Local Field Potentials Serve as a Better Predictor of Reaching Behavior than Calcium Fluorescent Intensity in the Primary Motor Cortex

Devin Chang, Pawel Vijayakumar, and Claire Gao

I. Abstract

We found that calcium fluorescence signals had a higher latency in response to reaching behavior and a more sporadic pattern of activity relative to local field potential (LFP) responses. Based on these differences, we are interested in whether the extracellular LFP in the primary motor cortex (M1) serves as a better predictor than intracellular calcium influx for reaching behavior in marmoset monkeys. To test this research question, we trained logistic regression and random forest classifiers to predict reaching behavior based on either LFP responses or calcium fluorescence signals. Both classifiers trained on LFP signals produce better results for reaching behavior than calcium imaging, and logistic regression predicts the behavior more reliably for both modalities.

II. Introduction

Extracellular recordings of action potentials have served as the foundation of neuroscience and were critical in discovering the neuronal basis for cognition, movement and learning[4,5,6]. These recordings measure changes in the membrane potential and represent the real time activity of neurons. Similarly, local field potentials (LFPs) are also real time recordings, however they quantify the activity from a population of neurons. For example, LFP recordings from the primary motor cortex (M1) have been shown to correlate strongly with movement onset[7]. On the other hand, calcium imaging is a new method that has become synonymous with neuronal spiking. Calcium influx into the cytosol of neurons is measured by genetically expressing a fluorescent marker bound to Calmodulin (GCaMP). Calmodulin binds to intracellular calcium and allows for an approximation of the calcium concentration based on the change in fluorescence.

LFP recordings determine neural activity by measuring summed synaptic activity from a population of neurons, while calcium imaging aims to observe neural activity by tracking calcium influx with fluorescence intensity. However, considering that intracellular calcium can originate from a variety of sources and has high signal latency, recent publications argue calcium influx represents a degraded signal of neuronal activation[3]. More specifically, the high signal latency can be attributed to the delay in calcium influx relative to neuronal electrophysiological response, which occurs simultaneously with its activation. Therefore, it is critical to consider the drawbacks of calcium imaging and whether it can faithfully predict neuronal activity.

To test our question, we compare how well calcium fluorescence and LFPs recorded from the primary motor cortex predict reaching behavior in marmoset monkeys. The primary motor cortex is responsible for executing movement[5]. Therefore, we found two papers with datasets that measure the neuronal activity from primary motor cortex with either LFPs[1] or calcium imaging[2]. Although the electrophysiology dataset reports reaching tasks and the calcium imaging dataset records push/pull behavior in marmoset monkeys, both involve similar behavioral paradigms in which the animal extends its arm.

Since previous research has suggested that calcium imaging represents a degraded signal of neural activity[4] we suspect that our calcium heatmaps will demonstrate a higher latency when compared to the LFP responses from M1. As a result, we hypothesize that both our classification models will predict reaching behavior more accurately and precisely using the LFP responses when compared to the calcium imaging data.

III. Results

3.1 Data wrangling

There are two marmoset monkey datasets that we loaded for data analysis and model training. The first dataset is from the marmoset monkey Chewie, which contains the local field potential (LFP) and spikes recorded from the motor cortex (M1) when the marmoset monkey is at rest or during reaching behavior[1]. Since wave features are difficult to extract from the sparse spikes using tsfresh, we use the data for LFP. We select only 10/178 trials for

analysis and model training to reduce computational complexity. For the LFP data frame of each trial, the rows represent the timepoints and the columns represent neurons. There are in total 864 neurons (columns) with varying number of rows (timepoints) since each recording duration is different. There are no null values in the LFP data for the 10 trials. Furthermore, we extracted behavioral data from the main dataset that includes the start/end time and movement onset for each trial. We determine the number of timepoints in each trial and generate labels for each time point by calculating the difference between start and end time. Timepoints before movement onset will be labeled as 0 (resting state), and timepoints after will be labeled as 1 (reaching behavior).

The second dataset we used is a calcium imaging dataset from one marmoset monkey, which includes fluorescent intensity recorded from each pixel of selected brain regions when the monkey is at rest, pulling, or pushing[2]. To make a fair comparison with the LFP data, we also selected 10 pushing/reaching trials for analysis and model training. The dataframe for each trial contains 4096 rows representing the pixels recorded in each brain region and a varying number of columns representing the timepoints (since the recording length is not the same). Moreover, since we are interested only in M1, we extracted the ROI indices given in the dataset, filtering the pixels in the fluorescent data by ROI indices to contain only pixels from M1. There are also no null values in the fluorescent intensity data for M1 over the 10 trials. Furthermore, we extracted the movement onset for all 10 trials, which we labeled timepoints before movement onset as 0 (resting) and after as 1 (reaching).

3.2 Data Analysis

In data wrangling, we incorporated behavioral data to help us conduct data analysis and model training with the wrangled datasets. We develop a 4-step pipeline that applies for both LFP and calcium imaging (fluorescent intensity) data to achieve this. Firstly, we performed an exploratory analysis on the raw LFP and fluorescence intensity data by examining their raw signals (Figure 1 and 2). For the fluorescence intensity data, we z-scored the responses in the dataframe, before averaging to ensure normality across trials. By observing raw LFP and fluorescent intensity signals, we confirmed if there are changes in LFP and fluorescent intensity in M1 before and after the movement onset through plots.

After confirming that there are differences in LFP and fluorescent intensity data before and after movement onset, we preprocessed both LFP and fluorescent intensity dataframes for M1 into long-form, which the timepoints from each trial serves as the ‘id’ column. This is because tsfresh package takes in long-form dataframes as inputs for feature extraction, and we are classifying whether the marmoset monkey is at rest or reaching at each timepoint based on LFP and fluorescent intensity data. Furthermore, we also downsample the fluorescent intensity data by aggregating data from every two timepoints by their mean, since there are twice as many data points for fluorescent intensity than LFP.

Next, we implemented tsfresh, a python package that assists in automated extraction of wave features from time series data, to help us extract features from LFP and fluorescent intensity data across timepoints in each trial. The tsfresh package extracts 783 features from LFP and fluorescent intensity data including but not limited to mean/median amplitude, standard deviation, and fourier components from the group of neurons at a specific time point. After imputing null values caused by features that are not applicable for our time series data, we selected only features that are statistically significant ($p<0.05$) for hypothesis test of relevance/importance using the select_feature() function. We then normalized the selected features for both LFP and fluorescent intensity data.

Finally, using the labels and normalized tsfresh features, we built and trained a logistic regression classifier and a random forest classifier for each data modality (LFP and fluorescent intensity). We employed a 80/20 stratified split for the training and testing set. Furthermore, we observed that the classes (resting vs. reaching) were imbalanced for both LFP and fluorescent intensity data, so the class_weight parameter for all four models is set to ‘balanced’ to give more weight to the minority (reaching) class. For the random forest classifier, hyperparameter tuning showed that 26 trees is optimal for both random forest classifiers trained on LFP and fluorescent intensity data, so the number of trees was set to 26 for both models. After training the logistic regression and random forest classifiers, the model was used to predict labels on the test set and metrics such as accuracy, precision, recall, and f1-score were calculated for each of the four models in classification reports. The classification reports show that

models trained on LFP signals performed better on all metrics compared to that of the calcium fluorescence signals. Furthermore, logistic regression classifiers perform better on all metrics compared to random forest.

3.3 Data Visualization

Bursts of high-amplitude LFP activity consistently occur across trials, aligning with neural events linked to reaching movements (~0.1s after onset, Figure 1). Calcium imaging heat maps reveal variable calcium influx in M1 neurons, persisting for 2s post-behavior and peaking ~1s after reaching (Figure 2).

Random forest classifiers applied to LFP and calcium imaging data show rapid training error reduction with increasing trees (Figure 3). The LFP-based model achieves lower validation errors, indicating superior classification performance.

Confusion matrices for logistic regression and random forest classifiers (Figure 4) show LFP data enables higher classification accuracy. Logistic regression achieves 213 true negatives, 53 true positives, and minimal misclassification (2 false positives, 0 false negatives). Calcium imaging yields a higher false positive (27) and false negative rate (20). Random forest with LFP excels in resting classification (215 true negatives, 0 false positives) but has a moderate false negative rate (15). On calcium imaging, random forest performs similarly to logistic regression but with higher false negatives (33). LFP-based models outperform calcium imaging, with random forest making more conservative predictions.

The ROC curves compare the performance of logistic regression classifier and random forest classifier trained on LFP and calcium imaging data to classify resting and reaching behavior in marmoset monkeys(figure 5). For both logistic regression and random forest model, the LFP data shows better performance with an AUC of 0.99, suggesting a better classification ability between two behavior states compared with calcium data with an AUC of 0.86. The logistic regression classifier (left panel) notably shows steeper ROC curves with a more pronounced initial rise than the random forest model (right panel) for both LFP and calcium imaging data, suggesting a better classification performance.

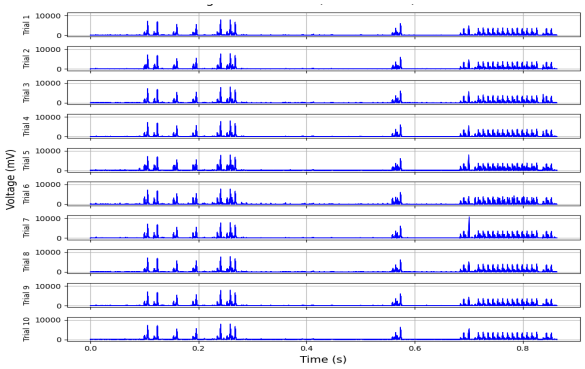


Figure 1: LFP Signals over Time in the Primary Motor Cortex after Reaching Behavior. 10 individual trials are labelled. The x-axis represents time in seconds, while the y-axis represents LFP amplitude in microvolts. Reaching behavior begins at 0 seconds.

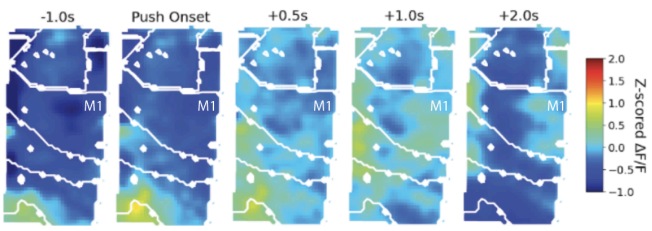


Figure 2: Calcium Imaging Heatmaps Relative to Reaching (Push) Behavior. The z-scored change in fluorescence over basal fluorescence (ΔF/F) signal, which corresponds to calcium influx, was averaged across 10 trials. The heatmaps are 40x64 pixels and the color indicates the intensity of the z-scored ΔF/F as indicated by the legend on the right. A region of interest (ROI) mask provided in the

dataset is mapped over each of the heatmaps and indicated by the white boundaries. M1 is labeled as within the boundaries that enclose the neurons in this region.

