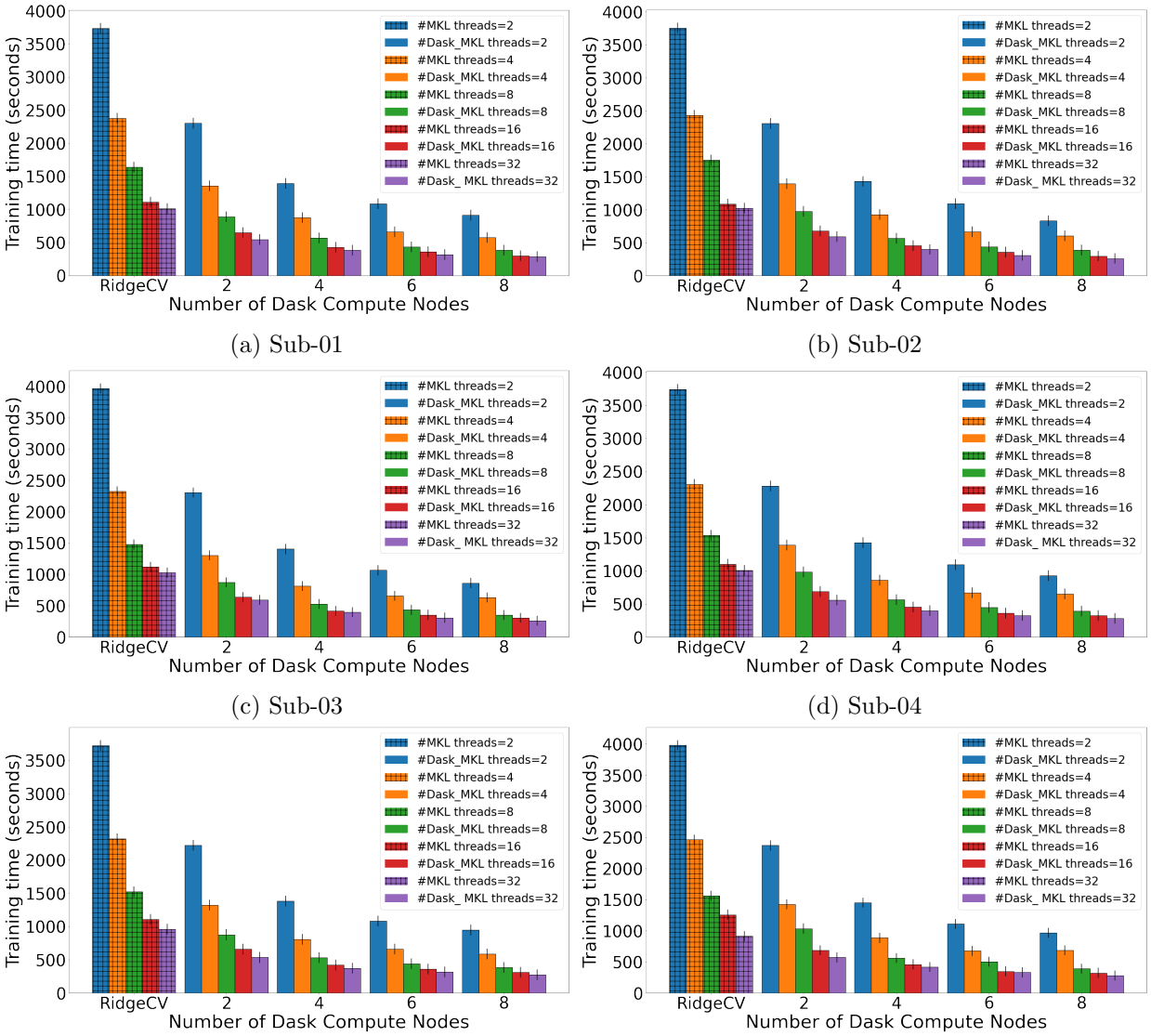
In this next experiment, we implemented scikit-learn's original MultiOutput ridge regression (MOR) within a Dask-based distributed parallelism setting for brain encoding tasks. As this approach results in a slow training process, we created a custom truncated version of whole-brain resolution, called whole-brain (MOR), as seen in Table 3.1. Figure 3.9 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 3.9 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 3.6.

3.4.7 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 3.10 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 3.11). 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.



(a) Sub-01

(b) Sub-02

(c) Sub-03

(d) Sub-04

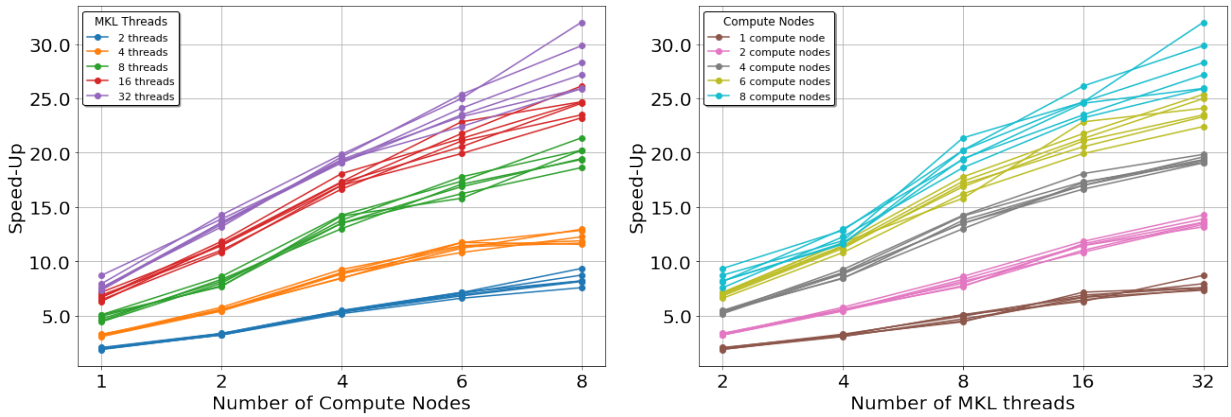
(e) Sub-05

(f) Sub-06

Figure 3.10: B-MOR ridgeCV training time 6 subjects with whole brain (B-MOR) data described in Table 3.1. B-MOR scales across compute nodes and threads, and provides substantial speed-up compared to scikit-learn's multi-threaded implementation (labelled as "RidgeCV").

3.5 Discussion

The most important property of B-MOR is its applicability to scenarios involving large datasets, typically exceeding 100,000 targets. This is crucial, as the division is based on the number of targets. By dividing the problem into subproblems, the size of the subproblems should differ significantly from the size of the data in the main problem. However, there are no conceptual limitations on the number of training samples, except the amount of memory available per worker. In this work, we focus on a truncated dataset version of B-MOR, which uses the same number of targets (whole-brain voxels) but a smaller number of



(a) Speed-Up vs Number of Threads

(b) Speed-Up vs Number of Compute Nodes

Figure 3.11: 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.

training samples, as this is the only approach feasible within our local laboratory cluster. Furthermore, it is worth noting that a larger number of training samples has the potential to further illustrate the efficiency of B-MOR for whole-brain encoding.

MOR, independent of the dataset size, is inefficient due to the significant computational overhead involved. In this work, to make the training time feasible, we applied it to a smaller dataset. Even for this truncated dataset, the training time for Scikit-learn was less than 1s, while the training time for MOR was the order of 1000. As the dataset size increases, the computational overhead and differences in training time would become even more significant. Our results highlight the differences in computational efficiency between B-MOR and MOR in handling whole-brain encoding. B-MOR divides the problem into subproblems, where the number of subproblems (c) corresponds to the number of workers, while MOR handles s independent ridge regressions, with s representing the number of voxels. These approaches demonstrate distinct impacts on scalability. However, due to the constraints on computational resources, it was not feasible to train both B-MOR and MOR on datasets of the same size. To account for this, we compared B-MOR with RidgeCV and MOR with RidgeCV using separate figures and datasets of varying sizes.

In this chapter, our research primarily focuses on CPU-based implementations. However, it is important to note that GPUs provide significant advantages for machine learning tasks by offering access to a higher number of computation nodes, enabling faster parallel processing on large datasets. This leads to substantial speedups over traditional CPU implementations. Despite these benefits, GPUs come with limitations, particularly in terms of memory capacity, which requires careful management to avoid bottlenecks. Among the various available GPU-based approaches, we have explored the Himalaya [37] implementation for efficient ridge regression for brain encoding. Himalaya offers several advantages, such as the ability to estimate linear models on large numbers of targets (e.g., $>$ 100k), compatibility with GPU hardware, and the provision of estimators that integrate seamlessly with scikit-learn's API. Given that GPU memory is often smaller than CPU memory, careful management is essential to prevent running out of memory during computations. For instance, Himalaya

addresses this by implementing several options to limit GPU memory usage, often trading off computational speed for memory efficiency. For instance, some solvers in the Himalaya library perform computations in batches, such as `n_targets_batch` or `n_alphas_batch` to manage the size of intermediate arrays. These considerations are crucial for researchers deciding between CPU and GPU resources, as they must balance the need for speed with the constraints of memory capacity.

Our preliminary experiments (not shown) suggested that memory constraints would prevent us from implementing the large-scale brain encoding experiments performed in this work using Himalaya on a single GPU, specifically one NVIDIA Tesla T4 GPU (with CUDA Version 12.3) equipped with 15,360 MB of memory and 2,560 CUDA cores. These experiments were limited in scope, and further research is needed to clarify the data regimes and hardware constraints where GPU-based approaches achieve a better performance than CPU-based approaches.

While our results are presented within the context of brain encoding, it is crucial to emphasize that the approach we have developed for speeding up and scaling ridge regression is broadly applicable, especially in scenarios involving a large number of target vectors (e.g., $>$ 100k). For instance, in genomics, researchers often use ridge regression to predict gene expression levels across various conditions. In this context, each gene's expression level in different samples or under different conditions can be treated as a separate target variable. Given that the number of genes in an organism can exceed 100,000, managing and processing such a vast number of target vectors requires efficient computational techniques like the one we propose.

It is worth mentioning that distributed Ridge regression can be applied to problems where the matrix X is combined with different feature spaces. For example, in banded ridge regression [37], the matrix X is created based on the combination of activations of 7 layers, where each feature space has a specific value for the normalization hyperparameter (λ). In these cases, the training process can be divided among multiple Ridge regressions, each trained independently in parallel. This approach significantly assists in selecting the feature space among neural network layers.

3.6 Conclusion

In this chapter, 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 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 whithin 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.

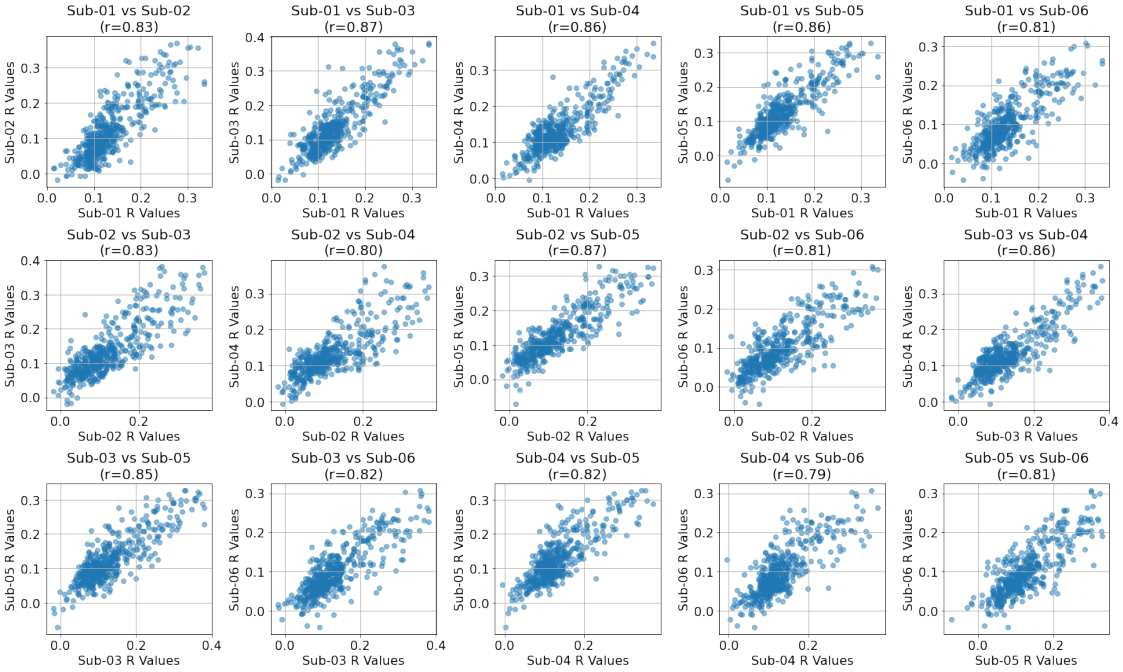
3.7 Availability of code and data

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.

