Abstract

A grand challenge in neuroscience is to understand how a human brain functions. Much previous research has focused on studying the activities of brain regions. However, the functional interactions between brain regions are not well understood. This dissertation focuses on computational methods to analyze data from functional magnetic resonance imaging (fMRI) scanners to study the functional interactions in the human brain.

This dissertation proposes, designs and implements efficient systems to conduct full correlation matrix analysis (FCMA) as an unbiased way to explore functional interactions in the human brain. Since a straightforward way to study FCMA would take years to complete one run with a typical neuroscience study dataset on a modern compute server, no previous attempt has been made in the past. This dissertation makes several contributions. First, we proposed and implemented parallel algorithms and optimizations on a multi-processor cluster, which improved FCMA computation by three orders of magnitude.

Second, we demonstrated that FCMA can effectively show functional interactions in the human brain by conducting a neuroscience study on an fMRI dataset. Our study successfully identified functional interactions between certain brain regions.

Third, we proposed and implemented optimization methods for FCMA for emerging many-core processors such as the Intel® Xeon Phi™ coprocessors and improved the performance of computing FCMA by another order of magnitude. On a 96-node Xeon Phi cluster, our system can finish an FCMA study with a typical dataset in minutes.

Finally, we proposed, designed and implemented a service for real-time, closed loop neuroscience studies. Our real-time FCMA can process and analyze brain volumes from multiple fMRI experiments on a 40-node compute cluster simultaneously and send the neurofeedback to each fMRI scanner over the Internet within 1.5 sec-
onds. This system uses a novel method to improve the performance and utilization of compute nodes while meeting the real-time requirements in the presence of node failures.
Acknowledgements

First and foremost, I would like to express my special appreciation to my adviser Kai Li for the continuous guidance, encouragement and support throughout my PhD study. Kai not only showed me how to do great research, but also advised me how to build my career. It is my fortune to have such a wonderful mentor.

I am very grateful to Jon Cohen and Nick Turk-Browne for advising me from the neuroscience perspective and being my dissertation readers. My interdisciplinary research will not be possible without their guidance.

I would like to thank Olga Troyanskaya and Sebastian Seung for serving in my dissertation committee.

I would also like to thank Ted Willke and researchers in his group, Mike Anderson, Mihai Capota, Bryn Keller and Narayanan Sundaram, from Intel Corporation for the fruitful collaboration.

I especially thank my colleagues Zhe Wang, Dong Wei and the retired graduate coordinator Melissa Lawson for their help during my early days in Princeton, my colleagues Yungang Bao and Linpeng Tang for all the inspiring discussions. I also thank people in Cohen Lab, Turk-Browne Lab and the PNI technical staff for their various kinds of support.

I feel lucky to have met a lot of good friends in Princeton. The time we spent together is forever memorized.

Finally, special thanks to my beloved wife Siming Li for being part of my life; my dearest grandma for her endless love; my parents and parents-in-law for their great support and my daughter for bringing me the most special joy.

This work is supported by Intel Corporation, the J. Insley Blair Pyne fund at Princeton University, the John Templeton Foundation, the National Science Foundation (MRI BCS1229597), and the National Institutes of Health (R01 EY021755).
To my family.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xii</td>
</tr>
</tbody>
</table>

### 1 Introduction

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Functional Magnetic Resonance Imaging</td>
<td>1</td>
</tr>
<tr>
<td>1.2 FMRI Data Analysis</td>
<td>3</td>
</tr>
<tr>
<td>1.2.1 Processing Pipeline</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2 Challenges</td>
<td>7</td>
</tr>
<tr>
<td>1.2.3 Previous Methods</td>
<td>9</td>
</tr>
<tr>
<td>1.3 Contributions</td>
<td>15</td>
</tr>
</tbody>
</table>

### 2 Full Correlation Matrix Analysis

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Introduction</td>
<td>20</td>
</tr>
<tr>
<td>2.2 Challenges</td>
<td>23</td>
</tr>
<tr>
<td>2.3 Implementation</td>
<td>25</td>
</tr>
<tr>
<td>2.3.1 Overview</td>
<td>25</td>
</tr>
<tr>
<td>2.3.2 Optimization of Correlation Computation</td>
<td>25</td>
</tr>
<tr>
<td>2.3.3 Parallelization of Correlation Computation</td>
<td>27</td>
</tr>
<tr>
<td>2.3.4 Parallelization of Voxel-wise Classifier Analysis</td>
<td>29</td>
</tr>
</tbody>
</table>
2.3.5 Classification of Correlation Patterns .................................. 30
2.4 Performance Benchmarking .................................................. 32
2.5 FCMA Toolbox ................................................................. 32
  2.5.1 Code Availability .......................................................... 32
  2.5.2 Other Functionality ....................................................... 34
  2.5.3 Generalizability ............................................................ 35
2.6 Related Work ................................................................. 35
2.7 Summary ............................................................................. 36

3 A Case Study of FCMA ............................................................ 38
  3.1 Introduction ........................................................................ 38
  3.2 Case Study ......................................................................... 39
    3.2.1 Dataset ........................................................................ 39
    3.2.2 Data Preparation .......................................................... 40
    3.2.3 Classification ............................................................... 41
    3.2.4 Information Mapping ...................................................... 44
    3.2.5 Vector Change Analysis .................................................. 48
    3.2.6 Relationship of mPFC to classic visual cortex .................... 50
    3.2.7 Background Connectivity Analysis .................................. 55
    3.2.8 Seed-based Analysis ...................................................... 58
  3.3 Related Work and Discussion ............................................... 59
  3.4 Summary ............................................................................. 61

4 Optimization of FCMA on Many-core Processors ...................... 62
  4.1 Introduction ........................................................................ 62
  4.2 Intel® Xeon Phi™ Architecture ............................................. 66
  4.3 Algorithms ......................................................................... 68
    4.3.1 Overview of FCMA Voxel Selection ................................. 68
4.3.2 Basic Implementation ........................................ 73
4.3.3 Performance Analysis ...................................... 75

4.4 Optimizations .............................................. 78
  4.4.1 Optimization Ideas .................................... 78
  4.4.2 Correlation Computation ................................ 79
  4.4.3 Within-subject normalization .......................... 80
  4.4.4 SVM cross validation .................................. 81

4.5 Evaluation ................................................ 84
  4.5.1 Experimental Setup ................................... 85
  4.5.2 System Performance .................................. 85
  4.5.3 Speedup ............................................... 88
  4.5.4 Performance Implications of Optimizations ........ 88
  4.5.5 Performance on Intel® Xeon® Processors ............ 93

4.6 Related Work ............................................ 95

4.7 Summary .................................................. 96

5 A Real-time FCMA System ................................. 98
  5.1 Introduction ............................................. 98
    5.1.1 Motivation ......................................... 98
    5.1.2 Challenges ......................................... 100
    5.1.3 Requirements ...................................... 101
  5.2 Real-time FCMA ......................................... 103
    5.2.1 FCMA in Real-time Study .......................... 103
    5.2.2 RtFCMA Algorithm ................................ 105

5.3 Fault Tolerance .......................................... 111
  5.3.1 Fault Detection ...................................... 112
  5.3.2 Master Fault Recovery .............................. 113
  5.3.3 Worker Fault Recovery .............................. 114
5.4 System Architecture .................................................... 118
  5.4.1 Design .............................................................. 118
  5.4.2 Implementation .................................................... 120
5.5 Evaluation .............................................................. 124
  5.5.1 Experimental setup ............................................... 124
  5.5.2 RtFCMA Evaluation ................................................. 125
  5.5.3 Fault Tolerance Evaluation ....................................... 134
5.6 Related Work .......................................................... 138
5.7 Summary ............................................................... 140

6 Conclusion ....................................................................... 142
  6.1 Contributions ............................................................ 142
  6.2 Future Work .............................................................. 144

Bibliography .................................................................... 147
List of Tables

2.1 FCMA benchmark ................................................. 33

3.1 Randomization test of voxel frequencies ........................ 45
3.2 Randomization test of cluster sizes ................................ 46

4.1 Instrumentation results of basic FCMA .......................... 76
4.2 FMRI datasets .................................................... 85
4.3 Elapsed times of offline data analysis ............................ 87
4.4 Elapsed times of emulated online data analysis ................. 87
4.5 Matrix multiplication performance results ....................... 91
4.6 Matrix multiplication instrument results ......................... 91
4.7 Performance comparisons of retaining L2 cache contents .......... 92
4.8 Performance of SVM cross validation ........................... 93

5.1 Time break-down of a TR in rtFCMA ............................ 102
5.2 Performance of multiple services ................................ 132
5.3 Time to handle a master failure .................................. 135
5.4 Time to handle a classification worker failure ................... 136
5.5 Classification worker node recovery time ......................... 136
5.6 Voxel selection worker failure handling time ................... 137
List of Figures

1.1 FMRI yearly publication ............................................. 2
1.2 FMRI data processing pipeline .................................... 4
1.3 General linear model .................................................. 10
1.4 MVPA workflow ...................................................... 12
1.5 Functional connectivity .............................................. 14

2.1 FCMA workflow ...................................................... 28
2.2 FCMA classification procedure .................................... 31

3.1 Robustness of feature selection .................................... 43
3.2 Information mapping .................................................. 47
3.3 Vector change analysis .............................................. 49
3.4 Relationship of mPFC to FFA/PPA ................................ 52
3.5 Correlation between mPFC and visual cortex .................. 54
3.6 Spurious correlation .................................................. 55
3.7 Feature selection in background connectivity analysis ........ 57
3.8 Information mapping on residualized data ...................... 58
3.9 Seed-based analysis .................................................. 60

4.1 Closed-loop fMRI experiment ...................................... 63
4.2 Intel® Xeon Phi™ architecture ...................................... 67
4.3 Three-stage voxel selection .......................................... 69
4.4 Correlation computation ........................................... 72
4.5 Reain correlation values across stages .......................... 79
4.6 The $z$-score computation within L2 cache ...................... 81
4.7 Tall-skinny matrix multiplication for SVM kernel matrices .... 83
4.8 Scalability of the optimized code ................................ 88
4.9 Single worker task improvement .................................. 90
4.10 Performance improvement of our optimization on the coprocessor .... 94
4.11 Performance comparison between processor and coprocessor .... 95

5.1 Conceptual diagram for rtFCMA ................................. 102
5.2 Data processing flow of FCMA .................................... 105
5.3 Master-worker model of rtFCMA ................................. 106
5.4 Incremental voxel selection ....................................... 108
5.5 Sliding window of training samples ............................... 109
5.6 Fault tolerance ....................................................... 112
5.7 RtFCMA system architecture ...................................... 121
5.8 Voxel selection speedups and single node processing percentage ... 126
5.9 Duration of voxel selection and model update .................. 129
5.10 Duration of classification ......................................... 130
5.11 Voxel selection time comparison between rtFCMA and batchFCMA .... 133
5.12 Accuracy comparison between rtFCMA and batchFCMA .......... 134
5.13 Classification accuracy change in selection worker failures ....... 138
Chapter 1

Introduction

1.1 Functional Magnetic Resonance Imaging

This dissertation studies the functional interactions in the human brain based on neuroimaging data produced by functional magnetic resonance imaging (fMRI). FMRI is a functional neuroimaging procedure to detect the oxygenation level change of the blood flow in the brain in a non-invasive way. It is well-known that cerebral blood flow and neuronal activity are positively coupled, i.e. the more active a brain area is, the more blood flows through this area. Thus, fMRI is able to reflect metabolic activity, and by inference, neural activity, of the brain via measuring the blood oxygenation level dependent (BOLD) contrast.

Since the early 1990s, fMRI quickly became popular in the cognitive neuroscience community and served as the most common method for studying human brains [96]. The number of papers that mention fMRI grew exponentially according to the PubMed database (see Figure 1.1).

In cognitive neuroscience, there are mainly two ways that neuroscientists study how the mental function is organized in the human brain. The first way is examining the brains of individuals who had suffered particular neural system damage due to
stroke, infection, or injury. Its progress is limited by lacking of samples and proper quantitative methods.

The second approach, which is apparently more advanced, uses neuroimaging techniques to record brain functions while individuals (who can be normal people around us) are performing mental tasks to understand how mental functions relate to brain processes. Among other methods usually used in functional neuroimaging such as position emission tomography (PET) and electroencephalography (EEG), fMRI dominates the field because it provides an unprecedented ability to noninvasively record the brain activity with fine spatial resolution and relatively good temporal resolution which will be described below. In addition, fMRI is safe and can be repeatedly used in a broad range of individuals.
FMRI data are generated by fMRI scanners. When a human subject (or subject in short) is lying down in the scanner, the scanner acquires the brain data periodically. Overall, the fMRI data live in four-dimensional (4D) space, consisting of different three-dimensional (3D) brain volumes in different time points. Nowadays, the temporal resolution (TR) of an fMRI scanner to reliably acquire a brain volume is $1 - 2$ seconds, that is, an fMRI scanner generates one 3D brain volume every $1 - 2$ seconds. A typical fMRI experiment lasts tens of minutes, resulting in hundreds of thousands of TRs per subject.

In terms of spatial resolution, during one TR, a common configuration of the fMRI scanner measures the BOLD contrast in a $64 \times 64 \times 34$ box, each cell of the box has edges in $3 - 5$ mm. The 3D brain volume locates in the box and can be contoured by applying a mask to the box. The masked brain has $\sim 35,000$ cells (recognized as and called voxels henceforth), each voxel has its own value to depict the neural activity of its corresponding region in each TR.

1.2 FMRI Data Analysis

The dissertation works on analyzing the fMRI data in terms of functional interactions. In this section, we first describe the fMRI data processing pipeline to show how fMRI data analysis fits in the pipeline, and then list the challenges of the fMRI data analysis and the previous works.

1.2.1 Processing Pipeline

Figure 1.2 shows a four-stage pipeline of fMRI data processing in high level which is typically used in fMRI experiments. In this subsection we will first describe the fMRI experiment generating the fMRI data, and then briefly explain the steps in fMRI data analysis pipeline one by one.
Figure 1.2: FMRI data processing pipeline. FMRI data are acquired and reconstructed in the fMRI scanner, then sent to some external computing resource to do the preprocessing and analysis.

Experiment

FMRI data analysis applies to fMRI datasets, which are collected from fMRI experiments. In order to understand the brain functionality as the response to different mental tasks, neuroscientists first need to design proper fMRI experiments. A typical fMRI experiment consists of multiple experimental runs, each of which lasts for several minutes. During each experimental run, the subject being scanned is requested to perform some carefully designed mental tasks, e.g. viewing a series of images of faces or scenes, or paying attention to the left or right part of the screen. In addition, some auxiliary anatomical runs may be performed as well to attain the brain shape of the subject. The same experimental configuration (defined by the similar sequence of mental tasks) usually applies to tens of subjects to form an fMRI dataset consisting of multiple subjects’ data to analyze together.

Data acquisition

The fMRI data are acquired by the fMRI scanners (Siemens 3-Tesla Allegra in our case). When a subject is lying down in the scanner and doing the required mental tasks, the scanner measures the signal of BOLD contrast of the brain every 1 – 2 seconds (depending on the duration of a TR). The data are collected in k-space, which is the 3D Fourier transform of the measured magnetic resonance image.
Image reconstruction

Image reconstruction is a step to transform the k-space data acquired by the fMRI scanner into 3D brain images [59], which happens right after one scan of the brain data is collected. The image reconstruction takes place within the scanner like the data acquisition, and is typically done by graphical processing units (GPUs). The scanner operators (neuroscientists who conduct the fMRI experiments) need to coordinate with scanner vendors if they want to customize this step. The elapsed time of image reconstruction depends on the temporal resolution of the scan, the finer the temporal resolution is, the longer it takes to do a reconstruction. For a 1.5-second TR, it takes 300 – 700 ms to reconstruct one 3D image in our scanner (Siemens 3-Tesla Allegra). After reconstruction, the fMRI scanner usually will put together the meta data of the image as a header and attach it to the beginning of the image data.

Image preprocessing

The reconstructed 3D images are exported from the fMRI scanner for analysis. However, these data are too rudimentary to make scientific sense. Therefore, preprocessing is usually applied to the data before analysis. Typically, the image preprocessing involves the following components:

- Motion correction: to realign the brain data across scans (TRs) to correct for the head motion during the course of the experiment;

- Slice-time acquisition: to correct the difference in timing across the different slices in the image, because essentially the 3D image is formed by several 2D slices taken at slightly different time points within the range of a TR;

- Spatial smoothing: to blur the data spatially for reducing the noise;
• High-pass temporal filtering: to remove low-frequency noise of the data by applying a high-pass filter (i.e. only the signal with a frequency higher than a certain threshold will pass the filter);

• Spatial normalization: to align (warp) brain data from different subjects to a standard template (usually the Montreal Neurological Institute (MNI) template).

These preprocessing components are helpful to the subsequent data analysis in various ways.

Data analysis

The data analysis applies to the preprocessed brain data. Neuroscientists analyze the data to understand human brain functions in different cognitive tasks. Methodologically, there are in general two directions that neuroscientists pursue to analyze the data. The first one is univariate analysis which constructs statistical models and fit the data in order to estimate the response of individual voxels to the curtain mental tasks or stimuli.

The second set of methods attempts to extract patterns from the combination of a group of voxels. The pattern can be expressed as the activities of voxels (first order pattern) or the temporal correlations between activities of voxels (second order pattern). Known as multi-voxel pattern analysis (MVPA), this kind of data-driven approach captures more reliable brain function patterns than univariate analysis and is able to apply machine learning techniques to make prediction on new data rather than only describing the existing data. The dissertation focuses mainly on the data-driven methods to analyze the fMRI data in patterns of correlation to reveal functional interactions in the human brain.
1.2.2 Challenges

The emergence of fMRI techniques largely accelerates the scientific understanding of human brains in cognitive neuroscience. However, fMRI data analysis is exceedingly complex because of the following reasons.

Noisy data

FMRI data contain a considerable amount of noise compared to the underlying signal it conveys, i.e. the signal-to-noise ratio (SNR) is low. The noise comes from various sources, such as the scanner and the subject. The fMRI scanner has intrinsic artifacts such as magnetic drifting. Physiological elements of the subject like the head motion, changes in rate of blood flow, blood volume and use of oxygen over time, also contribute to the noise. In addition, unpredictable random thoughts of the subject which are not relevant to the designed mental tasks produce neural activity independent of the experiment manipulation.

The preprocessing components described in section 1.2.1 handle some noise of the data. However, it is not possible to control and constrain all noisy factors during an fMRI experiment even with the best experimental design and most advanced fMRI scanner. Sophisticated algorithms are needed to pick out useful signals from the data.

Irregular data shape

There are generally only 10 – 50 subjects in an fMRI dataset. Typically one subject accomplishes tens of mental tasks during the entire experiment. Therefore, unlike many modern big data problems which has millions or even billions of samples, when applying machine learning techniques to fMRI data, we are only processing tens to hundreds of samples, if we treat one task that the subject does as a sample.

As a contrast, the dimensionality of the samples in fMRI data analysis can be huge. The whole brain analysis produces samples in $O(n)$ dimensions if analyzing in
activity and in $O(n^2)$ dimensions if analyzing in correlation, where $n$ is the number of voxels of a brain ($\sim 35,000$).

In machine learning, when working on prediction problems over the data with irregular shape in which the number of features $p$ (in our case, tens of thousands to billions) is much larger the number of samples $N$ (in our case, tens to hundreds), it is easy to get overfitting [39]. Careful feature selection is usually applied before making sense out of data with $p >> N$.

**Intensive computation**

The computing power that an fMRI data analysis requires varies a lot in different analyses. The computational burden of some univariate analysis of individual voxels is so trivial that one can easily use Matlab or similar tools to obtain the results on a single machine. In contrast, MVPA typically requires more computing power, particularly if dealing with full brain correlation which may reveal the underlying functional interaction patterns in the human brain, it is extremely computationally intensive (details in chapter 2).

Neuroscientists often find the computationally intensive analysis intractable and have to make assumptions to release the computational burden, hence the analysis may be biased. High performance computing (HPC) techniques are more and more appealing in fMRI data analysis to help neuroscientists get rid of unnecessary assumptions.

**Lack of real-time data analysis tools**

To date, almost all existing fMRI studies are conducted in an offline fashion – data analysis occurs only after all data have been collected long after the subject has been taken out of the fMRI scanner. Although sufficient for many purposes, this offline approach assumes that every subject should perform similarly, if not exactly
the same, under the same experiment condition, missing opportunities to customize the experiment to an individual's cognitive abilities. In addition, it is impossible to generate online feedback to the subject using the information acquired on the fly to enhance participation.

Therefore, there is an increasing desire for conducting real-time fMRI studies in which the data is immediately processed after acquiring. However, it is not easy for neuroscientists to do this because of the lack of computing power/technique, and furthermore, the lack of readily accessible standardized tools for conducting real-time experiments.

1.2.3 Previous Methods

In order to make sense out of fMRI data, people have been developing various of data analysis approaches for decades. In this subsection we summarize the previous work of fMRI data analysis to date.

Univariate analysis

The most common univariate analysis is General Linear Model (GLM). GLM is often used for detecting task-related neural activities of voxels. It assumes those voxels have the similar patterns which can be summarized as a general model (e.g. getting larger activity values when the task is on and smaller activity values when the task is off). In GLM, the time dimension of the fMRI data plays the role of dependent variables. Normally the model is derived from the timing of the stimuli (caused by tasks) that was applied to the subject in the fMRI scanner, then the voxels whose activity along its time course fits the model are probably responding to the tasks.

A simple GLM of two stimuli can be described as

$$ y(t) = \beta_1 * x_1(t) + \beta_2 * x_2(t) + c + e(t) \quad (1.1) $$
where $y(t)$ is a vector containing the activity values of one voxel in a time course $t$, one value per TR; $x_1(t)$ and $x_2(t)$ are two models corresponding to two stimuli in the same time course containing values of the expected patterns related to their tasks, also one value per TR, which are often referred to as explanatory variables (EVs); $\beta_1$ and $\beta_2$ are the parameter estimates (PEs) for the models describing the “goodness-of-fit” to the models at this voxel; $c$ is a tuning constant; $e(t)$ is the error term. If a particular voxel responds strongly to stimulus (task) 1 but is irrelevant to stimulus (task) 2, GLM will result in a large value for $\beta_1$ and a small value (close to 0) for $\beta_2$ (see Figure 1.3).

![Figure 1.3: The general linear model in fMRI data analysis. The demo data responds to stimulus 1 but not to stimulus 2.](image)

The $\beta$ values then need to be converted into useful statistics to describe the significance mathematically. A common conversion is to calculate the $t$ values from the $\beta$ values by the following formula:

$$t = \beta / S_E(\beta)$$  \hspace{1cm} (1.2)
where $S_E(\beta)$ is the standard error of the $\beta$ values of the entire brain, which depicts the estimate uncertainty of the $\beta$ population. A voxel with higher $t$ value indicates that it is more unlikely that this voxel fits the predefined model well by chance alone. Furthermore, $z$ values and $p$ (probability) values can be derived from $t$ values using standard statistical transformation, describing how significant the data are related to the model. For example, a conventional threshold to define the statistic significance is $p < 0.05$, which means that the probability of getting a false positive (i.e. the voxel’s data fits the model only by chance) is less than 5%.

However, if we analyze 35,000 voxels (the entire brain) in the same way one by one, using $p < 0.05$ as the criterion for checking significance still ends up with 1750 false positives in theory. This can be corrected by only considering the voxels with $p < 0.05$ that are physically grouped together, since the real brain tends to activate in clusters of physically adjacent voxels.

The same statistical analysis pipeline also applies to the difference between two $\beta$ values, which quantifies which model of the compared two is more relevant to the voxel.

The analysis described above is done in a single subject, which is called first-level analysis. The same statistical principles can be used to compare the values across multiple subjects in second/group-level analysis. Classical cognitive neuroimaging discoveries such as fusiform face area (FFA) [62] and parahippocampal place area (PPA) [32] can be revealed by the univariate analysis. Specifically, we used group-level analysis of GLM to identify FFA and PPA for comparison to our work in Section 3.2.6.

**Multi-voxel pattern analysis**

Instead of fitting a predefined model voxel by voxel, MVPA tries to extract patterns from the combination of a group of voxels [88]. MVPA is essentially a pattern classification problem, that is, a pattern of brain activity is recognized to be associated
with one cognitive task versus another. Therefore, once the expression of the data is fixed (i.e. how to extract feature vectors from the data), all kinds of machine learning techniques that have been developed for classification, such as support vector machine (SVM), logistic regression, K-nearest neighbor search (KNN), all the way to the recently super popular deep learning, can be effectively applied to fMRI data analysis. Figure 1.4 shows the typical workflow of MVPA.