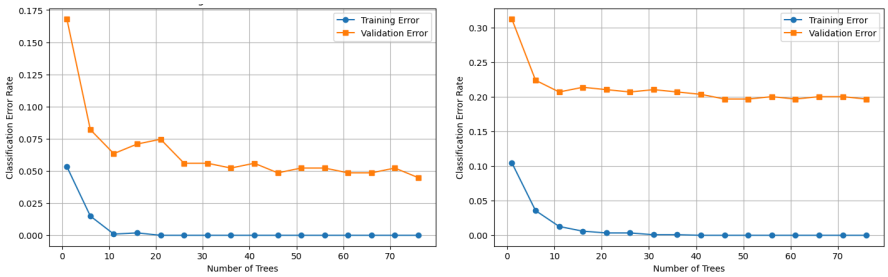


Figure 3: Training and validation error rates for random forest classification model and regression model based on LFP and calcium imaging data. The left panel shows error rates for the model trained on LFP data, while the right panel corresponds to the model trained on calcium imaging data. The x-axis represents the number of trees in the random forest, and the y-axis indicates the classification error rate. The blue line represents the training error line, while the orange line represents the validation error line.

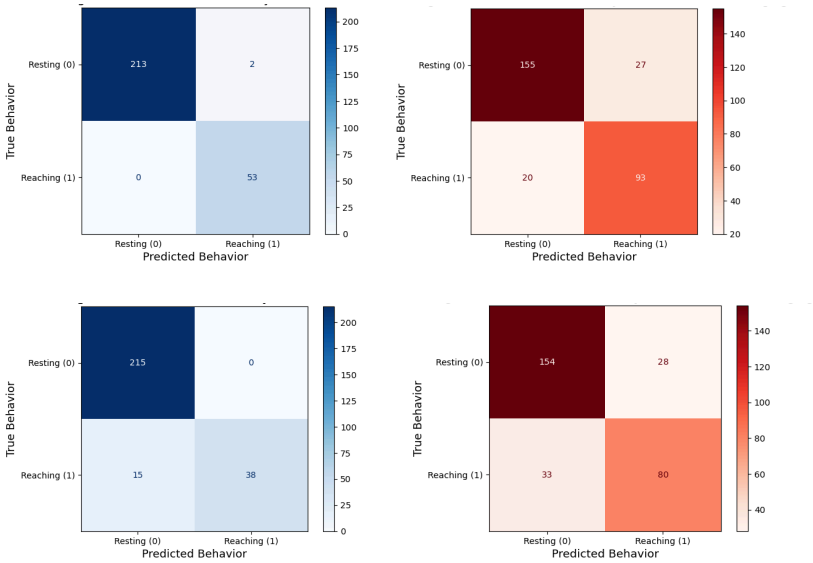


Figure 4: Confusion matrices for logistic regression(top) and random forest classification models(bottom) predicting reaching behavior in marmoset monkeys based on LFP and calcium imaging data. The blue and red matrices present the LFP data and the calcium imaging data respectively. The top two matrices represent the logistic regression model while the bottom two represent the random forest classification model. The numbers of success of predicting for the resting and reaching behaviors are shown in the top-right and bottom-left corners for each matrix respectively. The top-right corner and the bottom-left corner show failure in predicting resting and reaching behavior respectively.

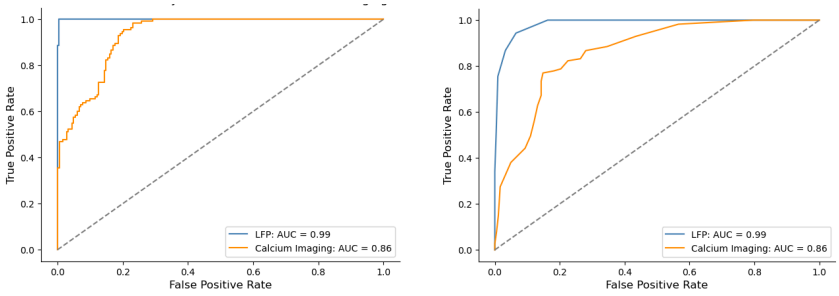


Figure 5: Receiver Operating Characteristics (ROC) curves for logistic regression classifier (left) and random forest classifier (right) trained on local field potential (LFP) and calcium imaging (fluorescent intensity) data to classify resting vs. reaching behavior in marmoset monkeys. The blue line is the ROC curve for models trained on LFP data, and orange line represents the ROC curve for models trained on calcium imaging data. The area under the curve (AUC) for each curve is noted in the bottom right legend for each subplot. The dashed black line indicates the ROC curve for a baseline model generating random predictions.

IV. Discussion/Conclusion

We found that the LFP responses had a lower latency (0.1 seconds) when compared to the calcium imaging data (1 second). This is consistent with previous research that suggests calcium influx represents a degraded signal of neuronal activity[4]. Considering that calcium imaging has lower temporal resolution when compared with LFP recordings, due to the delay in calcium influx into the cell, this makes sense. We reason that this delayed latency contributed to the poorer ability for our classification models to predict reaching behavior, since the calcium fluorescence signals don't closely align with behavior onset. Additionally, the LFP responses had a clear pattern of activity following the onset of behavior, where increases in the voltages corresponded across trials, while calcium fluorescence signals were more variable and took longer to decay. The fluctuation in the calcium signals is likely due to the different sources of calcium that can be detected, in addition to calcium influx there are intracellular sources of calcium that could contribute to the signal[8]. This will also decrease the ability for the classification models to predict the behavior from calcium signals, considering the fluctuations in the pattern of neuronal activity and longer period of decay. Although both classification models were able to predict reaching behavior with greater accuracy using the LFP responses when compared to the calcium fluorescence signals, the logistic regression model predicted the behavior better using both modalities. This could be due to the type of decision boundary that fits the model best, it is likely that the data features are better separated with a linear boundary or plane. Logistic regression is less prone to overfitting features in smaller datasets than random forests, which is a reasonable explanation considering we are only using 10 trials for both datasets.

There are several limitations to our study. Firstly, since the LFP and fluorescent intensity data are obtained from different experiments using different marmoset monkeys, there might be variations between the nervous system which we obtained the data from. Furthermore, although the monkeys in both experiments execute behavior involving extending their arms, the tasks they were asked to perform were slightly different. The LFP marmoset monkey reaches towards the monitor in response to visual stimulus, whereas calcium imaging monkey grabs onto a pole in response to visual stimulus. This might produce slight variations in the LFP and fluorescent intensity patterns. Secondly, we are only building our model on 10 trials of data for LFP and fluorescent intensity due to limitations in computational resources. The models we trained have limited generalizability and statistical power despite sufficient data. Thirdly, we only tested our hypothesis on two types of models, and not all hyperparameters are optimized for the random forest model, limiting the reliability of our conclusion. Finally, we left out some statistically significantly relevant features for calcium imaging (fluorescent intensity) data to match the number of features for LFP data used for training. This might inevitably reduce the performance of the logistic regression and random forest classifiers trained on fluorescent intensity data.

Reflection

Our proposal sought to address the question of whether calcium imaging data can be used as a proxy for neuronal activity when compared to electrophysiological signals, such as LFPs. Initially we considered aligning calcium imaging data on the same time scale as LFP signals and comparing the two side by side relative to reaching behavior. However, feedback on the proposal revealed the difficulties in doing so, considering the difference in the temporal resolution of the two data modalities. Therefore, we decided to plot the two modalities separately and consider the latency of the signal following onset of reaching behavior. We reasoned that a higher latency may make it more difficult for classification models to predict the behavior. We also wanted to compare the signal pattern shown in the plots for both modalities and see if there were recognizable patterns across time. If following the signal following the behavior had consistent signals, we reasoned that the classification models would be able to more faithfully predict the behavior.

Team Member Contributions and AI Usage Statements

Devin Chang: data wrangling, feature engineering, building/evaluating ML models, writing results and discussion
Pawel Vijayakumar: analysis for calcium imaging data, writing abstract, introduction, results, and discussion
Claire Gao: exploratory analysis for LFP data, writing results and figure captions, revise the whole paper

We used and modified code written by ChatGPT in extracting information from mat files, plotting raw LFP and fluorescent intensity signals, and code for tuning hyperparameters for random forest classifiers.

IV. References

1) Gallego-Carracedo C., Perich M., Chowdhury R., Miller L., Gallego J. (2022) Local field potentials reflect cortical population dynamics in a region-specific and frequency-dependent manner *eLife* **11**:e73155. https://doi.org/10.7554/eLife.73155

2) Ebina, T., Sasagawa, A., Hong, D. *et al.* (2024) Dynamics of directional motor tuning in the primate premotor and primary motor cortices during sensorimotor learning. *Nat Commun* 15, 7127 https://doi.org/10.1038/s41467-024-51425-3

3) Huang, L., Ledochowitsch, P., Knoblich, U., Lecoq, J., Murphy, G. J., Reid, R. C., de Vries, S. E., Koch, C., Zeng, H., Buice, M. A., Waters, J., & Li, L. (2021). Relationship between simultaneously recorded spiking activity and fluorescence signal in GCaMP6 transgenic mice. *eLife*, *10*, e51675. https://doi.org/10.7554/eLife.51675

4) Buzsáki, G., & Tingley, D. (2018). Space and Time: The Hippocampus as a Sequence Generator. *Trends in cognitive sciences*, *22*(10), 853–869. https://doi.org/10.1016/j.tics.2018.07.006

5) Shenoy, K. V., Sahani, M., & Churchland, M. M. (2013). Cortical control of arm movements: a dynamical systems perspective. *Annual review of neuroscience*, *36*, 337–359. https://doi.org/10.1146/annurev-neuro-062111-150509

6) Yael Niv (2009). Reinforcement learning in the brain. *Journal of Mathematical Psychology*, *53*(3), 139-154 https://doi.org/10.1016/j.jmp.2008.12.005.

7) Witham CL, Wang M, Baker SN (2007) Cells in somatosensory areas show synchrony with beta oscillations in monkey motor cortex *The European Journal of Neuroscience* **26**:2677–2686.

8) Bagur, R., & Hajnóczky, G. (2017). Intracellular Ca2+ Sensing: Its Role in Calcium Homeostasis and Signaling. *Molecular cell*, *66*(6), 780–788. https://doi.org/10.1016/j.molcel.2017.05.028

Import Packages

In [240…
```
# Import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
import h5py
```

Classification of Marmoset Reaching Behavior Using Motor Cortex (M1) LFP Data

Data Wrangling

First, we load in the electrophysiology data from one marmoset monkey (Chewie) at resting state vs. during reaching behavior recorded on 2015/06/30 and used in the paper. The dataset is a mat file, which contains keys and sub-keys shown as below. We will mostly be focusing on determining whether the marmoset monkey is performing reaching behavior based on the LFP data collected from the motor cortex (M1), so we are isolating information in M1_lfp. Furthermore, we need the behavioral data in the dataset to know when the trial starts/end, as well as when the reaching behavior begins, which we have extracted and stored in behavioral_data_df.