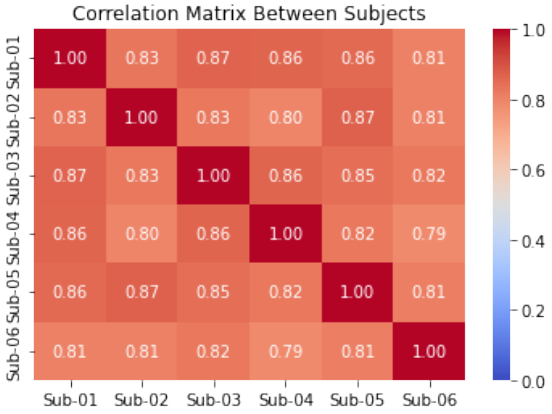
3.8 Appendix

Table 3.4: 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



(a) Scatter plots showing the relationship between r brain encoding values across all brain parcels for a pair of distinct participants. In total 15 pairs of subjects are presented, in the group of 6 subjects.



(b) Correlation between r brain encoding values across all brain parcels for each pair of distinct participants. In total 15 pairs of subjects are presented, in the group of 6 subjects.

Figure 3.12: spatial correlation between brain encoding maps of different subjects

Chapter 4

Training Compute-Optimal Vision Transformers for Brain Encoding

Abstract

The optimal training of a vision transformer for brain encoding depends on three factors: model size, data size, and computational resources. This study investigates these three pillars, focusing on the effects of data scaling, model scaling, and high-performance computing on brain encoding results. Using VideoGPT to extract efficient spatiotemporal features from videos and training a Ridge model to predict brain activity based on these features, we conducted benchmark experiments with varying data sizes (10k, 100k, 1M, 6M) and different model configurations of GPT-2, including hidden layer dimensions, number of layers, and number of attention heads. We also evaluated the effects of training models with 32-bit vs 16-bit floating point representations. Our results demonstrate that increasing the hidden layer dimensions significantly improves brain encoding performance, as evidenced by higher Pearson correlation coefficients across all subjects. In contrast, the number of attention heads does not have a significant effect on the encoding results. Additionally, increasing the number of layers shows some improvement in brain encoding correlations, but the trend is not as consistent as that observed with hidden layer dimensions. The data scaling results show that larger training datasets lead to improved brain encoding performance, with the highest Pearson correlation coefficients observed for the largest dataset size (6M). These findings highlight that the effects of data scaling are more significant compared to model scaling in enhancing brain encoding performance. Furthermore, we explored the impact of floating-point precision by comparing 32-bit and 16-bit representations. Training with 16-bit precision yielded the same brain encoding accuracy as 32-bit, while reducing training time by 1.17 times, demonstrating its efficiency for high-performance computing tasks.

4.1 Introduction

Brain encoding aims to predict neural responses to stimuli by leveraging computational models. Traditionally, Convolutional Neural Networks (CNNs) such as AlexNet[16], VGG16 [67], and ResNets [22] have been employed to mimic brain activity patterns, extracting semantic information from visual stimuli [33, 34, 26, 7, 58, 65]. However, deeper CNN architectures have not consistently replicated brain-like responses across all regions and recent advancements highlight the potential of transformers [53, 36, 19]. Transformer-based architecture [73] offer several distinct advantages for brain encoding, including superior performance in capturing spatial and temporal features compared to CNNs, RNNs and LSTMs. The attention mechanisms in transformers enhance the selective integration of visual inputs, which is crucial for understanding neural activity. Furthermore, generative self-supervised models demonstrate predictive capabilities comparable to supervised models. Finally, multi-modal architectures, such as VisualBERT [39] and CLIP [59], effectively leverage semantic correlations across different modalities, offering robust performance in visio-linguistic tasks. Inspired by these advancements, neuroscientists have developed transformer-based brain encoding models that significantly improve the accuracy of fMRI encoding across the entire brain [55, 53, 36, 19, 51].

In the pursuit of developing compute-optimal models, the interplay among computational resources, model dimensions, and dataset sizes holds paramount importance [24, 3, 2, 82]. Scaling laws serve as fundamental frameworks elucidating the dynamics between a model's efficacy and these three pillars. For instance, the authors in [24] delve into the optimal configuration of model size and training data for transformer language models based on computing resources budget. Their findings highlight that large-scale models often undergo undertraining due to an emphasis on scaling model sizes while keeping the training dataset size constant. By scaling both model size and training data size, the proposed model, Chinchilla, surpasses larger counterparts in performance metrics while utilizing fewer computational resources. Similarly, in [3], the authors advance scaling laws for vision transformers to propose compute-optimal model architectures regarding width and depth instead of solely focusing on the number of parameters.

Recent research has extended scaling laws to brain encoding tasks, mirroring discussions found in studies of end-to-end language and vision models (trained from scratch on stimuli) alongside their brain encoding correlates. These studies aim to address fundamental questions such as: (1) How does the sample size to train transformers impact on brain encoding prediction accuracy? (2) How does brain encoding prediction accuracy vary with the parameter size (model size) of transformer? For example, a recent study [5] explored the effectiveness of larger language models, such as those from the OPT [83] and LLaMA [69] families (30B parameters), in predicting brain responses compared to traditional model GPT-2 (125M parameters). The results indicated a logarithmic relationship between brain prediction performance and model size. Similarly, scaling laws were observed with increasing training dataset, demonstrating that brain encoding prediction accuracy increases on a logarithmic scale with both the size of the training samples and the parameter size of the language models employed. The study in [43] explores the construction of a high-performance vision encoding model, assessing how changes in the sample size of the fMRI training set and

the parameter size of vision models affect prediction accuracy. Various vision models with parameter sizes ranging from 86M to 4.3B were employed to extract features from stimuli presented to the subjects. Results demonstrate that increasing the training sample size and the parameter size of vision models enhances prediction accuracy of brain encoding according to the scaling law.

In this study, we investigate the optimization of vision transformers for brain encoding using the Shinobi dataset [20], which includes approximately 10 hours of fMRI data collected as subjects engaged with the Shinobi video game. This dataset provides a diverse range of cognitive engagements across different game levels, offering a rich foundation for brain encoding experiments. To extract efficient spatiotemporal features from this dataset, we train an end-to-end VideoGPT [80] model with tens of millions of parameters. VideoGPT is able to capture complex spatial and temporal patterns in video data, which are essential for understanding brain responses to visual stimuli. The extracted features from VideoGPT are then paired with corresponding fMRI data, and we train a Ridge regression [23] model on these pairs to predict brain activity for unseen frames, enabling the decoding of neural responses to new visual stimuli. Our investigation is structured to address several key aspects: 1) exploring the effects of varying dataset sizes (10k, 100k, 1M, 6M) on brain encoding performance, 2) examining different model configurations such as hidden layer dimensions, number of layers, and attention heads, 3) assessing the impact of mixed precision on model training time and brain encoding accuracy.

4.2 Materials and Methods

4.2.1 fMRI datasets

The fMRI Shinobi videogame dataset was collected in the context of the Courtois Neuromod Project. This game has been selected to effectively engage subjects with multiple cognitive components simultaneously, such as perception of the environment, strategic planning, decision making and taking action. In the Shinobi dataset, about 10 hours of fMRI data was recorded while the subjects play the Shinobi video game. In each run, subjects played 3 levels in cycles and in the same order each time. These levels were: Level-1) corresponded to round 1 of the original game which included one mini-boss and one boss fight. Level-4) corresponded to the beginning of round 4 of the original game which included no mini-boss or boss fight. Level-5) corresponded to the beginning of round 5 of the original game, which included one mini-boss fight and no boss fight. Subject moved to the next level in two cases: they successfully completed a level, or lost three lives. The duration of each run is a minimum of ten minutes. A run was completed as soon as its duration exceeded 10 minutes and the participant completed a level, as was just defined. The duration of each run was thus variable, depending on the individual gameplay of the participant. As there are fixed order in the levels, Level-1 was repeated more frequently than Level-4 and Level-5. For more information on this dataset, visit the CNeuroMod dataset documentation page(CNeuroMod web page).

4.2.1.1 Participants

The Shinobi dataset includes fMRI time series collected on four participants in good general health, 2 women (sub-04, and sub-06) and 2 men (sub-01, sub-02). 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.

4.2.1.2 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 [64, 79] developed at the University of Minnesota, as part of the Human Connectome (HCP) Project [72]. 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.

4.2.1.3 Preprocessing

All fMRI data were preprocessed using the fMRIprep pipeline version 20.2.3 [17]. 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` tool[1] (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).

4.2.1.4 Brain parcellation

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) [70]. MIST provides a hierarchical decomposition of functional brain networks in nine levels, and we used here the largest available resolution ($MIST_{AtOM}$). For each subject, the validation data is the data from the session 004, the test data is from session 005 and the training is the rest of the sessions. Details of fMRI data representation using $MIST_{AtOM}$ parcellation is presented in Table 4.1.

[1] https://github.com/simexp/load_confounds

Table 4.1: Shinobi fMRI data representation using $MIST_{AtOM}$ parcellation

fMRI data representing	parcel-wise train	parcel-wise test
shape of sub-01 data	(15313, 1095)	(1513, 1095)
shape of sub-02 data	(17996, 1095)	(829, 1095)
shape of sub-04 data	(14046, 1095)	(1867, 1095)
shape of sub-06 data	(13966, 1095)	(1479, 1095)
Size	$\sim 151M$	$\sim 12M$

4.2.2 Generating Video Data

The generation of video data for the CNeuromod Shinobi gameplay recordings involved transforming *keypress* logs into visual frames that represent the gameplay experience. This process is pivotal for creating a dataset that captures not only the player's actions but also the corresponding visual responses from the game environment. The method employed integrates game emulation with keypress playback to achieve a structured dataset suitable for subsequent analysis.

The process begins with the setup of the *emulator* using the *retro* library, which facilitates the playback of Sega Genesis games. Specifically, the emulator is configured to run *"ShinobiIIIReturnOfTheNinjaMaster-Genesis"* leveraging custom integrations to ensure compatibility with the CNeuromod dataset. This step is critical as it establishes the foundation for accurately replaying gameplay sessions based on recorded inputs. Once the emulator is set up, the keypress logs, stored in *.bk2* files, are loaded into the system. Each log file contains a sequential record of player inputs, and the emulator is reset to its initial state, ensuring it can accurately reflect the gameplay as originally experienced. This resetting process allows for the emulator to start from the exact point in the game where the recorded session began, thereby preserving the integrity of the gameplay dynamics.

As the emulator replays each log, it simulates the exact sequence of actions taken by the player, advancing frame by frame. The gameplay environment's visual output is captured at each step, resulting in a series of images that represent the game state over time. To ensure consistency, the captured frames are resized to a standard resolution of 64×64 (or 128×128) pixels, maintaining their visual quality while fitting within the requirements of the dataset.

4.2.3 Brain encoding

In this work, an end-to-end VideoGPT model was trained on the Shinobi dataset, which consists of more than 6 million frames totaling 309GB and includes 10-hours video recording of Shinobi gameplay across 4 subjects. After training the VideoGPT model, we used the trained model to extract spatio-temporal features from stimuli for brain encoding.