![Figure 1.4: MVPA workflow. The feature vectors in high dimensional space with different labels (indicating as different colors in the figure) are formed from the fMRI data according to the analysis purpose, and then applied to train a machine learning model for classification.](image)

The neuroscientific domain knowledge is leveraged in feature extraction, that is, what features are useful and how they are generated depend on what neuroscientific analysis is to be conducted. In practice, MVPA mostly applies to amplitude values of the voxel activity which present the subject’s brain state at particular time point. In order to get more stable signal, the activity values can also be calculated from the average of adjacent TRs belonging to the same cognitive task (a cognitive task typically lasts for multiple TRs). In addition, most of the time only the activity of (pre-known) relevant voxels are considered (e.g. visual cortex is largely analyzed in object recognition studies). This approach is effective in various studies [49, 66, 88]. In Chapter 2 and 3, MVPA based on the amplitude values of voxel activity is conducted as the comparison to our full correlation study.
Theoretically, in order to study the functional interactions in the human brain, MVPA can take the correlational data of multiple voxels as features to conduct pattern recognition. However, very few people actually do this over a large amount of voxels because of the huge computational burden it would bring. Instead, neuroscientists study the functional interactions in the human brain in a seed-based manner as described in the next subsection.

**Functional interaction analysis**

Neuroscientists believe that in terms of mental functions, there is some degree of localization across the brain (i.e. different brain regions take care of different mental functions). Meanwhile, brain regions interact with each other to achieve coherent mental function and behavior [41].

Functional interactions in the human brain can be depicted as the temporal correlations of voxel activities in fMRI data. The analysis of them often refers to *functional connectivity*, which essentially studies the statistical dependencies of voxel pairs, e.g. [38 106]. The typical pipeline of these analyses first identifies a small set of “seed” regions of interest (ROIs) which are known to be category selective from previous studies, or are picked based on coarse anatomical parcellations of the brain, then examines the correlation between these seeds and other voxels in the brain (Figure 1.5). These functional connectivity analyses are biased by the predefined seeds.

Before this dissertation, to the best of our knowledge, there is little previous work on exhaustive, unbiased analysis of temporal correlations of voxel activities in fMRI data.
Figure 1.5: Functional connectivity analysis of fMRI data. The brain area in light green is a predefined ROI which is a seed to correlate with other regions of the brain.

**Real-time data analysis**

Real-time fMRI data analysis is increasingly valuable for cognitive neuroscience [20, 69, 117], but still largely less developed than it desires. To date, there are many forms of real-time fMRI: “triggering” involves initiating experimental trials based on the amount or pattern of activity in a brain region, allowing for stronger inferences about the region and behavior [122]; “adaptive design” involves altering experimental parameters such as stimuli and tasks, in order to optimize the experiment to recruit particular brain regions [75]; and “neurofeedback” involves returning to subjects some visualization of the amount or pattern of activity in a brain region, helping them better engage this neural representation [105, 107].

However, in order to keep pace of the rate of data acquisition to provide real-time feedback, all of the above studies processed only a small set of voxels. In other words, the existing real-time studies are largely limited by the computing power it can leverage. Nowadays, in order to conduct real-time fMRI studies, different teams have to set up their own experimental environments in their scanning rooms, typically using a standalone machine sitting besides the fMRI scanner to receive and analyze the incoming data stream. This introduces a lot of duplicate effort in the neuroscience community. Before this dissertation, to the best of our knowledge, there
is no readily accessible, standardized tools for implementing real-time analysis on top of high performance compute clusters.

1.3 Contributions

This dissertation proposes, designs and implements full correlation matrix analysis (FCMA) to study functional interactions in the human brain in an unbiased way using a compute cluster within a tractable time. The dissertation also designs and implements a real-time system with FCMA as an analysis application. Specifically, the dissertation makes the following contributions.

Propose, design and implement full correlation matrix analysis

Currently, the approaches neuroscientists use to study the brain imaging data generated by fMRI scanners are limited and incomplete. For example, in neuroimaging fMRI studies which seek to associate different cognitive processes to different patterns of neural activities, the most popular method in the community is to apply MVPA to the amplitude of BOLD activity across a set of voxels as described in Section 1.2.3. While revealing information about neural representations, such spatial patterns detected by MVPA fail to capture interactions between brain regions over time that support neural processes. As a result, functional connectivity analysis comes in to focus on the functional interactions in the human brain. However, due to the limitation of computing power, instead of accounting for billions of pairwise relations between all measured brain voxels, such analyses are often biased by only studying the interactions based on seed regions that are identified by MVPA on the amplitude of BOLD activity or brain parcellations to reduce the computational challenge. These assumptions bias the neuroscientific studies and restrict neuroscientists from well understanding functional interactions in the human brain across cognitive tasks.
With the advance of parallel computing and machine learning techniques and
the advent of more powerful computing resources, this dissertation surmounts the
aforementioned limitations by making the exhaustive computation and analysis of
functional interactions in the human brain possible. In the dissertation, we propose
full correlation matrix analysis (FCMA) as the first attempt in the neuroscience
community to exhaustively study the functional interactions in the human brain in an
unbiased way by applying multivariate analyses to patterns of correlation rather than
the amplitude of BOLD activity, and extracting correlations in an unbiased manner
over the whole brain rather than among seeds. We introduce the processing pipeline
of FCMA which runs on a multi-processor compute cluster to efficiently analyze the
pairwise correlations of all voxels in the brain by leveraging and optimizing algorithms
from parallel computing and machine learning, which improves FCMA computation
by three orders of magnitude. We make FCMA open source to enable the exhaustive
brain correlation analysis in the neuroscience community.

**Conduct a case study of FCMA and make new scientific discovery**

This dissertation then conducts a case study to show the effectiveness of FCMA. We
demonstrate that FCMA is able to unveil previously vague functional interactions in
the human brain even during very simple cognitive tasks such as showing participants
either face or scene images when lying down in the fMRI scanner. Specifically, by
applying FCMA to a simple face-scene localizer dataset, in addition to the commonly
known visual cortex, *medial prefrontal cortex (mPFC)* is also identified to be category-
selective in terms of its correlation with the whole brain, which is not revealed in
traditional MVPA. We perform various analyses to verify this finding as well as depict
the connectivity characteristics of mPFC with the visual cortex.

Our proof of concept suggests a broader research program to examine how patterns
of voxel correlation, but not just individual voxel amplitude activity, may advance
our understanding of human brain function. Our various approaches to analyzing the FCMA-identified ROIs such as information mapping, vector change analysis, background connectivity analysis, etc. are generalizable to a lot of other datasets after applying FCMA.

**Optimize FCMA for a cluster of many-core processors**

Our basic optimization of FCMA works reasonably fast to finish the analysis on a typical fMRI dataset in tens of minutes using a compute cluster. However, there is room to further optimize the code for achieving better performance. In this dissertation, we focus on the optimization of the most computationally intensive part of FCMA, the voxel selection, on the emerging many-core processors such as Intel® Xeon Phi™ coprocessors. Although working on similar programming paradigm, compared to multi-core processors, many-core processors have much smaller on board memory and cache, wider vector processing units (VPUs), and support much larger number of threads running simultaneously. According to these architecture characteristics, we come up with several optimization ideas which are generalizable to other similar applications, and apply them to our code.

We show that by carefully designing the data structure and workflow of our code to be cache and vectorization friendly as well as supportive to many threads running together, voxel selection of FCMA can achieve 5x-16x speedup on real fMRI datasets compared to our basic optimization with commonly used MKL and LibSVM libraries on the many-core processors. We also show that our optimizations for the many-core processors yield a faster implementation on multi-core processors because the optimization fully utilizes the computational resources.

Our optimizations make the analysis of FCMA on a typical fMRI dataset possible in seconds to minutes, as well as enable the possibility to do previously intractable FCMA in real-time.
Design and implement a real-time FCMA system

Real-time fMRI study, which generates the neurofeedback in the pace of data acquisition, is in a growing need in the neuroscience community since it allows a lot of interactive experimental design by leveraging the on-the-fly neurofeedback from the subject. There are very few real-time fMRI studies for now, and none of the existing ones analyze the fMRI data in full correlation fashion due to lack of computing power and a feasible real-time experiment framework.

This dissertation designs and implements a distributed real-time fMRI system which runs FCMA on a compute cluster in the back end. The system can process and analyze brain volumes from multiple fMRI experiments on a 40-node compute cluster simultaneously and send the neurofeedback to each fMRI scanner over the Internet within 1.5 seconds. We also modify our optimized FCMA voxel selection algorithm in the incremental fashion to deal with the incoming fMRI data on the fly efficiently. The incremental voxel selection algorithm running on our real-time fMRI system performs 1.8x-4.3x faster than using the offline version of FCMA in the real-time context. In addition, our system proceeds correctly in the presence of node failures. Finally, our real-time system is constructed in Software as a Service (SaaS) mode, which is potentially able to run on the cloud to provide real-time fMRI experiment service to neuroscientists from MRI centers around the world.

The materials of Chapter 2-4 have been used in the following publications:


[Chapter 4] Yida Wang, Michael J. Anderson, Jonathan D. Cohen, Alexander Heinecke, Kai Li, Nadathur Satish, Narayanan Sundaram, Nicholas B. Turk-
Chapter 2

Full Correlation Matrix Analysis

2.1 Introduction

Until recently, one of the mainstreams of fMRI studies seeks to associate cognitive processes with brain regions. A classic finding of this type is the category selectivity of occipital and temporal cortex during visual perception, with punctuate subregions responsive to objects from particular categories [13, 27, 61]. However, this univariate, sometimes phrenological approach has been challenged by multivariate (multi-voxel) pattern analysis (MVPA) methods [66, 88]. In the case of object perception, MVPA demonstrates that category selectivity is not restricted to specific areas, but is also reflected in patterns of activity throughout occipital and temporal cortex [49]. Such findings have been offered as evidence that representations in the brain can be distributed over multiple areas. In this dissertation, we extend this view, suggesting that the neural mechanisms supporting cognitive processing may be distributed over the entire brain, outside of traditional domain-specific systems. It may have been difficult to fully recognize this with existing analysis methods because of two implicit biases in such approaches.
First, the basic currency of MVPA is typically the amplitude of blood oxygenation level-dependent (BOLD) activity across a set of voxels. Although such spatial patterns can reveal information about neural representations, they may fail to capture interactions between brain regions over time that support neural processes. Even if neurons show identical average firing rates across behavioral tasks, when and how they interact with each other can distinguish between tasks [21, 22, 97, 112]. In fMRI, the BOLD activity of a voxel may fluctuate synchronously with certain other voxels in one task and with different voxels in another task, without the time-locked average amplitude of activity in any of these voxels reliably differing between the tasks or from baseline. A brain region X, for example, which contributes to object recognition and contains voxels that behave in this way, may show either no evoked response or similar responses to all object categories, and therefore would not be identified as category-selective by standard analyses that contrast or classify patterns of activity. If region X’s interactions with other regions differ by category, however, it could show category-selective patterns of correlation.

Second, although the analysis of correlations in fMRI data, or functional connectivity [38, 106], has become prevalent, this approach is subject to certain limitations. Such analyses often involve first identifying a small set of seed regions of interest that show task-related activity and then examining correlations between these seeds and other voxels in the brain. When seed correlations are calculated during tasks, efforts are made to eliminate the spurious correlations induced by evoked responses, such as by modeling these responses and the task-related seed timecourse simultaneously (as in psychophysiological interaction analysis [42]) or by removing these responses prior to estimating correlations (as in background connectivity analysis [4]). Regardless, by choosing seeds based on activity, these procedures are affected by the limitations of activity-based methods highlighted above and would fail to identify region X (since its activity is not category-selective). Some studies have attempted to consider func-
tional connectivity more broadly by using MVPA to classify patterns of correlation amongst multiple regions both at rest (e.g., [31]) and during tasks (e.g., [85, 92, 116]). However, in these studies, only a small number of regions were selected relative to the total number of voxels in the brain, and this selection was based on activity in the same or other studies, or on coarse anatomical parcellations of the brain. To avoid any selection bias and consider the full range of data when identifying task-related interactions, the correlation of every voxel with every other voxel may be needed at the outset.

In this chapter we describe the technical details of a method, full correlation matrix analysis (FCMA), that surmounts these limitations by performing unbiased multivariate analyses of whole-brain functional connectivity. The basic currency of FCMA is the full correlation matrix: the temporal correlation in BOLD activity of every voxel in the brain with every other voxel. A separate matrix is computed for each temporal epoch of interest in a task, such as trial or block, just as would be done for mean activity or activity patterns. These matrices are then labeled with the task condition for that epoch and submitted to MVPA. Critically, the input to the classifier is now correlation patterns, rather than activity patterns. This analysis determines, in an exhaustive manner, which correlations distinguish between conditions. In principle, region X above can therefore be found because it has differential interactions as a function of category, which will be verified in the proof of concept detailed in Chapter 3.

We use Pearson correlation and thus assume that the relationship between activity in one region and another is linear—a common assumption in fMRI studies of functional connectivity. It remains likely that meaningful interactions between regions are non-linear, and thus future development of FCMA will incorporate other more flexible relationship metrics [100].
2.2 Challenges

Conceptually, exhausting all possible voxel pairs of the brain may sound straightforward, but it is currently computationally intractable using a naive approach. Consider a small fMRI dataset (called face-scene henceforth) that we will use in the subsequent chapters for proof of concept and performance evaluation, which contains 18 subjects, 12 epochs per subject over two experimental conditions, and 34,470 voxels in the brain. The first, and easiest, part of FCMA is calculating the pairwise voxel autocorrelation matrix of every epoch. On a machine with two 4-core 2.6 GHz Xeon CPUs, running Matlab’s corr function in batch mode, the computation of all 216 matrices (each with more than 594 million unique entries) takes 2.5 hours and requires 478 GB of disk space at the end (and much more memory at intermediate stages). Rewriting the Pearson correlation computation in C++ using matrix multiplication and optimized linear algebra packages shortens running time to 348 seconds on the same machine. Thus, computing correlation matrices from fMRI data per se is not the hard problem and, in fact, there are already efficient tools, such as InstaCorr in AFNI [25], that have been used for this purpose [47].

The more challenging problem arises when this massive amount of data needs to be analyzed. Typical seed-based functional connectivity maps are 3D, reflecting the correlation of one seed voxel with all other voxels in the brain. Such data can be analyzed in a voxel-wise manner by examining which voxels have correlations with the seed that are reliably positive or negative, or that differ between conditions, using simple t-tests over subjects. The full correlation matrix, on the other hand, can be thought of as 6D, reflecting one of these 3D maps for every voxel in the 3D brain. The sample dataset above produces 216 of these 6D matrices, each with correlation values for more than 594 million unique voxel pairs. That is, there are 4-5 orders of magnitude more variables to analyze, depending on the number of voxels. At that scale, algorithmic optimization and parallelization are necessary for the analysis to be
tractable on current hardware, and machine learning techniques are needed to make sense of the data.

In particular, the set of full correlation matrices can be mined with MVPA to identify which pairs, and combinations of pairs, reliably discriminate between experimental conditions across subjects. To do so, the correlations are first preprocessed, including normalizing each coefficient with the Fisher transformation and then z-scoring all coefficients across the matrix within subject. To classify two conditions, these correlation matrices are then divided into training and classification sets, such as by leaving out one subject at a time, which enables random-effects cross-validation. Using this approach on the dataset above, the classifier would be trained on 204 matrices to find a boundary separating the conditions in a 594 million dimensional hyperspace and classified on the remaining 12 matrices to obtain a classification accuracy. With the data-driven feature selection approach described below, training and classification a basic linear classifier (e.g., linear support vector machine, SVM) on the correlation matrices in C++ would take 36 days on the machine described above. For all 18 subjects, this process must be repeated 17 more times for a total of 646 days.

We designed FCMA to address these challenges, by incorporating and refining algorithms for parallel computing and machine learning. Applying this method on a compute cluster, the time required for the analysis above was reduced from two years to one hour. In the next subsection, we describe the technical details of implementing FCMA.
2.3 Implementation

2.3.1 Overview

FCMA incorporates efficient algorithms for correlation computation, massive parallelization to manage the scale of the data, and on-demand analysis without the need for storage space or time. It runs on a compute cluster with any number of nodes, using a master-worker model, and takes standard NIfTI-formatted (NifTI stands for neuroimaging informatics technology initiative) images as input. A dataset consisting of fMRI activity over time in a set of voxels is divided in multiple temporal epochs and stored on a disk accessible to each node. These (relatively small) data are copied into the memory of each worker and the master dynamically allocates a subset of voxels for computation when a node is available. A vector is computed for every epoch of each voxel using matrix multiplication, reflecting its correlation with every other voxel in that epoch. To handle the high dimensionality of these vectors without imposing biases, they are submitted to MVPA and the resulting cross-validation accuracy is assigned to the voxel. These accuracy scores can be used for nested feature selection, leading to a final round of MVPA over the correlation matrices of the automatically selected voxels.

2.3.2 Optimization of Correlation Computation

The Pearson correlation between several pairs of variables can be reduced to a matrix multiplication by normalizing the data. Specifically, the mean activity is first subtracted from each timecourse, and this mean-centered timecourse is then divided by its root sum of squares. The Pearson correlation between two of the resulting normalized timecourses is their pointwise product, which for an arbitrary number of timecourses becomes a dot product. A derivation of this is provided below.

[Link to NIfTI website](http://nifti.nimh.nih.gov/)
The formula for the population Pearson product moment correlation is:

\[ corr(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} \quad (2.1) \]

This can be re-written based on estimates of covariance and standard deviations:

\[
corr(X, Y) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{(n - 1) \sqrt{\frac{\sum_{j=1}^{n} x_j^2 - n\bar{x}^2}{n-1}} \sqrt{\frac{\sum_{j=1}^{n} y_j^2 - n\bar{y}^2}{n-1}}} \quad (2.2)
\]

\[
= \sum_{i=1}^{n} \left( \frac{x_i - \bar{x}}{\sqrt{\sum_{j=1}^{n} x_j^2 - n\bar{x}^2}} \right) \left( \frac{y_i - \bar{y}}{\sqrt{\sum_{j=1}^{n} y_j^2 - n\bar{y}^2}} \right)
\]

Applying the mean subtraction and root sum-of-squares division, let:

\[
x_i' = \frac{(x_i - \bar{x})}{\sqrt{\sum_{j=1}^{n} x_j^2 - n\bar{x}^2}} \quad (2.3)
\]

The formula above then becomes:

\[
corr(X, Y) = X' \cdot Y' \quad (2.4)
\]

This can be expanded from vectors to matrices:

\[
X' = (x'_1, x'_2, \cdots, x'_n) \quad (2.5)
\]

and

\[
A = (X'_1, X'_2, \cdots, X'_n)^T \quad (2.6)
\]
Then the correlation matrix, where every voxel is represented by a row and column, is given by:

\[ C = AA^T \] (2.7)

To sum up, the timeseries for a given voxel and epoch is normalized by subtracting the mean and then dividing each value by the root sum of squares of the mean-centered data. Apparently, the number of these operations scales linearly with the number of voxels, and thus computation poses little burden.

Matrix multiplication is significantly faster than other approaches for computing correlations. In addition, it can benefit from generally available technological advances in modern CPUs, such as the single instruction, multiple data (SIMD) set. FCMA implements advanced linear algebra algorithms from the GotoBlas library [46] and/or Intel math kernel library (MKL) to exploit the hardware. At peak performance, with SSE instructions over 128-bit XMM registers, these algorithms allow for 8 single-precision floating-point operations (4 additions and 4 multiplications) in one CPU cycle.

2.3.3 Parallelization of Correlation Computation

FCMA is designed to run on a compute cluster with modern commercial machines and X86 architecture; each machine needs a reasonable amount of memory (e.g., 16 GB). Large-scale computing techniques are leveraged to accelerate computation and analysis. A master-worker model is used as shown in Figure 2.1, in which a master process coordinates numerous worker processes running on multiple machines with the Message Passing Interface (MPI), which will be replaced with more advanced and reliable communication in Chapter 5. The master allocates computation and analysis tasks to the workers; typically, one process is assigned to each node in order to fully

utilize its resources. Each process consists of multiple threads to compute and analyze multiple voxels simultaneously within one node.

Figure 2.1: Workflow overview. FCMA uses a master-worker model, in which each worker first loads the full data into memory. The full data consist of a matrix with $V$ voxels in rows and $T$ timepoints in columns; the timepoints can be subdivided into $E$ epochs, each with $T_E$ timepoints (inset depicts two voxels and epochs). The master process does the following: assigns a subset $S$ of voxels to each of $W$ workers; instructs the worker to compute the correlation between each of these voxels and the rest of the brain in each epoch; instructs the worker to analyze the correlation vectors for each voxel across epochs with MVPA and supplied condition labels; collects the analysis result (i.e., cross-validation accuracy) for each voxel and loads it into memory; and returns to the first step to assign another subset of voxels until there are none left. Finally, the master writes the results to disk.

The first step of FCMA is to read in preprocessed fMRI data (e.g., corrected for head motion and other sources of noise) and text files specifying the experimental design. Based on this design, the data are partitioned into epochs. For every voxel and epoch, the data are normalized as described above. The master then broadcasts all available worker processes the full data and dynamically assigns each a subset of voxels to analyze. In other words, the full correlation matrix is automatically divided
into groups of rows, and they are spread across worker processes. When the analysis in a worker finishes, the master collects the results, stores them in memory, and assigns another group of voxels.

By distributing the full correlation computation in this way over a 66-node cluster (spec in Section 2.4), and by using data normalization and optimized matrix multiplication algorithms in GotoBlas, total correlation computation time for the face-scene dataset is reduced from 2.5 hours to 0.73 seconds.

2.3.4 Parallelization of Voxel-wise Classifier Analysis

To avoid the burden of storing full correlation matrices to disk (and associated write/read time), analysis is performed online within the nodes, immediately after correlation computation. After a worker process has computed a correlation matrix of the assigned subset of voxels with the rest of the voxels for each epoch, the same row of all matrices is extracted. Each of these rows comprises a vector of the correlations between a given voxel and all other voxels in the brain for one epoch. These correlation vectors are then labeled with the condition of the experimental design to which the epoch corresponds, normalized via Fisher transformation and z-scoring within subject (detailed in Chapter 4 when we discuss the further optimization of these computations), and submitted to MVPA as training (or classification) patterns. Each vector reflects a point in a high-dimensional space, and the goal of MVPA is to determine how accurately the points with different labels can be separated. MVPA is run with multiple threads; each thread processes the correlation vectors across different epochs for one voxel.

The default FCMA configuration uses a linear Support Vector Machine (SVM) classifier based on LibSVM. On a server with two 4-core 2.6GHz Xeon CPUs, the standard version of LibSVM in serial mode takes about 90 seconds to perform cross-validation on a set of 204 correlation vectors from one voxel (each of length
34,470) in the face-scene dataset. To speed up this process, we accelerate the original LibSVM algorithm by pre-computing the linear kernel matrices with GotoBlas. This reduces the running time to 2 seconds per voxel in a single thread.

### 2.3.5 Classification of Correlation Patterns

Applying the classifier to the correlation vectors produces a cross-validation accuracy for each voxel, reflecting how informative its correlations with the rest of the brain are about the task conditions. FCMA can stop here, providing an unbiased, voxel-wise “map” of the extent to which every brain region show task-related changes in whole-brain functional connectivity.

The results of the voxel-wise classifier analysis described above can also be used as feature selection for MVPA, by ranking voxels according to the accuracy with which their individual patterns of correlation differentiate conditions and then selecting a subset that exceeds some threshold (e.g., percentage of voxels, absolute accuracy level, statistical significance over cross-validation folds, etc.). FCMA implements this step as well: A chosen subset of voxels can be submitted again to obtain and analyze with MVPA the correlation matrix for only these voxels. This again requires that the original data be partitioned into training and classification sets, with the feature selection step only applied to the training sets and the final classification accuracy derived from the held-out classification sets. Using FCMA on our cluster, the total running time can be reduced from 646 days to 72 min.

The overall procedure of FCMA, including voxel selection and classification based on selected voxels, is shown in Figure 2.2. Notice that in addition to FCMA procedure, Figure 2.2 also illustrates the a similar procedure with the only difference being processing based on the amplitude of BOLD activity of voxels instead of based on the correlation between voxel pairs. We will described this procedure as the comparison to our FCMA procedure in detail in the case study in Chapter 3.
Figure 2.2: Classification procedure. (a) The preprocessed fMRI data set contains $n$ subjects, each represented with a $k$ voxels by $t$ epochs matrix. (b) For standard MVPA of activity patterns, vectors are defined for each subject and epoch as the average BOLD signal over time in every voxel ($\mu_i$). For MVPA of correlation patterns, vectors are defined for each subject and epoch as the pairwise correlation of the BOLD signal over time between every voxel and every other voxel ($r_{i,j}$). (c) The same nested cross-validation pipeline can be applied to activity and correlation patterns. The inner loop serves to select features (voxels) for classification: A training set ($S_i$) is divided into $m$ pieces to do an $m$-fold cross-validation that identifies the voxels with highest performance. (d) The outer loop is $n$-fold, with each fold leveraging the selected voxels to train a model on $S_i$ and classification it on the left-out classification set ($T_i$). This results in a classification accuracy ($P_i$), which is then averaged across folds ($P$) to quantify overall performance.
2.4 Performance Benchmarking

We benchmark the performance of our implemented FCMA described above by running analysis on the aforementioned face-scene dataset on a 66-node compute cluster. Each node was equipped with two Intel® Xeon® E5430 processors running at a 2.66 GHz clock rate, 16 GB memory, and 4 TB local disk, and could run 8 threads simultaneously at peak. The nodes are inter-connected by 1G Ethernet.