**Note: The code below for extracting behavioral data from mat files are modified code from ChatGPT.

In [241…
```
# Keys in the marmoset electrophysiology/reaching behavior dataset
ephys_file = h5py.File("/Users/Devin Chang/Downloads/Chewie_CO_20150630.mat", 'r')
print(ephys_file.keys())
```

```
<KeysViewHDF5 ['#refs#', 'trial_data']>
```

In [242…
```
# Information contained in the trial data
print(ephys_file['trial_data'].keys())
```

```
<KeysViewHDF5 ['M1_lfp', 'M1_lfp_guide', 'M1_spikes', 'M1_unit_guide', 'bin_size', 'date_time', 'idx_endTime', 'idx_goCueTime',
'idx_movement_on', 'idx_startTime', 'idx_tgtOnTime', 'idx_trial_end', 'idx_trial_start', 'monkey', 'result', 'tgtDir', 'vel']>
```

In [243…
```
# List of behavioral information that we are interested
behavioral_data_list = ['idx_endTime', 'idx_goCueTime', 'idx_movement_on', 'idx_startTime', 'idx_tgtOnTime', 'idx_trial_end',
behavioral_data_df = np.array([])

# Extract specified information from trial data
for info in behavioral_data_list:
    data = ephys_file['trial_data'][info]
    extracted_data = []
    for i in range(data.shape[0]):
        row_data = []
        for j in range(data.shape[1]):
            ref = data[i, j]  # Get the reference
            if isinstance(ref, h5py.Reference):  # Check if it's a reference
                actual_data = ephys_file[ref][()]  # Dereference and get data
                row_data.append(actual_data)
            else:
                row_data.append(ref)  # Directly append non-referenced data
        extracted_data.append(row_data)

    # Convert to a NumPy array and then to DataFrame (if applicable)
    np_data = np.array(extracted_data).reshape(178,)
    behavioral_data_df = np.hstack((behavioral_data_df, np_data))

# Organize behavioral data into dataframe
behavioral_data_df = pd.DataFrame(behavioral_data_df.reshape(len(behavioral_data_list), 178).T,
                                  columns=behavioral_data_list)
```

In [245…
```
# look at the first 5 rows of the behavioral dataframe
behavioral_data_df.head()
```

Out[245…

	idx_endTime	idx_goCueTime	idx_movement_on	idx_startTime	idx_tgtOnTime	idx_trial_end	idx_trial_start
0	142.0	107.0	108.0	1.0	73.0	NaN	1.0
1	167.0	135.0	141.0	1.0	93.0	176.0	1.0
2	114.0	75.0	88.0	1.0	53.0	122.0	1.0
3	134.0	97.0	106.0	1.0	64.0	142.0	1.0
4	118.0	85.0	93.0	1.0	45.0	126.0	1.0

Here, since we have in total 178 trials for the marmoset monkey Chewie in this dataset, we have obtained a dataframe that contains 178 rows and columns with information including trial start/end time as well as time when movement onset. We then store the behavioral start/end times and movement onsets for each trial into arrays below.

In [246…
```
# Storing the start/end time and movement onset of each trial in an array
trial_starttime = np.array(behavioral_data_df['idx_startTime']) - 1
trial_endtime = np.array(behavioral_data_df['idx_endTime']) - 1
trial_movement_on = np.array(behavioral_data_df['idx_movement_on'])
```

We then load the LFP data in from the mat file. Note that the code below is a modified version of code written by ChatGPT for extracting information from mat files. A sample dataframe that contains information of LFP of each neuron over time is shown below. LFP data are recorded from 864 neuron at each trial, but the number of timepoints recorded are different given the same time bin.

In [275…
```
# Load in all M1 LFP data from the Chewie marmoset monkey as dataframes
with h5py.File("/Users/Devin Chang/Downloads/Chewie_CO_20150630.mat", 'r') as mat:
    data_list = []
    data = mat['trial_data']['M1_lfp']
    # Convert to NumPy array (MATLAB stores in column-major order, so transpose if needed)
    if isinstance(data[0, 0], h5py.Reference):
        # Extract actual data from references
        extracted_data = []
        for i in range(data.shape[0]):  # Iterate over rows
            row_data = []
            for j in range(data.shape[1]):  # Iterate over columns
                ref = data[i, j]  # Get the reference
                if isinstance(ref, h5py.Reference):  # Check if it's a reference
                    actual_data = mat[ref][()]  # Dereference and get data
                    row_data.append(actual_data)
                else:
                    row_data.append(ref)  # Directly append non-referenced data
            extracted_data.append(row_data)

        # Convert to a NumPy array and then to DataFrame (if applicable)
        np_data = extracted_data#, dtype=object)
        lfp_df = pd.DataFrame(np_data)

# Store all M1 LFP dataframes into a list
lfp_df_list = []
for i in range(lfp_df.shape[0]):
    lfp_df_list.append(pd.DataFrame(lfp_df[0][i]).T)
```

In [289…
```
lfp_df_list[0]
```

Out[289…

	0	1	2	3	4	5	6	7	8	9	…	854
0	8.257605	42.878782	63.301639	1.283340	15.801977	31.863385	44.897776	72.045651	51.539563	15.346799	…	701.08267
1	1.281180	42.859230	56.917981	2.666293	15.971701	46.606807	31.429457	67.583616	41.308509	6.611114	…	692.36449
2	-7.685382	42.711337	55.778981	6.675004	16.990480	43.712009	46.306036	55.344703	38.607804	-1.384134	…	706.09728
3	-10.719503	42.380179	54.586844	13.088107	19.931322	50.536605	68.939265	51.641724	35.171302	-3.401443	…	694.38922
4	-5.274688	41.879501	44.956520	21.140904	20.355948	76.888430	74.356965	63.603587	35.040695	6.266364	…	704.74860
…	…	…	…	…	…	…	…	…	…	…	…	…
144	-15.950196	72.235273	27.111106	9.582767	22.616733	25.843046	36.013704	68.937066	24.007891	-30.653910	…	697.56804
145	-16.506689	73.371396	32.477607	5.576015	25.991430	31.107557	45.646061	76.902404	33.498279	-25.344021	…	776.51317
146	-9.366235	74.333296	34.125276	3.938636	24.622420	36.478300	47.978507	71.507306	32.556722	-13.786378	…	780.80768
147	0.663022	74.671967	39.916477	2.346640	22.801099	49.223705	44.739138	57.394521	31.533487	-1.961522	…	756.10281
148	7.268667	74.251060	43.729536	2.203949	19.574597	51.984260	36.185935	48.884475	26.400500	3.983650	…	670.30147

149 rows × 864 columns

We then checked if there are any null values in LFP data for all trials. There are no null values in the LFP data for all ten trials.

In [293…
```
# Check if there are any null values in LFP data
num_null_values = []
for df in lfp_df_list:
    num_null_values.append(df.isnull().sum().sum())
sum(num_null_values)
```

Out[293… 0

Exploring LFP Signals for Each Trial

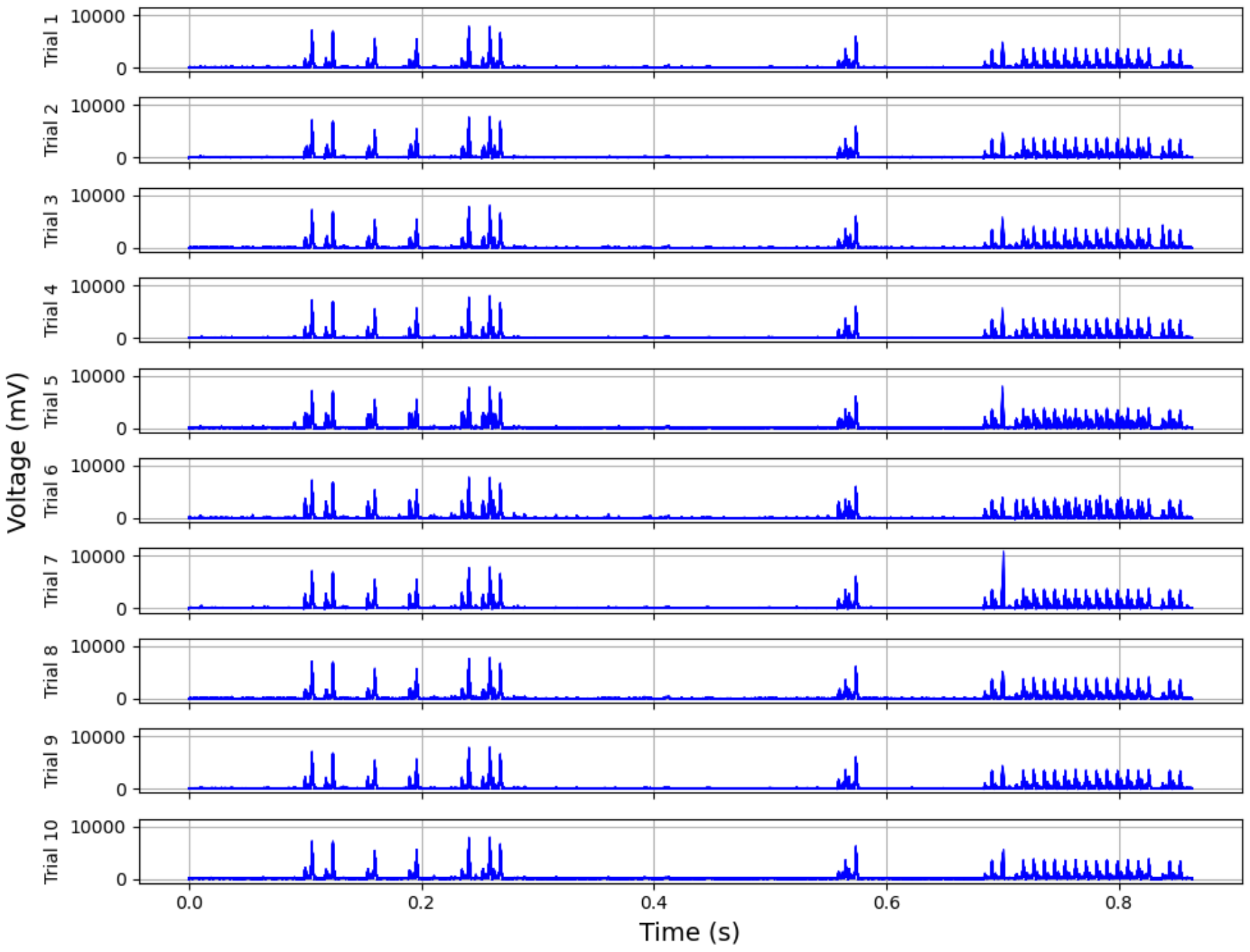
In [278…
```
lfp_data = ephys_file['trial_data']['M1_lfp'][:]
dereferenced_lfp_data = []
```

```
for i in range(lfp_data.shape[0]):
    ref = lfp_data[i, 0]  # Each entry in spike_data is a reference
    if isinstance(ref, h5py.Reference):  # Check if it's a reference
        actual_data = ephys_file[ref][()]  # Dereference and get actual data (spike train)
        dereferenced_lfp_data.append(actual_data)

print("First few dereferenced lfp data entries:", dereferenced_lfp_data[:3])
```

```
First few dereferenced lfp data entries: [array([[  8.25760511,   1.28117995,  -7.6853823 , ...,  -9.36623511,
          0.66302155,   7.26866694],
       [ 42.87878155,  42.85923044,  42.71133739, ...,  74.33329646,
         74.671967  ,  74.25105982],
       [ 63.30163916,  56.91798101,  55.77898055, ...,  34.12527615,
         39.91647681,  43.72953584],
       ...,
       [ 31.05704444,  31.54764108,  43.01594377, ...,  67.60052589,
         65.31063961,  64.34749013],
       [142.16892331, 155.46692775, 139.95452356, ..., 155.39846802,
        156.48370434, 162.39390066],
       [ 27.14150786,  27.0805987 ,  21.76053676, ...,  23.152977  ,
         19.38651627,  23.01673615]]), array([[  7.52417904,   4.11947544,   3.18948721, ...,   6.97946834,
         11.63401358,   9.38977961],
       [ 70.9957223 ,  67.21002591,  62.8050974 , ...,  54.50958701,
         51.92080404,  48.78120789],
       [ 40.87755205,  44.51658557,  43.0147504 , ...,  29.07884461,
         33.56803795,  34.51496027],
       ...,
       [ 89.72978787,  86.02280625,  79.6792148 , ...,  27.36936756,
         28.48121445,  47.35962527],
       [121.6591395 , 100.60688112,  89.85076848, ..., 121.40355751,
        123.35015084, 116.90149407],
       [ 38.18526374,  37.14660319,  33.00370105, ...,  35.87422194,
         32.13737907,  35.38243498]]), array([[ 3.97400148e+00, -2.48445498e-01, -1.88219748e+00, ...,
        -5.81073791e-03,  8.34204466e+00,  2.13815613e+01],
       [ 4.57244967e+01,  4.50196946e+01,  4.54506139e+01, ...,
         5.44024266e+01,  4.71551499e+01,  4.52050974e+01],
       [ 3.34446164e+01,  4.11371815e+01,  3.77653190e+01, ...,
         6.03062060e+01,  5.79930988e+01,  5.79403671e+01],
       ...,
       [ 5.22511970e+01,  5.24017081e+01,  5.42157097e+01, ...,
         4.92217310e+01,  4.96859696e+01,  3.60138674e+01],
       [ 1.46722250e+02,  1.69090098e+02,  1.40727912e+02, ...,
         5.72526952e+01,  5.66052398e+01,  7.08709742e+01],
       [ 3.83602455e+01,  4.55012653e+01,  4.40775700e+01, ...,
         4.38366783e+01,  4.40899109e+01,  4.15790729e+01]])]
```