We specifically used the "attn_stack.attn_nets.4.post_fc_dp" layer to represent the activations of VideoGPT. This layer was selected because the model includes 8 blocks, and we chose the fourth block to extract mid-level features, compare to the high-level features in later layers or low-level features in earlier layers. Additionally, we conducted a benchmark analysis among the layers in the 4th block and found that the 11attn_stack.attn_nets.4.post_fc_dp" layer had

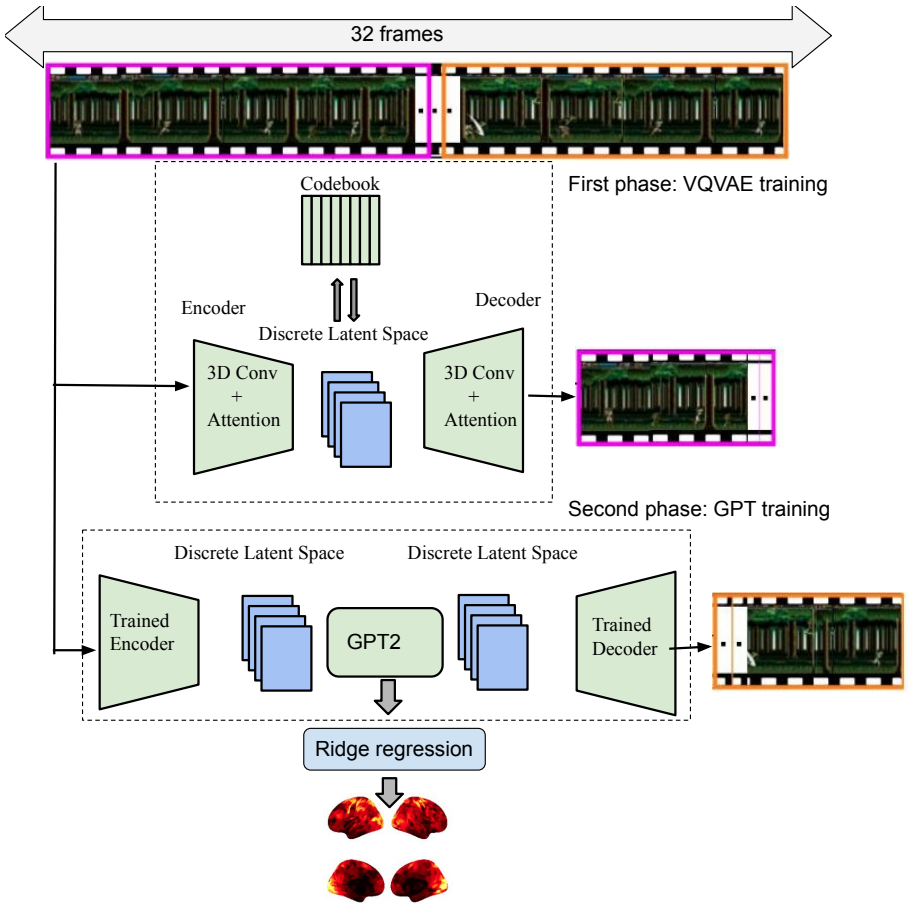


Figure 4.1: Two main steps of brain encoding: Extracting features from movie frames using GPT-2 model and predicting brain response using ridge regression

lower activation size compared to other layers and provided a higher correlation for brain encoding (see Figure 4.7 and Table 4.5 in 4.6).
After extracting features from VideoGPT, we applied a 3TR (4.5s) delay. These final features were then used to train ridge model using ridge regression on pairs of extracted features, brain activity. Figure 4.1 illustrates the two main steps of brain encoding: extracting intricate patterns and temporal dependencies in the video sequences and predicting brain responses using ridge regression.

4.2.3.1 VideoGPT model

The architecture of the VideoGPT model is an adaptation of VQ-VAE [71] and GPT-2 [60] architectures. In the first phase of VideoGPT, we trained VQ-VAE to reconstruct 16 sequences of frames. In the second phase, we train GPT to predict the next 16 sequence of frames based on the previous 16 sequences of frames. In the second phased, we use VQ-VAE as a pretrained network to represent sequences of frames with the codebook as the input for GPT.
Training VQ-VAE The VQ-VAE [71] architecture further extends Autoencoders [25] using discrete latent variables, inspired by vector quantization (VQ). In this approach, the posterior

and prior distributions are considered categorical (discrete). The output of the encoder is compared to all vectors in a (learned) codebook, then the closest codebook vector in the Euclidean distance is selected as input to the decoder. The parameter set of VQ-VAE includes parameters of the encoder, decoder, and the embedding space e.
In the first phased of Video GPT, a set of discrete latent codes will be trained for the Shinobi dataset through the VQ-VAE, in effect downsampling windows of video frames (sequence length of 16) into a discrete space-time codebook. The encoder of the VQ-VAE will include a series of 3D convolutions followed by attention residual blocks (a replacement for standard residual blocks) to better capture complex spatiotemporal dependencies within video data. In this architecture, each attention residual block includes Convolution, LayerNorm, position embedding and axial attention layers. The position embedding is shared between all axial attention layers in the encoder and decoder. The architecture of the decoder starts with attention residual blocks which are then followed by a series of 3D transposed convolution (reverse of encoder) to upsample the video frames across space-time.
Multi-Head Attention: Multi-Head Attention works by projecting the input tensor X into multiple subspaces using separate attention heads, each computing queries Q_i, keys K_i, and values V_i with distinct linear transformations. For each head i, the attention is calculated as:

$$\text{Attention}_i(X) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i$$

where d_k is the dimensionality of the keys. The outputs from all attention heads are concatenated and projected to obtain the final multi-head attention result:

$$\text{MultiHeadAttention}(X) = \text{Concat}(\text{Attention}_1(X), \ldots, \text{Attention}_H(X))W^O$$

This multi-head attention allows the model to process various aspects of the shinobi video data simultaneously, improving its capacity to learn complex patterns.
Axial Attention: Axial Attentionis then applied to efficiently manage the high-dimensional video data. Instead of processing the entire tensor at once, axial attention operates along different dimensions separately—width, height, and temporal—allowing the model to focus on specific aspects of the data. Attention is computed separately along each dimension:
- Width Attention:

$$\text{Attention}_{\text{width}}(X) = \text{softmax}\left(\frac{Q_{\text{width}} K_{\text{width}}^T}{\sqrt{d_k}}\right) V_{\text{width}}$$

- Height Attention:

$$\text{Attention}_{\text{height}}(X) = \text{softmax}\left(\frac{Q_{\text{height}} K_{\text{height}}^T}{\sqrt{d_k}}\right) V_{\text{height}}$$

- Temporal Attention:

$$\text{Attention}_{\text{temporal}}(X) = \text{softmax}\left(\frac{Q_{\text{temporal}} K_{\text{temporal}}^T}{\sqrt{d_k}}\right) V_{\text{temporal}}$$

The combined axial attention is represented as:

$$\text{Attention}_{\text{combined}}(X) = \text{Attention}_{\text{width}}(X) + \text{Attention}_{\text{height}}(X) + \text{Attention}_{\text{temporal}}(X)$$

This approach allows the model to process different dimensions of the video independently, enhancing its ability to manage complex spatiotemporal structures. Finally, after the attention mechanism, the input tensor is passed through a series of convolutional layers for further feature extraction:

$$X' = \text{Conv3D}(X)$$

Axial attention is applied through an axial block:

$$X'' = \text{AxialBlock}(X')$$

The output of the axial block is then combined with the original input through a residual connection:

$$Y = X + X''$$

This residual connection ensures the retention of the original input features while incorporating the newly learned ones, facilitating smoother training and improving the model's overall ability to capture both spatial and temporal patterns in video data.
Loss Function: In VideoGPT, the VQ-VAE is trained using the following loss function:

$$L = \|x - D(e)\|_2^2 + \|\,\text{sg}[E(x)] - e\|_2^2 + \beta\|\,\text{sg}[e] - E(x)\|_2^2$$

In the loss function, x represents the input frames, and e denotes the quantized representation obtained from the codebook. The function $D(e)$ refers to the decoder's output when given the quantized representation e, and $E(x)$ represents the encoder output for the input x. The term $\|\cdot\|_2^2$ indicates the squared L_2-norm, which is used to measure the difference between two vectors. The operator $\text{sg}[\cdot]$ refers to a stop-gradient operation, which prevents gradients from flowing through the specified term during backpropagation. Finally, β is a hyperparameter that weights the contribution of the commitment loss in the total loss function. These notations collectively define the components of the loss function and their roles in training the VQ-VAE model. Table 4.2 presents the notations ans symbols.

Table 4.2: VQ-VAE notations

x	input frames
e	quantized representation
D(x)	decoder
E(x)	encoder

The loss function includes reconstruction loss L_{recon}, a codebook loss $L_{codebook}$, and a commitment loss L_{commit}. The reconstruction loss controls the VQ-VAE training process to learn efficient representations of frames with minimizing the difference between original and reconstructed frame features. The codebook loss ensures that the codebook embeddings and their

corresponding encoder outputs are closely matched based on nearest neighbors lookup. In the VQ-VAE model, the encoder outputs may oscillate between different code vectors for the same input sample. To tackle this problem, the commitment loss is employed to encourage the encoder outputs to commit to a particular code vector. The commitment loss is weighted by a hyperparameter β to regularise the VQ-VAE training process with penalizes the encoder for switching between code vectors. In the lost function, by stopping the gradients for e and pre-trained weights $E(x)$, their values remain fixed. In other words, stop gradient leads to preserving the representations they have already learned.

Optimizer: The model uses the Adam optimizer (*betas=(0.9, 0.999)*) for training. The model starts with a learning rate of 3e-4, and then the *CosineAnnealingLR* scheduler [42] gradually reduces this learning rate over time according to a cosine function. CosineAnnealingLR scheduler, defined as:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{t}{T_{\max}}\pi\right)\right) \tag{4.1}$$

where $\eta_{\max}$ is the initial learning rate, and $T_{\max}$ is the maximum number of training steps. This scheduling allows for a smooth reduction in learning rate, fostering a balance between fast convergence in the initial training phase and fine-tuning as the model nears convergence. The codebook in the VQ-VAE model is updated using an Exponential Moving Average (EMA) [71] method to maintain stable and meaningful learned representations. Specifically, the counts N and the running average of the embeddings z_avg are updated as follows:

$$\mathrm{N} \leftarrow 0.99 \times \mathrm{N} + 0.01 \times \mathrm{n\_total} \tag{4.2}$$

$$\mathrm{z\_avg} \leftarrow 0.99 \times \mathrm{z\_avg} + 0.01 \times \mathrm{encode\_sum} \tag{4.3}$$

where n_total is the sum of one-hot encoded vectors indicating the frequency of each code being selected, and encode_sum is the sum of the input vectors corresponding to each code. The embeddings are then normalized based on their frequency:

$$\mathrm{weights} = \frac{(\mathrm{N} + 1e-7)}{(\sum \mathrm{N}) + \mathrm{n\_codes} \times 1e-7} \times \sum \mathrm{N} \tag{4.4}$$

$$\mathrm{embeddings} \leftarrow \frac{\mathrm{z\_avg}}{\mathrm{weights}} \tag{4.5}$$

Unused embeddings (where $\mathrm{N} < 1$) are reinitialized with randomly sampled vectors from the data. This EMA-based update mechanism ensures that the codebook embeddings adapt to new data while remaining stable and representative of the underlying input distribution.

Hyperparameters: As Table 4.3 shows, the main hyperparameters of the VQ-VAE model include the embedding dimension (*embedding_dim*), the number of codes in the codebook (*n_codes*), the number of hidden layers (*n_hiddens*), the number of residual layers (*n_res_layers*), and the downsampling factors for the encoder and upsampling factors for the decoder (*downsample*, *upsample*).