As described in Section 2.3, FCMA accelerates the unbiased, exhaustive analysis of functional interactions in the human brain by optimizing correlation and machine learning algorithms and by leveraging multi-thread parallelism within nodes and multi-node parallelism across nodes. For each step of optimization, the performance of FCMA when applied to the example dataset with the sample hardware is provided in Table 2.1.

FCMA has been ported to other systems as well, including a 128-node cluster at the Princeton Institute for Computational Science and Engineering (PICSciE) and a dedicated 50-node cluster called metacortex in Princeton Neuroscience Institute (PNI), featuring two Intel® Xeon® E5-2670 processors and two Intel® Xeon Phi™ cards per node (total of 800 processor cores and 6,000 coprocessor cores), which is be heavily used in Chapter 4 when further optimizing the FCMA code and in Chapter 5 for building the real-time fMRI system.

2.5 FCMA Toolbox

2.5.1 Code Availability

In order to benefit the entire neuroscience community, we implement the functionality we described above as the FCMA toolbox which is open sourced in GitHub at [http://princetonuniversity.github.io/fcma-toolbox](http://princetonuniversity.github.io/fcma-toolbox). The source code is written in
<table>
<thead>
<tr>
<th>Optimization</th>
<th>One voxel, one classification subject (34,470 correlations)</th>
<th>All voxels, one classification subject (34,470 voxels)</th>
<th>All voxels, all classification subjects (18 subjects)</th>
<th>Performance speedup (x Baseline)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>1.5</td>
<td>51,705</td>
<td>930,528</td>
<td>-</td>
</tr>
<tr>
<td>Improved linear SVM</td>
<td>0.03</td>
<td>1174</td>
<td>21,138</td>
<td>44</td>
</tr>
<tr>
<td>Multi-core parallelism (8 cores)</td>
<td>0.03*</td>
<td>201</td>
<td>3618</td>
<td>257</td>
</tr>
<tr>
<td>Multi-node parallelism (66 nodes)</td>
<td>0.03*</td>
<td>4</td>
<td>72.3</td>
<td>12,855</td>
</tr>
</tbody>
</table>

Table 2.1: Benchmark times for different optimizations in minutes. Each time refers to the training and classification of an SVM classifier model, in which one subject is held out for the final classification, and cross-validation is performed among the remaining training subjects to select voxels whose correlations are used for the final classification. The baseline code was written in C++ using GotoBLAS library and LibSVM in single-thread mode. Improved linear SVM code pre-computed the linear kernel matrices used in LibSVM. Multi-core parallelism code used OpenMP to launch 8 shared memory threads in one cluster node to run the program in parallel. Multi-node parallelism code runs MPI to coordinate 66 cluster nodes to work together in controller-worker mode. Long elapsed times were estimated via extrapolation.

*Note that the benefit of parallelism is only realized when multiple voxels correlation matrices are analyzed conjointly and can take advantage of multi-threading; a single thread is recruited for analyses restricted to one voxel.

C++ and requires dependencies including OpenMP, MPI, and GotoBlas/Intel MKL packages. A compute cluster is recommended for running the toolbox, although it can also be run on a standalone workstation. The current stable version is 1.7 and it is being actively updated.

Currently FCMA toolbox works in command-line interface (CLI). We also provide some scripts to facilitate the usage. The template scripts as well as an example dataset (face-scene) is available at [http://pni.princeton.edu/pni-software-tools/fcma-toolbox](http://pni.princeton.edu/pni-software-tools/fcma-toolbox).
2.5.2 Other Functionality

In addition to the main functionality such as correlation-based full brain voxel selection, training and classification (illustrated in Figure 2.2), FCMA toolbox also provides the following features:

- A similar processing pipeline as what we described above but using patterns of activity as the features. The patterns of activity, representing the individual amplitude of neural activity, can be used to do the full brain voxel selection, training and classification as well. It is essentially the traditional MVPA approach. We use it as the comparison in our case study in Chapter 3.

- Summation of the correlation coefficients voxel by voxel. For each voxel, the toolbox computes its correlation with all other voxels in the brain to form a correlation vector, and collapse the vector by summing up the absolute values of its elements together. The summation of the correlation coefficients potentially reveals which voxels strongly correlate with the full brain during a particular epoch.

- Outputting the correlation coefficients in 4D NIfTI format file for visualization. Specifically, if we have $N$ voxels’ correlation with the full brain to be visualized, the toolbox can generate an N-TR 4D brain file. Each “TR” of this file shows a 3D brain containing the correlation between one voxel and the full brain. This 4D NIfTI format file can be easily visualized by FSLView[^3]. The visualization offers a straightforward view to show which parts of the brain have significant correlation with the investigated voxels.

The toolbox is still under active development to continue implementing new requirements from neuroscientists.

[^3]: [http://fsl.fmrib.ox.ac.uk/fsl/fslview/](http://fsl.fmrib.ox.ac.uk/fsl/fslview/)
2.5.3 Generalizability

Though applied to voxel pairs in fMRI, the techniques FCMA toolbox provide could help uncover relationships between variables across a range of applications [100]. Even within cognitive neuroscience, our case study involved the autocorrelation of one dataset, but two or more datasets (e.g., from different phase offsets, regions, or even brains) could be submitted for unbiased analysis of cross-correlations. Moreover, although developed to work efficiently at the scale individual voxels, FCMA also works with any larger spatial unit of measurement that reduces the problem size, such as anatomical [115] or functional [98] parcellations. Finally, different classifier algorithms could be specified, and any division of the data into training and test sets is possible (e.g., within rather than between subjects). We believe FCMA toolbox provides a high-performance platform for applying machine learning techniques to correlational data. By releasing our code, we hope that researchers will explore these and other avenues, to help further evaluate the utility of this approach.

2.6 Related Work

There are existing commonly used fMRI data analysis tools such as FSL (FMRIB Software Library) [58], SPM (Statistical Parametric Mapping) [40], AFNI (Analysis of Functional NeuroImages) [24], and BrainVoyager [45]. In terms of data analysis, these tools are mainly for model-based, statistical analyses. They are not designed for model-free analysis such as MVPA, neither based on patterns of activity nor more computationally intensive patterns of correlation.

Princeton multi-voxel pattern analysis (MVPA) toolbox [30] is an open source software written in Matlab to provide the functionality for various kinds of multivariate analysis of fMRI data, especially those employing pattern classification methods and regression. However, since Matlab code typically does not run in high perfor-
mance, this toolbox can only deal with patterns of activity which requires relatively light computing power. Analyzing the patterns of (full) correlation is intractable to Princeton MVPA toolbox and a lot of other similar fMRI data analysis tools in Matlab.

The backbone of FCMA toolbox is correlation computation. Being a classic problem, there are lots of tools that support computing the correlation between different vectors. For example, Matlab has a build-in function `corr` to do this in one line of code. However, when the data scale grows to the full brain, i.e. computing the correlation between every voxel and every other voxel in the brain in a given epoch, it takes hours for the `corr` function of Matlab to finish, which is not acceptable especially if tens or hundreds of epochs’ correlation values need to be computed. Some other existing tools, such as InstaCorr in AFNI [25], on the other hand, are able to solve the correlation computation in a more efficient way. But when computing the full brain correlation, these tools can only generate a huge amount of correlation coefficients to the memory without the efficient and effective use of them.

### 2.7 Summary

To the best of our knowledge, FCMA is the first toolbox in the neuroscience community that takes advantage of high performance computing to efficiently analyze the full correlation matrix of fMRI data. In summary, FCMA takes advantage of high-performance computing to efficiently analyze the full correlation matrix of fMRI data. It deals gracefully with enormous amounts data, optimally splitting and scheduling problems based on the latest techniques in large-scale computing and minimizing the need for slow data transfer by managing memory intelligently during online analysis. Beyond the flexible parallelization, FCMA accelerates analysis by improving standard algorithms for correlation and classification. FCMA can be run on a compute cluster.
with any number of nodes, limited only by the computation resources of the hardware. In the next chapter, we will show some new neuroscientific findings obtained by applying FCMA.
Chapter 3

A Case Study of FCMA

3.1 Introduction

In this chapter, we perform a case study to show the effectiveness of FCMA. The main goal of FCMA is to identify the brain ROIs which show distinguished patterns of correlation with the full brain in different cognitive tasks (conditions). In some cases, the correlational ROIs do not necessarily overlap with the ROIs identified via traditional model-based approaches and/or MVPA based on amplitude of voxel activity. As discussed in Section 2.1, consider a region that is always engaged (or seems to be not engaged) during all conducted cognitive tasks, but whose interactions with other regions depend upon the category. This region might only be identified as category selective based on its patterns of correlation with other brain regions. However, standard analyses that contrast or classify patterns of activity would not be able to identify this kind of regions since they by nature do not show activity difference across tasks. Typical approaches for assessing correlations in neuroimaging data could also fail to identify this kind of regions, since they require selecting seed regions based on activity differences.
We apply FCMA to a localizer dataset in which subjects viewed blocks of faces or scenes as the proof of concept. The main purpose of a localizer dataset is to localize the brain regions which are typically responsive to one of the displayed stimuli (in this case, face or scene images). Being different from this, in the following case study, FCMA tries to distinguish face and scene blocks in terms of correlation. Although the cognitive tasks (faces vs. scenes) are already well-studied, e.g. GLM and traditional MVPA approaches have revealed areas of occipital and temporal cortex that are activated by objects of particular categories [27, 49, 61], here we take a fresh look at category selectivity by applying multivariate analyses to patterns of correlation rather than activity, and extracting correlations in an unbiased manner over the full brain rather than among seeds, which is not typically analyzed before. To examine whether FCMA can provide added value, we test whether it emphasizes other, less-traditional brain areas as being category-selective in terms of their patterns of correlation. Our hope is that releasing the software toolbox will prompt additional applications of FCMA, but this example illustrates the different steps involved and the variety of results that can be obtained.

3.2 Case Study

3.2.1 Dataset

We used fMRI data from 18 subjects (localizer data from [110]), who viewed sequences of face and scene images in alternating blocks, while performing male/female and indoor/outdoor judgment tasks, respectively. Each subject completed one functional run that contained 12 blocks (6 involving faces and 6 involving scenes). Each block contained 12 images from one category and lasted for 18 seconds, followed by blank fixation period of 12 seconds between blocks. In the context of FCMA, we treated each block to be an epoch of interest over which the correlation will be computed.
Data were acquired using a Siemens 3T Allegra scanner, with a T2*-weighted EPI sequence: volumes = 224, slices = 26, TR = 1500 ms, TE = 28 ms, matrix = 64, field of view = 224 mm, flip angle = 64°, and thickness = 5 mm (3.5 x 3.5 x 5 mm voxels). Data were preprocessed prior to FCMA with corrections for head motion and slice-time acquisition, spatial smoothing (5 mm), and high-pass temporal filtering (128-second period). Brains were anatomically aligned to MNI space using standard linear registration methods and masked to remove non-brain voxels. After alignment, every brain contained 34,470 voxels.

3.2.2 Data Preparation

Whole-brain activity and correlation patterns were calculated for each block. For activity patterns, the BOLD signal in each voxel was averaged over the 12 timepoints in that block, shifted forward by two TRs to account for the hemodynamic lag and capture peak response amplitude. We then normalized within subject by z-scoring these values across blocks within each voxel. The analysis on patterns of activity provides a comparison to the results revealed by patterns of correlation via FCMA.

For correlation patterns, the BOLD signal in all pairs of voxels was correlated over the timecourse of each block. In pilot testing, variance during the rise of the hemodynamic response meaningfully contributed to correlations and thus the timecourses were not shifted prior to being correlated. Although a sample size of 12 timepoints for computing correlations is small and leads to volatile estimates, we reasoned that the classifier would be able to extract meaningful correlation patterns across training examples. Any concerns about statistical power worked against finding reliable classification performance. Additional preprocessing steps (mean-centering, division by root sum of squares) were applied to each timecourse to optimize correlation computation. The resulting coefficients were Fisher-transformed and normalized within subject by z-scoring across blocks within each voxel pair.
3.2.3 Classification

Method

As shown in Figure 2.2, the SVM algorithm was applied to activity and correlation patterns with a leave-one-subject-out nested n-fold cross-validation approach (where \( n = 18 \) subjects in our studying dataset). An inner loop was used for feature (voxel) selection on the training set, while an outer loop used the chosen voxels to train a final classifier on the training set for cross-validation against the test set. The goal of feature selection was to automatically identify voxels whose activity/correlation patterns distinguish between classes, as well as to reduce the dimensionality of final classifier training. For each training set from the outer loop, we did an inner leave-one-subject-out \( m \)-fold cross validation for this feature selection (where \( m = \) the remaining 17 subjects). Note that leaving one subject out at a time is not a requirement of FCMA itself, which can be applied just as easily across trials or blocks within subject. This approach is convenient, however, as the resulting sample of cross-validation accuracies (one per subject) can be analyzed statistically treating subject as a random effect.

The feature selection for activity and correlation patterns was similar, but differed due to the nature of the data. For activity patterns, we used a searchlight approach [65]. For every block in the inner-loop training set (\( m - 1 \) or 16 subjects \( \times \) 12 blocks/subject = 192 blocks), we generated a spherical pattern of activity surrounding each voxel, made up of the average activity over that block for each voxel in the sphere. We then trained a linear SVM classifier to distinguish these patterns as having been obtained from face vs. scene blocks. The cross-validation classification performance on the \( m^{th} \) (17th) subject was assigned to this voxel, and the process was repeated for all 34,470 voxels. For correlation patterns, the procedure was very similar except in terms of how the block patterns for each voxel were generated. Rather than
the average pattern of activity of the surrounding voxels, we used the pairwise correlation of a given voxel with all other voxels, resulting in a vector of 34,470 coefficients for that voxel for every block. These vectors were then submitted to the same kind of classifier to distinguish face vs. scene blocks, and the classification performance was assigned to that voxel. The process was repeated for all voxels.

This entire process was repeated $m$ times for the activity searchlights and correlation vectors for each voxel, with final voxel-wise classifier performance computed as the average performance across folds. The $k$ most informative voxels were chosen from these voxel-wise average classifier performance maps for each type of classification, to be used for the outer-loop training and classification. Specifically, these $k$ voxels were used to generate patterns of activity and correlation for the full set of $m$ subjects in the training set of the current outer-loop fold. Activity patterns reflected the average activity for each block from all selected voxels (i.e., vector of length $k$). Correlation patterns consisted of a correlation matrix for every block, with the selected voxels in the rows and columns (rearranged into a vector of length $k^2$). These activity and correlation patterns were used to train a linear SVM classifier, which was then tested on the $n^{th}$ ($18^{th}$) subject. To evaluate the robustness of decoding with respect to the choice of $k$, the average outer-loop classification accuracy across folds was calculated for $k = 10, 20, 50, 100, 200, 500, 1000, 2000$ (used in main analysis), and 5000 voxels. Random-effects reliability against chance (50%) was assessed with a permutation test over the subject folds.

**Results**

The aforementioned classification was applied to the face-scene localizer dataset in which 18 subjects viewed sequences of faces or scenes in separate, recurring blocks of trials. The feature-selection procedure identified the 2000 most informative voxels because top-2000 voxels yield the best classification performance in both patterns of
activity and correlation (see Figure 3.1). These voxels were used to generate final activity and correlation patterns for each block, which were in turn used to estimate overall classification performance.

Figure 3.1: Robustness of feature selection. Classification accuracy for activity and correlation patterns, as a function of the number of voxels selected during nested cross-validation. Error bars reflect ±1 SEM over folds.

Having conducted feature selection and classifier training on \( n - 1 \) (17) subjects, we tested the ability of the classifier to predict block category in the remaining subject. For activity patterns, average cross-validation accuracy was 98\% \((p = 2.0 \times 10^{-6})\). This is consistent with previous findings [49]. For correlation patterns, average cross-validation accuracy was 95\% \((p = 7.0 \times 10^{-6})\). Although less accurate in relative terms, the absolute level was nevertheless quite high and the correlation analysis started with lower statistical power in two ways. First, the unit of measurement was a single correlation value per block, compared to the multiple repeated samples of activity per block that went into the average activity. Second, the correlation-based classifier
involved 4,000,000 inputs (2,000 selected voxels squared), compared to 2,000 inputs for the activity-based classifier, which increases the risk of overfitting. Nevertheless, for both activity and correlation classifiers, performance was robust over a range of features.

3.2.4 Information Mapping

The similarity in performance for the two classifiers raises the question: Are they identifying the same neural mechanisms? To address this issue, we examined the frequency with which each voxel was chosen during both types of feature selection. That is, we summed the number of times that each voxel was chosen across outer-loop classification folds and assessed the reliability of these counts using a non-parametric procedure that accounted for the inherent overlap in the training sets.

The problem is how to properly identify the statistically significant voxels from the top voxels generated by FCMA as well as the traditional activity-based MVPA comparison. These \( n \) sets of \( k \) top voxels were not independent from each other because any two training sets had exactly \( n - 2 \) subjects in common. Due to this dependence, binomial or parametric tests were not appropriate for assessing the statistical significance of the frequencies. Thus, we used a non-parametric method to simulate the null distribution. We independently randomized the spatial location of voxels in every subject’s preprocessed data, and then repeated the same full suite of analyses, selecting the top 2000 voxels in each of the 18 folds. Since the voxels in all subjects were shuffled, this led to a null distribution of 34,470 counts of how many times a voxel would be chosen in feature selection by chance as shown in Table 3.1. We ran this simulation separately for activity- and correlation-based analyses, and obtained nearly identical results: less than 5% of voxels had counts greater than or equal to 8/18 (i.e., \( p < .05 \)).
Simulated activity  Simulated correlation

<table>
<thead>
<tr>
<th>Frequency</th>
<th># voxels</th>
<th>p-value</th>
<th># voxels</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25212</td>
<td>1.000</td>
<td>25643</td>
<td>1.000</td>
</tr>
<tr>
<td>1</td>
<td>3420</td>
<td>0.269</td>
<td>2891</td>
<td>0.256</td>
</tr>
<tr>
<td>2</td>
<td>1580</td>
<td>0.169</td>
<td>1500</td>
<td>0.172</td>
</tr>
<tr>
<td>3</td>
<td>943</td>
<td>0.124</td>
<td>949</td>
<td>0.129</td>
</tr>
<tr>
<td>4</td>
<td>686</td>
<td>0.096</td>
<td>691</td>
<td>0.101</td>
</tr>
<tr>
<td>5</td>
<td>487</td>
<td>0.076</td>
<td>528</td>
<td>0.081</td>
</tr>
<tr>
<td>6</td>
<td>387</td>
<td>0.062</td>
<td>439</td>
<td>0.066</td>
</tr>
<tr>
<td>7</td>
<td>297</td>
<td>0.051</td>
<td>340</td>
<td>0.053</td>
</tr>
<tr>
<td>8</td>
<td>266</td>
<td>0.042</td>
<td>290</td>
<td>0.043</td>
</tr>
<tr>
<td>9</td>
<td>186</td>
<td>0.035</td>
<td>242</td>
<td>0.035</td>
</tr>
<tr>
<td>10</td>
<td>187</td>
<td>0.029</td>
<td>208</td>
<td>0.028</td>
</tr>
<tr>
<td>11</td>
<td>156</td>
<td>0.024</td>
<td>165</td>
<td>0.022</td>
</tr>
<tr>
<td>12</td>
<td>122</td>
<td>0.019</td>
<td>134</td>
<td>0.017</td>
</tr>
<tr>
<td>13</td>
<td>129</td>
<td>0.016</td>
<td>93</td>
<td>0.013</td>
</tr>
<tr>
<td>14</td>
<td>106</td>
<td>0.012</td>
<td>93</td>
<td>0.010</td>
</tr>
<tr>
<td>15</td>
<td>86</td>
<td>0.009</td>
<td>78</td>
<td>0.008</td>
</tr>
<tr>
<td>16</td>
<td>97</td>
<td>0.006</td>
<td>66</td>
<td>0.005</td>
</tr>
<tr>
<td>17</td>
<td>58</td>
<td>0.004</td>
<td>54</td>
<td>0.003</td>
</tr>
<tr>
<td>18</td>
<td>65</td>
<td>0.002</td>
<td>66</td>
<td>0.002</td>
</tr>
</tbody>
</table>

Table 3.1: Randomization test of voxel frequencies. Voxel locations were scrambled for each subject, and then activity-based and correlation-based classification analyses were repeated. The cumulative probability distribution of voxel frequencies was calculated.

This procedure produced a voxel-wise map of p values based on the frequencies for activity and correlation, corrected for the dependence of the training sets. To further correct for multiple comparisons across voxels, we used cluster-size thresholding [37]. The initial cluster-forming threshold for individual voxels was set at a frequency of 8/18 (p < .05 uncorrected). For each of 10,000 iterations, the true distribution of frequencies was randomized over the brain, the cluster-forming threshold was applied, and the number of clusters of each possible extent was counted. This procedure resulted in a null distribution of cluster sizes summarized in Table 3.2, with clusters of size 8 or more voxels corresponding to a whole-brain corrected threshold of p < .05. We attempted to correct for spatial autocorrelation by estimating the smoothness of
the true frequency map and applying a matched Gaussian kernel after the random-
ization step of each iteration. However, due to the strength of the results and the
relative sparseness of the frequencies, no amount of smoothing resulted in a more
conservative cluster-size threshold. Thus, we retained the cluster-size threshold of 8
voxels.

<table>
<thead>
<tr>
<th>Cluster size</th>
<th># clusters</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11551153</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>1211371</td>
<td>1.000</td>
</tr>
<tr>
<td>3</td>
<td>220917</td>
<td>1.000</td>
</tr>
<tr>
<td>4</td>
<td>46038</td>
<td>0.997</td>
</tr>
<tr>
<td>5</td>
<td>10509</td>
<td>0.755</td>
</tr>
<tr>
<td>6</td>
<td>2573</td>
<td>0.293</td>
</tr>
<tr>
<td>7</td>
<td>661</td>
<td>0.087</td>
</tr>
<tr>
<td>8</td>
<td>181</td>
<td>0.024</td>
</tr>
<tr>
<td>9</td>
<td>46</td>
<td>0.007</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
<td>0.002</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>0.001</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>&lt; 0.001</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>&lt; 0.001</td>
</tr>
</tbody>
</table>

Table 3.2: Randomization test of cluster sizes. The locations of voxel counts were
scrambled 10,000 times. This resulted in a null distribution of cluster sizes. The
probability of obtaining one or more cluster with number of voxels $\geq$ each size was
calculated over the permutations.

We produced voxel-wise maps of significance values for these counts in Figure 3.2,
which were corrected for multiple comparisons as discussed above. The results re-
vealed which areas of the brain contributed to successful classification, These areas
should be ROIs in neuroscience that is worth further studying.

Consistent with many previous findings, classification of activity patterns for faces
and scenes was supported by regions of ventral temporal cortex, as well as known
category-selective areas of dorsal occipital and retrosplenial cortex (Figure 3.2a).
Classification of correlation patterns was partly supported by overlapping regions.
Figure 3.2: (a) Activity-based analysis. Sagittal, coronal, and axial sections depicting voxels in which surrounding activity led to reliable classification of object category. These voxels were found in areas of ventral temporal, dorsal occipital, and retrosplenial cortex. (b) Correlation-based analysis. The same sections depicting voxels whose correlations with all other voxels led to reliable classification of object category. These voxels were found in areas overlapping with the activity-based analysis, but also in mPFC, early visual cortex, and precuneus. The color of each voxel reflects the frequency with which it was selected across cross-validation folds. P = posterior, A = anterior, R = right, L = left.
Additionally, however, they were joined by an area of medial prefrontal cortex (mPFC), as well as by part of early visual cortex and the precuneus (Figure 3.2).  

### 3.2.5 Vector Change Analysis

The involvement of mPFC in the classification of correlation but not activity patterns fits the description of the hypothetical region X in Section 2.1. This area may participate in category-selective object perception not in terms of its patterns of activity, but rather in terms of its patterns of correlation. To examine how the patterns of correlation changed across categories (based on the results of feature selection), we isolated significant clusters in mPFC (81 voxels) and occipital/temporal cortex (1337 “visual” voxels). For each block and subject, we computed the correlation between every mPFC voxel and every visual voxel (81 * 1337 pairs). This matrix can be represented as 81 mPFC correlation vectors of length 1337 (one for each mPFC voxel) and 1337 visual vectors of length 81 for each block (one for each visual voxel). These two sets of vectors were then averaged across blocks from the same category, resulting in 81 face mPFC vectors, 81 scene mPFC vectors, 1337 face visual vectors, and 1337 scene visual vectors.

There are two explanations for how this could arise (Figure 3.3): by mPFC interacting preferentially with different areas for different categories (“switching” hypothesis), or by mPFC interacting with the same areas to a greater or lesser degree based on category (“strength” hypothesis).