In [285…

```
num_trials = 10
sampling_rate = 1000  # Adjust this based on your dataset
time_window = 2  # Time window in seconds to display (adjust as needed)

# Create the figure and subplots
fig, axes = plt.subplots(num_trials, 1, figsize=(10, 8), sharex=True, sharey=True)

for i in range(num_trials):
    if i >= len(dereferenced_lfp_data):  # Prevent index error
        break

    lfp_trial = np.squeeze(dereferenced_lfp_data[i])  # Convert to 1D if needed
    time = np.arange(0, len(lfp_trial)) / sampling_rate  # Generate time axis

    # Limit to the first 'time_window' seconds
    time_idx = int(time_window * sampling_rate)
    time = time[:time_idx]
    lfp_trial = lfp_trial[:time_idx]

    axes[i].plot(time, lfp_trial, color='b', linewidth=0.8)
    axes[i].set_ylabel(f'Trial {i+1}')
    axes[i].grid(True)

# Labels for the last subplot
axes[-1].set_xlabel('Time (s)', fontsize=14)
fig.supylabel('Voltage (mV)', fontsize=14)
fig.suptitle('LFP Signals Over Time (First 10 Trials)', fontsize=16)
plt.tight_layout()
plt.show()
```



Preprocessing for LFP Data

Since we have a lot of trials, we will be using data from 10 trials for analysis and modeling to reduce computational complexity. We intend to use tsfresh package for feature extraction, which is a python package that automates wave feature extraction for time series data. Since the tsfresh functions takes inputs as long-form dataframe, we will organize our lists of LFP dataframes into a single long-form dataframe. Furthermore, since we are interested in classifying marmoset monkey's reaching behavior based on LFP at each time point, we will make the timepoints as the unique 'id' column named 'time'.

At the same time, based on the behavioral data, we have the start/end time and movement onset for each trial. Thus we can create label array for each trial with length end time - start time. Also, since the paper does not specify the duration of the reaching behavior, we will assume LFP for reaching behavior remains after movement onset for rest of the time (this makes sense because we can see LFP spikes after the movement onset and towards the end time in the LFP plot). Therefore, we will label every timepoints after movement onset with 1 (reaching) and before movement onset as 0 (resting).

In [249…

```
num_trials = 10
num_neurons = 864

# Initializing lists to hold the long-form LFP dataframe and labels
all_trials_lfp_list = []
y_labels_list = []

for i in range(num_trials):
    # Preprocessing dataframe to become long-form and making label array for each trial
    X = lfp_df_list[i]
    X = X.iloc[int(trial_starttime[i]):int(trial_endtime[i])]
    y = np.zeros(X.shape[0])
    y[int(trial_movement_on[i]):] = 1
    X = X.reset_index().drop(columns='index')
    y_labels_list.append(y)

    df_list = []

    # Make long-form LFP dataframe
    for time in X.index:
        current_df = pd.DataFrame({'time':f"{i}_{time}", 'neuron':X.columns,
                                   'value': X.iloc[time, :]})
        df_list.append(current_df)
```

```
    # Assemble a single long-form LFP dataframe for all trials
    long_df = pd.concat(df_list, ignore_index=True)
    all_trials_lfp_list.append(long_df)
all_trials_lfp_df = pd.concat(all_trials_lfp_list, ignore_index=True)

# Assembling labels for all trials into a single array
y_labels = np.concatenate(y_labels_list)
```

In [250…
```
# Take a look at the long-form dataframe
all_trials_lfp_df
```

Out[250…

	time	neuron	value
0	0_0	0	8.257605
1	0_0	1	42.878782
2	0_0	2	63.301639
3	0_0	3	1.283340
4	0_0	4	15.801977
…	…	…	…
1156027	9_128	859	101.711083
1156028	9_128	860	54.748275
1156029	9_128	861	112.671200
1156030	9_128	862	182.233188
1156031	9_128	863	56.856474

1156032 rows × 3 columns

Feature Extraction of LFP data via tsfresh

For feature extraction, we will be using tsfresh to extract automated selected features across each time point using the long-form LFP dataframe and labels.

In [11]:
```
# Import tsfresh packages for feature extraction
from tsfresh import extract_features, select_features
from tsfresh.utilities.dataframe_functions import impute
```

In [193…
```
# Perform feature extraction
lfp_extracted_features = extract_features(all_trials_lfp_df, column_id="time", column_sort="neuron")
```

```
Feature Extraction: 100%|██████████| 20/20 [17:21<00:00, 52.07s/it]
```

Since we want the row order of the resulting feature matrix to match the order of the labels, we will sort the index for the feature matrix.

In [66]:
```
# This function sorts the index of the feature matrix
def sort_trialtime_index(df):
    df.index.name = "trial_time"
    df = df.reset_index()

    # Split the index into two numeric columns
    df[['trial', 'timepoint']] = df['trial_time'].str.split('_', expand=True).astype(int)

    # # Sort by trial number first, then by timepoint
    df = df.sort_values(by=['trial', 'timepoint']).set_index('trial_time')
    df.drop(columns=['trial', 'timepoint'], inplace=True)
    return df
```

In [195…
```
# Sort the extracted feature matrix by its index
lfp_extracted_features = sort_trialtime_index(lfp_extracted_features)
lfp_extracted_features
```

Out[195...

trial_time	value__variance_larger_than_standard_deviation	value__has_duplicate_max	value__has_duplicate_min	value__has_duplicate	valu
0_0	1.0	0.0	0.0	1.0	
0_1	1.0	0.0	0.0	1.0	
0_2	1.0	0.0	0.0	1.0	
0_3	1.0	0.0	0.0	1.0	
0_4	1.0	0.0	0.0	1.0	
...	...	...	...	...	
9_124	1.0	0.0	0.0	1.0	
9_125	1.0	0.0	0.0	1.0	
9_126	1.0	0.0	0.0	1.0	
9_127	1.0	0.0	0.0	1.0	
9_128	1.0	0.0	0.0	1.0	

1338 rows × 783 columns

After extracting the features, we will first impute any missing value in the feature matrix. These missing values are due to the fact that some features extracted by tsfresh does not apply to our LFP data. Next, we will select only features that are statistically significant for relevance in the tsfresh feature matrix using the select_features function, which applies statistical tests on each feature to assess their relevance ($p<0.05$). At last, there are in total 363 statistically significantly relevant features out of the 783 features extracted, and we use only those for modeling training.

In [196...

```
# Drop NaN and impute missing values
lfp_extracted_features = impute(lfp_extracted_features)

# Select only statistically significant relevent/important features
lfp_selected_features = select_features(lfp_extracted_features, y_labels)
```

```
C:\Users\Devin Chang\anaconda3\Lib\site-packages\tsfresh\utilities\dataframe_functions.py:198: RuntimeWarning: The columns ['va
lue__query_similarity_count__query_None__threshold_0.0'] did not have any finite values. Filling with zeros.
  warnings.warn(
```

In [251...

```
# First five rows of the feature dataframe after selecting statistically relevant features
lfp_selected_features.head()
```

Out[251...

	value__fft_coefficient__attr_"abs"__coeff_39	value__fft_coefficient__attr_"imag"__coeff_39	value__mean_change	value__change_quantiles_
0	18481.985140	18144.983087	0.021882	
1	18518.272516	18182.850685	0.029895	
2	19283.948707	18960.320720	0.034120	
3	20003.439741	19681.261654	0.037011	
4	21094.226401	20902.029182	0.031925	

5 rows × 363 columns

Since it takes a long time to extract and select features, we saved the selected feature matrix as a csv in a github folder.

In [12]:

```
# Writing the selected features dataframe as csv
# lfp_selected_features.to_csv('/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/lfp_features.csv', index=True)

# Loading the selected feature matrix if needed
lfp_selected_features = pd.read_csv('/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/tsfresh_feature_matrix/lfp_featur
lfp_selected_features = lfp_selected_features.iloc[:, 1:]
```

Next, we normalized each features (columns) in the selected feature matrix using StandardScaler. We also make sure that the features are properly normalized by checking each column's mean, which is close to 0.

In [17]:

```
# Import StandardScaler for normalization
from sklearn.preprocessing import StandardScaler

# Normalize each features
scaler = StandardScaler()
lfp_normalized_features = scaler.fit_transform(lfp_selected_features)
lfp_normalized_features = pd.DataFrame(lfp_normalized_features, columns=lfp_selected_features.columns)
lfp_normalized_features
```

Out[17]:

	value__fft_coefficient__attr_"abs"__coeff_39	value__fft_coefficient__attr_"imag"__coeff_39	value__mean_change	value__change_quant
0	-1.147695	-1.140738	-1.070894	
1	-1.130384	-1.122978	-0.670096	
2	-0.765122	-0.758331	-0.458758	
3	-0.421892	-0.420198	-0.314182	
4	0.098462	0.152362	-0.568546	
...	...	...	...	
1333	0.322724	0.212779	-0.720987	
1334	0.530650	0.444512	-0.127833	
1335	0.315809	0.304494	0.515113	
1336	0.403375	0.438668	1.740648	
1337	-0.015954	0.050393	1.586106	

1338 rows × 363 columns

In [18]:
```
lfp_normalized_features.mean()
```

Out[18]:
```
value__fft_coefficient__attr_"abs"__coeff_39                          -8.284355e-16
value__fft_coefficient__attr_"imag"__coeff_39                          1.455073e-15
value__mean_change                                                    -2.097641e-16
value__change_quantiles__f_agg_"mean"__isabs_False__qh_1.0__ql_0.0     2.310060e-16
value__fft_coefficient__attr_"imag"__coeff_42                          1.115202e-15
                                                                           ...
value__fft_coefficient__attr_"abs"__coeff_82                          -6.797419e-16
value__fourier_entropy__bins_2                                         1.306379e-15
value__agg_linear_trend__attr_"stderr"__chunk_len_50__f_agg_"mean"     1.688734e-15
value__fft_coefficient__attr_"angle"__coeff_49                        -6.160161e-16
value__fft_coefficient__attr_"angle"__coeff_83                         2.124194e-17
Length: 363, dtype: float64
```

We also write the normalized feature matrix into csv and save it in the folder.