Training autoregressive GPT-2 In time series data such as videos, autoregressive models train to predict a time step value (frame in the video) using previous time step values. The VideoGPT autoregressive architecture includes a stack of transformer encoder layers

to generate videos (predict next frames) from a latent space. In VideoGPT, through the VQ-VAE (first phase of videoGPT training process), we learn the latent codes, and in the next step, we leverage GPT-2 to model the prior over the latent space. The input to GPT-2 is a sequence of discrete latent codes, produced by the VQ-VAE encoder. The GPT-2 model is trained to generate new sequences of latent codes, which are then passed through the VQ-VAE decoder to produce new video frames.
By employing a self-supervised and autoregressive approach, GPT effectively learns from Shinobi video data, generating meaningful representations of the spatio-temporal dynamics present in the video. This capability enables the model to capture not only the visual information of individual frames but also the temporal transitions that define the sequence. As GPT learns to predict future frames based on previously observed content, it produces contextualized embeddings that reflect the temporal dependencies inherent in video sequences. This rich representation allows the model to discern essential characteristics of movement, object interaction, and scene evolution, making it particularly advantageous for brain encoding, where understanding the flow of time and space of stimuli is crucial.
Loss Function: The GPT loss function used in the VideoGPT is Cross-Entropy Loss. This loss function is used for classification tasks and is well-suited for training models like GPT, which predict the next token (or codebook in the context of VQ-VAE) in a sequence. The loss function measures the difference between the predicted probability distribution and the actual distribution (which is typically a one-hot encoded vector for classification). The formula for the cross-entropy loss is:

$$L_{\text{cross-entropy}} = -\sum_{i=1}^{C} y_i \log(p_i) \tag{4.6}$$

where C represents the number of possible latent codes. For each latent code i, y_i is a one-hot indicator (0 or 1) indicating whether latent code i is the correct representation for the current input. Additionally, p_i denotes the predicted probability for latent code i, which is obtained by applying the softmax function to the logits.
Optimizer: Optimization is performed using the Adam optimizer (*betas=(0.9, 0.999)*) with a learning rate of 3e-4. The *CosineAnnealingLR* scheduler adjusts the learning rate according to a cosine function over the training steps, starting at a higher value to enable faster convergence initially, and then decreasing gradually to promote more refined learning as convergence is approached.
Hyperparameters: As Table 4.3 shows, the GPT-2 model employs several hyperparameters, including a hidden dimension (*hidden_dim*) of 576, 4 attention heads (*heads*), 8 transformer layers (*layers*), and dropout rates of 0.2 for the attention mechanism (*attn_dropout*) and 0.3 for other layers (*dropout*).

Table 4.3: Model Sizes

(a) VQVAE Model Size

Name	Type	Parameters
encoder	encoder	18.7 M
decoder	decoder	17.1 M
pre_vq_conv	SamePadConv3d	30.8 K
post_vq_conv	SamePadConv3d	31.0 K
codebook	Codebook	0
Trainable params		35.8 M
Non-trainable params		0
Total params		35.8 M
Total model params size (MB)		143.306

(b) GPT-2 Model Size

Name	Type	Parameters
vqvae	VQ-VAE	35.8 M
resnet	ResidualBlock	2.4 M
fc_in	Linear	73.7 K
attn_stack	AttentionStack	37.2 M
fc_out	Linear	589 K
Trainable params		40.3 M
Non-trainable params		35.8 M
Total params		76.2 M
Total model params size (MB)		304.606

(c) Hyperparameters for VQVAE and GPT-2 components

Parameter	**VQVAE**	**GPT-2**
embedding_dim	256	-
n_codes	2048	-
n_hiddens	240	-
n_res_layers	4	-
downsample	(4, 4, 4)	-
hidden_dim	-	576
heads	2	4
layers	-	8
dropout	-	0.2
attn_dropout	-	0.3

4.2.3.2 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. 3.1 based on the training set only. 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. 3.1), based on the grid:

$$\lambda \in \{0.1, 1, 100\}\,.$$

4.2.3.3 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.

4.2.4 Scaling up VideoGPT model

4.2.4.1 Scaling up GPT in terms of Dataset Size

The GPT model was trained on datasets of varying sizes: 10k, 100k, 1M, and 6M samples. These datasets were derived from the entire available dataset collected from four subjects who played the Shinobi game while the videos were recorded. After training the GPT model on these different dataset sizes, we extracted activations from the model and compared the brain encoding results. In all scenarios of varying dataset size, the hyperparameters related to model size were fixed according to Table 4.3.

4.2.4.2 Scaling up GPT in terms of Model Size

In this study, the scaling of the GPT model size was explored by varying the following hyperparameters:

- **Number of hidden layers**: 1, 2, 4, 8
- **Dimension of hidden layers**: 6, 15, 30, 63, 126, 255, 576, 1024 (note: the hidden dimension must be a multiple of 3 and greater than 3)
- **Number of heads in multi-head attention mechanism**: 1, 2, 4, 8

In each scenario, the other model hyperparameters, as detailed in Table 4.3, were kept fixed. Furthermore, the GPT model was trained on a dataset of 6 million samples across all scenarios. After training with these configurations, activations were extracted from the GPT model, and the brain encoding results were compared using actual fMRI data. Specifically, we used the "attn_stack.attn_nets.4.post_fc_dp" layer to represent the GPT activations for experiments involving the hidden dimension and the number of heads, but only for the case of the number of layers. Table 4.4 shows the selected layer for brain encoding:

Table 4.4: Selected GPT layers for brain encoding based on the number of layers

Number of Layers	Selected GPT Layer for Brain Encoding
1	attn_stack.attn_nets.0.post_fc_dp
2	attn_stack.attn_nets.1.post_fc_dp
4	attn_stack.attn_nets.3.post_fc_dp
8	attn_stack.attn_nets.7.post_fc_dp

4.2.4.3 High Performance computing during GPT training process

In [45], the authors proposed an approach for training deep neural networks based on half-precision floating-point numbers, without losing model accuracy or modifying the hyperparameters. In this work, the results show that the mixed-precision approach has two main advantages: 1) approximately halves the memory requirements, 2) accelerating arithmetic on GPU. In the proposed approach weights, activations, and gradients are stored in half-precision format. Updating the parameters contains two key steps: Firstly, a single-precision

copy of weights is maintained to accumulate the gradients after backpropagation. Secondly, scaling the loss function to preserve gradient values with small magnitudes. The mixed precision approach works across a wide variety of modern large scale Deep Learning model architectures, trained on large datasets.
Two commonly used 16-bit formats are float 16 and bfloat16, each with distinct advantages. Float 16, adhering to the IEEE 754 standard, uses 1 bit for the sign, 5 bits for the exponent, and 10 bits for the mantissa. Its compact nature reduces memory usage and accelerates computations but limits precision and dynamic range. It is widely used in scenarios where speed and memory efficiency are critical, though it struggles with representing very large or small numbers. In contrast, bfloat16 allocates 1 bit for the sign, 8 bits for the exponent, and 7 bits for the mantissa, offering the same dynamic range as float 32 but with reduced precision. This format is optimized for modern hardware, including Google's TPUs and NVIDIA GPUs such as A100, making it a popular choice in machine learning.
Recently, the FP8 format has emerged as a breakthrough for further improving memory and computational efficiency. With NVIDIA's H100 (Hopper architecture), two new FP8 formats are introduced:

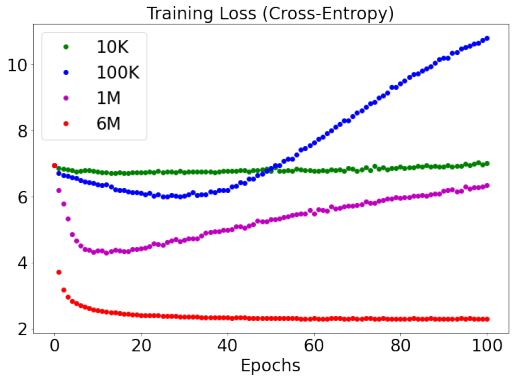
- E5M2 (5 bits for the exponent, 2 bits for the mantissa).
- E4M3 (4 bits for the exponent, 3 bits for the mantissa).

These formats strike a balance between performance and precision, allowing for even faster computations and more efficient memory usage, making them ideal for deep learning workloads. Hopper's dynamic precision capabilities ensure seamless switching between FP8, FP16, and FP32, adapting to the specific needs of the task at hand and maximizing overall efficiency.
In this work, we focus on comparing the training of GPT-2 using 32-bit and standard 16-bit precision format, and we discuss the impacts of these precision levels on brain encoding.

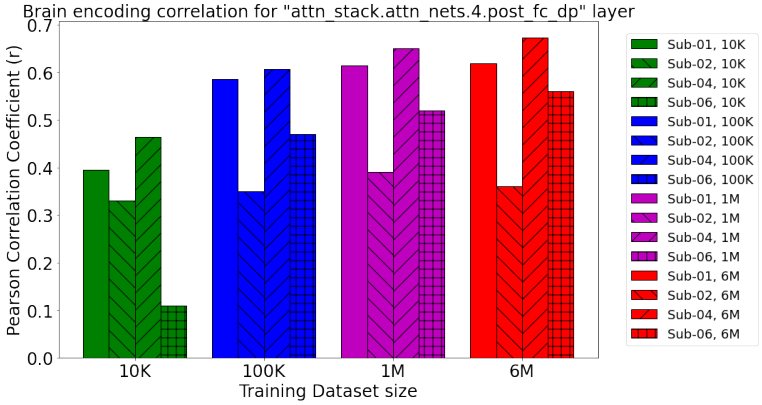
4.3 Results

4.3.1 Scaling up the dataset size for training VideoGPT resulted in a significant improvement in brain encoding performance

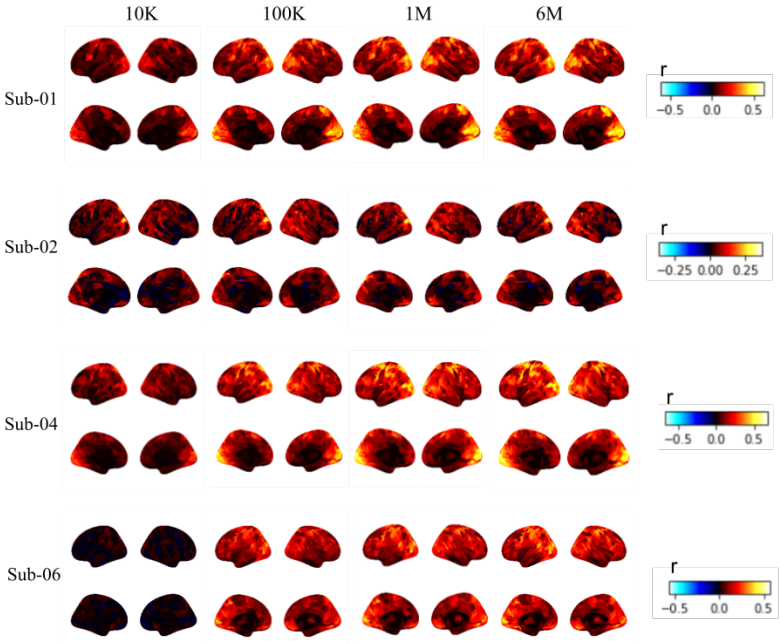
In this work, we explored the impact of varying training dataset sizes of GPT model in brain encoding tasks. Specifically, we trained the GPT model on datasets of sizes 10K, 100K, 1M, and 6M samples, derived from video recordings of subjects playing the Shinobi game.
The training loss decreases more rapidly and stabilizes at a lower value with larger datasets, as shown in Figure 4.2a. Specifically, GPT trained on the 6M dataset (red curve) shows a rapid decline in cross-entropy loss, reaching a stable point around epoch 40. In contrast, the models trained on smaller datasets, such as the 10K (green curve) and 100K (blue curve) datasets, exhibit slower convergence and higher final loss values, with the 100K dataset even experiencing an upward trend after epoch 60, indicating overfitting. Smaller datasets lack sufficient diversity and information for the model to generalize well. As training progresses, the model starts to overfit to the limited data, causing the loss to increase over time. In



(a) GPT training loss



(b) Brain encoding correlations across subjects



(c) Brain maps across subjects

Figure 4.2: GPT training across different training dataset sizes

contrast, larger datasets allowing the model to learn more effectively and maintain a stable or decreasing loss throughout training. The model trained on the 1M dataset (purple curve) falls in between these extremes, achieving better performance than the smaller datasets but not reaching the low loss values of the 6M dataset. This trend demonstrates that increasing the dataset size leads to better GPT convergence, with the larger datasets enabling the model to generalize better and achieve a lower cross-entropy loss, thus improving overall training efficiency and effectiveness.