We tested these two possibilities, which are not mutually exclusive, by examining the correlations between 81 mPFC voxels and the 1337 voxels in occipital and temporal cortex that showed significant category selectivity in the correlation-based analysis. For any given block, the pattern of these mPFC correlations can be considered a point in a 1337-dimensional space, or a vector whose angle and length can be defined relative to the origin. According to the switching hypothesis, mPFC should
Figure 3.3: Analysis of mPFC correlation vectors. For each subject and category, a vector of the average correlation between mPFC and each of the 1337 voxels in occipital and temporal cortex was computed. A 2-D projection of these vectors for each subject is depicted in gray, with the average across subjects in color. Each vector pair belonging to the same subject was placed symmetrically around the vertical meridian, and the angle between them was the real angle in the high-dimensional space. The lengths were rendered proportional to the group mean.

be correlated with a subset of occipitotemporal voxels for one category and a different subset for the other category. Therefore, certain dimensions will have higher values for one category and other dimensions will have higher values for the other, which can be quantified as the difference in the angle of the vectors for each category. In contrast, the strength hypothesis predicts that the mPFC should always be correlated with the same subset of occipitotemporal voxels, just to different degrees. Therefore, the same dimensions will have higher values for one category and lower values for the other, which can be quantified as the difference in the length of the vectors for each category.

We defined these correlation vectors for every subject, block, and mPFC voxel. We then averaged them, within subject, over all mPFC voxels and over all blocks of the same category. This resulted in two correlation vectors for each subject, one
for face blocks and the other for scene blocks. From these vectors, we extracted two metrics: angle change (cosine distance) and length change (difference in L^2 norms). Since it is unknown how these metrics are distributed, we first normalized them with respect to their own null distribution. Specifically, we randomized the block labels 1000 times within each subject, calculating the cosine distance and L^2-norm difference for every voxel in each iteration. This generated null distributions against which the true values could be converted to z scores. For each subject, the 81 z values for mPFC voxels and the 1337 z values for visual voxels for each of cosine distance and L^2-norm difference were averaged within region to produce four values: average mPFC angle change, average visual angle change, average mPFC length change, and average visual length change. These values were compared across subjects with permutation tests.

In support of the switching hypothesis, there was a significant change in angle \((p = 1.6 \times 10^{-4})\) of the mPFC vectors for face and scene blocks. Although length also changed reliably \((p = .02)\), the angle change in mPFC was significantly stronger than the length change \((p = .002)\). Finally, the angle change in mPFC was greater than when the analysis was inverted, to examine the change in the angle of occipital and temporal correlation vectors with mPFC \((p = .02)\). This finding suggests that some of mPFC’s category selectivity derives from interactions with different voxels for each category. It also provides an example of how FCMA not only identifies task-related interactions between brain regions, but also permits more in-depth analysis of the nature of these interactions.

### 3.2.6 Relationship of mPFC to classic visual cortex

To further characterize the interactions between mPFC and the visual cortex which is well-studied in object perception, we tested a natural hypothesis: that mPFC correlates with face-selective regions (e.g., fusiform face area, FFA [62]) during face blocks, and with scene-selective regions (e.g., parahippocampal place area, PPA [32]).
during scene blocks. To define these areas, we applied a standard univariate GLM to the data with FSL, modeling face and scene blocks with a canonical hemodynamic response function. The resulting parameter estimates were contrasted within subject, and the reliability of these contrasts was assessed at the group level with a higher-level FLAME mixed-effects model. We identified the FFA and PPA bilaterally by finding the peak face and scene selective voxel in the group analysis within the posterior lateral fusiform gyrus and the collateral sulcus/parahippocampal gyrus, respectively. We intersected a 10-mm radius sphere around these peaks with the visual voxels obtained from FCMA, obtaining 13 FFA voxels and 89 PPA voxels.

We then examined the correlations between these voxels and mPFC for each subject, and tested reliability across subjects with permutation tests. Although these correlations were already known to differ between categories, the purpose of this analysis was to characterize the specific pattern of the correlations. Indeed, there are many patterns of correlation that could support the binary classification performed by FCMA. What we obtained was contrary to our initial hypothesis - and to what was suggested from prior findings [4, 16] - demonstrating the potential for FCMA to reveal unexpected findings.

The correlation of mPFC to FFA/PPA is shown in Figure 3.4a. Specifically, the pattern of results across subjects was different than would be expected based on the category selectivity of FFA and PPA. There was no correlation on average between mPFC and FFA during face blocks ($p = .15$) or between mPFC and PPA during scene blocks ($p = .33$). Indeed, the only reliable correlation was between mPFC and PPA during face blocks ($p = 2.8 \times 10^{-5}$), which was also stronger than the mPFC-FFA correlation in these blocks ($p = 8.3 \times 10^{-4}$). The mPFC-FFA correlation during scene blocks was not reliable ($p = .24$). More generally, mPFC was not obtained in a standard seed-based whole-brain analysis of functional connectivity with FFA and
Figure 3.4: (a) The average correlations of FFA and PPA voxels with mPFC plotted as a function of block type. (b) The average evoked BOLD activity of FFA, PPA, and mPFC voxels plotted as a function of block type. The FFA and PPA voxels were selected as face- and scene-selective, respectively, using a GLM. They were thus guaranteed to show this pattern of activity, which we display here only for visualization purposes.
PPA which will be discussed in Section 3.2.8, highlighting the potential added value of FCMA.

Considering these correlations in the context of the average activity of FFA, PPA, and mPFC as shown in Figure 3.4b, helps rule out alternative explanations for the category-selective correlations in mPFC based on evoked activity. For example, if mPFC responded to both faces and scenes, and FFA and PPA responded selectively to faces and scenes, respectively, then greater mPFC-FFA correlations for faces and mPFC-PPA correlations for scenes would be spurious and unrelated to their interactions. The actual evoked activity and correlation in these regions are inconsistent with this kind of explanation in several ways:

1. FFA showed a stronger response than PPA to faces but a weaker correlation with mPFC;

2. FFA strongly responded to faces above baseline but did not correlate with mPFC;

3. mPFC did not respond to faces above baseline but strongly correlated with PPA;

4. PPA strongly responded to scenes above baseline and mPFC strongly responded to scenes below baseline but there was no (negative) correlation between PPA and mPFC.

Moreover, when we calculated correlations over the average timecourses in Figure 3.4b (rather than the concatenated raw timeseries, as used to generate Figure 3.4a), FFA was again less correlated with mPFC than PPA in face blocks \((p = 3.1 \times 10^{-4})\), despite its greater activity.

The complex patterns of correlation between mPFC, FFA, and PPA can be visualized over all selected voxels in visual cortex using a circular graph in Figure 3.5.
One possible explanation for the overall trends in this graph is that mPFC is more focused on face information and thus sends excitatory input to visual cortex during face blocks (leading to positive correlations) and inhibitory input to visual cortex during scene blocks (leading to negative correlations). This occurred over a broad swath of visual cortex, which would be plausible if face- and scene-selective neurons were widely distributed [49]. Regardless, what determined the nature of mPFC correlations was the perceptual process being engaged rather than the activity-based identity of the target region (e.g., mPFC was strongly correlated with PPA on face blocks).

Figure 3.5: mPFC correlation patterns for face and scene blocks. Each point on the circle represents a voxel, with mPFC voxels shown in black and visual voxels arranged clockwise according to their activity-based selectivity, from face (green) to scene (purple). Links are drawn for correlations between pairs of voxels that were reliably positive (red) or negative (blue) in a one-sample t-test across subjects. The surrounding histogram reflects the frequency (8-18) with which each voxel was selected across cross-validation folds (as in Figure 3.2). Created with the Circos graphical tool [67].
This result was unexpected and will require further study. However, it at least provides a demonstration that findings from FCMA need not align with those of seed-based analyses, which in this case would have assumed that category-selective correlation would align with category-selective activity.

### 3.2.7 Background Connectivity Analysis

Although the mPFC correlation results are difficult to explain by way of the patterns of evoked responses in mPFC, FFA, and PPA, a more general concern about the contribution of evoked responses to correlation-based classification remains. In particular, a voxel that responds during both face and scene blocks would be more correlated with other face-responsive voxels during face blocks and with other scene-responsive voxels during scene blocks as shown in Figure 3.6. This voxel’s category-selective correlations would thus be spurious, reflecting shared responses to a common influence rather than intrinsic interactions.

![Figure 3.6: Illustration of a spurious correlation. Voxel A responds to both face and scene stimuli, while voxel A and B respond only to face or scene stimulus, respectively. In this case, the correlation of X to A and B is category-selective but does not necessarily relate to the intrinsic interaction between regions.](image-url)

55
If the successful correlation-based classification we observed reflects only spurious correlations, then removing such evoked responses from the data should result in chance-level decoding. We tested this possibility using background connectivity analysis [4], in which we scrubbed the data of evoked responses prior to applying FCMA and repeating the analyses.

Notice that the analyses above were applied to data that had only been preprocessed. These data contain stimulus-evoked responses (the significant rise and drop of voxel signals shown in Figure 3.6). In order to eliminate the potential confounding effects of them on task-related functional connectivity, we removed these evoked responses from the data and analyze on the residualized data, so that only the intrinsic interaction (between the fluctuation of voxel signals shown in Figure 3.6) will be captured. There were two steps to get rid of the evoked responses from the preprocessed data:

1. fitting a “nuisance” GLM to the preprocessed data with regressors of no interest for 15 nuisance variables (6 degrees of freedom for head motion, 4 seeds in bilateral anterior and posterior ventricles, 4 seeds in bilateral anterior and posterior white matter, global mean signal), as is done in analyses of resting connectivity;

2. fitting a finite impulse response (FIR) model to the residuals of the nuisance GLM with separate candlestick functions for each condition and timepoint in the block, to capture stimulus-evoked responses in a data-driven and voxel-specific manner.

After that, we applied FCMA in the same way as for the preprocessed data, but to the residuals of the FIR model.

As a check that this procedure worked, we re-ran activity-based classification, which previously produced 98% accuracy, and obtained chance performance (Fig-
ure 3.7), with no voxels reliably selected (Figure 3.8a). The critical question was whether FCMA would still be able to classify object category from correlations after evoked responses were removed. Indeed, cross-validation accuracy was still reliably above chance ($p = .002$, or 62% as shown in Figure 3.7). Moreover, overlapping mPFC and precuneus regions were obtained (Figure 3.8b). These regions were again not obtained in a seed-based analysis of the residualized data as discussed in Section 3.2.8.

![Figure 3.7](image)

**Figure 3.7:** Feature selection in background connectivity analysis. Classification accuracy for activity and correlation patterns in the residuals of FIR models that removed stimulus-evoked responses, as a function of the number of voxels selected during nested cross-validation. Error bars reflect ±1 SEM over folds.

Unlike mPFC and precuneus, early visual cortex was no longer observed. Thus, we cannot rule out the possibility that the category-selective correlations in this region were a byproduct of evoked responses. This point extends to later visual areas in ventral temporal cortex, dorsal occipital cortex, and retrosplenial cortex as well, which
were robust in the original analysis but much weaker in the background connectivity analysis. These findings indicate that parts of frontal and parietal cortices may fit the profile of regions whose correlations carry unique information, beyond what is reflected in activity.

### 3.2.8 Seed-based Analysis

To examine whether the FCMA results would be evident with a standard seed-based approach, we constructed a GLM with two regressors based on FFA and PPA activity:
1. the “selective” regressor consisted of the concatenated timeseries of activity from FFA in face blocks and PPA in scene blocks;

2. the “non-selective” regressor consisted of the concatenated timeseries of activity from PPA in face blocks and from FFA in scene blocks.

The resulting parameter estimates were contrasted (selective > non-selective), to identify voxels that correlated more strongly with category-selective regions during blocks of their preferred stimuli. Statistical significance was calculated across subjects using a randomization test and corrected for multiple comparisons with threshold-free cluster enhancement. This analysis was applied to both the preprocessed data and residualized data from the background connectivity analysis.

The results are in Figure 3.9. From the figure we see that 1) mPFC did not stand out in either the preprocessed data or the residual data via seed-based analysis; 2) seed-based analysis does not reveal any brain areas in the residual data. These results indicate that the analysis based on any given “seeds” may be biased and prevent the neuroscientists from discovering new insights, and the “seeds” identified from activity do not work on the background connectivity analysis at all.

### 3.3 Related Work and Discussion

By running FCMA on a face-scene localizeer datset, the mPFC is identified as category-selective in terms of correlation with the full brain. The mPFC has previously been implicated in the perception of social and emotional information [81, 101], including self-concept, attitudes, and mentalizing [84]. As an important social stimulus, faces have also been shown to elicit activity in mPFC [5, 68]. However, to the best our knowledge, no previous work exhausted all voxel pair correlation in an unbiased way to reveal the modulating functionality of mPFC. Although we did not obtain mPFC in our whole-brain activity-based classification, and the category
Figure 3.9: (a) Seed-based analysis of preprocessed data. Sagittal, coronal, and axial sections depicting voxels that showed greater functional connectivity with FFA (vs. PPA) during face blocks and PPA (vs. FFA) during scene blocks. (b) The same seed-based analysis applied to the residual data. P = posterior, A = anterior, R = right, L = left.
selectivity of mPFC correlations persisted despite regressing out average activity, the mPFC may play a role in modulating face processing in ways that can manifest in both local activity and long-range correlations. This is consistent with our finding that mPFC showed robust positive correlations with visual cortex (both face- and scene-preferring voxels) during face but not scene blocks.

Our case study also demonstrates that FCMA complements other approaches for the analysis of functional connectivity, helping identify task-related correlations in an unbiased manner, and providing a high-performance platform for applying machine learning techniques to correlational data. In addition to the use of FCMA, the various analysis approaches we used to prove the reliability of the new discovery FCMA made and interpret the neuroscientific meaning of the correlation-based ROIs, can be applied to a lot of other similar studies as well.

3.4 Summary

Because what you find depends on how you look, FCMA might produce findings that are less apparent with existing methods since it looks the data in an unbiased correlational way which is totally different from the existing approaches. Specifically, in this chapter we applied FCMA to a simple face-scene localizer dataset which revealed the involvement of mPFC and precuneus in category-selective object perception. After controlling for evoked responses, FCMA emphasized these regions more than the occipital and temporal regions typically linked to object perception.

In summary, as we show from the case study, FCMA enables the possibility to examine interactions between brain areas in an unbiased, exhaustive manner. Examining such interactions can reveal the engagement of neural systems transparent to traditional, activity-based analysis methods.
Chapter 4

Optimization of FCMA on Many-core Processors

4.1 Introduction

Although the basic version of FCMA implemented in Chapter 2 produced new insight to cognitive neuroscience in Chapter 3 by running on a localizer dataset, it is still not sufficient for neuroscientists for two reasons. First, the basic FCMA finishes the analysis on a small (244 TRs per subject), low resolution (34,470 voxels per brain image) localizer dataset in more than an hour. When going to larger (> 1000 TRs per subject) and higher resolution (> 100,000 voxels per brain image) dataset, it will take tens of hours or several days to run an offline analysis on a compute cluster, which is much more than a comfortable amount of waiting time to the neuroscientists.

Second, there is an emerging requirement of real-time fMRI experiments which may accelerate discoveries in neuroscience in various ways (details will be covered in Chapter 5). As a promising new fMRI analysis technique, FCMA ought to be used in the real-time fMRI studies, because a system that permits real-time analysis of brain interactions in the human brain will allow neuroscientists to conduct new
scientific and clinical studies at a time when interest in real-time fMRI neurofeedback is rising rapidly [107]. However, real-time experiments require the analysis to generate the neurofeedback that keeps pace with data acquisition, i.e. the analysis of the current brain image, together with several other necessary data processing steps (reconstruction, preprocessing, etc.) should be done while the next brain image is being acquired. A brain image is acquired in a TR, which typically lasts 1-2 seconds. Thus, this online analysis in general has to finish in sub-second level. Apparently, the performance of basic FCMA is far away from it.

Because of these two reasons, it is a compelling need to further optimize the basic version of FCMA for the usage of FCMA. Making FCMA run more efficiently on cost-effective computational platforms presents a challenge, however it holds the promise of substantially boosting progress in neuroscience. In this chapter, we optimize the most computationally intensive component of FCMA, the voxel selection, on the many-core processors such as Intel® Xeon Phi™ coprocessors (henceforth referred to as coprocessor in this chapter), for preparation of developing a closed-loop neuroscience research system with an fMRI scanner and a compute cluster, as shown in Figure 4.1.

**Figure 4.1:** A closed-loop neuroscience research system with an fMRI scanner and a cluster featuring both Intel® Xeon® processors and Intel® Xeon Phi™ coprocessors.
The fMRI scanner produces an entire brain’s worth of data every 1-2 seconds as a human subject is exposed to stimuli and/or asked to perform tasks. The stream of brain data is sent to a compute cluster with coprocessors that runs FCMA. The FCMA software analyzes the brain data in two ways: performing offline correlation analysis of the human brain after collecting multiple subjects’ data; or selecting voxels to train a classifier using one subject’s data online to provide real-time feedback as described in the following experiments. The software must achieve satisfactory wall-clock time in both scenarios.

The challenge is to achieve satisfactory performance for both offline analysis and online real-time study without substantial computational hardware costs. There are two main FCMA design goals for FCMA closed-loop system. The first is a scalable implementation that can achieve linear or near-linear speedups on a large cluster. The second is to ensure that the code running on each individual node fully exploits the hardware capability of a coprocessor.

To recap what we discussed in Chapter 2 and 3, the input data for FCMA is a stream of 3D human brain image data (volumes of voxels) over time. Including the time dimension, the input is a 4D dataset. A separated full correlation matrix (i.e. the temporal correlation in BOLD activity of every voxel in the brain with every other voxel) is computed for each time epoch of interest (defined by the experimental blocks) during the fMRI scan. Each matrix may then be categorically labelled based on the experimental conditions experienced by the subject during each epoch (e.g., the stimulus or task condition). The goal of FCMA is to make an unbiased determination of which correlations distinguish between conditions, thereby identifying regions of interests (ROIs) that have different patterns of correlation as a function of condition.

The data that FCMA deals with shapes as tall-skinny matrices since the number of voxels (up to 100,000) is much larger than the number of TRs (typically dozens to hundreds). Although it is fairly straightforward to parallelize FCMA using cluster-
level data partitioning as what we did in Chapter 2, it is challenging to fully utilize the hardware capabilities of modern architectures, specifically the increasing amounts of thread- and data-level parallelism, and smaller amounts of cache per core. Many-core architectures are leading these trends and hence benefit more from optimizations such as vectorization and blocking. Our basic implementation using Intel MKL and LibSVM libraries achieved respectable performance, but was found to significantly underutilize the hardware, especially on the coprocessors. These libraries, and others like them, use cache conscious algorithms to implement their functions, but they do not co-optimize functions or handle data with special characteristics such as tall-skinny matrices well, things that we believe would help in our application.

This chapter describes several optimizations for FCMA on many-core architectures, including blocking tall-skinny matrices for multiplication, retaining L2 cache contents across computation stages, and designing data layout and workflow to be vectorization friendly. We have implemented an optimized version of FCMA that incorporates these ideas, as well as optimized support vector machine (SVM) algorithm.

Our evaluation shows that the optimized implementation on a single coprocessor runs 5x-16x faster than the baseline with Intel MKL and LibSVM libraries. Our optimized SVM runs 10x faster than the popular LibSVM package[18] on a single coprocessor. Our parallel FCMA implementation on a cluster featuring with Intel® Xeon Phi™ coprocessors achieves near linear speedup on up to 96 coprocessors or 5760 cores. Although being applied to FCMA as a case study, our optimizations will also enhance the performance of other applications that involve datasets with similar characteristics on the many-core processors like Intel® Xeon Phi™ coprocessors.

We also show that our optimizations for the coprocessor yield a faster implementation on the multi-core processors like Intel® Xeon® processors. The optimized
implementation on an E5-2670 processor runs 1.4x-2.5x faster than the basic implementation with MKL and LibSVM libraries.

4.2 Intel® Xeon Phi™ Architecture

In this chapter, our optimization targets on FCMA runs on the many-core processors such as Intel® Xeon Phi™ coprocessors. Before we start to discuss the optimization, we briefly describe this many-core hardware architecture in this section.

The Intel® Xeon Phi™ is a coprocessor based on the Intel Many Integrated Core (MIC) architecture. The coprocessor provides a general-purpose programming environment similar to that of an Intel® Xeon® processor.

Figure 4.2 illustrates the high-level architecture of the 5110P coprocessor. This model has 60 CPU cores, each of which runs at a fixed clock rate of 1053MHz and supports up to 4 hardware threads compensating for its in-order instruction execution. Each core has a 32KB L1 data cache, a 32KB L1 instruction cache, and a 512KB unified L2 cache shared by up to four threads. The L2 caches belonging to different cores are interconnected via a bidirectional ring. Cache coherence is maintained by a global-distributed tag directory.

An L2 cache miss triggered by a core can be satisfied by either a remote cache or the memory with slightly different latencies. A previous empirical study showed that the latency of an L2 cache miss on the coprocessor takes 250 CPU cycles from a remote L2 cache location and 302 CPU cycles from the main memory [35]. Both L1 and L2 caches use a cache line size of 64B, therefore, a cache miss will bring 16 single precision or 8 double precision floating point numbers into the cache.

Each core also has a 512-bit wide vector processing unit (VPU), which allows 16 single precision or 8 double precision floating point numbers to be processed in a single CPU cycle. The wide VPU size makes vectorization challenging. If the
vectorization intensity, defined as the number of vectorized elements divided by the number of executed VPU instructions, is low, the VPU is not fully utilized.

By using all available threads and VPU in the most efficient way, the peak floating point performance of the coprocessor can reach 2.02 TFLOPS for single precision and 1.01 TFLOPS for double precision.

Each 5110P coprocessor board has 8GB DRAM, out of which ~2GB are dedicated to the operating system, leaving ~6GB memory available for applications.
4.3 Algorithms

4.3.1 Overview of FCMA Voxel Selection

FCMA works on fMRI datasets. An fMRI dataset contains the fMRI data from a neuroscientific experiment, often conducted over multiple human subjects. A brain volume is comprised of a number of voxels, depending on the resolution of the fMRI scanner and its scanning speed. An epoch of interest consists of a series of continuous time points during which the subject was doing some specific task. The time epoch can be labeled based on the types of task.

The fMRI scanner used in this research (Siemens Skyra) can be configured in various ways, but a relatively common set of image parameters might be 35,000 voxels every 1.5 seconds. The neuroscience datasets we consider contain fMRI data from tens of subjects, each of whom has dozens of time epochs of interest labeled in two conditions, meaning that hundreds of full correlation matrices must be computed. Although the size of such a dataset is approximately a gigabyte, the size of the corresponding full correlation matrices will be in terabytes.

The most computationally intensive component of FCMA, the correlation-based voxel selection, involves a three-stage pipeline, as shown in Figure 4.3. Before computation, FCMA reads in the preprocessed fMRI data (e.g., corrected for head motion and other noise sources) and the text files specifying the labeled time epochs over which the correlation is to be computed.

The first stage of FCMA is correlation computation. We operationalize the temporal interaction between two brain voxels during a time epoch of interest using Pearson correlation, which is computed as

\[
\text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}
\]  

(4.1)
where $X$ and $Y$ are two vectors depicting the BOLD activity of voxel $x$ and $y$ in a time epoch, and $\text{cov}$ and $\sigma$ refer to the covariance and the standard deviation of the vectors, respectively. We reduce the computation of the Pearson correlation between voxel pairs to the multiplication of a voxel-by-time matrix and its transpose by normalizing the data within each time epoch\[119\]. Specifically, the reduction first subtracts the mean value of the time epoch vector, and then divides this mean-centered vector by its root sum of squares as

$$x_i' = \frac{(x_i - \bar{x})}{\sqrt{\sum_{j=1}^{n} x_j^2 - n\bar{x}^2}}$$ \hspace{1cm} (4.2)$$

where $x_i$ and $x_j$ are elements of time epoch vector $X$, and $\bar{x}$ depicts its mean value. The Pearson correlation between two of the resulting vectors $X'$ and $Y'$ is their pointwise product

$$\text{corr}(X, Y) = X' \cdot Y'$$ \hspace{1cm} (4.3)$$

which for an arbitrary number of vectors becomes an inner product matrix multiplication. A more detailed derivation can be found in Section 2.3. To obtain the full correlation matrix for a given time epoch, $\text{corr}(X, X)$ is calculated by taking the product of $X'$ and its transpose. Such reduction simplifies the computation. Suppose
there are $N$ voxels in the whole brain (here $N \approx 35,000$), each matrix multiplication yields an $N \times N$ full correlation matrix $C$, in which entry $C_{ij}$ represents the correlation over time in an epoch between voxel $i$ and $j$.