In [19]:
```
# Write the normalized features dataframe into csv
# lfp_normalized_features.to_csv('/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/normalized_lfp_features.csv', index=

# Load in normalized features dataframe if needed
lfp_normalized_features = pd.read_csv('/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/tsfresh_feature_matrix/normaliz
lfp_normalized_features = lfp_normalized_features.iloc[:, 1:]
```

Classification of Marmoset Reaching Behavior using LFP Data

Import packages

In [204…
```
# Import packages for data preparation and classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

Preparing and examine training and test set

Firstly, we split the data (normalized features) and labels into training set and test set. The split is made with stratification to ensure representativeness to the full dataset.

In [21]:
```
# Split into train and test sets
lfp_X_train, lfp_X_test, lfp_y_train, lfp_y_test = train_test_split(lfp_normalized_features,
                                                                    y_labels, test_size=0.2, random_state=0, stratify=y_labels
```

After splitting the data and labels into training set and test set, we examine how many datapoints are in each class (behavior). We see that there is a class imbalance in the LFP data, which the points for resting state is about 4 times more than that of the reaching datapoints.

In [22]:
```
# Examine how many datapoints are for each class (behavior)
pd.Series(lfp_y_train).value_counts()
```

Out[22]:
```
0.0    860
1.0    210
Name: count, dtype: int64
```

Hyperparameter Tuning and Model Training

Logistic Regression Model:

Starting from simpler models, we first construct a logistic regression classifier to classify marmoset monkey's resting vs reaching behavior based on LFP data. Since there is class imbalanced, we specify the class_weight parameter to 'balanced' to give more weights to the minority class (reaching). After fitting on the training data and evaluating on the test set, we can see from the classification report below that training and predicting reaching behavior based on LFP data yields a high accuracy of 0.99 along with high precision, recall, and f1-score for both classes.

In [225…
```
# Train a Logistic Classifier
lfp_lr_clf = LogisticRegression(class_weight='balanced', max_iter=1000, random_state=0)
lfp_lr_clf.fit(lfp_X_train, lfp_y_train)

# Make predictions
lfp_lr_y_pred = lfp_lr_clf.predict(lfp_X_test)

# Evaluate
print("Accuracy:", accuracy_score(lfp_y_test, lfp_lr_y_pred))
print(classification_report(lfp_y_test, lfp_lr_y_pred))
```

```
Accuracy: 0.9925373134328358
              precision    recall  f1-score   support

         0.0       1.00      0.99      1.00       215
         1.0       0.96      1.00      0.98        53

    accuracy                           0.99       268
   macro avg       0.98      1.00      0.99       268
weighted avg       0.99      0.99      0.99       268
```

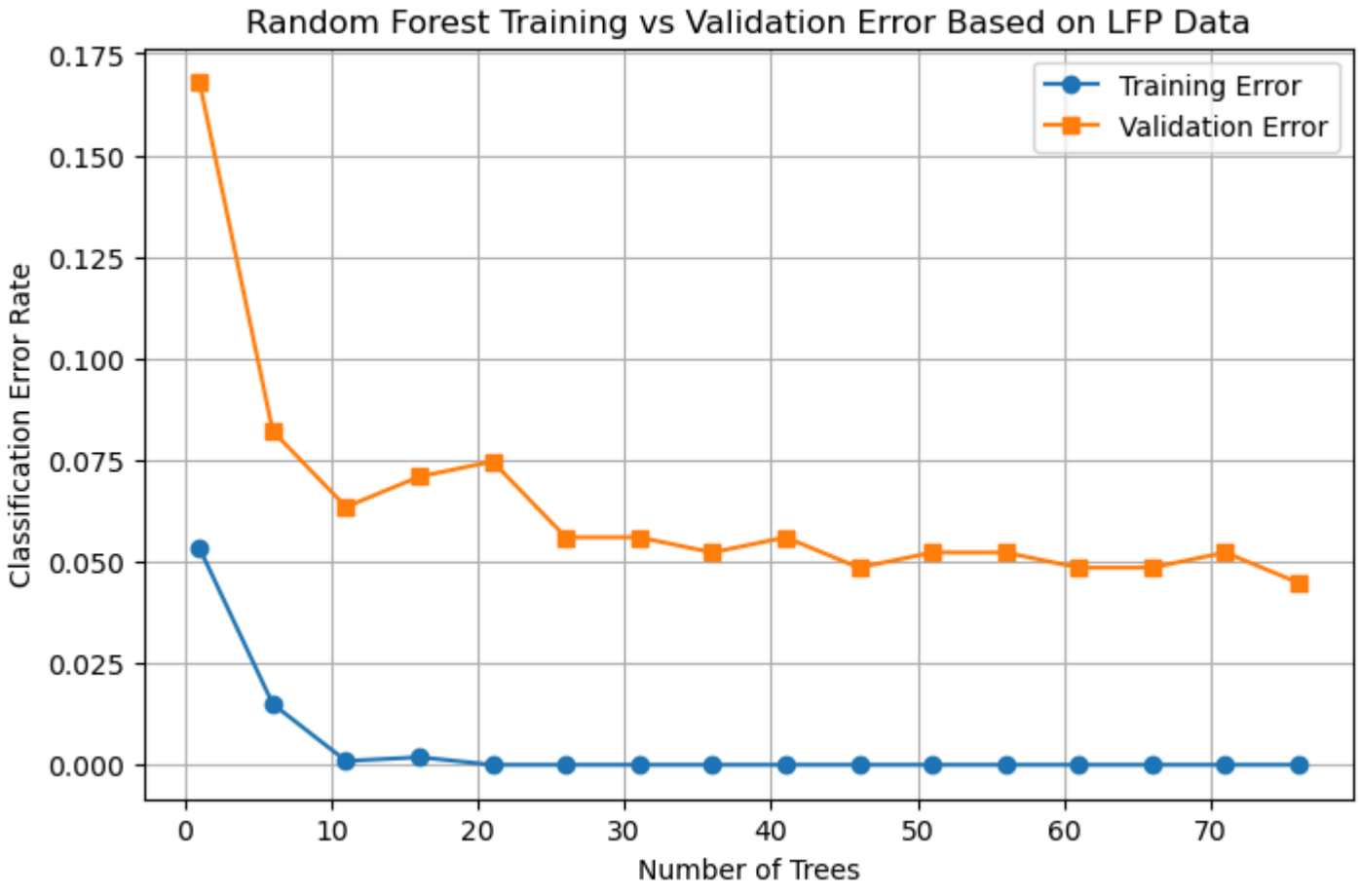
Random Forest Model:

Next, we further tried random forest model for classification of reaching behavior. Different from logistic regression, the random forest is an ensemble model than contains a specified number of estimators (trees), which the decisions made by each will be taken account to formulate the final prediction based on majority voting. Thus, we will tune the hyperparameter (the number of trees) before implementing the classifier.

From the classification error rate vs. number of trees plot below, we can see that the classification error rate for the training and testing data plateaus at around 26 estimators. We will thus use 26 trees for our random forest classifier.

In [227…
```
clf_accuracies = []
train_errors, val_errors = [], []
num_estimators = np.arange(1, 80, 5)
for num in num_estimators:
    temp_clf = RandomForestClassifier(n_estimators=num, class_weight='balanced', random_state=3)
    temp_clf.fit(lfp_X_train, lfp_y_train)
    temp_test_pred = temp_clf.predict(lfp_X_test)
    temp_train_pred = temp_clf.predict(lfp_X_train)
    clf_accuracies.append(accuracy_score(lfp_y_test, temp_test_pred))
    train_errors.append(1 - accuracy_score(lfp_y_train, temp_train_pred))
    val_errors.append(1 - accuracy_score(lfp_y_test, temp_test_pred))
```

In [229…
```
plt.figure(figsize=(8, 5))
plt.plot(num_estimators, train_errors, label="Training Error", marker='o')
plt.plot(num_estimators, val_errors, label="Validation Error", marker='s')
plt.xlabel("Number of Trees")
plt.ylabel("Classification Error Rate")
plt.title("Random Forest Training vs Validation Error Based on LFP Data")
plt.legend()
plt.grid()
plt.show()
```



Having obtained the optimal hyperparameter, we trained a random forest classifier with 26 trees on the LFP data and labels. Again, the class_weight is specified to be 'balanced' to account for the imbalanced classes. We see that it yields lower accuracy (0.94), precision, recall, and f1-score than using logistic regression. Thus, the logistic regression model performed better on the classifying marmoset monkey's reaching behavior vs resting using LFP data.

In [230…
```
# Train a Random Forest Classifier
lfp_rf_clf = RandomForestClassifier(n_estimators=26, class_weight='balanced', random_state=3)
lfp_rf_clf.fit(lfp_X_train, lfp_y_train)

# Make predictions
lfp_y_pred = lfp_rf_clf.predict(lfp_X_test)

# Evaluate
print("Accuracy:", accuracy_score(lfp_y_test, lfp_y_pred))
print(classification_report(lfp_y_test, lfp_y_pred))
```

```
Accuracy: 0.9440298507462687
              precision    recall  f1-score   support

         0.0       0.93      1.00      0.97       215
         1.0       1.00      0.72      0.84        53

    accuracy                           0.94       268
   macro avg       0.97      0.86      0.90       268
weighted avg       0.95      0.94      0.94       268
```

Classification of Marmoset Pushing Behavior Using Motor Cortex (M1) Calcium Imaging Data (Fluorescent Intensity)

In this section, we will use the calcium imaging data from marmoset monkey when at resting and performing pushing behavior. Since we could not find a calcium imaging dataset specifically for reaching behavior, we will use data on pushing behavior, which, same as reaching behavior, involves the extension of the arm and hands. The paper also called pushing a "reaching" task.

Data Wrangling

Loading in Calcium Imaging dataset of Marmoset Monkey's Pushing Behavior

Loading in the calcium imaging dataset, we see that there are two keys, roi and session_data. The roi contains information about what indices in the fluorescent intensity data correspondes to each of the brain regions recorded. Since it is hard to work with mat files in python, we used matlab to open the mat files and extract csv files for the information we are interested in. We load in those csv files here instead.

In [141…
```
# Pring the keys for the calcium imaging dataset
calcium_imaging_file = h5py.File('/Users/Devin Chang/Documents/Marmoset1/Marmoset1-data.mat', 'r')
print(calcium_imaging_file.keys())
```