Figure 4.2b demonstrates the impact of training dataset size on brain encoding correlations, as indicated by the Pearson correlation coefficient (r) between actual and predicted brain activity across various subjects. The GPT model trained on the 6M dataset (red bars) generally achieves the highest correlation values across all subjects, highlighting the enhanced brain encoding predictive accuracy with increased GPT training data.

The brain encoding results of Subject 01 show a noticeable improvement in correlation from approximately 0.4 with the 10K dataset to nearly 0.6 with the 6M dataset, representing an increase by a factor of about 1.5. Similarly, Subject 06 exhibits a significant enhancement, with correlations rising from around 0.15 with the 10K dataset to almost 0.55 with the 6M dataset, leading to an improvement by a factor of approximately 3.7. On the other hand, Subject 02 shows a modest improvement, where the correlation increases from around 0.35 with the 10K dataset to about 0.4 with the 6M dataset, yielding only a 1.14 times improvement.

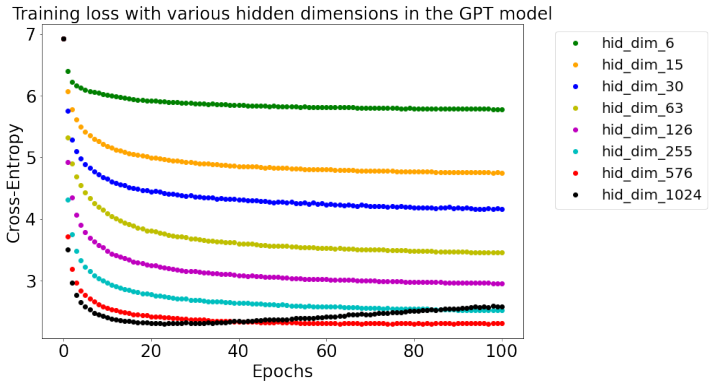
This pattern suggests that while larger training datasets generally lead to more accurate brain encoding, the extent of improvement varies across subjects. For example, Sub-06 experienced significant gains, while Sub-02 saw less substantial improvements, indicating that the effectiveness of dataset scaling may differ depending on individual subject characteristics. Figure 4.2c illustrates brain maps across four subjects.

4.3.2 Scaling up the GPT model size in terms of hidden dimensions resulted in a significant improvement in brain encoding performance

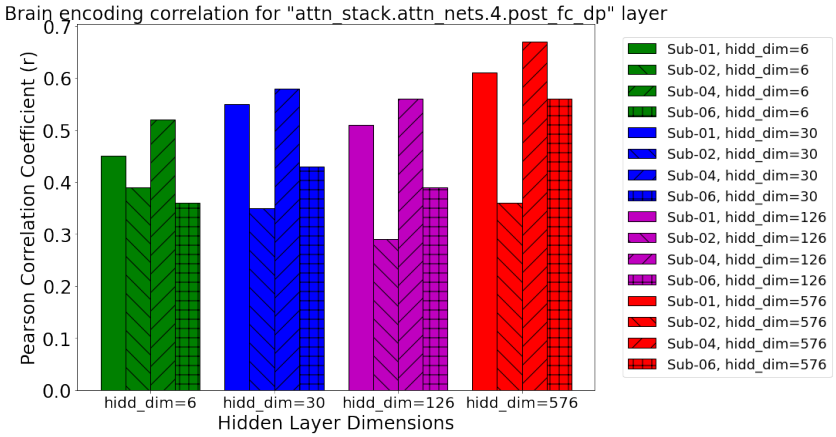
In this section, we investigated the impact of increasing the hidden dimensions in the GPT model on brain encoding performance. The VideoGPT model was trained with various hidden dimensions, while other hyperparameters were kept constant, as outlined in Table 4.3.

Figure 4.3a displays the training loss curves across different hidden dimensions. As the hidden dimensions increase from 6 to 576, the training loss decreases more rapidly and stabilizes at lower cross-entropy values. This indicates that models with larger hidden dimensions have greater capacity, leading to better convergence during training. However, after epoch 30, the loss for the model with 1024 hidden dimensions begins to increase slightly, suggesting overfitting. Then, the results hint at a slight decline in performance beyond a certain threshold, as evidenced by the small drop in correlation for some subjects at the highest hidden dimension tested (1024).

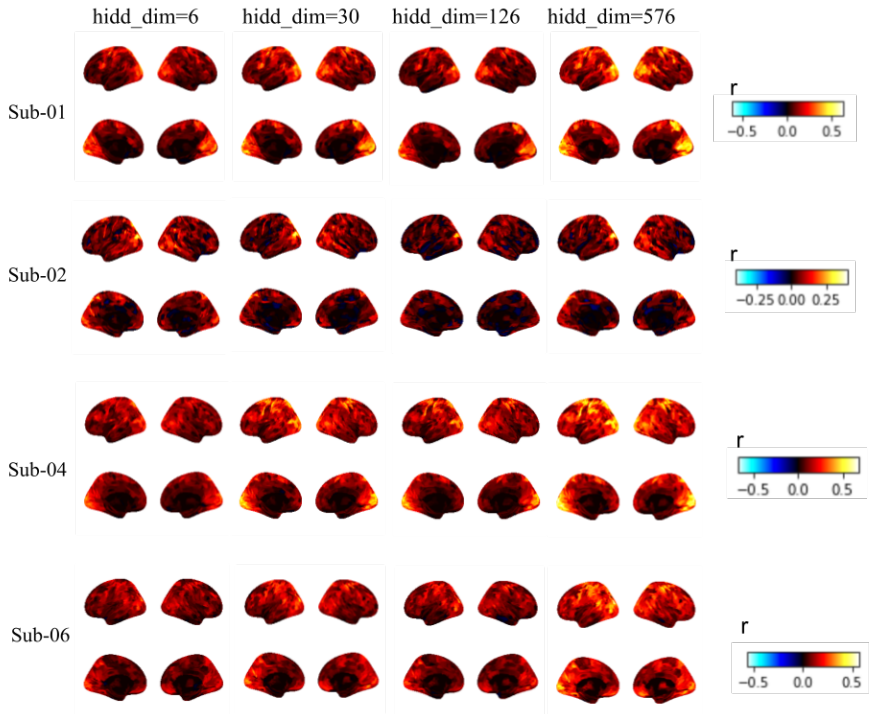
Figure 4.3b shows the Pearson correlation coefficients between the model predictions and actual brain activity across different subjects. The results demonstrate that models with



(a) GPT training loss



(b) Brain encoding correlation coefficients across subjects



(c) Brain maps across subjects

Figure 4.3: GPT training across different hidden dimensions

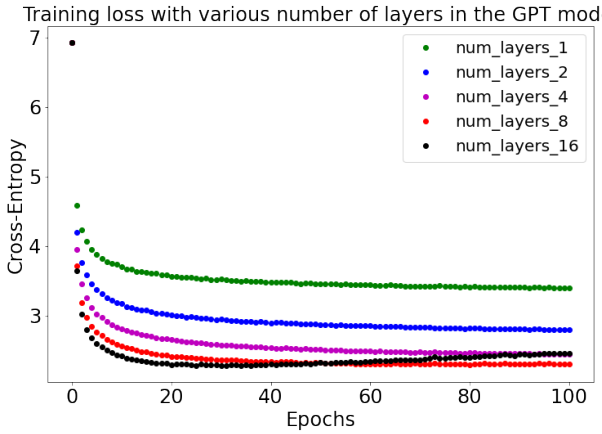
larger hidden dimensions generally achieve higher brain encoding correlations. Notably, at a hidden dimension of 576, the highest correlations are observed, particularly for Subjects 01, 04, and 06, where the improvements are significant compared to the hidden dimension of 126.
However, the pattern is not entirely consistent across all subjects and hidden dimensions. For instance, at a hidden dimension of 30, Subject 02 exhibits a decrease in brain encoding performance compared to its performance at a hidden dimension of 6, while other subjects show an increase. Furthermore, at a hidden dimension of 126, all subjects experience a decrease in correlation compared to the results at 30. These fluctuations highlight the nuanced relationship between model complexity and brain encoding accuracy. Figure 4.3c presents brain activation maps for four subjects across different hidden dimensions.

4.3.3 Effect of increasing layers of GPT on brain encoding performance is limited

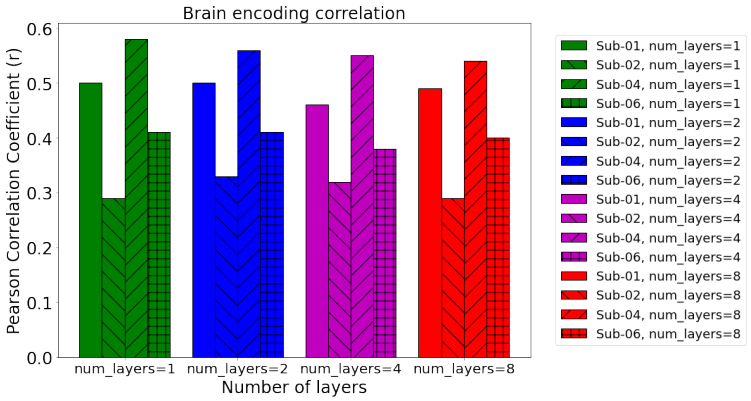
In this section, we explored the effect of varying the number of layers in the GPT model on brain encoding performance. The GPT model was trained with 1, 2, 4, 8 and 16 layers, while all other hyperparameters were kept constant, as detailed in Table 4.3.
Figure 4.4a illustrates the training loss curves across different layer configurations. As the number of layers increases from 1 to 8, we observe a notable decrease in the training loss, particularly during the early epochs. This indicates that deeper models have a greater capacity for capturing complex patterns in the data, leading to more effective convergence. However, beyond 4 layers, the rate of improvement in the training loss begins to plateau, suggesting that the benefits of additional layers reduce as the model becomes deeper. Additionally, the model with 16 layers shows signs of overfitting after epoch 30, as indicated by a minor increase in the training loss, implying that excessively deep models may lead to overfitting on the training data.
Figure 4.4b presents the Pearson correlation coefficients between the model predictions and actual brain activity across different subjects (sub-01, sub-02, sub-04, and sub-06) for varying numbers of layers (1, 2, 4, and 8). The corresponding GPT layers selected for brain encoding in each scenario are listed in Table 4.4. The results show that models with 1 layer generally perform well, with sub-01, sub-04 and sub-06, achieving the highest correlation. When increasing the number of layers to 2, there is an observable improvement in performance across Sub-02. The correlation coefficients do not show consistent improvement with increases in number of layers. This suggests that there is a point where increasing of GPT model layer depth does not yield better performance of brain encoding and may even result in diminishing in correlations. Figure 4.4c illustrate the brain maps across six subjects.