The second stage is *within-subject normalization*. The full correlation matrices with various labels from the first stage are the input of this stage. Because Pearson correlation coefficients are bounded $[-1, 1]$ and not normally distributed near the bounds, we apply the Fisher transformation to every resulting correlation coefficient as

$$z = \frac{1}{2} \ln\left(\frac{1 + r}{1 - r}\right)$$

(4.4)

where $r$ is the correlation coefficient between any two voxels. After that, to put the correlation coefficients from different subjects on the same scale for the cross-subject classification, we also apply z-score transformation within each subject

$$\forall z \in P, z' = \frac{z - \mu}{\sigma}$$

(4.5)

where $P$ is the population of Fisher-transformed correlation coefficients within subject, and $\mu$ and $\sigma$ are the mean and standard deviation of the population, respectively.

The output of the second stage is then grouped by voxels so that the correlation vectors for the same voxel from all epochs are stored together as the input of the third stage (different labels depicted as dark red and light pink in Figure 4.3).

The third stage is *SVM cross validation*. This stage performs voxel-wise cross validation to identify which voxels are informative in terms of their correlations with other voxels. Note that for each voxel there are only a few hundred correlation vector samples (for cross-subject classification, number of epochs per subject $\times$ number of subjects) while each sample has $\sim 35,000$ dimensions (corresponding to the number of voxels in the brain), so we use linear SVM to avoid overfitting. Linear SVM handles high-dimensional data better than SVM with other kernels such as polynomial or
Gaussian, as well as other classification algorithms such as logistic regression. For each voxel, we extract its corresponding rows from all correlation matrices. Each of these rows contains the normalized correlation values (stage 2) between this voxel and all the other voxels in the brain in a time epoch. These correlation vectors are then labeled with the experimental conditions to which their epochs correspond and fed into linear SVM as samples. Linear SVM runs cross-validation across subjects (leave one subject out at a time) to assign a classification accuracy value to each voxel, quantifying its ability to distinguish between conditions.

Cluster parallelization framework

We use a task-based parallel framework on a compute cluster to process this pipeline, in which a master assigns tasks to workers. The master node first distributes brain data to the worker nodes and then sends tasks to the workers to process in parallel. A worker works on one task at a time. When a worker finishes a task, it will receive a new task from the master.

The tasks are defined by partitioning the correlation matrices along their rows. Each task is one run of the three-stage FCMA algorithm for the assigned number of voxels.

Three-stage algorithm on a worker node

We then describe the three-stage algorithm on a single worker node after being assigned a number of voxels as a task. It was briefly covered in Section 2.3 with less detail.

In stage one, a worker node computes correlation vectors for all epochs of its given voxels as shown in Fig. 4.4. The number of given voxels is typically a few hundred or less to permit all correlation data to fit into memory. Since in most cases one time epoch consists of less than 20 time points, the matrix multiplications
used to compute correlation have one very small dimension (the $k$ dimension) which limits performance. On the other hand, in order to facilitate the subsequent steps, we arrange the data in memory such that all correlation vectors corresponding to a single voxel are contiguous, which means the result of the correlation matrix multiplications must be interleaved row by row. Fig. 4.4 illustrates the data layout in different colors, where the activity values of the first and second epochs are colored in dark red and light pink, respectively. The first voxel out of the $V$ voxels computes its correlation vector of the first epoch and places as the first row of the correlation data of the first voxel depicted in dark red; the last voxel computes its correlation vector of the second epoch and places as the second row of the correlation data of the last voxel depicted in light pink.

![Correlation computation diagram](image)

Figure 4.4: Correlation computation for $V$ voxels in $M$ epochs, the results are grouped by voxels and then normalized within subject (assuming $E$ epochs per subject).
In stage two, the worker node applies Fisher transformation and within subject z-scoring to the resulting correlation coefficients of its assigned voxels from stage one. As depicted in the bottom half of Figure 4.4, a voxel’s $M$ correlation vectors are partitioned into $E$ epochs per subject (dashed lines). For within-subject normalization, a sub-column of $E$ values belonging to the same subject (e.g. the vertical black line in the second subject of the first voxel in Figure 4.4) is extracted as a population $P$ in Formula 4.5 to process together.

In stage three, the normalized correlation vectors for the assigned voxels with two different labels can then be sent to the linear SVM classifier to determine how well the vectors differentiate the labeled categories via cross validation. The attained accuracies of these voxels are sent back to the master node.

Finally, the master node collects all voxels and sorts them by their resulting accuracies of cross validation. The brain regions constituted by top voxels are identified as ROIs in terms of correlation for following studies.

4.3.2 Basic Implementation

We implemented the three-stage algorithm described above in C++. All floating point values are represented in single precision. Our basic implementation can be viewed as a typical implementation using sound programming discipline and the state-of-the-art libraries.

For correlation computation in the first stage, we allocate continuous aligned space in the memory to store the correlation vectors of all voxels so that the cblas_sgemm routine of Intel MKL can be called to compute the correlation vectors epoch by epoch and place the results in an alternate way grouping by voxels via specifying the parameter $ldc$ appropriately. The normalization of the second stage is parallelized along all voxels while applying vectorization within z-scoring. The SVM training that takes place in the third stage is essentially an algorithm that repeatedly calculates the
matrix-vector product of one voxel’s corresponding data matrix with different rows of the matrix itself. Since there are many more voxels (corresponding to the length of a correlation vector) than there are training examples (one time epoch corresponds to one sample), we precomputed all such matrix-vector products (also called the kernel matrix) before beginning SVM cross validation. Given that it is a linear SVM and the kernel function is a dot product, this kernel matrix can be cast as a symmetric matrix multiplication and solved with the cblas_ssyrk routine of MKL over one voxel’s corresponding data matrix and its transpose. The SVM cross validation works based on the precomputed kernel matrix by applying the sequential minimal optimization (SMO) algorithm implemented in the LibSVM [18] package for training. SMO is an iterative method to solve large quadratic programming problems in the training phase of SVM. It breaks the problem into smallest possible sub-problems and each time solves one analytically until convergence [95]. In this way, one thread takes care of one voxel’s kernel matrix computation and cross validation at a time so that different voxels can progress simultaneously.

We have tuned this implementation by carefully designing the data structures to utilize the high performance Intel MKL routines. We have also deliberately precomputed the kernel matrices to avoid duplicate pairwise kernel computation and to keep more frequently used data in the cache. In practice, this results in good performance on a cluster consisting of nodes with Intel® Xeon® processors, where FCMA runs in the master-worker model communicating via MPI calls, and the master node allocates different sets of voxels to different workers for processing.

Besides what we discussed in Section 2.3, the basic implementation pays additional attention to the cache locality by reorganizing the data layout through the process to make the data to be processed together live in the memory closer. In general, it gets 9.7x speedup by running this basic implementation of FCMA voxel selection on a cluster of 96 coprocessors (details in Section 4.5) than what we reported in Chapter 2.
While some of the performance difference can be attributed to the different processor generations used, much of it is due to the algorithm and performance optimization. In other words, this basic implementation runs in a respectable performance.

4.3.3 Performance Analysis

This subsection reports our analysis of the baseline implementation on the Intel® Xeon Phi™ coprocessors. All of our measurements were collected by running the basic implementation on the face-scene dataset (used in both Chapter 2 and 3), which contains brain data with 34,470 voxels, in 216 12-time-point epochs with two different labels. The master node assigns 120 voxels to a worker node as a single task for processing. We analyzed the performance of a single worker task.

Low efficiency of matrix multiplication

The first finding is that the matrix multiplications of Intel MKL does not perform efficiently for our tall-skinny matrices on the coprocessor.

At the correlation computation stage, a worker is responsible for 120 voxels. It calls cblas_sgemm to perform 216 matrix multiplications between $120 \times 12$ and $12 \times 34,470$ matrix pairs to obtain correlation coefficients between the assigned 120 voxels and the entire brain over 216 epochs. At the SVM cross validation stage, it computes 120 symmetric kernel matrices between $216 \times 34470$ matrices and the corresponding transposes for 120 voxels using cblas_ssyrk.

The first row of Table 4.1 summarizes the performance of the matrix multiplication routines using Intel® vTune™ Amplifier. There are three performance issues. The first issue is that the number of memory references is too high. Our instrumentation shows that there are 34.9 billion memory references, whereas the matrix multiplications for the correlation computation and for SVM cross validation stages should have fewer than 10 billion.
<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th># mem refs</th>
<th>L2 miss</th>
<th>Vector intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix multiplication</td>
<td>1830 ms</td>
<td>34.9 billion</td>
<td>709 million</td>
<td>3.6</td>
</tr>
<tr>
<td>Normalization</td>
<td>766 ms</td>
<td>6.2 billion</td>
<td>179 million</td>
<td>8.5</td>
</tr>
<tr>
<td>LibSVM</td>
<td>3600 ms</td>
<td>23.0 billion</td>
<td>7 million</td>
<td>1.9</td>
</tr>
</tbody>
</table>

Table 4.1: The instrumentation results of the basic implementation.

The second issue is that the cache miss overhead is high. Since empirically the latency of a L2 cache miss on the coprocessor is \(~250\) CPU cycles from remote L2 cache and \(~302\) CPU cycles from memory\[^35\], and the clock rate of the coprocessor (5110P) is 1053MHz, we can estimate the latency of an L2 cache miss to be \(~300\) ns so the total latency of L2 cache misses could be as high as \(~880\) ms if not well hidden by other operations, which is significant compared to the total elapsed time (1830 ms).

The third issue is that the vectorization intensity value is only 3.6 while the ideal vectorization intensity is 16. Only 23% of the VPU capability is used during computation.

These three observations show that when computing our tall-skinny matrices, Intel MKL doesn’t leverage L2 cache well and does poorly to take advantage of the VPUs of the coprocessor.

**Lack of cached data reuse between stages**

By manually calculation, we noticed that the second stage of FCMA causes \(~112\) million compulsory L2 cache misses. The first stage (correlation computation) generates correlations and writes them into their data structures. The second stage (within-subject normalization) reads the data back to perform Fisher transformation and z-scoring.

Optimization within a function cannot avoid such compulsory L2 cache misses between function calls. When such situations happen between two stages of the
processing pipeline, retaining cache contents becomes difficult. To avoid such cache misses, we need to have higher-level optimizations.

In addition, Table 4.1 shows that the vectorization intensity of within-subject normalization is 8.5, indicating that there are rooms to improve the utilization of the vector unit.

**Poor VPU utilization in SVM cross validation**

We noticed that the SVM cross validation stage takes a lot of time and most of the time spent in the LibSVM library.

The first reason is that the vectorization intensity of LibSVM is only 1.9 (Table 4.1), indicating it does not take advantage of the vector unit of the coprocessor well. As we started looking at the code of LibSVM, we found that it stores data in sparse index set instead of dense matrix.

The second reason is LibSVM does unnecessary data type conversions during computation and uses double precision values in the computationally intensive loops.

The third reason is about the memory limitation of the coprocessor. The implementation uses one thread to run cross validation for one voxel. Therefore, the master node needs to assign at least 240 voxels at a time to the coprocessor for fully utilizing its available 240 threads. However, each 5110P coprocessor has only about 6GB memory available to applications. 240 voxels’ correlation vectors will consume 8.3GB memory. This forces the master node to only assign a small number of voxels to the coprocessor once, consequently the computing resource is under utilized during the linear SVM cross validation stage.

The poor performance of LibSVM motivated us to optimize the LibSVM. In addition, we implemented *PhiSVM* based on a GPU SVM implementation [15]. We will report the performance of these implementations in Section 4.5.
4.4 Optimizations

Based on the optimization opportunities identified above, we came up with three optimization ideas. This section first describes our ideas and then presents how we optimized our implementation.

4.4.1 Optimization Ideas

We employed three optimization ideas to optimize the FCMA algorithm for a single worker node task:

1. Partitioning tall-skinny matrices for blocking to fit small amount of L2 cache for each thread. The traditional way of blocking is to change the looping structure of the code to process a square block of data in inner loops. However, this approach would exceed the relatively small L2 cache per thread on the coprocessor;

2. Retaining cache contents across stages of the procedure pipeline. Our approach is to look at the contents of L2 cache in the current stage. If the contents will be reused frequently in the next stage, we will consider merging the two stages to avoid cache misses at the next stage. Typically, a cache conscious algorithm of the current stage uses blocked data structure to reduce cache misses. When finishing the computation with the blocks, it will proceed with the next stage computation without waiting for other blocks of the current stage to complete their computations. Obviously, merging stages will reduce modularity of the program. So one needs to be careful when applying this idea;

3. Designing data structures and workflow for vectorization. A vector unit typically requires its data to be layout in memory in consecutive fashion so that they can be moved into and out of its large register file quickly. This optimiza-
tion ensures that the data structures fit the required layout to maximize the utilization of the vector unit.

### 4.4.2 Correlation Computation

As mentioned before, correlation computation is reduced to matrix multiplication. At this stage, a single worker node needs to correlate its assigned \( V \) voxels with all brain voxels over all epochs of interest, so the job is essentially doing matrix multiplications between relatively small matrices and tall-skinny matrices as illustrated in the top half of Figure 4.5. Our optimization idea #1 is applied to block voxel data as depicted in dark red in Figure 4.5. Performing computation within blocks will not trigger unnecessary L2 cache misses. In order to fully utilize the VPUs, we defined the size of blocks to be integral multiples of the vector unit width. Also, we consciously transpose the block of tall-skinny matrix to better utilize the VPUs (idea #3).

![Figure 4.5: Retain correlations (red blocks) in L2 cache for the normalization stage.](image)

79
4.4.3 Within-subject normalization

Next, we consider the within-subject normalization stage of FCMA voxel selection, which is shown as the second pipeline stage in Figure 4.3. If treating this stage independently, we can see that a large number of correlation data have to be read in for processing and written back which causes a lot of compulsory L2 cache misses. The memory accessing latency cannot be hidden well by computing because this stage is memory-bound. Fortunately, this can be saved if our optimization idea #2 is applied.

Briefly speaking, the idea is, after computing the correlation values in local blocks, the normalization can be applied to the data before they are written back. What we need to take into consideration, besides the regular blocking, is that the data necessary for a complete normalization should reside in the same block. It is a cache locality driven job scheduling approach. Figure 4.5 indicates the merging process. Out of $V$ assigned voxels, each thread only takes $B$ voxels to compute their correlations with some other $B'$ voxels for $E$ epochs belonged to one subject, yielding $B$ portion of within-subject correlation values that can be normalized. Notice that all blocks in red can be fit into L2 cache, we will not spend additional time to fetch the data between stages. Using different threads to handle different blocks in this way, all correlation data would be computed and immediately normalized then written to the memory. The idea of merging adjacent stages in the procedure pipeline can be generalized to all memory-bound processes where the memory latency cannot be hidden completely by the computation.

In addition to merging stages, we also apply our optimization idea #3 to this stage. Our approach to vectorizing the normalization procedure is visualized in Figure 4.6, which shows the detailed information of processing happened inside one red block in the right part of Figure 4.5. The correlation vectors are placed contiguously into a temporary block of memory. We process the data in chunks of 16 voxels using SIMD instructions and registers (enabled with the `#pragma SIMD` directive). To
compute the Fisher transformation, we must perform a $logf$ operation on each data element. On the coprocessors, computing $logf$ benefits from hardware support for single precision transcendental functions in the extended math unit (EMU). During the first pass through the data, we also compute the mean and standard deviation across correlation vectors. We use the $E[X^2] - E[X]^2$ formulation for variance in order to compute both the mean and standard deviation in one pass. As one block of 16 voxels are processed, we also prefetch the next set of voxels into the L1 cache using `_mm_prefetch(_MM_HINT_T0)` We do not do L2 software prefetching because the data are presumed L2 resident by our blocking design. After the mean and standard deviation are computed, a second pass through the data subtracts the mean and scales by the inverse of the standard deviation.

![Diagram](image)

**Figure 4.6:** The $z$-score computation begins with an L2 block of correlation vectors and processes 16 voxels at a time using SIMD instructions. The mean and variance at each voxel is computed for a given subject’s correlation vectors.

### 4.4.4 SVM cross validation

This stage consists of two parts, a linear kernel matrix precomputation followed by linear SVM cross validation over the precomputed kernel matrices.
SVM kernel matrix precomputation in FCMA is essentially a matrix multiplication between a voxel’s $M \times N$ data matrix and its transpose, where $M$ is the number of epochs over which the correlation is computed, and $N$ is the number of voxels in the brain, normally $M << N$. We implemented a custom symmetric matrix multiplication function for the coprocessor which attempts to optimize for our particular setting by applying our optimization ideas #1 and #3 similar to the first stage.

Figure 4.7 shows the workflow of our optimized implementation. Since one worker node deals with certain number of voxels simultaneously, a number of independent matrix multiplications are running in parallel, one per voxel, as shown in the depth dimension. The size of a data matrix is typically $\sim 60$ MB (400 epochs times 35,000 voxels matrix stored in single precision values), making it possible to consume the $\sim 6$GB of on-board memory of the coprocessor by processing only a 100 voxels’ worth of matrices. Therefore the number of independent, concurrently executed matrix multiplications is limited. They cannot saturate the coprocessor, which compels us to split the problems across multiple threads and use OpenMP locks to control access to the C matrices.

Each thread proceeds down the long dimension of the matrix in blocks of 96 rows (an integral multiple of VPU length). These blocks are copied into a local buffer ($A_{\text{local}}$). Smaller blocks of $A_{\text{local}}$ are then transposed and copied into a smaller buffer ($A^T_{\text{local}}$). Once the data is ready, we call an auto-generated $16 \times 9 \times 96$ assembly-level matrix multiply routine in the inner loop to generate each block of $C_{\text{local}}$. We pad $A_{\text{local}}$ with zeros for the last block if $A$’s height is not a perfect multiple of 96, and use vectorized loops for the other dimensions. After the thread completes its portion of the matrix multiply, it takes a lock corresponding to the $C$ matrix and adds its contribution to $C_{[91]}$.

Regarding the SVM cross validation algorithm over the precomputed kernel matrix, in order to circumvent the drawbacks of LibSVM mentioned above, we adopt
Figure 4.7: Multiple tall-skinny matrix multiplications (sryk) are performed to pre-compute the SVM kernel matrices.

A fast SVM algorithm [15] implemented for GPUs and rewrote the CUDA code into C++ to run on the coprocessor. Like LibSVM, this fast SVM applies SMO algorithm to solve an SVM training problem. Typically, a single iteration of SMO algorithm involves choosing two rows from the kernel matrix and using them to update information for all other rows. The choice of the two rows is done heuristically. But instead of only using the LibSVM heuristic in the working set selection, our fast SVM adaptively chooses the faster heuristic (either first order [63] or second order [34]) based on the convergence rate on the specific training data. The original CUDA implementation aims at solving huge SVM problems involving tens of thousands of samples by coordinating all GPU cores to work together. In our problem setting, we face
a large number of smaller scale SVM problems, one per voxel, each of which only contains a few hundred of samples and has the kernel matrix (linear kernel) precomputed. Therefore, we make it so that a thread takes full responsibility for the cross validation of one voxel. Moreover, using our optimization idea #3, we vectorize the most computationally intensive part of the code for better usage of VPUs. We call the adopted fast SVM algorithm implemented on the coprocessor PhiSVM.

Another challenge is finding enough parallelism (i.e. independent SVM problems) to fully utilize the coprocessor during the cross validation stage without exceeding the limits of the on-board memory. This can be solved by redesigning the computing procedure, in which we accumulate a least 240 voxels’ kernel matrices before conducting the SVM cross validation. Since a kernel matrix is significantly smaller than a data matrix, reducing to kernel matrices can save a lot of space so that doing SVM cross validation for at least 240 voxels becomes possible, therefore no available computing power will be wasted during the SVM cross validation.

4.5 Evaluation

To evaluate the proposed optimization ideas, we pursued answers to following questions:

1. What is the system performance for the offline and online data analysis cases?

2. Is the system scalable as we add more coprocessor nodes to the cluster?

3. For a typical dataset, what are the performance contributions of the proposed optimization ideas?

4. How does the resulting system work on general-purpose processors?

To answer these questions, we will first describe our experimental setup and then present our results and analysis.
4.5.1 Experimental Setup

We ran our experiments on a 48-node cluster named Metacortex located at the Princeton Neuroscience Institute (PNI), interconnected by an Arista 10GE switch. Each node of the cluster has a motherboard with: 2 Intel® Xeon® E5-2670 processors, both running at a 2.6GHz clock rate, 2 Intel® Xeon Phi™ 5110P coprocessors running at 1053MHz, connected via PCI-e slots, 256GB memory, 8×3TB SATA disks, and 1.65TB FusionIO ScaleIO Flash memory card.

The host node runs CentOS 6.3 and each coprocessor runs its on-board Linux (version 2.6.38.8). The software packages used in our experiments include MPSS (version 3.3), Intel MPI (version 5.0.2.044), and Intel compiler (version 15.0.2). Intel® MKL (version 11.2) and LibSVM (version 3.20) libraries were applied to the baseline implementation for comparison. We used Intel® vTune™ Amplifier (version 2015.2.0.39344) to collect the performance data.

Our experiments used two fMRI datasets. The first one is the face-scene localizer dataset consisting of fMRI data from 18 subjects who passively viewed either face or scene images as described in Chapter 2 and 3. The second one is an attention dataset used in [55] consisting of fMRI data from 30 subjects who were asked to look at either left or right images on a screen while being scanned. Detailed information on the datasets is listed in Table 4.2.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Voxels</th>
<th>Subjects</th>
<th>Epochs</th>
<th>Epoch length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face-scene</td>
<td>34,470</td>
<td>18</td>
<td>216</td>
<td>12</td>
</tr>
<tr>
<td>Attention</td>
<td>25,260</td>
<td>30</td>
<td>540</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 4.2: The fMRI datasets used in the experiments.

4.5.2 System Performance

In this section, we present the results for both offline and emulated online data analysis. The offline case is time consuming because it involved leave-one-subject-out
nested $n$-fold cross validation, which we elaborate on below, on all $n$ subjects in the dataset. The emulated online data analysis involves selecting voxels to train a classifier using one subject’s data on the fly to provide real-time neurofeedback in subsequent experiments.

**Offline analysis performance**

We ran the nested leave-one-subject-out $n$-fold cross validation illustrated in Figure 2.2 on both *face-scene* and *attention* datasets, where $n$ is the total number of subjects in a dataset. In each fold of the outer loop cross validation, a training set consisting of $n - 1$ subjects was used for voxel selection by conducting another level of leave-one-subject-out cross validation. Voxels were selected based on their classification accuracies of their correlation vectors, determined by the procedure illustrated in Figure 4.3. After voxel selection in each fold, a final classifier can be trained using the correlation patterns of the selected voxels to test on the left out subject of the outer loop to verify the selection. In addition, the selected voxels across different folds can be statistically compared to identify the reliable voxels whose correlation patterns with the rest of the brain are informative as we did in the case study of Chapter 3.

Table 4.3 shows the elapsed time in seconds as a function of the number of coprocessor nodes. For the *face-scene* dataset, the experiment ran 18 folds of leave-one-subject-out validation. The whole process took 85 seconds using 96 coprocessors. On average, each fold took 4.7 seconds.

For the *attention* dataset, the experiment ran 30 folds of leave-one-subject-out validation. The whole process took 741 seconds using 96 coprocessors. On average, each fold took 24.7 seconds. The *attention* experiment took longer because it involved more subjects and each subject performed more epochs.

We reproduced the results used in Chapter 3 and [55].
Table 4.3: The elapsed time in seconds of using different numbers of coprocessors worker nodes to run an offline analysis.

<table>
<thead>
<tr>
<th># nodes</th>
<th>1</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face-scene</td>
<td>5101</td>
<td>694</td>
<td>385</td>
<td>242</td>
<td>124</td>
<td>85</td>
</tr>
<tr>
<td>Attention</td>
<td>54506</td>
<td>6813</td>
<td>3620</td>
<td>2172</td>
<td>1099</td>
<td>741</td>
</tr>
</tbody>
</table>

Table 4.4: The elapsed time in seconds of using different numbers of coprocessors worker nodes to select voxels for building online classifier.

<table>
<thead>
<tr>
<th># nodes</th>
<th>1</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face-scene</td>
<td>12.00</td>
<td>3.18</td>
<td>2.51</td>
<td>2.26</td>
<td>2.24</td>
<td>2.21</td>
</tr>
<tr>
<td>Attention</td>
<td>16.50</td>
<td>3.96</td>
<td>2.97</td>
<td>2.59</td>
<td>2.52</td>
<td>2.51</td>
</tr>
</tbody>
</table>

**Emulated online analysis performance**

In closed-loop real-time fMRI study of functional interactions in the human brain, a classifier may need to be trained online using the voxels selected by FCMA based on the whole-brain correlation. This classifier will be used for sending feedback to the subject while being scanned. The voxel selection procedure is similar to the offline analysis, except that instead of taking data from multiple subjects to process in batch, we only use the data received from the subject being scanned, and no nested cross validation is applied.

We performed the emulated online analysis on one subject’s data from each of the face-scene and attention datasets. The voxels for building the classifier were selected using the subject’s data. Table 4.4 shows the elapsed time in seconds as a function of the number of coprocessors. Using 96 coprocessors, voxels were selected within 3 seconds which is fast enough to build an online classifier to provide real-time feedback to the subject in the subsequent experiments.
4.5.3 Speedup

We studied the scalability of our optimized implementation of FCMA by varying the number of coprocessors used in the offline analysis.