```
<KeysViewHDF5 ['#refs#', 'roi', 'session_data']>
```

We now read in the dataframe that contains information about the brain regions and their corresponding indices in the fluorescent intensity data. We organized the dataframe so that the idx column contains all indices for a brain region in a list.

In [142…
```
# This function transform the columns of indices into lists of indices
def make_idx_list(ser):
    return list(ser.dropna().apply(int))
```

In [143…
```
# Read in and process the roi dataframe that contains indices for different brain region
roi_idx_df = pd.read_csv("/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/calcium_imaging_data/roi_idx.csv").set_index
roi_idx_df['idx'] = roi_idx_df.apply(make_idx_list, axis=1)
roi_idx_df = roi_idx_df[['idx']]
```

In [144…
```
# Take a look at the roi-indices dataframe
roi_idx_df
```

Out[144…

label	idx
8aV	[82, 83, 84, 85, 86, 87, 88, 89, 90, 144, 145,...
8b	[108, 109, 171, 172, 173, 174, 175, 176, 234, ...
8c	[153, 154, 217, 218, 280, 281, 282, 343, 344, ...
6Dr	[38, 39, 91, 92, 93, 94, 95, 96, 97, 98, 99, 1...
6Dc	[602, 603, 604, 605, 664, 665, 666, 667, 668, ...
6V	[272, 273, 274, 275, 276, 277, 278, 336, 337, ...
6M	[496, 554, 555, 556, 557, 558, 559, 560, 618, ...
4ab	[1239, 1240, 1241, 1302, 1303, 1304, 1305, 130...
4c	[848, 912, 913, 914, 976, 977, 978, 979, 1040,...
A3a	[1167, 1231, 1295, 1296, 1359, 1360, 1423, 142...
A3b	[1743, 1744, 1807, 1808, 1809, 1871, 1872, 187...
A12	[2127, 2191, 2192, 2255, 2256, 2257, 2258, 231...
PF	[2703, 2704, 2767, 2768, 2769, 2770, 2831, 283...
PE	[2901, 2902, 2965, 2966, 2967, 2968, 2969, 297...
PFG	[2959, 3023, 3024, 3025, 3087, 3088, 3089, 309...
PG	[3790, 3791, 3792, 3793, 3794, 3854, 3855, 385...
AIP	[3540, 3541, 3604, 3605, 3606, 3607, 3608, 366...

Then, we load in the fluorescent intensity dataset for each trial. 10 trials with push behavior (instead of pull) are selected to make comparison with the LFP data which we also loaded 10 trials. Each trial's dataframe contains in total of 4096 rows representing 4096 pixels that were recorded in calcium imaging, with variable number of columns (timepoints) for each trial due to different recording times.

In [145…
```
# Read in fluorescent intensity data from 10 trials into a list of dataframes
num_df = 10
calcium_imaging_df_list = []
calcium_imaging_folder = '/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/calcium_imaging_data'
for i in range(1, num_df + 1):
    df = pd.read_csv(calcium_imaging_folder + f'/trial_{i}.csv', header=None)
    calcium_imaging_df_list.append(df)
```

In [146…
```
# Take a look at the first trial's data
calcium_imaging_df_list[0]
```

Out[146…

	0	1	2	3	4	5	6	7	8	
0	2.919705e-06	0.000002	0.000002	1.882468e-06	1.534436e-06	1.866604e-06	1.836362e-06	1.518441e-06	0.000002	1.42
1	-3.533586e-05	-0.000038	-0.000040	-3.371209e-05	-3.306586e-05	-3.254881e-05	-3.372005e-05	-3.407252e-05	-0.000032	-3
2	-1.344078e-05	-0.000006	-0.000015	-9.995447e-06	-1.132800e-05	-1.189011e-05	-7.270114e-06	-9.712302e-06	-0.000007	-7
3	-3.441154e-06	-0.000009	-0.000004	-1.825091e-06	-4.640276e-06	-1.279701e-06	-9.647399e-07	-5.995730e-07	-0.000002	-4
4	1.034998e-07	0.000005	0.000002	5.469345e-07	4.603168e-07	-5.653084e-07	-8.153557e-08	-1.418958e-06	-0.000001	-1
...	...	...	...	...	...	...	...	...	...	
4091	0.000000e+00	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.00
4092	0.000000e+00	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.00
4093	0.000000e+00	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.00
4094	0.000000e+00	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.00
4095	0.000000e+00	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.00

4096 rows × 296 columns

There are no null values in the fluorescent intensity data for all 10 trials.

In [295…
```
# Check if there are any null values in fluorescent intensity data
num_null_values = []
for df in calcium_imaging_df_list:
    num_null_values.append(df.isnull().sum().sum())
sum(num_null_values)
```

Out[295… `0`

Then, we organized a csv that contains all information needed that records the important change in monkey's behavior at different trials. The onset_cue column shows which timepoint the cue for pushing behavior was given, and onset_push contains timepoint which the pushing behavior started (onset_pull is -1 because only trials that contains only pushing behavior are extracted).

In [254…
```
# Read in the dataframe containing behavioral data
ci_behavior_df = pd.read_excel('/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/calcium_imaging_data/calcium_imaging_b
ci_behavior_df
```

Out[254…

	Trial	onset_cue	onset_pull	onset_push	success	trial_type	first_choice
0	1	100	-1	183	1	1	1
1	2	99	-1	152	1	1	1
2	3	101	-1	388	1	1	1
3	4	97	-1	168	1	1	1
4	5	96	-1	165	1	1	1
5	6	99	-1	136	1	1	1
6	7	102	-1	139	1	1	1
7	8	150	-1	198	1	1	1
8	9	100	-1	138	1	1	1
9	10	99	-1	142	1	1	1

Exploring Fluorescent Intensity over Time

In [256…
```
from scipy.ndimage import gaussian_filter
from scipy.ndimage import label, binary_dilation, binary_closing, binary_fill_holes, binary_dilation
from skimage.measure import regionprops
```

In [259…
```
df_roi = pd.read_csv('/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/calcium_imaging_data/roi_idx.csv')
```

In [264…
```
# Ensure all trials have the same shape
min_time_bins = min(df.shape[1] for df in calcium_imaging_df_list)  # Find shortest trial
df_trials = [df.iloc[:, :min_time_bins] for df in calcium_imaging_df_list]  # Trim to same length

# Compute the average calcium signal across trials (ignoring NaNs)
df_trial_avg = pd.concat(df_trials).groupby(level=0).mean()

# Define constants
```

```
sampling_rate = 30  # Hz
pixel_size = 64  # 64x64 pixels
num_time_bins = df_trial_avg.shape[1]  # Total available time bins

# Compute total available recording time
total_recording_time = num_time_bins / sampling_rate

# Extract push onset time and ensure it is within range
push_time = ci_behavior_df.iloc[0]["onset_push"] if "onset_push" in ci_behavior_df.columns else None
if push_time is None:
    raise ValueError("Column 'onset_push' not found in behavioral data.")

# Cap push time if it's too high
push_time = min(push_time, total_recording_time - 2)
```

In [265…

```
# Define time points relative to push onset
time_offsets = [-1.0, 0, 0.5, 1.0, 2.0]  # Before, during, and after push onset
time_labels = ["-1.0s", "Push Onset", "+0.5s", "+1.0s", "+2.0s"]

# **Shift push bin by 1 second (30 frames)**
corrected_push_bin = max(int(push_time * sampling_rate) - 30, 0)  # Shift forward by 30 frames

# **Compute corrected time bins (Ensure all bins exist in dataset)**
time_bins = [
    max(min(corrected_push_bin + int(offset * sampling_rate), num_time_bins - 1), 0)
    for offset in time_offsets
]

# **Debugging: Print the new push bin and time bins**
print(f"New Corrected Push Bin: {corrected_push_bin}")
print(f"Updated Time Bins: {time_bins}")
```

```
New Corrected Push Bin: 158
Updated Time Bins: [128, 158, 173, 188, 218]
```

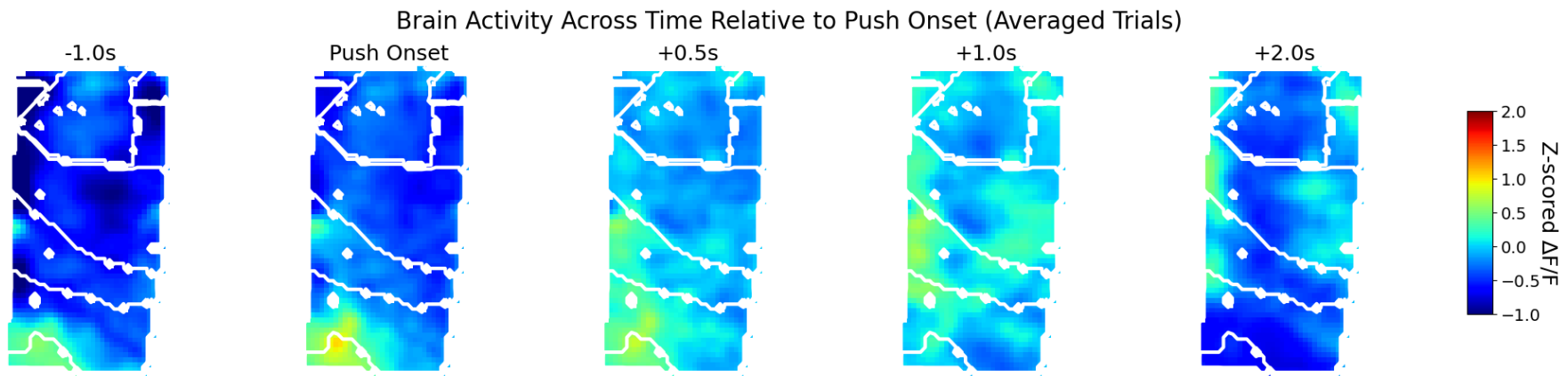
In [262…

```
roi_mask = np.zeros((pixel_size, pixel_size))
for i, row in df_roi.iterrows():
    roi_pixels = row[1:].dropna().astype(int).values  # Get valid pixel indices
    roi_mask[np.unravel_index(roi_pixels, (pixel_size, pixel_size))] = i + 1  # Assign unique index
```

In [274…

```
# **Better Figure Size**
fig, axes = plt.subplots(1, 5, figsize=(20, 4))

# Define fixed color scale
vmin, vmax = -1, 2

# Generate plots for each time point
for idx, (time_bin, label) in enumerate(zip(time_bins, time_labels)):
    if time_bin >= num_time_bins:
        print(f"Skipping {label}, time_bin {time_bin} is out of range.")
        continue

    # Extract brain activity data
    brain_activity = df_trial_avg.iloc[:, time_bin].values.reshape((pixel_size, pixel_size))
    brain_activity = np.nan_to_num(brain_activity)  # Replace NaNs
    brain_activity = gaussian_filter(brain_activity, sigma=1.0)  # Smoothing

    # Create subplot
    ax = axes[idx]

    # **Ensure No Extra Background**
    ax.set_facecolor("none")  # Remove any background color

    # **Ensure Proper Aspect Ratio**
    ax.set_aspect('auto')

    # **Larger Heatmap Without Extra Space**
    img = sns.heatmap(
        brain_activity,
        cmap="jet",
        cbar=False,
        ax=ax,
        vmin=vmin,
        vmax=vmax,
        mask=(roi_mask == 0),
        square=False,  # Avoid forced padding
        xticklabels=False,  # Remove axis labels
        yticklabels=False
    )

    # **Overlay ROI Boundaries**
    ax.contour(roi_mask, colors="white", linewidths=3)

    ax.set_title(label, fontsize=18)
    ax.axis("off")

# **Reduce Space Between Plots Even More**
```

```
plt.subplots_adjust(wspace=0.001)  # Remove almost all space

# **Remove Extra White Padding**
plt.tight_layout(pad=0)  # Fully removes excess padding

# **Adjust Color Bar to Avoid Overlapping**
fig.subplots_adjust(right=0.88)  # Keep color bar properly positioned
cbar_ax = fig.add_axes([0.9, 0.2, 0.015, 0.6])  # Adjust size
norm = mpl.colors.Normalize(vmin=vmin, vmax=vmax)
cbar = mpl.colorbar.ColorbarBase(cbar_ax, cmap="jet", norm=norm)