4.3.4 Increasing the number of attention heads does not translate to better brain encoding performance

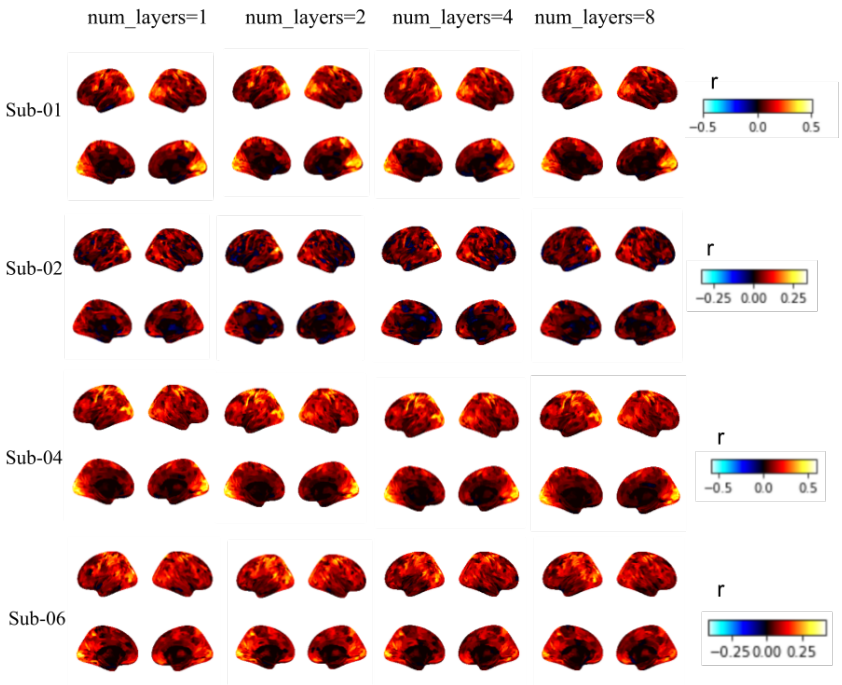
In this section, we investigated the effect of changing the number of attention heads in the GPT model on brain encoding performance. The model was trained with varying numbers of attention heads (1, 4, and 8), while other hyperparameters were kept constant in Table



(a) GPT training loss

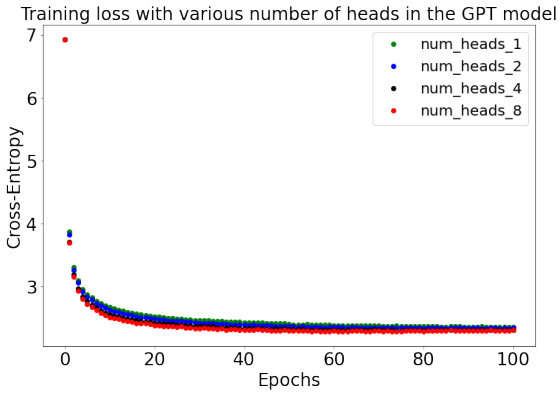


(b) Brain encoding correlation coefficients across subjects

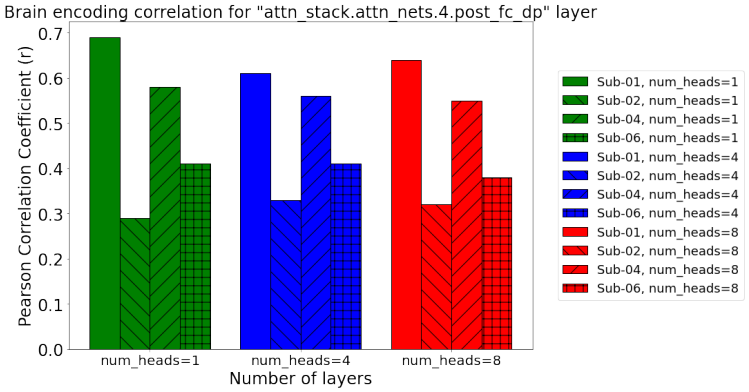


(c) Brain maps across subjects

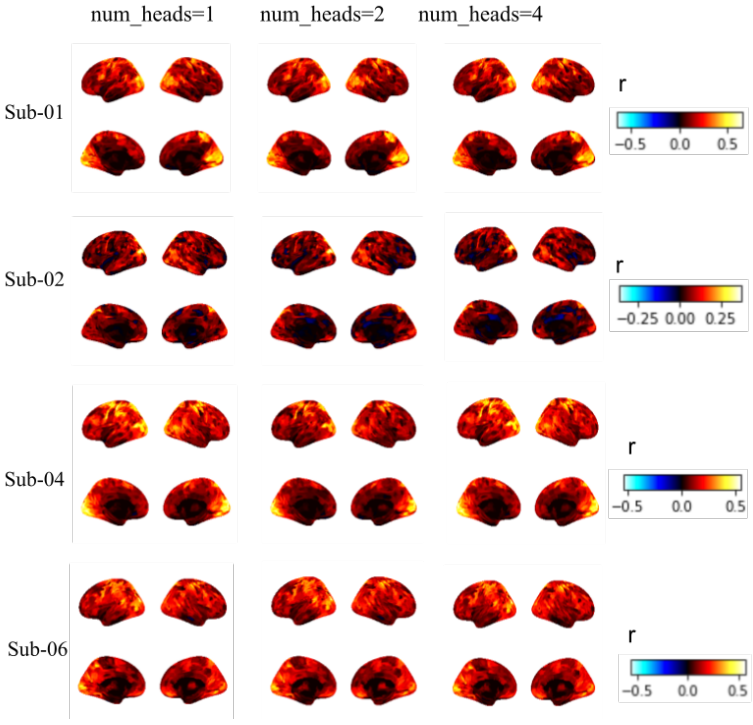
Figure 4.4: GPT training across different number of layers



(a) GPT training loss



(b) Brain encoding correlation coefficients across subjects



(c) Brain maps across subjects

Figure 4.5: Training GPT with FP32 and FP16 precision

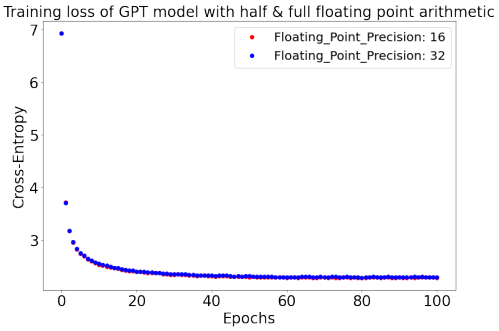
4.3.
Figure 4.5a shows the training loss curves across different numbers of attention heads. The training loss decreases steadily and converges similarly across all configurations, regardless of the number of heads. However, there is no significant difference in the rate of convergence or the final cross-entropy values, suggesting that the number of attention heads does not have impact on training of GPT efficiency.
Figure 4.5b presents the Pearson correlation coefficients between model predictions and actual brain activity across different subjects. The results indicate that brain encoding performance is somewhat sensitive to the number of attention heads. Notably, models with 1 attention head exhibit noticeable higher correlation values for Sub-01 compared to models with 4 and 8 attention heads. However, for Sub-02, models with 2 attention heads show slightly better performance compared to models with 4 and 8 heads. For Sub-04 and Sub-06, varying the number of attention heads does not significantly affect brain encoding performance. Increasing the number of attention heads does not always translate to better brain encoding performance.

4.3.5 Training GPT with 32-bit and 16-bit floating-point precision yields exactly the same results for brain encoding

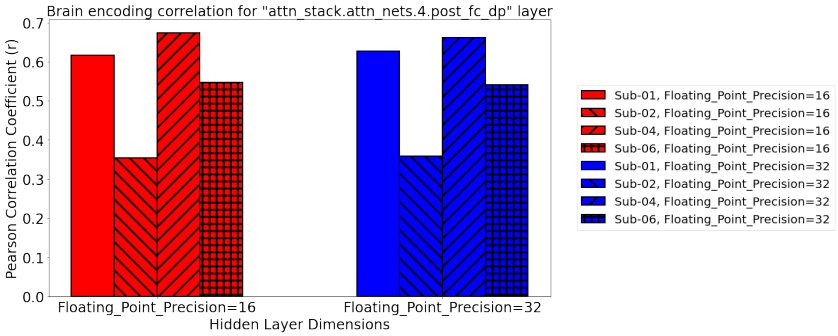
The training time of GPT with 32-bit floating-point precision was 35 hours, while training with 16-bit floating-point precision took 30 hours. The computational environment included 5 CPUs and 4x Tesla V100-SXM2-16GB GPUs in the Beluga HPC computing cluster. This means that using standard 16-bit precision sped up training by 1.17 times.
For this experiment, we leveraged PyTorch Lightning framework to enable mixed precision GPT training. The model size of GPT is presented in Table.4.3. We utilized the standard FP16 format, which consists of 1 sign bit, 5 exponent bits, and 10 mantissa bits. The Beluga dashboard provided execution time and memory statistics, indicating that 16-bit precision was more efficient.
As shown in Figure 4.6a, training GPT with both 32-bit and 16-bit precision results in the same loss function behavior over epochs, with identical convergence rates. We then benchmarked the effects on brain encoding and found that the results were exactly the same for all six subjects, as depicted in Figure 4.6b. Thus, training with 16-bit floating-point precision (standard FP16 format) leads to 1.17 times speed-up without any impact on brain encoding prediction accuracy. The loss values for training GPT were identical in both scenarios. In terms of brain encoding results, the brain map regions were compatible, and our findings indicated that the maximum correlation values were the same in both scenarios.

4.4 Conclusion

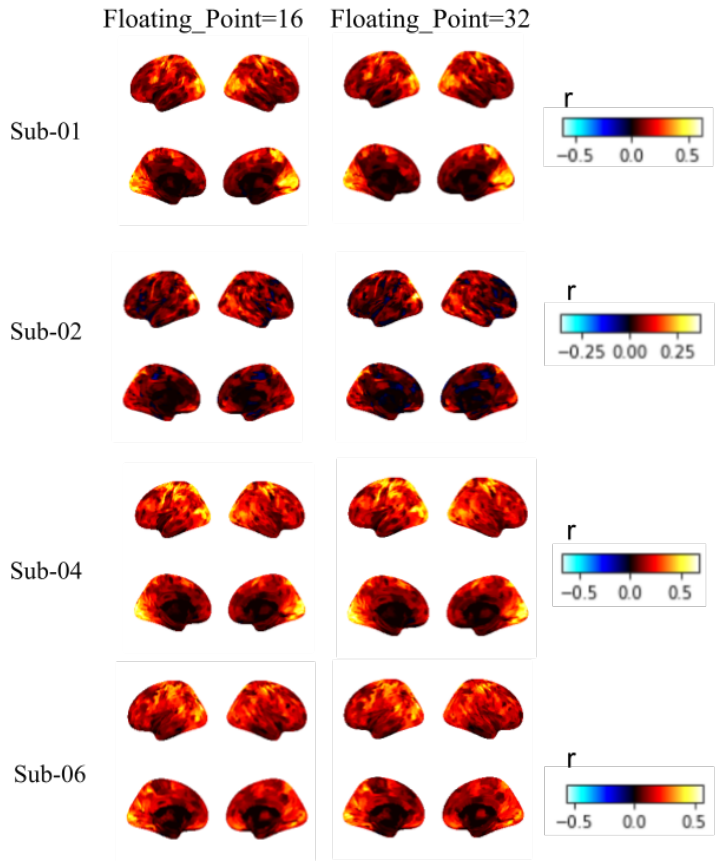
In this study, we assessed the influence of scaling dataset size, hidden dimensions, layers, attention heads, and floating-point precision on the performance of the VideoGPT model in brain encoding tasks. The findings reveal that increasing the training dataset size leads to substantial improvements in brain encoding performance, as evidenced by lower training losses and higher Pearson correlation coefficients across subjects. Notably, models trained on



(a) GPT training loss



(b) Brain encoding correlation coefficients across subjects



(c) Brain maps across across subjects

Figure 4.6: Training GPT with FP32 and FP16 precision

larger datasets enhanced predictive accuracy, underscoring the importance of data quantity for effective brain encoding. Similarly, scaling the model's hidden dimensions improved performance, with an optimal increase observed at 576 hidden dimensions, while further increases resulted in diminishing returns and potential overfitting. Conversely, the number of layers had a limited effect on performance, with noticeable gains up to eight layers, beyond which the improvements plateaued and overfitting became evident. The investigation into the number of attention heads revealed that changes in this parameter had little impact on overall performance, indicating that the effectiveness of attention mechanisms may not directly correlate with their quantity. Finally, training GPT with both 32-bit and 16-bit floating-point precision yielded identical results, indicating that the choice of precision does not significantly impact the model's performance in this context. This finding enables us to utilize mixed precision training to accelerate the training time of transformer models on stimuli without sacrificing brain encoding correlations. Overall, the results highlight the critical role of dataset size and model complexity in enhancing brain encoding capabilities, providing insights for future research and applications in neural decoding and cognitive modeling.