Figure 4.8 shows the speedup of both datasets as a function of the number of coprocessors. With 96 coprocessors, we achieved a 59.8x on the face-scene dataset and a 73.5x on the attention dataset. The speedup is greater for the attention dataset due to its larger size.

![Figure 4.8: Speedups of the optimized implementation with face-scene and attention datasets.](image)

4.5.4 Performance Implications of Optimizations

In this subsection, we focused on the performance of a single coprocessor to demonstrate the contributions of our proposed optimization ideas.

**Performance of a three-stage task**

We first compared the overall performance of a single coprocessor for our optimized implementation and the baseline implementation with MKL and LibSVM libraries. Both face-scene and attention datasets were used.
As we described in Section 4.3.1 in the parallel FCMA framework, the master node distributes tasks to worker nodes by partitioning all voxels into pieces. In the baseline implementation, due to the memory limitation of the coprocessor, the master only can allocate 120 voxels of the face-scene dataset or 60 voxels of the attention dataset to a coprocessor for processing, respectively. As a result, the third stage of the baseline implementation of the FCMA pipeline cannot fully exploit the available hardware capability of the coprocessor, since one thread takes care of only one voxel’s cross validation. This constraint is largely relaxed in the optimized implementation in which a coprocessor can take more voxels (e.g. 240) by reducing the large correlation data into much smaller precomputed kernel matrices portion by portion to guarantee the consumed memory size does not go beyond the available on-board memory of the coprocessor.

Since the number of voxels one coprocessor can take differs between the baseline and optimized implementations, we normalized the performance to processing time per voxel. Figure 4.9 shows the speedup of the optimized implementation over the baseline. The performance of the baseline was set to 1 for both datasets. Our optimized implementation runs 5.24x and 16.39x faster than the baseline, respectively. For attention dataset, the fraction of time spent in SVM computation is significantly larger, and hence benefits much more from our optimizations.

Next, we break down the contributions based on the proposed optimizations. We compared the performance of processing a task of 120 voxels from the face-scene dataset in a single coprocessor.

**Blocking skinny matrices (vs. Intel MKL)**

There are two matrix multiplications in two stages of the FCMA processing pipeline: in correlation computation and in SVM cross validation. For offline analysis on face-scene dataset, at the correlation computation stage, FCMA performs 216 (epochs)
multiplications of matrix of $A[120, 12]$ with $B[12, 34470]$ and writes results to matrix $C[120, 34470]$ (illustrated in Figure 4.4). 21.443 billion floating-point operations and 4.136 million memory writes are performed.

In the SVM cross validation stage, FCMA performs a multiplication of matrix $A[204, 34470]$ with its transpose $A^T$ and writes results to matrix $C[204, 204]$ (illustrated in Figure 4.7). Since $A$ is multiplied with its transpose, only upper or lower triangle of the resulting matrix needs to be computed. This matrix multiplication performs 172.14 billion floating point operations but only 20,088 memory writes.

Table 4.5 reports the elapsed times and Giga FlOating Operations/Seconds (GFLOPS) for both cases. Our optimized matrix multiplication achieved 126 GFLOPS in the correlation computation stage and 430 GFLOPS in the SVM cross validation stage. The matrix multiplication using Intel MKL achieved 93 GFLOPS for the correlation computation stage, and 108 GFLOPS for the SVM cross validation stage.
The matrix multiplication in the correlation computation stage produced many more writes than the SVM cross validation stage did, explaining why the latter reached 3.4x higher GFLOPS.

<table>
<thead>
<tr>
<th>Function</th>
<th>Time</th>
<th>GFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our blocking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>correlation matrix computation</td>
<td>170 ms</td>
<td>126</td>
</tr>
<tr>
<td>SVM kernel matrix computation</td>
<td>400 ms</td>
<td>430</td>
</tr>
<tr>
<td>MKL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>correlation matrix computation</td>
<td>230 ms</td>
<td>93</td>
</tr>
<tr>
<td>SVM kernel matrix computation</td>
<td>1600 ms</td>
<td>108</td>
</tr>
</tbody>
</table>

Table 4.5: The performance results of matrix multiplication routines used in both correlation computation and SVM cross validation stages.

Table 4.6 shows the total number of memory references, the number of L2 cache misses, and the vectorization intensities of the matrix multiplication routines in our optimized implementation and in Intel MKL. These are the combined results of both stages. The vectorization intensity measured from vTune for our optimized implementation was close to the theoretical peak value 16, whereas that for Intel MKL was 3.6. The results show that our implementation took full advantage of the vector unit, whereas Intel MKL only utilized 23% of the hardware capability. Our implementation had 5.82x fewer L2 cache misses than Intel MKL (121.8 vs. 708.9 million). These results show that Intel MKL performed relatively poorly because of its large number of L2 cache misses and low vectorization intensity.

Our measurements also indicate that Intel MKL made 3.49x more memory references than our implementation (34,858.37 vs. 9,974.87 millions).

<table>
<thead>
<tr>
<th></th>
<th># memory refs</th>
<th>L2 miss</th>
<th>Vector intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our blocking</td>
<td>9,974,870,500</td>
<td>121,800,000</td>
<td>16</td>
</tr>
<tr>
<td>MKL</td>
<td>34,858,368,500</td>
<td>708,900,000</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Table 4.6: Memory references, L2 cache misses and vector intensity of the matrix multiplication routines.
Retaining cache contents

To understand the performance impact of modifying the FCMA algorithm to retain cache contents across the correlation computation and normalization stages, we implemented two cases: separated and merged.

As we discussed in Section 4.4.3, the merged implementation performed normalization on correlations as soon as they were computed, without waiting for the entire correlation computation stage to finish. The data in the L2 cache was retained for within-subject normalization without writing out to memory and reading back in again. Conversely, the separated implementation finished all correlation computation before moving forward to the normalization stage.

Table 4.7 shows the results of elapsed times, number of memory references, and number of L2 cache misses for both implementations. The merged implementation had a fewer number of memory references (1.93 vs. 4.35 billion) and a fewer number of L2 cache misses (67.5 vs. 188.1 million), resulting in a 24% reduction in elapsed time.

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
<th># memory refs</th>
<th>L2 miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>merged</td>
<td>320 ms</td>
<td>1,925,806,500</td>
<td>67,500,000</td>
</tr>
<tr>
<td>separated</td>
<td>420 ms</td>
<td>4,347,490,500</td>
<td>188,100,000</td>
</tr>
</tbody>
</table>

Table 4.7: Performance comparisons of retaining L2 cache contents (merged stages vs separated stages).

Vectorization for SVM

To determine the performance impact of vectorization in the coprocessor, we compared LibSVM, our optimized LibSVM, and our optimized PhiSVM.

In both optimized implementations, we reorganized the data layout and workflow for the computationally intensive loops in order to better utilize the VPUs of the coprocessor. Since single precision floating point numbers are accurate enough for our
application, we used \textit{float} type in PhiSVM. For fair comparison, we also converted all \textit{double} type values in LibSVM to \textit{float} type so that the VPU can process an equal number of values in a single \textit{SIMD} instruction.

Table 4.8 shows the elapsed times and vectorization intensities for all three implementations. Optimized PhiSVM took 390 ms whereas optimized LibSVM and the single precision LibSVM took 1,150 ms and 3,600 ms, respectively. PhiSVM outperformed LibSVM even after its careful vectorization because of the advances in algorithm and data structure as described in section 3.3.3.

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Vector intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>LibSVM</td>
<td>3600 ms</td>
<td>1.9</td>
</tr>
<tr>
<td>Optimized LibSVM</td>
<td>1150 ms</td>
<td>8.9</td>
</tr>
<tr>
<td>PhiSVM</td>
<td>390 ms</td>
<td>9.8</td>
</tr>
</tbody>
</table>

Table 4.8: The performance comparison of SVM cross validation.

4.5.5 \textbf{Performance on Intel® Xeon® Processors}

To determine how well our optimizations for the coprocessor would work on a general-purpose processor, we compared our optimized implementation with the basic implementation on a single E5-2670 processor in one node of our cluster, described in Section 4.5.1. This experiment was identical to what is described in the first subsection of Section 4.5.4 (Performance of a three-stage task).

Figure 4.10 shows that our optimizations for the coprocessor worked quite well for the processor. Our optimized implementation ran 1.4x and 2.5x faster than the basic one for the \textit{face-scene} and \textit{attention} datasets, respectively.

The performance improvements on the processor were significant but less dramatic than on the coprocessor for several reasons. First, the processor has a relatively large Last Level Cache (LLC) per CPU core or per thread. It has 8 CPU cores and 16 hyperthreads and 20MB LLC. On average, each thread has 1.28MB LLC per thread,
Figure 4.10: The performance improvement of our optimized version over the basic implementation on Intel® Xeon® E5-2670 processor. The basic performance is normalized to 1.

which is an order of magnitude larger than that for the coprocessor. The large cache allows fewer LLC cache misses, making the performance tuning for L2 cache misses less important.

Second, the width of vector registers on the processor is 256-bit, only half of that on the coprocessor. The narrower vector unit makes the effect of vectorization less significant.

Third, the processor supports 2 hyperthreading threads, whereas the coprocessor supports 4 per core. The total number of concurrent threads on the processor is 16 versus 240 on the coprocessor. Therefore, the thread starvation issue presenting during SVM cross validation of the baseline implementation on the coprocessor does not exist on the processor.

We also compared the basic and the optimized implementation on the processor and coprocessor. Figure 4.11 shows that the optimized implementation on the coprocessor outperformed the same code running on the processor for both face-scene and attention datasets.
Figure 4.11: The performance comparisons between Xeon\textsuperscript{®} E5-2670 processor and Xeon Phi\textsuperscript{TM} 5110P coprocessor (a) face-scene dataset; (b) attention dataset. The basic performance of E5-2670 processor is normalized to 1.

4.6 Related Work

Many scientific computing applications in multiple disciplines such as physics and chemistry have taken advantage of many-core processors such as the Intel\textsuperscript{®} Xeon Phi\textsuperscript{TM} coprocessors \cite{7, 53}. To the best of our knowledge, this paper presents the first neuroscience application that adopts the coprocessor for achieving performance goals (for both offline and online analysis).

Data locality optimizing algorithms for improving the efficiency of accessing data residing in memory hierarchies have been well studied for a long time \cite{70, 118}. Several optimizations have been performed to fully exploit the processor architecture in order to achieve high performance of linear algebra operations especially matrix multiplication \cite{46}, based on which different versions of BLAS routines such as MKL and GotoBlas were implemented. There are studies about optimizing matrix multiplication on many-core architectures such as GPUs \cite{77, 108, 114} and some recent work on Intel\textsuperscript{®} Xeon Phi\textsuperscript{TM} coprocessors \cite{43, 51}. Most of the optimizations for \textit{GEMM} focus on coordinating multiple threads to conquer huge, nearly-square matrices. Our application, on the other hand, requires a single thread to work on one matrix multiplication between matrices with one small dimension. Dense matrix multiplication involving tall-skinny matrices is known to be difficult to optimize \cite{29}. 
Tall-skinny matrix operations appear in other contexts as well, such as QR decomposition or eigenvalue problems \[3, 6, 9, 80\]. Cache locality optimizations are among the most important optimizations required for tall-skinny problems. Cache locality driven thread scheduling (e.g. \[93\]) is a general way to block data efficiently. We have implemented similar ideas (Section 4.4.2 and 4.4.3) in our pipeline for better L2 cache reuse.

While other techniques for solving linear SVM exist (such as \[33\]), we use PhiSVM (which is derived from SMO-based techniques such as \[15\] and \[34\]) as it is fast and efficient for small SVM problems that we solve for FCMA. PhiSVM is also usable in other applications that require an efficient coprocessor-based SVM library.

The optimization ideas presented in this paper (such as tall-skinny matrix multiplication) are relevant and generalizable to a lot of other applications as well (e.g. \[80\]).

### 4.7 Summary

This chapter described the optimization of the most computationally intensive component of FCMA, the voxel selection, on the many-core processors such as Intel\textsuperscript{®} Xeon Phi\textsuperscript{TM} coprocessors. Our optimized implementation for a single node task on the coprocessor ran 5x-16x faster than an optimized basic version with Intel MKL and LibSVM libraries for two different datasets. Our optimization also improved the performance on the E5-2670 processor by a factor 2. In addition, we showed that our parallel code achieved near linear speedup on 5760 coprocessor cores. This work reduced the computing and analyzing time of the full correlation matrix in an fMRI dataset of the human brain to minutes for offline analysis, and seconds for online analysis. This latter finding made plugging FCMA into established closed-loop real-time fMRI studies possible (which will be discussed in Chapter 5).
Due to increasing amounts of parallelism as we move from multi-core to many-core architectures, optimizations to exploit these hardware features become increasingly important. This chapter optimized FCMA code by redesigning matrix multiplication for tall-skinny matrices, merging adjacent memory-bound stages in the procedure pipeline, and rewriting more vectorization-friendly SVM algorithms, in the consideration of making efficient use of the available hardware (cache and VPUs). The optimizations described in this chapter can be generalized as independent components that have many other applications. We also showed that most of the optimizations done on the coprocessor worked as well on the processor, although to a lesser degree, due to similar memory hierarchical structure and vectorization techniques. In addition, we believe our implementation can be migrated on to the next generation of Intel® Xeon Phi™ (KNL) with moderate effort.

In summary, we successfully optimized FCMA running on many-core processors which is applicable to multi-core processors as well. With the optimization done in this chapter, we are able to push FCMA to the real-time study which will be discussed in the next chapter.
Chapter 5

A Real-time FCMA System

5.1 Introduction

5.1.1 Motivation

As the dominant technique for investigating human brain activity in neuroscience research, currently almost all existing fMRI studies are conducted in an offline fashion – statistical analysis occurs only after all data have been acquired and sent to a file server or lab for processing, long after the research subject has been taken out of the scanner. This offline approach allows researchers to design an experiment, perform it in multiple subjects, and then analyze the reliability of the data by examining consistency across subjects. Although sufficient for many purposes, this approach has three major limitations: First, the pace of discovery is very slow, with typical fMRI studies lasting 6-12 months total, and often at least 2-3 months before even tentative results are known. Second, it is assumed that the same exact experiment should be run in each participant, missing opportunities to tailor the design (for example, level of difficulty) to an individual’s cognitive abilities and to assess whether enough or the right kind of data has been collected in a given session. Third, it misses the
opportunity to use information acquired during scanning as feedback to the subject, either to enhance participation and/or training.

Recently, it has become possible to conduct online or real-time fMRI (rtfMRI) studies, in which data are preprocessed and analyzed as they are collected, keeping pace with the rate of data acquisition [20, 69, 117]. As an example, rtfMRI with neurofeedback was recently used to train subjects to better sustain their attention, by externalizing their internal attentional states such that they could be better monitored and controlled [28].

Most analyses of fMRI data and, to date, all analyses used in rtfMRI, treat the activity observed in each voxel independently of one another. Examining patterns of activity in this way has led to considerable progress in brain research. However, brain function relies not only on the isolated activity of different areas, but as if not more critically on interactions between different brain systems, which can be reflected in correlations of activity among these. Such correlations, sometimes referred to as functional connectivity analysis, have become an increasingly important focus of neuroimaging research [109]. Just as FCMA has already produced novel insights when used for offline analysis in Chapter 3, patterns of correlation may be as, and perhaps even more, valuable for real-time analysis. For example, subtle changes in brain states associated with cognition (e.g., as a person thinks about or remembers something) may involve changes in the patterns of correlation among subsystems, without a change in the overall level of their engagement in real-time. This effect would not be observable in analyses of mean activity (using standard methods), but would be observable in an analysis of patterns of correlation (using FCMA). Therefore, it is a demanding requirement to also conduct rtfMRI experiments based on patterns of correlation.
5.1.2 Challenges

Despite the advent of rtfMRI, there still exists a major gulf between online and offline measures. One critical constraint concerns the type of analyses that are possible. Because of computational constraints, studying functional interactions in the human brain has typically been largely restricted. To address the limitation, in the previous chapters we proposed, implemented and optimized an exhaustive correlation analysis tool of the human brain named FCMA, allowing unbiased analysis of patterns of correlation in voxel activity over the entire human brain. In Chapter 3 we have shown that FCMA can reveal aspects of brain function not revealed by traditional, non-correlational methods (mPFC being identified in analyzing face-scene localizer dataset). However, due to the computational demands, to date it has not been possible to incorporate this method into studies of rtfMRI. In this chapter, we design and implement a distributed system to run FCMA in the back-end that is able to conduct the full correlation analysis in real-time, and provide proof-of-concept for its use in rtfMRI.

There is another problem that has constrained the use of rtfMRI – the lack of readily accessible, standardized tools for implementing it. In current neuroscience community, different research teams have to set up their own rtfMRI experimental environments, typically using a standalone machine sitting beside the scanner to receive and analyze the incoming data stream. This results in considerable duplication of effort, and often inefficient solutions, or ones limited by local resources. To address this problem, we describe a new, general distributed system for rtfMRI that runs on a compute cluster that can be deployed in Software as a Service (SaaS) mode, which will allow neuroscientists from MRI centers around the world to conduct their own real-time neurofeedback experiments, leveraging and creating shared computing resources. We have built the system to require very minimal resources in the scanner room, and the back-end, which resides on a local compute cluster or potentially a
cloud service provider, has a simple REST HTTPS API that allows easy integration and fits easily into existing network administration rules. The REST server is a gateway to the distributed back-end, which provides a flexible set of processes to handle classification, training, and incremental feature selection (voxel selection) in various topologies, with different experimental configurations, and using a variety of algorithms. The system provides multiple services simultaneously, allowing multiple rtfMRI experiments from different MRI centers to run in the same time. In addition, we have made the system be reliable in the presence of node failures during a real-time experiment, instead of having a failover which requires redoing the experiment.

5.1.3 Requirements

We designed and implemented the real-time FCMA as a neuroscientific application to our system (henceforth referred to as rtFCMA). We developed the incremental voxel selection algorithm based on the fully optimized batch FCMA code described in Chapter 4 by leveraging the characteristics of real-time data streams to only process the incoming data. RtFCMA is useful for showcasing our general framework, as it is an especially computationally intensive analysis that was not possible even in offline analysis until recently, and represents an upper limit on current computational demands for neuroimaging analysis.

Figure 5.1 shows the general concept of the rtFCMA system. An fMRI machine continuously scans the brain of a human subject while they perform certain tasks to generate a data stream of 3D brain volumes. The data are sent to rtFCMA that runs on a compute cluster. RtFCMA processes the brain volumes and produces feedback to send back to the fMRI scanner in real time to form a closed data transferring loop. Our system is the first ever rtfMRI system with low latency closed-loop feedback running on a compute cluster.
Figure 5.1: Conceptual diagram for rtFCMA, an fMRI system with closed-loop neurofeedback running real-time FCMA data analysis.

The rtFCMA system receives brain volumes from the fMRI scanner to do multiple processes. On one hand, it requires generating the neurofeedback with short latency. As an example, a typical fMRI scanner spends 1.5 seconds to produce a brain volume. To keep pace with the brain data stream from the scanner, rtFCMA needs to complete its process for a given volume (particularly correlation computation and multivariate classification) before the end of the next volume. The latency from the completion of a brain volume to its delivery to rtFCMA can be as high as 780 ms in practice, which leaves rtFCMA a time budget of about 720 ms (see Table 5.1 for time break-down). Although it is not a hard real-time requirement, i.e. missing some of the neurofeedback would only degrade the accuracy, our system meets the 720-ms requirement well.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reconstruction</td>
<td>500 ms</td>
</tr>
<tr>
<td>Transmission</td>
<td>174 ms</td>
</tr>
<tr>
<td>Preprocessing</td>
<td>106 ms</td>
</tr>
<tr>
<td>RtFCMA processing</td>
<td>720 ms</td>
</tr>
<tr>
<td><strong>Total TR length</strong></td>
<td><strong>1500 ms</strong></td>
</tr>
</tbody>
</table>

Table 5.1: The time break-down of a TR in rtFCMA. The durations of reconstruction, transmission and preprocessing vary across TRs, of which we take the maximal numbers to get a stringent estimate for rtFCMA processing time.
On the other hand, as an option, it is worthwhile to dynamically adjust the classification model according to the newly coming data. Updating the classification model has relatively looser latency requirement, but still expects the results, i.e. the updated model, in minutes at most. As a consequence, we design the system in the master-worker model, where the master coordinates different components as workers and dispatches the brain volumes to them accordingly. Experiments show that rtFCMA was able to conduct an rtFCMA study and provide neurofeedback with < 180 ms of lag. The incremental FCMA voxel selection running on our rtfMRI system performed 1.8x-4.3x faster than the offline batch version in the real-time context, while maintaining the same level of accuracy.

The rtFCMA system also needs to be reliable in the presence of node failures. The classification worker should be able to provide the neurofeedback in time even if it fails during the classification; the computationally intensive voxel selection component which the most nodes dedicate to should be able to update the voxel list and the model when some of the nodes fail during the process. We implemented schemes to tolerate node failures in rtFCMA. Working in the master-worker model, the master watches the voxel selection workers as well as the classification worker and once any of them fail, it restarts the failed ones and makes them recover the states. The reliability of the master is guaranteed by duplicating it and applying primary election via the Paxos protocol. Experiments show that our system continues to provide neurofeedback in time as well as performs voxel selection normally in the presence of node failures.

5.2 Real-time FCMA

5.2.1 FCMA in Real-time Study

As we discussed in the previous chapters, FCMA studies the fMRI data in terms of correlation between different brain regions. It is beneficial to incorporate FCMA into
rtfMRI studies to understand subtle changes in functional interactions in the human brain during different cognitive tasks as we discussed in Section 5.1. A closed-loop rtFCMA system can use a “pre-trained” model along with its “pre-selected” voxels to classify the incoming data in terms of correlation on the fly. Therefore only the classification component of FCMA is needed in the pipeline, which must return the classification result as the neurofeedback in a short time. However, in order to better characterize functional interactions in the human brain, it is advantageous to leverage the data collected from the current scanning subject to asynchronously update the model and the corresponding “pre-selected” voxels. As a result, the voxel selection component of FCMA also needs to perform efficiently.

A typical data processing of FCMA is shown in Figure 5.2. FCMA first picks informative voxels whose correlation with the whole brain is category-selective, then uses these voxels to construct correlation matrix of selected voxels to train an SVM model, which can be used for further classification based on the same selected voxel set. In the real-time context, the classification is in the closed-loop, while the training is done previously, or works as an auxiliary step.

The rtFCMA system is a second-order machine learning system. Unlike most distributed machine learning systems, it does not directly take the incoming data to form samples for training and classification. Instead of using the amplitude of voxel activity directly, it uses the correlation between voxel activities, so it must compute the correlation based on incoming data before applying any machine learning techniques. Additionally, using the correlation data as input for machine learning expands the data size by roughly three orders of magnitude. In order to meet the real-time requirement, high performance computing techniques are critical in this scenario.
Figure 5.2: The data processing flow of FCMA. The training updates the voxel list via voxel selection and uses the latest voxel list to update the SVM model; the classification uses the voxel list and SVM model generated from the training phase to classify.

5.2.2 RtFCMA Algorithm

The FCMA processing pipeline consists of voxel selection, model training and classification. Like in the offline mode, rtFCMA works in the master-worker model as shown in Figure 5.3. The master coordinates the selection workers to select voxels for training a model, and uses the model to classify samples in the classification worker. In order to perform FCMA in the real-time environment efficiently, we modify the existing FCMA voxel selection algorithm processing the fMRI data in batch to work in an incremental fashion. In the real-time setting, when a new brain volume comes in, we can leverage what we already have from previous computation so as to avoid re-computation as much as possible. Model training and classification of rtFCMA are processed similarly as in the offline mode. However, in the real-time context, no cross validation is needed to perform. Instead, the classification must be done quickly for
providing the neurofeedback in time. In this subsection, we describe how rtFCMA works component by component.

### Voxel selection

Voxel selection represents the feature engineering step in FCMA. It is a voxel-wise screening to pick the most category-selective voxels in terms of correlation. Specifically, for each voxel, the selection worker computes one correlation vector per epoch which consists of the correlation between the voxel and every other voxel in the brain; the correlation vector represents the row of the full correlation matrix corresponding to the voxel. To assess the prediction potential of voxels, the voxel selection worker builds for each voxel a machine learning model using the correlation vectors as training data. The voxel selection worker tests the accuracy of each voxel model using cross validation. It then sorts the voxels according to the accuracy of their models, which is an indicator of their category selectivity.

The algorithm can be described as a three-stage pipeline of *correlation computation*, *correlation normalization* and *cross validation* (see Figure 4.3). The correlation computation builds the full correlation matrix, composed of correlation vectors. The normalization stage brings data from different subjects to the same scale for between-subject analysis. The cross validation stage uses subjects as folds and trains a linear SVM classifier to test in each fold for each voxel. That is, for $m$ subjects, we build
linear SVM classifiers for each voxel using correlation vectors from \(m - 1\) subjects as training data and the \(m\)th subject as test data.