# **Enhance Color Bar Labeling**
cbar.set_label("Z-scored ΔF/F", fontsize=18, rotation=270, labelpad=15)
cbar.ax.tick_params(labelsize=14)

# **Final Title & Layout**
plt.suptitle("Brain Activity Across Time Relative to Push Onset (Averaged Trials)", fontsize=20, y=1.1)
plt.show()
```



Preprocessing for Calcium Fluorescent Intensity Data

Since we are only interested in the motor cortex (M1), we extract the roi indices only for the M1 data, which is printed below. According to the paper the label '4ab' corresponds to the M1 cortex, and the array is off by 1 because matlab indexing starts from 1 and python starts from 0.

In [252…

```
# Extract indices of pixels corresponding to motor cortex
M1_roi_idx = np.array(roi_idx_df.loc['4ab']['idx']) - 1
print(M1_roi_idx)
```

```
[1238 1239 1240 1301 1302 1303 1304 1305 1306 1307 1308 1309 1310 1311
 1312 1313 1314 1365 1366 1367 1368 1369 1370 1371 1372 1373 1374 1375
 1376 1377 1378 1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389
 1390 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441
 1442 1443 1444 1445 1446 1447 1448 1449 1450 1451 1452 1453 1454 1455
 1493 1494 1495 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506
 1507 1508 1509 1510 1511 1512 1513 1514 1515 1516 1517 1518 1519 1557
 1558 1559 1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1570 1571
 1572 1573 1574 1575 1576 1577 1578 1579 1580 1581 1582 1583 1621 1622
 1623 1624 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636
 1637 1638 1639 1640 1641 1642 1643 1644 1645 1646 1647 1685 1686 1687
 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1699 1700 1701
 1702 1703 1704 1705 1706 1707 1708 1709 1710 1711 1750 1751 1752 1753
 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765 1766 1767
 1768 1769 1770 1771 1772 1773 1774 1775 1816 1817 1818 1819 1820 1821
 1822 1823 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834 1835
 1836 1837 1838 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892
 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1948 1949 1950 1951
 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965
 1966 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025
 2026 2027 2028 2029 2030 2079 2080 2081 2082 2083 2084 2085 2086 2087
 2088 2089 2090 2091 2092 2093 2094 2144 2145 2146 2147 2148 2149 2150
 2151 2152 2153 2154 2155 2156 2157 2158 2210 2211 2212 2213 2214 2215
 2216 2217 2218 2219 2220 2221 2222 2276 2277 2278 2279 2280 2281 2282
 2283 2284 2285 2286 2342 2343 2344 2345 2346 2347 2348 2349 2350]
```

For each dataframe in the list of fluorescent intensity dataframe, we apply the roi mask so that it only contains the data on pixels corresponding to our region of interest, the motor cortex (M1).

In [150…

```
# Applying roi mask for M1 cortex
ci_M1_df_list = []
for data in calcium_imaging_df_list:
    data = data.loc[M1_roi_idx].T
    ci_M1_df_list.append(data)
ci_M1_df_list[0]
```

Out[150…

	1238	1239	1240	1301	1302	1303	1304	1305	1306	1307	...	2286	
0	-0.397130	-0.653119	-0.829642	-0.237469	-0.545765	-0.593284	-0.585078	-0.701838	-0.723307	-0.659907	...	-0.660166	-0.6
1	-0.473834	-0.740661	-0.927336	-0.312852	-0.637405	-0.692798	-0.694614	-0.790996	-0.808154	-0.773173	...	-0.425994	-0.7
2	-0.660657	-0.900975	-1.096124	-0.473538	-0.811514	-0.874250	-0.847355	-0.955417	-1.011256	-0.993379	...	-0.907777	-0.9
3	-0.665365	-0.870937	-1.025627	-0.476715	-0.802048	-0.862595	-0.814823	-0.890919	-0.949035	-0.953278	...	-1.432244	-0.9
4	-0.386687	-0.610903	-0.723226	-0.196402	-0.442993	-0.547433	-0.551416	-0.599971	-0.655308	-0.647794	...	-0.359641	-0.6
...	...	...	...	...	...	...	...	...	...	...	...	...	
291	0.168888	0.356962	0.298350	-0.039746	0.310585	0.373754	0.354586	0.315539	0.266251	0.253512	...	-0.305301	-0.5
292	0.333215	0.495604	0.428269	0.203875	0.495434	0.526268	0.503897	0.435872	0.368246	0.374198	...	0.115647	-0.3
293	0.683110	0.829113	0.734896	0.601391	0.834797	0.811999	0.744044	0.679031	0.609262	0.597736	...	0.216134	-0.2
294	0.645374	0.777535	0.700724	0.694264	0.852684	0.769015	0.736064	0.677237	0.602074	0.594615	...	0.018170	-0.2
295	0.847482	0.963309	0.862634	0.964177	1.128199	1.004807	0.921786	0.875324	0.803899	0.771643	...	0.056498	-0.2

296 rows × 349 columns

Based on behavioral data, we know the timepoints when the pushing/reaching behavior started, based on that, we assigned labels to each timepoint for each trial, which 0 signifies resting state and 1 represents pushing/reaching behavior.

In [151…
```
# Assigning label based on movement onset in behavioral data
trial_push_onset = list(ci_behavior_df['onset_push'])

y_list = []
for i in range(len(trial_push_onset)):
    temp_df = ci_M1_df_list[i]
    y = np.zeros(temp_df.shape[0])
    y[trial_push_onset[i]:] = 1
    y_list.append(y)
```

In [152…
```
# Take a look at the assigned label for one of the trials
y_list[0].shape
```

Out[152… (296,)

Next, since contains almost twice as many timepoints for calcium imaging data than LFP data, to make fair comparison for classification results, we downsample the fluorescent intensity data so that it contains approximately same number of datapoints as the LFP data. We achieve that by taking the average of the fluorescent intensity for every two timepoints for each trial. We also applied the changes to the labels so that the dimension matches.

In [153…
```
## Apply downsampling
for i in range(len(ci_M1_df_list)):
    group_size = 2
    ci_M1_df_list[i] = ci_M1_df_list[i].groupby(ci_M1_df_list[i].index // group_size).mean()
    y_list[i] = y_list[i][::2]
print(ci_M1_df_list[0].shape)
print(y_list[0].shape)
```

```
(148, 349)
(148,)
```

Then, same as what we did for LFP data, we transformed the list of dataframes for fluorescent intensity into a single long-form dataframe for tsfresh feature extraction. We also concatenated the labels for all the trials.

In [154…
```
# Concatenate the labels for all trials
ci_y = np.concatenate(y_list)
ci_y.shape
```

Out[154… (1472,)

In [155…
```
# Transform the list of fluorescent intensity dataframe into long-form
ci_long_df = pd.DataFrame()
for i in range(len(ci_M1_df_list)):
    temp_df = ci_M1_df_list[i].copy()
    temp_df['time'] = [f'{i}_{j}' for j in temp_df.index]
    temp_df = temp_df.melt(id_vars='time', var_name='neuron').sort_values(['time', 'neuron'])
    temp_df = temp_df.reset_index(drop=True)
    ci_long_df = pd.concat([ci_long_df, temp_df])
```

In [156…
```
# Take a look at the long-form dataframe
ci_long_df
```

Out[156...

	time	neuron	value
0	0_0	1238	-0.435482
1	0_0	1239	-0.696890
2	0_0	1240	-0.878489
3	0_0	1301	-0.275160
4	0_0	1302	-0.591585
...	...	...	...
45016	9_99	2346	0.127247
45017	9_99	2347	0.294129
45018	9_99	2348	0.278994
45019	9_99	2349	0.238811
45020	9_99	2350	-0.027169

513728 rows × 3 columns

Feature extraction of calcium imaging (fluorescent intensity) data using tsfresh

Same as for LFP data, we applied feature extraction for fluorescent intensity data and sort the index.

In [157...
```
# Extract tsfresh features
ci_extracted_features = extract_features(ci_long_df, column_id="time", column_sort="neuron")
```

```
Feature Extraction: 100%|██████████| 20/20 [03:00<00:00,  9.02s/it]
```

In [158...
```
# Sort index
ci_extracted_features = sort_trialtime_index(ci_extracted_features)
ci_extracted_features
```

Out[158...

trial_time	value__variance_larger_than_standard_deviation	value__has_duplicate_max	value__has_duplicate_min	value__has_duplicate	valu
0_0	0.0	0.0	0.0	0.0	
0_1	0.0	0.0	0.0	0.0	
0_2	0.0	0.0	0.0	0.0	
0_3	0.0	0.0	0.0	0.0	
0_4	0.0	0.0	0.0	0.0	
...	...	...	...	...	
9_124	0.0	0.0	0.0	0.0	
9_125	0.0	0.0	0.0	0.0	
9_126	0.0	0.0	0.0	0.0	
9_127	0.0	0.0	0.0	0.0	
9_128	0.0	0.0	0.0	0.0	

1472 rows × 783 columns

Next, due to the same reason, we also imputed the missing values for the extracted fluorescent intensity features. We also selected only the statistically significant relevant features from the extracted features. However, there are more statistically significant features for fluorescent intensity data than LFP data. To ensure that there are equal number of features included in model training for both LFP and fluorescent intensity data for fair comparison, we will include only the top 363 statistically significant fluorescent intensity features to match the number of features used in constructing classifier for LFP data

In [177...
```
# Drop NaN and impute missing values
ci_extracted_features = impute(ci_extracted_features)

# Select statistically significant relevant features
ci_selected_features = select_features(ci_extracted_features, ci_y)
print(ci_selected_features.shape)
```

```
(1472, 538)
```

In [178...
```
# Import relevance table
from tsfresh.feature_selection.relevance import calculate_relevance_table

# Sort and query for 363 features with highest relevance
```

```
relevance_table = calculate_relevance_table(ci_selected_features, pd.Series(ci_y, index=ci_selected_features.index))
top_363_features = relevance_table.nlargest(363, "p_value").index  # Lower p-value = more relevant

# Keep only the top 363 features
ci_selected_features = ci_selected_features[top_363_features]
```

In [255…

```
# Take a look at the selected features dataframe
ci_selected_features.head()
```

Out[255…

	value__fft_coefficient__attr_"abs"__coeff_72	value__agg_linear_trend__attr_"rvalue"__chunk_len_5__f_agg_"max"	value__fft_coefficient__attı
0	4.055416	-0.185764	
1	4.586821	-0.301961	
2	2.912076	-0.372886	
3	2.820196	-0.442166	
4	2.695957	-0.471792	

5 rows × 363 columns

We also write the selected feature matrix into csv for convinient retrieval.

In [181…

```
# Writing selected feature matrix into csv
# ci_selected_features.to_csv('/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/tsfresh_feature_matrix/ci_features.csv'

# Loading in the selected feature matrix if needed
ci_selected_features = pd.read_csv('/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/tsfresh_feature_matrix/ci_features
ci_selected_features = ci_selected_features.iloc[:, 1:]
ci_selected_features
```

Out[181…

	value__fft_coefficient__attr_"abs"__coeff_72	value__agg_linear_trend__attr_"rvalue"__chunk_len_5__f_agg_"max"	value__fft_coefficient_
0	4.055416	-0.185764	
1	4.586821	-0.301961	
2	2.912076	-0.372886	
3	2.820196	-0.442166	
4	2.695957	-0.471792	
...	...	...	
1467	2.206159	-0.521104	
1468	2.842053	-0.515537	
1469	1.133967	-0.575509	
1470	2.784085	-0.520522	
1471	3.007726	-0.509891	

1472 rows × 363 columns

We then also apply normalization to the selected features from the calcium fluorescent intensity data.

In [182…

```
# Apply normalziation of selected features with StandardScaler
scaler = StandardScaler()
ci_normalized_features = scaler.fit_transform(ci_selected_features)
ci_normalized_features = pd.DataFrame(ci_normalized_features, columns=ci_selected_features.columns)
ci_normalized_features
```

Out[182...

	value__fft_coefficient__attr_"abs"__coeff_72	value__agg_linear_trend__attr_"rvalue"__chunk_len_5__f_agg_"max"	value__fft_coefficient_
0	0.441245	-0.814156	
1	0.712446	-1.138497	
2	-0.142252	-1.336469	
3	-0.189143	-1.529850	
4	-0.252548	-1.612543	
...	...	...	
1467	-0.502514	-1.750187	
1468	-0.177988	-1.734649	
1469	-1.049701	-1.902048	
1470	-0.207572	-1.748563	
1471	-0.093438	-1.718889	

1472 rows × 363 columns

We then write the normalized feature matrix into csv and make sure that each column has a mean of 0 due to normalization.

In [184...