4.5 Availability of code and data

The code to reproduce our experiments is available at https://github.com/Sana3883/compute-optimal_GPT_brain-encoding .The CNeuroMod dataset is available at https://www.cneuromod.ca/gallery/datasets

4.6 Appendix

Table 4.5: Layers of block 4 and their corresponding activation shapes

Index	Name of layer	Activation shape
1	attn_stack.attn_nets.4.pre_attn_norm	[1, 4, 8, 8, 576]
2	attn_stack.attn_nets.4.post_attn_dp	[1, 4, 8, 8, 576]
3	attn_stack.attn_nets.4.attn.w_qs	[1, 4, 8, 8, 576]
4	attn_stack.attn_nets.4.attn.w_ks	[1, 4, 8, 8, 576]
5	attn_stack.attn_nets.4.attn.w_vs	[1, 4, 8, 8, 576]
6	attn_stack.attn_nets.4.attn.fc	[1, 4, 8, 8, 576]
7	attn_stack.attn_nets.4.attn.attn	[1, 4, 4, 8, 8, 144]
8	attn_stack.attn_nets.4.pre_enc_norm	[1, 4, 8, 8, 576]
9	attn_stack.attn_nets.4.post_enc_dp	[1, 4, 8, 8, 576]
10	attn_stack.attn_nets.4.enc_attn.w_qs	[1, 4, 8, 8, 576]
11	attn_stack.attn_nets.4.enc_attn.w_ks	[1, 4, 8, 8, 576]
12	attn_stack.attn_nets.4.enc_attn.w_vs	[1, 4, 8, 8, 240]
13	attn_stack.attn_nets.4.fc	[1, 4, 8, 8, 576]
14	attn_stack.attn_nets.5.enc_attn.attn	[1, 4, 4, 8, 8, 60]
15	attn_stack.attn_nets.4.pre_fc_norm	[1, 4, 8, 8, 576]
16	attn_stack.attn_nets.4.post_fc_dp	[1, 4, 8, 8, 576]
17	attn_stack.attn_nets.4.fc_block	[1, 4, 8, 8, 576]
18	attn_stack.attn_nets.4.fc_block.0	[1, 4, 8, 8, 2304]
19	attn_stack.attn_nets.4.fc_block.1	[1, 4, 8, 8, 2304]
20	attn_stack.attn_nets.4.fc_block.2	[1, 4, 8, 8, 576]

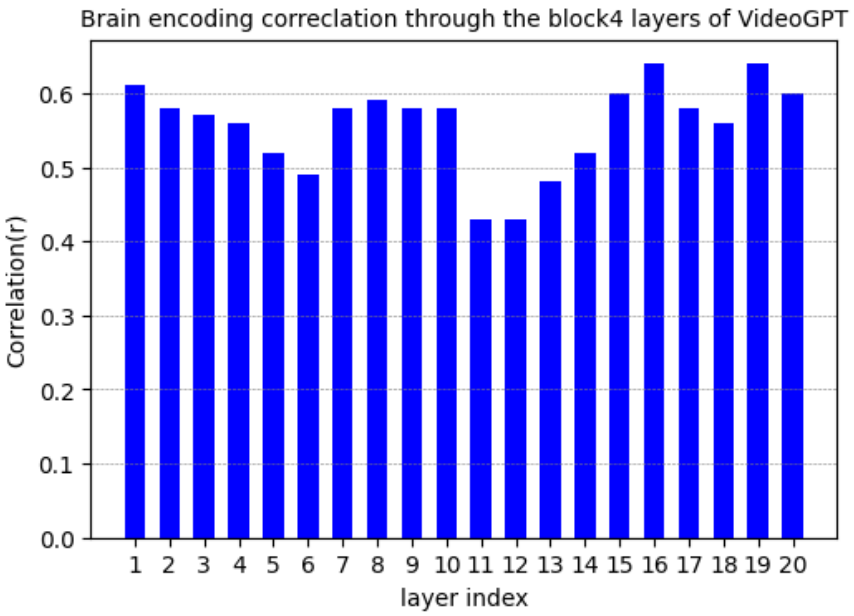


Figure 4.7: Pearson correlation prediction values for Sub-01 brain encoding across layers presented in Table 4.5

Chapter 5

Conclusions and Future work

Training brain encoding models at high spatial resolution presents a significant challenge, primarily due to the computational demands arising from both data size and model size. These models aim to decode brain processes from natural stimuli, such as visual scenes or language, and require extensive neuroimaging data collected at voxel-based fMRI recordings. Capturing neural activity at this level of detail requires massive datasets. The use of transformers-based architectures, further amplifies the computational requirements, as these architectures are particularly data-hungry and resource-intensive.
For research labs, the costs associated with necessary computational infrastructure, including powerful processors and high-memory storage can be prohibitive. As the field moves toward more complex brain encoding models capable of mapping fine spatial and temporal features of neural activity, the gap in computational resources becomes more pronounced. During this study, we focused on optimizing machine learning models, particularly emphasizing computational resource efficiency. My research involved two main projects: scaling up a Ridge regression model and scaling up the GPT-2 model.

Project 1: Scaling Up Ridge regression
In the first project, we proposed a scalable version of Scikit-learn's ridge regression, which is widely used for regression tasks due to its robust optimization techniques for closed-form solutions. Scikit-learn's implementation struggles with large-scale datasets, especially in domains like bioinformatics and neuroscience, where datasets can contain over 100,000 targets. Training on such datasets becomes extremely slow, and loading large matrices is a challenge even with high-memory resources. To address this issue, we evaluated parallelization techniques to accelerate the training process. Specifically, we compared multi-threading using Intel's Math Kernel Library (MKL) with OpenBLAS. The results showed that MKL was nearly twice as fast due to its optimization for Intel architectures, leveraging features like vectorization, parallelization, and cache management to improve matrix operation performance. However, multi-threading efficiency plateaued after about 32 threads, primarily due to limited shared resources, thread management overhead, and synchronization complexity, which diminished the benefits of additional parallelization.
We developed a scalable multi-CPU version of ridge regression using batch-based parallelization through the Dask distributed framework. Our approach divided the main problem into subproblems, each corresponding to a subset of target values. Each worker trained a ridge re-

gression model on its assigned subproblem and predicted the subset of targets independently, as Scikit-learn assumes target independence. The Dask scheduler managed the distributed workers, efficiently allocating resources across compute nodes and balancing the workload. The scheduler also handled failures by reallocating tasks, optimizing resource usage during distributed training. Our results demonstrated that batch ridge regression scaled effectively across multiple compute nodes and threads, and we plan to integrate this approach into an upcoming Scikit-learn release.
In future work, we aim to enhance this distributed system by introducing a master node responsible for performing matrix computations on the input data, particularly computing matrix M in 3.3 (see Section 3.2.3.1). The master node will then broadcast the result to the workers, improving training process efficiency. This approach is outlined as follows:

1. **Centralized Matrix Computation**: Assign one worker (the master node) to compute matrix M, ensuring this operation is performed only once.

2. **Broadcasting Results to Workers**: Use Dask's broadcasting functionality to send the computed matrix M to all workers, enabling them to use it in their computations without recalculating it.

This planned integration of a master node and optimized matrix computations will contribute to more efficient resource utilization and improved scalability. Additionally, we plan to extend this implementation for GPU usage, while accounting for the memory limitations of GPUs.

Project 2: Compute optimal vision transformers for brain encoding
The optimal training of a vision transformer for brain encoding depends on three key factors: model size, data scale, and computational resources. This study explores these pillars, focusing on how data scaling, model scaling, and mixed precision high-performance computing impact brain encoding outcomes. We employed VideoGPT to extract spatiotemporal features from videos and trained Ridge regression to predict brain activity based on these features. Benchmark experiments were conducted using varying data sizes (10k, 100k, 1M, 6M) and different configurations of GPT-2, adjusting hidden layer dimensions, number of layers, and attention heads. We also assessed the impact of 32-bit versus 16-bit floating point precision on training.
Our findings demonstrate that increasing the hidden layer dimensions consistently enhances brain encoding performance, as shown by higher Pearson correlation coefficients across all subjects. In contrast, the number of attention heads showed little influence on the results, while adding more layers led to marginal improvements, although not as pronounced as with hidden layer dimensions. Regarding data scaling, larger datasets significantly improved brain encoding performance, with the best results observed with the 6M dataset. These outcomes underscore that data scaling plays a more crucial role than model scaling in improving brain encoding accuracy.
The mixed precision (32-bit vs. 16-bit) experiments indicate minimal loss in performance when using 16-bit precision, making it a resource-efficient alternative for large-scale training. For future work, we propose extending this benchmarking approach to evaluate additional

high-performance techniques and designing computationally optimal models with smaller architectures. This includes:

1. **Parallelism Techniques:** Employing techniques such as pipeline, data, and tensor parallelism to provide distributed training processes.

2. **Efficient Model Design**: Developing models with smaller architectures that incorporate enhanced attention mechanisms, such as flash attention and sparse attention, to handle longer sequences while reducing resource consumption. Other strategies, such as pruning unnecessary parameters, model quantization, and knowledge distillation, can also be utilized to minimize model size without compromising brain encoding performance, allowing these smaller models to operate on par with larger models.

Smaller models not only require less computational power but also reduce memory usage, making it feasible for research labs to train and deploy these models on more accessible hardware. The impacts of these approaches have been largely underexplored in the neuroscience literature but may hold significant potential for enhancing brain encoding models. Testing their effects, similar to our investigations of low-precision training, could provide new insights and further optimize brain encoding efficiency when using transformers and large fMRI datasets.

Bibliography

[1] A. Abraham, F. Pedregosa, M. Eickenberg, P. Gervais, A. Mueller, J. Kossaifi, A. Gramfort, B. Thirion, and G. Varoquaux. Machine learning for neuroimaging with scikit-learn. *Frontiers in neuroinformatics*, page 14, 2014.

[2] I.M. Alabdulmohsin, B. Neyshabur, and X.. Zhai. Revisiting neural scaling laws in language and vision. *Advances in Neural Information Processing Systems*, 2022.

[3] I.M. Alabdulmohsin, X. Zhai, A. Kolesnikov, and L. Beyer. Getting vit in shape: Scaling laws for compute-optimal model design. *Advances in Neural Information Processing Systems*, page 36, 2024.

[4] E.J. Allen, G. St-Yves, Y. Wu, J.L. Breedlove, L.T. Dowdle, B. Caron, F. Pestilli, I. Charest, J.B. Hutchinson, T. Naselaris, and K. Kay. A massive 7t fmri dataset to bridge cognitive and computational neuroscience. *memory*, 2021.

[5] R. Antonello, A. Vaidya, and A. Huth. Scaling laws for language encoding models in fmri. *Advances in Neural Information Processing Systems*, page 36, 2024.

[6] M. Barth, F. Breuer, P.J. Koopmans, D.G. Norris, and B.A. Poser. Simultaneous multislice (sms) imaging techniques. *Magnetic resonance in medicine*, 75:63–81, 2016.

[7] R. Beliy, G. Gaziv, A. Hoogi, F. Strappini, T. Golan, and M Irani. From voxels to pixels and back: Self-supervision in natural-image reconstruction from fmri. *In Advances in Neural Information Processing Systems*, pages 6517–6527, 2019.

[8] N.C. Benson and J. Winawer. Bayesian analysis of retinotopic maps. *elife*, 7:e40224, 2018.

[9] J.A. Boyle and et al Pinsard, B. The courtois project on neuronal modeling - 2021 data release. *Poster 2224, 2021 Annual Meeting of the OHMB held virtually*, 2021.

[10] C. Caucheteux and J.R. King. Brains and algorithms partially converge in natural language processing. *Communications biology*, page 134, 2022.

[11] N. Chang, J.A. Pyles, A. Marcus, A. Gupta, M.J. Tarr, and E.M. Aminoff. Bold5000: a public fmri dataset while viewing 5000 visual images. *Scientific data*, pages 1–18, 2019.

[12] D. Changde, L. Jinpeng, H. Lijie, and H. Huiguang. Brain encoding and decoding in fmri with bidirectional deep generative models. *Nature communications*, 2019.

[13] C. Conwell, J.S. Prince, Alvarez G., and T. Konkle. Large-scale benchmarking of diverse artificial vision models in prediction of 7t human neuroimaging data. *bioRxiv*, 2022.