Since it needs to exhaustively go through all voxels of all subjects, voxel selection is the most computationally intensive component in FCMA. We have already optimized voxel selection for batch processing by carefully designing the processing pipeline and organizing the data layout to use the cache and vector processing units of the processor efficiently in Chapter 4, which works on both multi-core processors such as the Intel® Xeon® processor and many-core processors such as the Intel® Xeon Phi™ coprocessor. The batch algorithm works in a distributed way by sending different portions of voxels to different worker nodes via MPI to process in parallel. A master node will collect results from the workers to form the voxel list sorted by cross validation accuracy numbers.

However, the optimization above assumes that the algorithm reads all data in at once, which is not true in the real-time context. In rtfMRI studies, data come in brain volume by brain volume. The correlation computation happens every time an epoch of interest completes. Figure 5.4 shows the scenario in a worker node when the data of a new epoch of interest is completely received. Notice that this worker node is assigned \(V\) voxels by the master node. At that time, there are already \(M\) epochs’ data received and processed in the memory. When a new voxel selection takes place after receiving epoch \(M + 1\), we do not need to recompute the correlation and the SVM kernel matrices over all previous epochs of interest. Instead we only need to compute the correlation over epoch \(M + 1\) which is in dark red in Figure 5.4a, normalize it using the existing data of the same subject as depicted in Figure 5.4b, and compute the corresponding similarities with the existing correlation samples as shown in in Figure 5.4c. This saves recomputation overhead and overlaps the computation with the data acquisition.
Figure 5.4: Incremental voxel selection in one selection worker. The selection worker is assigned to process $v$ voxels of the brain, making sure all data resides in the memory in the entire process. (a) In the correlation computation stage, only the data of the newly coming epoch is computed; (b) In the normalization stage, only the data related to the newly computed correlation is normalized; (c) In the cross validation stage, when computing the kernel matrices, only the similarity values involving the newly normalized data are computed to expand the kernel matrices.

**Training with the latest data**

Training a precise model for real-time classification is critical to rtfMRI studies. Traditionally, researchers use a fixed model trained from previously collected data [28] because dynamically updating a model is usually not feasible in real-time by a single analysis machine. The fixed model does not take the latest brain data into consideration, so it may not reflect the neural status of the current subject, which therefore harms the neuroscientific studies.

In the rtFCMA algorithm, we implement a correlation-based training pipeline to incorporate the most recent data into the model. We first compute the correlation matrices of selected voxels consisting of correlations between the top-$K$ selected vox-
The sliding window of training samples. When the number of training samples reach a configurable threshold, the oldest sample will be discarded when a new one comes in.

In the real-time context, since the data keep coming in as a stream, it is impractical to enlarge the training set infinitely. Instead, we apply a moving window idea of the training data here. That is, when the number of training samples reach a configurable threshold, we start to discard the oldest training sample while incorporating the newly coming ones to maintain the training set scale as illustrated in Figure 5.5.

The training pipeline, together with the incremental voxel selection, work as independent processes to update the model asynchronously from classification.

**Real-time classification**

The voxel selection and training described above are only auxiliary components that refine the model. The critical path of closed-loop rtfMRI studies consists of using the
model to classify incoming data and sending the resulting prediction along with its confidence as neurofeedback back to the fMRI scanner. The classification must be done in real-time so that the neuroscientists can use the neurofeedback it produces for the subsequent steps in the experiment, e.g., adjusting the experimental task according to the fact that whether the prediction the system makes matches what the subject is really doing or not. Typically, an rtfMRI study requires the neurofeedback of brain volume $N$ to be produced within the time that brain volume $N+1$ is generated so that it can direct the cognitive task the subject performs when generating brain volume $N + 2$.

In rtFCMA, unlike the previous rtfMRI studies which work on the brain’s raw BOLD activity, the classification is done on the correlation values computed from incoming brain volumes over each epoch of interest. Because it is based on temporal correlation, classification is only performed after several brain volumes within the same epoch of interest are collected. Once it has accumulated enough brain volumes of the same epoch, the classification worker takes the top-$K$ voxels from the latest voxel list to compute a correlation matrix of selected voxels, normalizes the values using up-to-date data from the same subject, and applies the result to the latest model as a test sample. In order to run multiple correlation-based classifications for one epoch of interest, the system applies a sliding window of length $L$ to the epoch of interest. That is, each time a new brain volume within the epoch of interest is received, we drop the volume that was collected $L$ time points ago and incorporate the new one to compute the correlation. When $L$ is equal to the length of an epoch, we only conduct one correlation-based classification per epoch. The sliding window does not cross an epoch, so when a new epoch starts, the sliding window accumulation also needs to restart.

In summary, rtFCMA takes the fresh brain data to update the model and the voxel list on the fly, and always uses the latest model to classify the correlation computed
based on the current epoch including the newly incoming brain data. Experiments in Section 5.5.2 shows that the rtFCMA algorithms are able to conduct an rtfMRI experiment in terms of full brain correlation-based analysis.

## 5.3 Fault Tolerance

In an rtfMRI experiment, the brain volumes come in as a data stream. The system processes the streaming data to generate neurofeedback on the fly. Because of this online processing fashion, an rtfMRI experiment must proceed in the presence of node failure during the whole process. RtFCMA is designed to run on a number of compute nodes, there is no guarantee that all nodes can function healthily all the time. Therefore, fault tolerance is a “must have” feature to the rtFCMA system. In this section, we describe the fault tolerance design and algorithms to the rtFCMA system.

The design goal of the fault tolerance component is to make sure the system operates correctly, providing neurofeedback in time and updating the model as requested in the presence of node failures during the experiment. Specifically, the classification worker must provide neurofeedback within the 720 ms time budget even if it fails in the middle of a classification; the selection workers must be able to update the voxel list as well as the model when some of them fail during the voxel selection.

Fault tolerance in distributed systems has been well studied for decades [64, 26]. RtFCMA is a distributed streaming system in master-worker model with real-time requirement. So conceptually the design of it is not new. There exists some in-memory distributed streaming system with built-in fault tolerance features such as Apache Spark Streaming [124]. Apache Spark Streaming is a means of joining stream data with historical data or performing window-based operations on the stream. It chops a data stream up into mini-batches and processes them. However, Spark's
primary abstraction is the Resilient Distributed Dataset (RDD) \cite{123}, an immutable collection that provides fault tolerance but is inefficient at dealing with small changes to large data structures (e.g., updates to a small number of vectors in the correlation matrix).

Therefore, we argue that for our use case, it is easier to design some application specific schemes than using the existing generic ones. The fault tolerance ideas we apply to this system are not innovative but customized from state-of-the-art approaches to fit our application. Figure 5.6 shows the overview diagram of the fault tolerance component, which will be described in detail below. A complete fault tolerance component consists of two parts: fault detection and fault recovery.

5.3.1 Fault Detection

Our system deals with non-Byzantine fault. Byzantine failure includes the case that failed nodes generate arbitrary data and pretend the correctness. We assume that our nodes never return incorrect results handled by Byzantine fault tolerance \cite{73,82}. A node fails only when it becomes not responsive due to either node/process crash or network partitioning. In this case, monitoring the accessibility of the nodes via heartbeat messages is the most common scheme. Starting from the unreliable failure
detector [17], people developed several schemes and criteria to judge a node failure. The \( \phi \) accumal failure detector [50] introduced a threshold

\[
\phi = -\log_{10}(1 - F(\text{timeSinceLastHeartbeat})) \tag{5.1}
\]

where \( F \) is the cumulative distribution function of a normal distribution with mean and standard deviation estimated from historical heartbeat inter-arrival times, to determine if a node is failed. Gossip-style protocol [113] is a mechanism typically applied to large scale compute cluster to detect the failure in parallel. Falcon [76] and Pigeon [48] are recently developed fault detection approaches for more precise and more efficient fault detection to inform the applications in large distributed systems.

Since in an rtfMRI experiment instance, at most we have one master and tens of workers, we apply \( \phi \) accumal failure detector as the criteria to determine if a node is down. We use the master to monitor all workers, and use the master replicas (which will be described in Section 5.3.2) to monitor each other. Setting the \( \phi \) value is a trade-off between failure detection sensitivity (small \( \phi \) value) and failure detection correctness (large \( \phi \) value). The \( \phi \) value is configurable in our system. When the system runs on a local compute cluster which has fairly reliable connection between nodes, we set a small \( \phi \) value for more sensitive detection. If the system moves to the cloud, we can set a larger \( \phi \) value aiming at more reliable detection to account for the possible network issues.

5.3.2 Master Fault Recovery

The master node, which manages the entire processing pipeline, needs to be resilient to failure. In practice, people usually apply the replicated state machines [103] idea to identically duplicate the master in multiple nodes as active backups (replicas). Several distributed streaming systems [10, 104] used this idea to implement fault
tolerance. Among multiple master replicas, we need to elect a primary one. There
are a number of systems solving this classical distributed, asynchronous consensus
problem such as Viewstamped Replication [89, 78], ZooKeeper [54, 60], and Raft [90].
Paxos [71, 72] is a state-of-the-art protocol to handle this problem. Google’s Chubby
lock service [12] is a famous use case of Paxos.

We share the same idea as the Google Chubby lock service by deploying the
master in $K$ (which is an odd number) different nodes to form a cell. These $K$
masters (known as replicas) can access the same master state stored in a reliable
storage system. Among the $K$, we need to elect a primary one (depicted as the dark
circle in Figure 5.6) which directly talks to the workers and has the permission to
write states to the storage system. This can be done via a distributed consensus
protocol like Paxos. Once the primary master is down, one of the backup replicas can
be elected as the new primary master via the distributed consensus protocol. Since
we need a majority quorum to reach an agreement of the leading replica, a cell of $K$
replicas can tolerate $k/2 + 1$ failed masters at maximum. The number of replicas per

5.3.3 Worker Fault Recovery

The workers, including both voxel selection workers and classification workers, are
supervised by the master. Once a failed worker is identified, the master labels this
worker as inaccessible and starts to recover the state of the failed worker. The system
should be able to handle this inaccessibility and incorporate the recovered worker
once it comes back.

There are mainly two types of mechanism to recover a node from failure, replication
and rollback. Replication of workers is resource inefficiency especially when
the number of workers is large. Rollback, which replays the computation on a node

114
failure, saves computing resource but may take time to recover. Checkpointing \cite{74}, which periodically save the intermediate state of a node to a stable storage, enables the system to restart from the checked point instead of from the very beginning. However, since the intermediate states of our application, which consists of correlation data, are large, checkpointing them, even with more advanced methods such as concurrent checkpointing \cite{94} and incremental checkpointing \cite{44}, is not feasible. Instead, for recovering the correlation data state, we can apply the Upstream backup idea in distributed streaming systems \cite{23, 99, 2} by keeping only the raw data (upstream of the correlation data) for recomputation. This idea is similar to what most of the modern in-memory distributed systems do. In order to accelerate the rollback processing time, parallel recovery, a scheme in which multiple healthy workers work in parallel to speedup the process of recovering a failed worker, was used in several distributed streaming systems \cite{56, 14, 124, 121}. These solutions are more or less ad-hoc and introduce one more level of complexity to the system. As we will show later, in general we can recover the workers in time, and it is acceptable if the relatively long recovery of a voxel selection worker misses the next model updating. Therefore, we did not apply the parallel recovery idea to rtFCMA.

Since the essences of recovering a voxel selection worker and a classification worker are different, we describe their recovery design in separate as below.

**Voxel selection worker**

A voxel selection worker may maintain hundreds of gigabytes correlation data in memory when the model size is large. In order to save the data I/O throughput, we only checkpoint the raw fMRI data rather than the correlation data and recompute the correlation values on the fly when recovering the selection worker state. The raw fMRI data is much smaller, thus it poses less burden to the storage system. The recomputation of correlation data can be processed in batch instead of epoch by
epoch, therefore we can better leverage the optimization done in Chapter 4 to deal with it efficiently.

However, due to the large amount the correlation data could be, in the real-time context, we might not be able to recover the data to catch up the experiment process in time, namely, the failed selection worker may not yet come back in full to finish the currently assigned task when the system wants to update the model. We make the system continue functioning in this situation by discarding the voxels the failed nodes process. This is technically easy to implement but we need to make sure in terms of neuroscience we do not lose much scientific information due to discarding some of the voxels, even only temporarily, in the voxel selection.

When offloading different portions of voxels from the master to the selection workers, our original batch algorithm assigns voxels with continuous IDs which are physically adjacent to each other to a worker. In this case, once a worker fails, the losing voxels it processes correspond to a continuous brain area. According to the Figure 3.2 and 3.8, the informative voxels tend to be grouped in clusters. Therefore, if the nodes processing the clustered informative voxels are down and not back in time, we will lose the corresponding ROIs, which is not acceptable from the neuroscience perspective and may cause the degradation of the overall classification accuracy based on the top-N selected voxels.

A simple way to address this issue is to divide the voxels sent to the workers in an alternate way. Suppose we have $K$ selection workers for voxel selection, the master node can assign the voxels by partitioning them by the \textit{modulus} of voxel index over $K$. That is, \textit{worker 1} gets \textit{voxel 1, K+1, ..., worker 2 gets voxel 2, K+2, ..., etc.} By applying this method, no clustered informative voxels would be all lost when a selection worker is down. Since the voxels sitting besides each other tend to have \textit{spatial correlation} \cite{37}, the information loss can be easily compensated by the voxels around the missing ones. That is, on selection node failures, we essentially deal
with approximate brain volumes in a lower resolution while preserving the overall information well.

Apparently, the more selection workers an rtFCMA experiment recruits, the fewer voxels we will lose due to a selection worker failure. However, on the other hand, using more selection workers usually lead to larger communication and parallelization overhead. That is, adding more selection workers will start to get diminishing return at some point. As a result, we need to trade off these two factors by picking a proper number of selection workers assigned to an rtFCMA experiment.

In Section 5.5.3 we will show that, our system is able to function normally when one or more selection workers are down during the real-time experiment. The overall classification accuracy is not affected much when up to 25% voxels are missing, while the state of a selection workers can be recovered in minutes, which makes the corresponding voxels not gone forever.

**Classification worker**

Compared to the voxel selection worker, the classification worker runs in a much tighter time budget, namely, it must return the classification results as the neurofeedback in 720 ms. This makes the fault recovery of the classification worker be latency sensitive. After the master detects a classification worker failure, the system must recover the worker’s state, which essentially is the pre-trained model along with the to-be-classified sample, in a short time so that the neurofeedback can still be generated in time.

In practice, the state can be resent from the master as a binary stream and then deserialized from the worker side. However, the generic serialization works too slowly to meet the latency requirement. Thus, we designed a customized method to transmit the state between the master and classification worker in a way which directly maps the binary stream to the data structure of the model. This customized method, by
getting rid of unnecessary data packing/unpacking and copy, makes it possible for the master to recover the classification worker on the fly when a failure is detected under a tight latency requirement.

The evaluation in Section 5.5.3 shows that the classification worker node failure can be recovered to provide a neurofeedback in time for our tested datasets. Moreover, we demonstrate that multiple classification workers can be allocated to the same node while recovering all of them due to the unavailability of this node still meets the latency requirement.

5.4 System Architecture

5.4.1 Design

The aforementioned rtFCMA and fault tolerance algorithms would not practically function without an rtfMRI system. Our goal is to design and build a configurable system to support rtFCMA, and potentially, a lot of other rtfMRI analyses. The system should be configurable so that various processes involved can be assigned to whatever nodes by decision at launch time, and the communication and system organization are independent of the machine learning algorithms used. In this chapter, we configure the system to use linear SVM for classification, and FCMA as the analysis, but the framework of our system should allow any other algorithms that implement the interfaces the system requires.

The system should be able to provide multiple services simultaneously, allowing multiple rtFCMA experiments to be conducted in the same time. There are two options to assign computing resource for different rtFCMA experiments, either dedicating a small portion of nodes to each experiment exclusively, or using a large number of nodes to host multiple experiments in a shared fashion. We design the system in the latter way because of the following reasons. First, the classification worker
of an rtFCMA experiment does not consume much computing resource (only one correlation computation over the to-be-classified epoch followed by one classification computation), it is preferable to run multiple classification workers on the same node. Second, the voxel selection of an rtFCMA experiment only happens sporadically, so running only one voxel selection worker on one node will leave the compute nodes be completely idle when the current voxel selection is done before the next job comes in. Third, in general, running multiple experiments on the same group of nodes is a scalable solution because when the computing power of a node advances, more and more experiments can be assigned simultaneously without hurting the performance much.

In an rtfMRI experiment, the system receives one brain volume (< 500 KB) from the fMRI scanner every TR (1.5 seconds) and returns the neurofeedback (normally one or two single precision floating point values) to the scanner. Since the transmitting data size is small, it is easy to post the data from one side to the other via HTTPS.

On receiving the brain data, the system dispatches them to different workers for different purposes. Specifically for rtFCMA, the brain volumes either go the classification worker to get the prediction results as the neurofeedback, or to the voxel selection workers to assign accuracy number to each voxel for voxel selection and model updating, as discussed in Section 5.2. This behavior naturally fits the master-worker model, where the master is in charge of receiving the brain volumes and dispatching them to different workers conducting different analyses.

Among the workers, the classification worker is in the critical path of the closed-loop system, which must classify the brain volumes using a machine learning model (in this case, linear SVM) within a few hundred milliseconds, so that those results can be used as neurofeedback to direct the subsequent cognitive task assigned to the subject (e.g. showing which image to the subject). As an option, the brain volumes are also sent to the voxel selection workers for feature (voxel) selection so that the
system can update the classification model using the latest top voxels (described in Section 5.2.2). The actions of the voxel selection workers are not under the stringent time requirement like the classification worker, but it is still highly desirable that they could complete in at most minutes, so the outputs (models and voxel lists) can be leveraged as quickly as possible.

The fault tolerance algorithms designed in Section 5.3 is also based on the master-worker model, which fits our system design well. In general, the master watches all workers for failure and is guaranteed to be resilient itself by the multiple identical replicas with primary election. The specific worker recovery scheme depends on the task that worker processes. The system should allow the application programmers to configure if a node failure is expected to be recovered or should be discarded.

In summary, we design the rtFCMA system as Figure 5.7. The system can provide multiple services to accommodate multiple rtFCMA experiments by assigning the workers of different experiments to the same node. Each experiment instance uses a master node to communicate with the front-end fMRI scanner as well as coordinate the classification workers and voxel selection workers. In a closed-loop rtFCMA experiment, the master hands the brain volumes from the fMRI scanner to the classification worker for classification using a pre-loaded model. As an auxiliary step, the master also sends the brain volumes to a number of voxel selection workers for getting the latest voxel list so that it can update the model of the classification worker asynchronously. Finally, the master stores all the incoming brain volumes to a storage system for offline analysis in the future.

5.4.2 Implementation

In practice, rtFCMA uses the actor model [52], as implemented in the Akka [11] toolkit, to distribute computation across a compute cluster. The master, voxel selection worker, and classification worker are all abstracted as actors in Akka. These
Figure 5.7: The rtFCMA system architecture. The system is able to handle multiple rtFCMA experiments simultaneously. Via Internet, the scanners send brain volumes to the system and the system returns neurofeedback. An experiment instance runs in the master-worker model. One compute node can run multiple workers from different experiments.

actors are written in Scala and are lightweight and focused only on communication and management of the system.

The services rtFCMA provides are accessed using a RESTful web API via HTTPS. The REST server is implemented using the Spray framework[^1]. It does relatively little processing itself, instead it relies on the distributed actors in the system. There is a reaper runs on the fMRI computer. Its job is to send brain volumes to the REST server (using HTTP POST requests) as they are generated by fMRI scanner. The intent is that this should be a minimal daemon process that does not interfere at all with the normal operation of the fMRI scanner. The reaper is written in Python, and is borrowed from the SciTran project [102] with minimal modification.

Requests received by the web server are dispatched to various actors for processing. Although the actors facilitate communication in the system, for good performance of an actor system, it is important to have both small message sizes (no more than a few kilobytes) and to do all time consuming work on a different thread. Thus as messages

---

[^1]: http://spray.io
that require time to process arrive at one of the system’s actors, the message will be validated and then queued for processing by a different thread dedicated for that purpose, so that the actor is free to respond to the next message. Furthermore, the heavy analytical computation in the system is delegated via Java Native Interface (JNI) wrappers to C++ routines (in this case, rtFCMA algorithms) that analyze the brain volumes. As a matter of design, the system is built to be extended to use many different algorithms, not only FCMA. Therefore the actors communicate with the algorithm-specific code via algorithm-agnostic interfaces.

We then describe three different components of rtFCMA system one by one:

1. **Master**: The master is the hub of the experiment. It is the coordinator that sees the brain volumes through their life-cycle. It creates operational actors to implement the needed functions, assigning them to remote nodes based on configuration files. In the setting of rtFCMA, master creates classification worker for closed-loop rtFCMA analysis, as well as selection worker for voxel selection if required. When the selection worker is on, the master also updates the model via the latest voxel list to send to the classification worker. Since the master does relatively light jobs such as coordination and communication, in practice it can be deployed in the same node shared with other workers. In addition, note that there can be multiple experiments active at the same time, each with its own master created on-demand.

2. **Selection worker**: The selection worker’s job is to determine which voxels in the brain volumes are category-selective. The master typically creates a fleet of selection worker to process all voxels in parallel. These are analogous to (and share C++ code with) the voxel selection workers in the MPI job of the offline version of FCMA described in the previous chapters. Even in the original offline C++ version of FCMA, the communication required between voxel selection master and voxel selection workers is simple and does not use any advanced
features of MPI. With this in mind, we abandoned MPI and used Akka messages to distribute the work and collect results, which simplifies the deployment requirements for the system. Additionally, this simplifies the implementation on fault tolerance, as Akka, with its heritage from Erlang, is better suited to building fault tolerant systems than MPI.

3. **Classification worker**: The classification worker receives brain volumes and passes them through a pre-trained machine learning model to classify them. For this chapter, it is configured to use linear SVM. Its classification results are sent back to the master for storage until a client, such as the subject neurofeedback system, requests those results.

The fault tolerance algorithms are implemented based on the structure of these three components as well. The master crash is handled separately using primary election implemented in Paxos protocol. In practice, we duplicated the master to 3 different nodes, and made a specific actor in the master dedicate to the primary election. The state of the master stores in a reliable storage system GlusterFS\(^2\). All master replicas can access the state, but only the elected primary one has the write permit and will take charge of communicating with the workers. Once the primary master fails, a new election takes place and a new primary will be elected to take over.

In an rtFCMA experiment, the master watches the voxel selection workers and classification workers all the time. Once a worker node failure happens, the master restarts it accordingly. For the voxel selection worker, it is recovered by recomputing the correlation from the raw data it checkpointed as the “upstream”. The master does not necessarily wait until the failed selection worker has come back before proceeding to update the model, it can proceed to update the model without the voxels assigned to the failed node. As we discussed in Section 5.3.3, the more selection workers an

\(^2\)https://www.gluster.org/
rtFCMA experiment has, the fewer voxels are lost due to a worker node failure, and thus the less work it needs to recover the worker. Therefore, in practice we assign a large number of selection workers to an rtFCMA experiment. For a single experiment, it seems to be a waste of resource to do so since each selection worker may only have a light workload. However, one worker node can be shared by selection workers from multiple experiments running simultaneously which will make the resource efficiently used.

Regarding the classification worker, it is recovered by restoring the pre-trained model. When a classification worker node has multiple classification workers running in the same time, multiple pre-trained models need to be all restored. We implemented the model restoration using a customized direct mapping from the binary stream to the data structure of the model, instead of the packing and unpacking scheme provided by the generic serialization approaches. The customized method improves the model restoration performance by one order of magnitude.

5.5 Evaluation

5.5.1 Experimental setup

The evaluation was done on the same cluster Metacortex as we used in Section 4.5. The same two datasets in Section 4.5 (face-scene and attention, details can be found in Table 4.2) are used to simulate the rtfMRI studies.

In general, the evaluation is divided into two parts, one is to test out the performance of the rtFCMA algorithms, the other is for the performance of the fault tolerance. In the rtFCMA evaluation, we compared rtFCMA and the original FCMA toolbox described in the previous chapters, which we refer to as batchFCMA. Regarding the fault tolerance component, we tested on the rtFCMA system. We will evaluate these two parts one by one in the next two subsections.
5.5.2 RtFCMA Evaluation

To evaluate the performance of the rtFCMA algorithms, we pursued answers to the following questions:

1. How many selection workers does an rtFCMA experiment need?

2. What is the overall performance of the rtFCMA system? Specifically, can the classification always produce neurofeedback within the time of generating the next brain volume? Can the incremental voxel selection and training update the model efficiently?

3. What is the performance of rtFCMA system when multiple experiments are conducted simultaneously on the same nodes?

4. What is the performance gain of incremental voxel selection comparing to the optimized offline version?

5. Can the simulated real-time experiments achieve the comparable classification accuracy as the offline analysis?

RtFCMA experiment configuration

First of all, we need to specify the number of workers an rtFCMA experiment needs. As we discussed in Section 5.4.1, an rtFCMA experiment has only one classification worker but a fleet of selection workers which run on different nodes. There is a trade-off of how many selection workers should be recruited. The more selection workers an rtFCMA experiment has, the less workload is assigned to a selection worker, and consequently fewer voxels are lost due to a selection worker failure. On the other hand, more selection workers also lead to more communication and parallelization overhead. In addition, similar to the idea of Amdahl’s law, the maximal speedup our
Figure 5.8: (a) The voxel selection speedups of both face-scene and attention datasets as a function of the number of selection workers. The performance of 10 selection workers was normalized to 1. Face-scene got 2.60x speedup at 40 nodes and 2.64x speedup at 45 nodes; attention got 2.97x speedup at 40 nodes and 3.05x at 45 nodes. (b) The single node processing percentage of both face-scene and attention datasets as a function of the number of selection workers.

The system can get by adding more selection worker nodes is constrained by the portion of work that needs to be done using only one node.

The datasets we used have 25,000 – 35,000 voxels per brain scan. We would like to figure out the best number of selection workers that trades off the two factors above in this scale. Figure 5.8a shows the speedup of running voxel selection based on a pre-trained model attained from N − 1 subjects (where N is the number of subjects in a dataset) in both datasets by varying the number of selection workers used. At the end of the voxel selection, the system would need to update the model using the latest top voxels, which was processed using one node so would not benefit from including more nodes. The time of model updating became more and more significant as increasing the number of nodes to work on voxel selection, which caused the speedup decrease. Specifically, when using 40 selection worker nodes, 50.5% of the processing time on face-scene and 23.0% of the processing time on attention are dedicated to model updating (Figure 5.8b). Attention dataset got more speedup because it had more computation.
According to the results, we chose to use 40 selection workers per rtFCMA experiment in the subsequent evaluations since after 40 nodes we observed an obvious diminishing return when adding more nodes. We will show that our system is able to generate similar results as the offline classification while meeting the real-time requirement.

**Full system performance**

We then demonstrated the performance of the rtFCMA system. Two datasets mentioned above were used in this experiment. Note that in a practical real-time experiment design, a subject would typically go through multiple experimental runs. However, in these two simulated datasets, there was only one experimental run per subject. So we simulated the real-time experiment using multiple subjects’ data.

In order to fully simulate an rtfMRI experiment, we fed in our system brain volume by brain volume with a 1.5-second pause, which is the time interval of a brain volume being generated in the fMRI scanner; and inserted a 2-minute pause between subjects, which is a reasonable duration in practice between experimental runs. Since the scale of the model rtFCMA needs is still under discussion by neuroscientists, we tested out the performance of the incremental voxel selection and classification based on various model sizes, in which the model was constructed by 2 to $N - 1$ subjects, where $N$ is the number of subjects in a dataset (18 for *face-scene* and 30 for *attention*).

In practice, our system started from scratch, that is, there was no pre-trained model that could be used. Our system created the model and updated it using the incoming data stream while the real-time experiment was taking place. Like the batchFCMA, in voxel selection, our system did a leave-one-subject-out cross validation. The system invoked the voxel selection workers at the end of each epoch to incrementally compute the correlation. At the end of each subject, the voxel selection workers updated the voxel list based on the voxels’ accuracy numbers obtained
from the cross validation. Then a new model with the latest top-500 voxels can be constructed for the usage of the classification worker. We used top-500 voxels to construct the model since the number is enough to depict reasonable patterns of correlation according to the accuracy numbers reported in chapter 3 and [55].

Figure 5.9 shows the incremental voxel selection and model updating performance in our system using 40 voxel selection workers for both face-scene and attention datasets. The rtFCMA system always updated the voxel list and the model using the data of currently collected subjects. As more and more subjects’ data were collected, the model size grew larger and larger, and correspondingly the time it took to select voxels and update the model got longer and longer. The voxel list and model were updated after every subject but the correlation data were computed after each epoch to overlap the computation with the data acquisition. From Figure 5.9 we can tell that the model updating was able to complete before the next subject came in (2 minutes). That is, the classification worker always used an up-to-date model that was updated by the data from the latest subject.

The classification worker took the latest model along with the top-500 voxels in its corresponding voxel list to classify the correlation sample computed over a sliding window which included the newly incoming brain volume. In our experiment, we set the sliding window size to be 8, that is, when the number of TRs within the same epoch got to 8, the system started to compute correlation to classify, and when the 9th TR came in, the 1st TR was dropped to maintain the length of the time course over which the correlation was computed to be 8, so on and so forth until the end of the epoch.

Figure 5.10 shows the classification performance of our system on two datasets. We used a dedicated node to run the classification worker to prevent the unexpected interference introduced by other components of the system. On the other hand, since the classification computation is not heavy, we assign 8 threads of the node to run
Figure 5.9: Duration of voxel selection and model update as a function of the current model size. 40 selection worker nodes were used. Top: face-scene, bottom: attention one classification. As we mentioned in Section 5.1, although the time interval of generating a brain volume in our experiment is 1.5 seconds, the time budget left for classification is less than that considering the latency of other necessary steps in the rtfMRI pipeline, e.g. reconstruction and preprocessing, which in practice took up to 780 ms (see Table 5.1). Therefore, we set a deadline of 720 ms to the classification. From the figure we can see that the classification in our system always finished faster than 180 ms, which is well bellow the deadline, indicating that our system is able to satisfy the latency requirement of closed-loop real-time fMRI studies. As the model size increased, the classification time got longer in general although the fluctuation caused by the network transition existed.
Figure 5.10: Duration of classification as a function of model size. 1 classification worker node was used. Top: face-scene, bottom: attention

It should be noted that measured times of Figure 5.10 did not include network communication overhead between the scanner and the REST server. Even if we consider cross-continental links, if we use a conservative estimate of 200 ms (for instance based on [19]) for each HTTP call, and assume the feedback system polls the REST server for results every 100 ms, the overhead should be within 500 ms. Since the classification worker completed in less than 180 ms, the system meets the 720 ms deadline, let alone the additional care, such as use of keepalives or server side events (SSE) could well improve these numbers.
Performance of multiple services

Next, we evaluated the system performance when multiple services were running simultaneously. A service corresponds to one rtFCMA experiment instance in our case. As above, one classifier worker node and 40 selection worker nodes were used. We launched three services in total, two of them ran on the face-scene dataset, the rest one ran on the attention dataset. Each rtFCMA experiment was conducted based on a pre-trained model attained from $N-1$ subjects, where $N$ is the number of subjects a dataset has. One subject worth of data streamed through the system. Multiple classifications took place for each epoch of interest (sliding windows size is 8 as above). At the end of the subject, one voxel selection was performed. Table 5.2 shows that performance of voxel selection and classification in each experiment, also summarizes the same performance numbers when only one experiment was conducted (essentially the same results reported in Figure 5.9 and 5.10). The classification durations were the averaged across all classifications. From the table we see that the performance of classification was not affected much when multiple experiments were running in the same time because one classifier worker instance did not consume all computing resource of a worker node. The performance of voxel selection dropped a little when two face-scene and one attention experiments were conducting in the same time. However, the resulting performance was acceptable. In practice, because voxel selection only happens sporadically, allocating voxel selection workers belonging to different experiments to the same selection worker is feasible. Most of the time these selection workers will not consume the computing resource simultaneously.

Performance gain of incremental voxel selection

We then compared the performance gain we achieved using the incremental voxel selection algorithms. Our incremental voxel selection took advantage of the characteristics of data stream by accumulating and processing the data when they are
Table 5.2: The system performance when multiple services, specifically, two services of *face-scene* and one service of *attention*, are provided simultaneously. The performance numbers of single service are also listed for comparison. 40 selection worker nodes and 1 classification worker node were used.

Coming in. On the other hand, the offline algorithm, although fully optimized, had to read in all the data before process, which introduced a lot of overhead in the real-time context.

Figure 5.11 shows the performance comparison between the voxel selection of rtFCMA and batchFCMA in the last subject of both datasets, that is, doing leave-one-subject-out cross validation on 18 subjects of *face-scene*, and 30 subjects of *attention*. Like the full system performance evaluation, we used 40 selection workers for both *face-scene* and *attention* datasets, on both rtFCMA and batchFCMA. Notice that in the batch version, we excluded the time it took to read data in from disk and broadcast them to all worker nodes. The incremental voxel selection worked 1.8x-4.3x faster than the batch version. The performance gain was larger in *face-scene* because the correlation computation, which was handled by rtFCMA incrementally, took a larger portion there than in *attention*.

**Effectiveness of rtFCMA system**

To verify the effectiveness of our system, we compared the accuracy results it generated on both datasets with the results batchFCMA generated using leave-one-subject-out cross validation. In rtFCMA pipeline, we sent the data brain volume by brain volume to the system, eventually selected voxels and trained a model using $N - 1$ subjects (where $N$ is the number of subjects in a dataset) and used this model to test
on the last subject. We did this for $N$ times to allow all subjects to be tested. In batchFCMA, we replicated the pipeline used in chapter 3. The top-500 voxels were used in both cases.

Figure 5.12 shows the accuracy results of both datasets produced by rtFCMA and batchFCMA. In order to match what batchFCMA did, in rtFCMA we set the sliding window size of the classification to be the length of the epoch, that is, only one classification per epoch, which was based on the correlation over the entire epoch. We can see that our rtFCMA system got very similar results in both datasets to batchFCMA, which verified the correctness of the rtFCMA system. The results were slightly different because in the correlation normalization steps, rtFCMA could only normalize the data based on what it received so far, while batchFCMA used all data within the same subject to do the normalization.
5.5.3 Fault Tolerance Evaluation

To evaluate the performance of the fault tolerance component of the rtFCMA system, we pursued answers to the following questions:

1. How long does it take to detect a master node failure? In the presence of the primary master node failure, how long does it take to elect a new primary master node and restore the state?

2. How long does it take for the system to detect a failed classification worker node? How long does it take for the system to recover a classification worker node? How many classification workers can be allocated to a node so that all of them can be recovered to provide neurofeedback on time?

3. How long does it take for the system to detect and recover a failed selection worker node? If some of the failed selection worker nodes are not recovered when the system needs to update the model, will it largely affect the overall classification accuracy made by the real-time classification worker?
Master fault tolerance

We first evaluated the failure handling time in the presence of a master node failure. We used 3 master replicas in the system. When launching the system, the primary election implemented in Paxos protocol elected a replica as the primary one to work as the master node. Once the primary master node failure was detected by the replicas, a new round of primary election took place to elect a new primary master. The new master read the current state from a stable storage system to restore the functionality of the master node. Table 5.3 shows the time break-down to handle a master node failure in both face-scene and attention datasets. In general, it took less than 700 ms to recover from a master node failure for both datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Detection</th>
<th>Election</th>
<th>Restoration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face-scene</td>
<td>70</td>
<td>494</td>
<td>105</td>
</tr>
<tr>
<td>Attention</td>
<td>75</td>
<td>502</td>
<td>110</td>
</tr>
</tbody>
</table>

Table 5.3: Time to handle a master failure using Paxos protocol, which is divided into detection time, election time and restoration time. Unit: millisecond

Classification worker fault tolerance

We then evaluated if the classification worker failure of the rtFCMA system can be handled quickly to provide the neurofeedback in time. On a classification worker failure, the master detected it then recovered by reloading the model state. Table 5.4 shows the time it took to handle a classification worker failure for both face-scene and attention datasets when \( N - 1 \) subjects’ data were used to construct a model to classify the samples of the last subject, where \( N \) is the number of subjects a dataset has. The failure detection times were similar for both datasets. It took longer time to recover a classification worker failure for attention because the model size of attention was larger. In general, the classification worker failure could be handled quickly to provide the neurofeedback in time for both datasets.
Table 5.4: Time to handle a classification worker failure when $N - 1$ subjects were used to construct the model. The time is partitioned into two parts, fault detection and fault recovery. $N$: the number subjects of a dataset. Unit: millisecond

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fault detection</th>
<th>Fault recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face-scene</td>
<td>75</td>
<td>80</td>
</tr>
<tr>
<td>Attention</td>
<td>70</td>
<td>200</td>
</tr>
</tbody>
</table>

For the sake of resource usage efficiency, a classification worker node may handle multiple classification workers. A classification worker has its own model to classify the incoming brain volume to make a prediction. The feasibility of running multiple classification workers on the same node depends on the time it takes to recover all running classification workers once the node fails. Table 5.5 summarizes different classification worker node failure recovery times when multiple classification workers were running simultaneously. In general, more classification workers lead to longer recovery time because more model states need to be transmitted. From the table we can tell that if 720 ms is the deadline, and the classification takes up to 180 ms, we can at most run 4 classification workers for *face-scene* and 2 classification workers for *attention* on the same node in order to catch the deadline in the presence of a classification worker node failure.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>1 worker</th>
<th>2 workers</th>
<th>3 workers</th>
<th>4 workers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face-scene</td>
<td>80</td>
<td>152</td>
<td>239</td>
<td>304</td>
</tr>
<tr>
<td>Attention</td>
<td>200</td>
<td>380</td>
<td>577</td>
<td>747</td>
</tr>
</tbody>
</table>

Table 5.5: The classification worker node recovery time when different numbers of classification workers are running simultaneously for *face-scene* and *attention* datasets. Unit: millisecond

Voxel selection worker fault tolerance

Next, we tested out the resilience of the voxel selection workers of the rtFCMA system. Table 5.6 summarizes how much time it took to detect and recover the failure of a voxel selection worker once it failed for both *face-scene* and *attention* datasets when
$N - 1$ subjects’ data were processed for the voxel selection, where $N$ is the number of subjects a dataset has. Like in Section 5.5.2, we used 40 selection worker nodes for both face-scene and attention datasets. As discussed in Section 5.3.3, recovering the state of a selection worker is essentially recomputing the correlation data from the raw fMRI data. From the table we can see that compared to the fault recovery time of a selection worker, the fault detection time was negligible. The total failure handling times for both datasets were acceptable compared to the interval of doing a voxel selection (2 minutes in our case).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fault detection</th>
<th>Fault recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face-scene</td>
<td>0.08</td>
<td>9.89</td>
</tr>
<tr>
<td>Attention</td>
<td>0.08</td>
<td>15.53</td>
</tr>
</tbody>
</table>

Table 5.6: The voxel selection worker failure handling time for face-scene and attention datasets. 40 selection worker nodes were used. Unit: second

We also evaluated the overall classification accuracy change when some of the selection workers failed during the real-time experiment. We ran the same experiment as in Section 5.5.2 (Effectiveness of rtFCMA System), using 40 selection worker nodes and randomly killed some of them on purpose during the experiment. When a selection worker was killed, the master recorded it and ignored the voxels assigned to this worker, i.e. the lost voxels were excluded when picking the top voxels to update the model for classification.

Figure 5.13 shows the accuracy change as a function of the number of failed selection workers during the experiment. From the figure we can tell that, by assigning the voxels in an alternate way described in Section 5.3.3, the overall accuracy of the real-time classification did not degrade much when a portion of selection workers were down during the experiment. Even if 25% (10 out of 40) of the selection workers failed, which should not be common during an experiment in practice, the overall accuracy was still reasonably stable. The results indicated that if some of the voxel selection workers were gone during the rtFCMA experiments and did not come back
Figure 5.13: Classification accuracy change as a function of the number of selection workers fails. In total there were 40 selection workers launched, a portion of workers (1, 3, 5, 7, 10, respectively) went down during the experiment.

before the next model updating, the rtFCMA system could be still operational by discarding the voxels assigned to the failed selection workers temporally. From the neuroscience perspective, the results also demonstrated that the top voxels picked by FCMA have redundant information because of the spatial correlation characteristics of fMRI data, which points a potential future direction of FCMA to further reduce the number of selected features.

5.6 Related Work

In this section, we discuss some of the important related work that is not covered in the previous sections.

Though modern rtfMRI systems are capable of producing full brain scans every 1-2 seconds, analysis of these scans is typically performed offline. Incremental analysis of rtfMRI is a very recent advancement, motivated in part by a surge in neurofeed-
back research [107]. In one recent study [28], a whole brain-classifier was trained on single-subject voxel activity patterns by batch processing the data (on a single machine) between experimental runs while the subject was in the scanner. The classifier was used to predict whether a subject was attending to a face or a scene within a composite image presented to them and to reinforce the desired attentional behavior by changing the mixture of the composite stimulus. RtFCMA extends the features considered by the classifier to full-brain correlation patterns ($O(n^2)$ features instead of $O(n)$), automatically selects the most predictive voxels in terms of correlation in an incremental fashion, and uses the selected voxels to train classifiers to predict the incoming fMRI data from the correlation perspective. None of this would be possible without a parallel and distributed implementation of the algorithms.

Of course, many frameworks for stream-based parallel and distributed machine learning exist. These are largely memory-based systems that are efficient at iterative processing on streams such as Apache Spark Streaming [124]. Recently, the Spark machine learning library, MLlib, was extended with streaming versions of linear regression and the k-means clustering algorithm [83]. Distributed stream processing engines (DSPEs), like Apache Storm [2], S4 [87], and Samza [1], have emerged that are designed from the ground up for stream processing data in a reactive manner, one update at a time. The Apache incubator project SAMOA (Scalable Advanced Massive Online Analysis) provides a machine learning framework and library that utilizes DSPEs but abstracts away the execution engine.

Like some modern DSPEs and the original version of Apache Spark, rtFCMA uses the highly concurrent and resilient Akka runtime [11] to distribute messages (Akka is largely based on and inspired by Erlang [8]), and like SAMOA we represent our algorithms with a directed graph of nodes that communicate using messages. To the best of our knowledge, rtFCMA is the first to implement any form of stream-based
machine learning techniques on fMRI data and the first streaming implementation of full correlation matrix-based classification for any application.

The incremental voxel selection algorithm that we describe is more generally a form of feature selection. Feature selection is a very common technique in machine learning, often used to increase model accuracy, improve generalization of the model, and to speed up model training. Clustering techniques, component analysis, spectral transforms, and other methods are used broadly for fMRI data classification [120]. Though typically implemented as offline algorithms, many have incremental versions that can scale with parallel and distributed computing resources. However, data-driven feature selection using pairwise voxel correlations over the entire brain is unique to FCMA and more computationally challenging to implement incrementally than the classic techniques due to the large amount of memory and number of SVM models involved. As far as we know, our cluster instantiation of the pipeline is the most parallelized feature selection model ever applied to fMRI data.

5.7 Summary

This chapter proposed, designed and implemented a distributed system for the real-time full correlation matrix analysis of fMRI data. Our system is able to apply FCMA to process and analyze each brain volume and send a neurofeedback to the fMRI scanner within 1.5 seconds. The system also implements an incremental version of voxel selection of FCMA, allowing dynamical updating of the voxel list and the classification model in at most minutes using a compute cluster. In addition, this system tolerates various node failures to function resiliently during an rtfMRI experiment.

Our system is designed to be deployable as a service in the cloud, which potentially allows neuroscientists around the world to conduct their own real-time experiments. As a showcase, our system uses FCMA as the analysis application. However, other
analysis applications can be conducted using the same system framework as long as they implement the interfaces the system requires. We believe this distributed streaming system will help accelerate the pace of discovery in fMRI-based neuroscience research.
Chapter 6

Conclusion

6.1 Contributions

In this dissertation, we studied a relatively uncharted territory in neuroscience — the unbiased analysis of functional interactions in the human brain. We proposed, designed and implemented full correlation matrix analysis (FCMA) to explore functional interactions in the human brain across cognitive tasks. We made the following contributions:

- We proposed, designed and implemented a new method called full correlation matrix analysis (FCMA) to exhaustively study the functional interactions in the human brain under different cognitive tasks. FCMA extracted patterns of correlation, instead of the well-studied patterns of activity, from fMRI data in an unbiased way. Running on a multi-processor compute cluster of 66-nodes, FCMA improved the performance over four orders of magnitude (12,855x) compared to a naive, sequential implementation, making unbiased brain correlation analysis possible. We wrapped FCMA, along with some other correlation-related functionality into the FCMA toolbox, which is open source and available to the
neuroscience community, in hope that researchers will explore these exciting avenues and help further evaluate the utility of this approach.

- We conducted FCMA with a simple face-scene localizer dataset and successfully identified previously vague functional interactions between certain brain regions. Specifically, we demonstrated that mPFC participates in the object recognition via interacting with other regions of the brain, which may not be revealed in traditional methods. We performed multiple statistical analyses to reassure this finding. We also proposed several methods to characterize the interaction between the newly identified mPFC and the well-studied visual cortex in object recognition tasks, which can be reused in other similar correlation-based studies. Our case study suggests a broader research program to examine the patterns of correlation between voxels, highlighting the potential of FCMA.

- We optimized the most computationally intensive component of FCMA, the voxel selection, on the many-core processors such as Intel® Xeon Phi™ co-processors. The architecture of many-core processors is similar to the regular multi-core processors, but with smaller cache and on board memory sizes, wider vector processing units, and a larger number of cores. We proposed several optimization ideas accordingly to accelerate our code running on the many-core processors by a factor of 5 to 16. Our optimized code also had performance gain while porting back to the regular multi-core processors, running 1.4x-2.5x faster than the basic implementation. Our optimization ideas are generalizable to other applications on the many-core processors as well.

- We proposed, designed and implemented a real-time fMRI (rtfMRI) system running on a compute cluster as the first attempt of building a framework to support rtfMRI studies in general. We developed the rtfMRI system to be configurable with different kinds of fMRI analysis applications. Particularly, as
a showcase, we built an rtFCMA system which processes the incoming fMRI data stream in terms of full correlation. The rtFCMA system can process and analyze brain volumes from multiple fMRI experiments on a 40-node compute cluster simultaneously and send the neurofeedback to each fMRI scanner over the Internet within 1.5 seconds. In addition, it performed 1.8x-4.3x faster than using the offline FCMA to conduct the incremental voxel selection. We also implemented the fault tolerance component in the rtFCMA system to make sure the real-time experiment can proceed in the presence of node failures.

In summary, this dissertation made the previously intractable full correlation matrix analysis over the whole brain not only possible in offline analysis, but also applicable to real-time fMRI study. We built an offline FCMA toolbox as well as a configurable and reliable online distributed system that runs rtFCMA as a showcase. As the offline case study of this dissertation shows, FCMA starts to add new insight to cognitive neuroscience.

### 6.2 Future Work

Even though we have made substantial progress to bring the exhaustive analysis of functional interactions in the human brain from impossible to feasible even in real-time in the dissertation, this field is still young and requires much further effort. In addition, when exploring the full correlation analysis via various computer science techniques, we also see some shortcomings of current implementations. We envision the future work in at least the following directions.

- The analysis of functional interactions in the human brain is more than what we discussed in the dissertation. Current FCMA applies voxel selection to find the informative voxels in terms of correlation with the full brain without pointing out the specific regions the informative voxels correlate with, which should be
improved by more sophisticated algorithms. Moreover, FCMA does not extract
and process latent features of the fMRI data. The functional interactions ex-
pressed by the latent features would be interesting to study as well [79]. Last, in
this dissertation we described the functional interactions using Pearson product-
moment correlation, the exploration of applying other correlation metrics to the
functional interactions in the human brain still remains.

• Since the exhaustive study of functional interactions in the human brain is
new, there are not many neuroscientific experiments designed and conducted
for it. Throughout the dissertation, we only had two datasets to do the eval-
uation. And by now, no real-time neuroscience experiment for rtFCMA has
been established. In the future we should design and run real-time neuroscience
experiment using our rtFCMA system. It eventually requires the help from
the entire neuroscience community to collaboratively improve the existing tool-
box and real-time system to facilitate the neuroscientific research on functional
interactions in the human brain better.

• Our optimization work on FCMA shows the great potential of running applica-
tions on the many-core processors such as Intel® Xeon Phi™ coprocessors. As
many-core architecture becomes more and more popular in the high performance
computing (HPC) area, we think that developing a more general framework for
efficiently running a variety applications on the many-core processors would be
compelling, given the legacy code ported from the regular multi-core processor
to the many-core processor generally works less well.

• As our rtFCMA system is the first known effort to build an rtfMRI service, there
remains a great detail of future work. The system will be pushed to the cloud
(e.g. Amazon Web Services, AWS) so as to be easily accessible to neuroscien-
tists all over the world. Accordingly, the security issue while transferring data
to/from the cloud and storing the data remotely needs to be seriously addressed since the brain imaging data are typically sensitive. In addition, the system still needs a general design for the fine-grained resource allocation. Our rtFCMA system framework is also potentially generalizable to other applications with similar requirements.

As a closing remark, we think the interdisciplinary research between computer science and neuroscience is a very promising area. The advance of computer science techniques enables more neuroscientific discovery and innovations, which in turn drives innovations in computer science from machine learning to parallel and distributed computing. This dissertation shows how the research of functional interactions in the human brain interacts with the advanced computer science techniques. We envision more breakthroughs along this direction.
Bibliography