[14] Colin Conwell, Jacob S Prince, Kendrick N Kay, George A Alvarez, and Talia Konkle. What can 1.8 billion regressions tell us about the pressures shaping high-level visual representation in brains and machines? *BioRxiv*, 2022.

[15] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *IEEE conference on computer vision and pattern recognition*, pages 248–255, 2009.

[16] J. Deng, W. Dong, R. Li Socher, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *IEEE conference on computer vision and pattern recognition*, pages 248–255, 2009.

[17] O. Esteban, C.J. Markiewicz, R.W. Blair, C.A. Moodie, A.I. Isik, A. Erramuzpe, J.D. Kent, M. Goncalves, E. DuPre, M. Snyder, and H. Oya. fmriprep: a robust preprocessing pipeline for functional mri. nature methods. *Neuroimage*, 16:111–116, 2019.

[18] J. Goense, Y. Bohraus, and N.K. Logothetis. fmri at high spatial resolution: implications for bold-models. *Frontiers in computational neuroscience*, 10:p.66, 2016.

[19] A. Goldstein, E. Ham, S.A. Nastase, Z. Zada, A. Dabush, B.B. Aubrey, M. Schain, H. Gazula, A. Feder, W. Doyle, and S. Devore. Correspondence between the layered structure of deep language models and temporal structure of natural language processing in the human brain. *bioRxiv*, 2022.

[20] Y. Harel, A. Cyr, J. Boyle, B. Pinsard, K. Jerbi, and P. Bellec. Gamer in the scanner: Event-related analysis of fmri activity during retro videogame play guided by automated annotations of game content. 2023.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] A.E. Hoerl and R.W. Kennard. Ridge regression: applications to nonorthogonal problems. *Technometrics*, 12(1):69–82, 1970.

[24] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D.D.L. Casas, L.A. Hendricks, J. Welbl, A. Clark, and T. Hennigan. Training compute-optimal large language models. *In Proceedings of the 36th International Conference on Neural Information Processing Systems*, page 36, 2022.

[25] K.J. Holyoak. Parallel distributed processing: explorations in the microstructure of cognition. *Science*, pages 992–997, 1987.

[26] T. Horikawa, S.C. Aoki, M. Tsukamoto, and Y. Kamitani. Characterization of deep neural network features by decodability from human brain activity. *Scientific data*, page 190012, 2019.

[27] X. Hu, L. Guo, J. Han, and T. Liu. Decoding power-spectral profiles from fmri brain activities during naturalistic auditory experience. *Brain imaging and behavior*, 11(1):253–263, 2017.

[28] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q.V. Le, and Y. Wu. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *In Advances in neural information processing systems*, pages 103–112, 2019.

[29] Intel. Linear regression has never been faster. https://www.intel.com/content/www/us/en/developer/articles/technical/accelerating-linear-models-for-machine-learning.html, 2021.

[30] S. Jain and A. Huth. Incorporating context into language encoding models for fmri. *Advances in neural information processing systems*, page 31, 2018.

[31] J.B. Julian, E. Fedorenko, J. Webster, and N. Kanwisher. An algorithmic method for functionally defining regions of interest in the ventral visual pathway. *Neuroimage*, pages 2357–2364, 2012.

[32] K.N. Kay, T. Naselaris, R.J. Prenger, and J.L. Gallant. Identifying natural images from human brain activity. *Nature*, pages 352–355, 2008.

[33] K.N. Kay, T. Naselaris, R.J. Prenger, and J.L. Gallant. Identifying natural images from human brain activity. *Nature*, 452(7185):352–355, 2008.

[34] K.N. Kay, T. Naselaris, R.J. Prenger, and J.L. Gallant. Reconstructing visual experiences from brain activity evoked by natural movies. *Current Biology*, 21(19):1641–1646, 2011.

[35] A.J. Kell, D.L. Yamins, E.N. Shook, S.V. Norman-Haignere, and J.H. McDermott. A task-optimized neural network replicates human auditory behavior, predicts brain responses, and reveals a cortical processing hierarchy. *Neuron*, 98:630–644, 2018.

[36] S. Kumar, T. Sumers, T.R.and Yamakoshi, A. Goldstein, U. Hasson, K.A. Norman, T.L. Griffiths, R.D. Hawkins, and S.A. Nastase. Reconstructing the cascade of language processing in the brain using the internal computations of a transformer-based language model. *bioRxiv*, 2022.

[37] Tom Dupré la Tour, Michael Eickenberg, Anwar O Nunez-Elizalde, and Jack L Gallant. Feature-space selection with banded ridge regression. *NeuroImage*, 264:119728, 2022.

[38] M.D. Lescroart and J.L Gallant. Human scene-selective areas represent 3d configurations of surfaces. *Neuron*, 101:178–192, 2019.

[39] L.H. Li, M. Yatskar, D. Yin, C.J. Hsieh, and K.W. Chang. Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*, 2019.

[40] T.Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C.L. Zitnick. Microsoft coco: Common objects in context. *In European conference on computer vision*, pages 740–755, 2014.

[41] N.K. Logothetis, J. Pauls, M. Augath, T. Trinath, and A. Oeltermann. Neurophysiological investigation of the basis of the fmri signal. *nature*, pages 150–157, 2001.

[42] I. Loshchilov and F Hutter. Sgdr: Stochastic gradient descent with warm restarts. *Learning*, page 3, 2016.

[43] T. Matsuyama, K.S. Sasaki, and S. Nishimoto. Applicability of scaling laws to vision encoding models. *arXiv preprint arXiv:2308.00678*, 2023.

[44] R. Mayer and H.A. Jacobsen. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. *ACM Computing Surveys (CSUR)*, pages 1–37, 2020.

[45] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu. Mixed precision training. *In International Conference on Learning Representations.*, 2017.

[46] T.M. Mitchell, S.V. Shinkareva, A. Carlson, K.M. Chang, V.L. Malave, R.A. Mason, and M.A. Just. Predicting human brain activity associated with the meanings of nouns. *science*, 320(5880):1191–1195, 2008.

[47] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018.

[48] T. Naselaris, E. Allen, and K. Kay. Extensive sampling for complete models of individual brains. current opinion in behavioral sciences. *Current Opinion in Behavioral Sciences*, 40:45–51, 2021.

[49] T. Naselaris, E. Allen, and K. Kay. Extensive sampling for complete models of individual brains. current opinion in behavioral sciences. *Current Opinion in Behavioral Sciences*, 40:45–51, 2021.

[50] T. Naselaris, K.N. Kay, S. Nishimoto, and J.L. Gallant. Encoding and decoding in fmri. neuroimage. *Technometrics*, 56:400–410, 2011.

[51] S.R. Oota, J. Arora, V. Rowtula, M. Gupta, and R.S. Bapi. Visio-linguistic brain encoding. *The 29th International Conference on Computational Linguistics*, 2022.

[52] M. Ott, S. Edunov, D. Grangier, and M. Auli. Scaling neural machine translation. *In Proceedings of the Third Conference on Machine Translation: Research Papers*, 2018.

[53] A. Pasquiou, Y. Lakretz, J. Hale, B. Thirion, and C. Pallier. Neural language models are not born equal to fit brain data, but training helps. *In International Conference on Machine Learning*, 2022.

[54] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[55] F. Pereira, B. Lou, B. Pritchett, Gershman Ritter, S., N. S.J., Kanwisher, M. Botvinick, and E. Fedorenko. Toward a universal decoder of linguistic meaning from brain activation. *Nature communications*, 9:1–13, 2018.

[56] R.A. Poldrack, J.A. Mumford, and T.E Nichols. Handbook of functional mri data analysis. *Cambridge University Press*, 2011.

[57] R. Puri, R. Kirby, N. Yakovenko, and B. Catanzaro. Large scale language modeling: Converging on 40gb of text in four hours. *30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 290–297, 2018.

[58] K. Qiao, J. Chen, L. Wang, C. Zhang, L. Zeng, L. Tong, and 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.

[59] A. Radford, J.W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, and G. Krueger. Learning transferable visual models from natural language supervision. *In International conference on machine learning*, pages 8748–8763, 2021.

[60] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, page 9, 2019.

[61] C. ratton, T.O. Laumann, A.N. Nielsen, D.J. Greene, E.M. Gordon, A.W. Gilmore, S.M. Nelson, R.S. Coalson, A.Z. Snyder, B.L. Schlaggar, and N.U. Dosenbach. Functional brain networks are dominated by stable group and individual factors, not cognitive or daily variation. *Neuron*, pages 439–452, 2018.

[62] M Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. *In Proceedings of the 14th python in science conferenc*, 130:136, 2015.

[63] K. Seeliger, L. Ambrogioni, Y. Güçlütürk, L.M. van den Bulk, U. Güçlü, and M.A.J. van Gerven. End-to-end neural system identification with neural information flow. *PLOS Computational Biology*, 17(2), 2021.

[64] K. Setsompop, J. Cohen-Adad, B.A. Gagoski, T. Raij, A. Yendiki, B. Keil, V.J. Wedeen, and L.L. Wald. Improving diffusion mri using simultaneous multi-slice echo planar imaging. *Neuroimage*, 63:569–580, 2012.

[65] G. Shen, K. Dwivedi, K. Majima, T. Horikawa, and Y. Kamitani. End-to-end deep image reconstruction from human brain activity. *Frontiers in computational neuroscience*, 13:21, 2019.

[66] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv*, 2014.

[67] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv*, 2014.

[68] Cneuromod team. Cneuromod dataset. https://www.cneuromod.ca/gallery/datasets/, 2021.

[69] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.A. Lachaux, T. Lacroix, B. Rozière, andHambro E. Goyal, N., F. Azhar, and A. Rodriguez. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[70] S. Urchs, J. Armoza, C. Moreau, Y. Benhajali, J. St-Aubin, P. Orban, and P. Bellec. Mist: A multi-resolution parcellation of functional brain networks. *MNI Open Research*, 1:3, 2019.

[71] A. Van Den Oord and O. Vinyals. Neural discrete representation learning. *Advances in neural information processing systems*, page 30, 2017.

[72] D.C. Van Essen and M.F. Glasser. The human connectome project: Progress and prospects. in cerebrum: the dana forum on brain science. *Dana Foundation*, 63, 2016.

[73] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 2017.

[74] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang. Intel math kernel library. in high-performance. *Computing on the Intel® Xeon Phi™*, pages 167–188, 2014.

[75] X.W. Wang, D. Nie, and B.L. Lu. Emotional state classification from eeg data using machine learning approach. *Neurocomputing*, 129:94–106, 2014.

[76] L. Wehbe, A. Ramdas, R.C. Steorts, and C.R. Shalizi. Regularized brain reading with shrinkage and smoothing. *The annals of applied statistics*, page 1997, 2015.

[77] H. Wen, J. Shi, Y. Zhang, K.H. Lu, J. Cao, and Z.L Liu. Neural encoding and decoding with deep learning for dynamic natural vision. *Cerebral Cortex*, 28(12)):4136–4160, 2018.

[78] Z. Xianyi, W. Qian, and Z. Yunquan. Model-driven level 3 blas performance optimization on loongson 3a processor. *IEEE 18th international conference on parallel and distributed systems*, pages 1–18, 2012.

[79] J. Xu, S. Moeller, E.J. Auerbach, J. Strupp, S.M. Smith, D.A. Feinberg, E. Yacoub, and K. Uğurbil. Improving diffusion mri using simultaneous multi-slice echo planar imaging. *NeuroimageT*, 83:991–1001, 2013.

[80] W. Yan, Y. Zhang, P. Abbeel, and A. Srinivas. Videogpt: Video generation using vq-vae and transformers. *arXiv preprint arXiv:2104.10157*, 2021.

[81] S.D. Yun and N.J. Shah. Whole-brain high in-plane resolution fmri using accelerated epik for enhanced characterisation of functional areas at 3t. *PloS one*, 12(9):p.e0184759, 2017.

[82] Kolesnikov A. Houlsby N. Zhai, X. and L. Beyer. Scaling vision transformers. *In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12104–12113, 2022.

[83] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X.V. Lin, and T. Mihaylov. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.