```
# Writng normalized feature matrix into csv
# ci_normalized_features.to_csv('/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/tsfresh_feature_matrix/normalized_ci_

# Loading in normalized feature matrix if needed
ci_normalized_features = pd.read_csv('/Users/Devin Chang/Documents/BIPN162/BIPN162_Cell_Types/tsfresh_feature_matrix/normalize
ci_normalized_features = ci_normalized_features.iloc[:, 1:]
ci_normalized_features
```

Out[184...

	value__fft_coefficient__attr_"abs"__coeff_72	value__agg_linear_trend__attr_"rvalue"__chunk_len_5__f_agg_"max"	value__fft_coefficient_
0	0.441245	-0.814156	
1	0.712446	-1.138497	
2	-0.142252	-1.336469	
3	-0.189143	-1.529850	
4	-0.252548	-1.612543	
...	...	...	
1467	-0.502514	-1.750187	
1468	-0.177988	-1.734649	
1469	-1.049701	-1.902048	
1470	-0.207572	-1.748563	
1471	-0.093438	-1.718889	

1472 rows × 363 columns

In [185...

```
ci_normalized_features.mean()
```

Out[185...

```
value__fft_coefficient__attr_"abs"__coeff_72                          4.489163e-16
value__agg_linear_trend__attr_"rvalue"__chunk_len_5__f_agg_"max"      6.275174e-17
value__fft_coefficient__attr_"imag"__coeff_1                          1.930823e-17
value__fft_coefficient__attr_"angle"__coeff_51                        4.827057e-18
value__fft_coefficient__attr_"imag"__coeff_51                        -2.896234e-17
                                                                           ...
value__fft_coefficient__attr_"angle"__coeff_22                        2.413528e-17
value__fft_coefficient__attr_"angle"__coeff_92                        0.000000e+00
value__linear_trend__attr_"stderr"                                    4.827057e-17
value__fft_coefficient__attr_"real"__coeff_83                        -3.378940e-17
value__fft_coefficient__attr_"abs"__coeff_10                          1.206764e-17
Length: 363, dtype: float64
```

Classification of Marmoset Pushing/Reaching Behavior using Calcium Imaging (Fluorescent Intensity) Data

Same as for LFP data, we first split the fluorescent intensity data and labels into training and test sets. Then observe the number of datapoints for each class. We see that there is also a class imbalance in the dataset which there are more datapoints for resting state than reaching/pushing behavior.

In [209…
```
# Split into train and test sets
ci_X_train, ci_X_test, ci_y_train, ci_y_test = train_test_split(ci_normalized_features,
                                                                ci_y, test_size=0.2, random_state=2, stratify=ci_y)
```

In [210…
```
# Observe number of datapoints for each class
pd.Series(ci_y_train).value_counts()
```

Out[210…
```
0.0    724
1.0    453
Name: count, dtype: int64
```

Logistic Regression Model:

We also constructed a logistic regression classifier for the fluorescent intensity data. The settings are the same as for the logistic regression classifier for LFP data, with max_iter=1000 and class_weight='balanced' due to imbalanced classes. As shown below, it has a classification accuracy of about 0.84, with precision, recall and f1-score mostly within 0.8-0.9. The logistic regression model trained on fluorescent intensity data has lower performance that that of the model trained on LFP data.

In [232…
```
# Train a logistic regression classifier
ci_lr_clf = LogisticRegression(class_weight='balanced', max_iter=1000, random_state=1)
ci_lr_clf.fit(ci_X_train, ci_y_train)

# Make predictions
ci_lr_y_pred = ci_lr_clf.predict(ci_X_test)

# Evaluate
print("Accuracy:", accuracy_score(ci_y_test, ci_lr_y_pred))
print(classification_report(ci_y_test, ci_lr_y_pred))
```

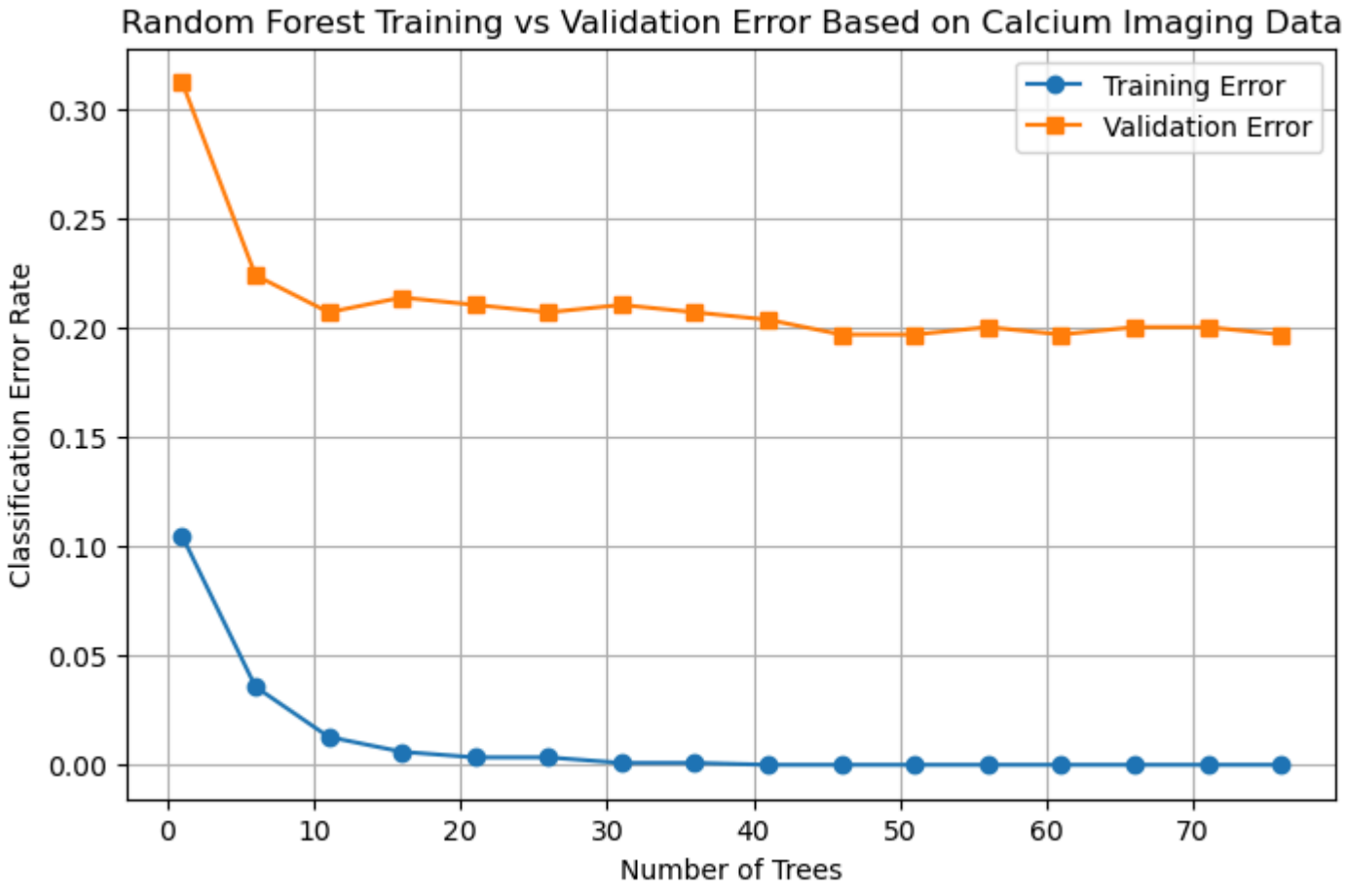
```
Accuracy: 0.8406779661016949
              precision    recall  f1-score   support

         0.0       0.89      0.85      0.87       182
         1.0       0.78      0.82      0.80       113

    accuracy                           0.84       295
   macro avg       0.83      0.84      0.83       295
weighted avg       0.84      0.84      0.84       295
```

Random Forest Model:

Then, we construct a random forest model for fluorescent intensity data. We also perform hyperparameter tuning for the random forest classifier. As shown in the classification error rate vs. number of trees plot below, the classification error rate for training and validation set both plateaus at about 21 trees. However, to make fair comparison between classifying LFP and fluorescent intensity data, we will also use 26 trees (same as LFP random forest classifier) for the classifier for fluorescent intensity data.

In [222…
```
# Finding train and validation errors of random forest for different number of trees
clf_accuracies = []
train_errors, val_errors = [], []
num_estimators = np.arange(1, 80, 5)
for num in num_estimators:
    temp_clf = RandomForestClassifier(n_estimators=num, class_weight='balanced', random_state=1)
    temp_clf.fit(ci_X_train, ci_y_train)
    temp_test_pred = temp_clf.predict(ci_X_test)
    temp_train_pred = temp_clf.predict(ci_X_train)
    clf_accuracies.append(accuracy_score(ci_y_test, temp_test_pred))
    train_errors.append(1 - accuracy_score(ci_y_train, temp_train_pred))
    val_errors.append(1 - accuracy_score(ci_y_test, temp_test_pred))
```

In [224…
```
# Plot the classification error vs. number of trees curve
plt.figure(figsize=(8, 5))
plt.plot(num_estimators, train_errors, label="Training Error", marker='o')
plt.plot(num_estimators, val_errors, label="Validation Error", marker='s')
plt.xlabel("Number of Trees")
plt.ylabel("Classification Error Rate")
plt.title("Random Forest Training vs Validation Error Based on Calcium Imaging Data")
plt.legend()
plt.grid()
plt.show()
```



Finally, we train the random forest classifier with 26 trees on the calcium imaging (fluorescent intensity) data. The class_weight is also set as 'balanced' to account for imbalanced classes. As shown below, the classification accuracy is 0.79 with precision, recall, and f1-score around 0.8. The performance of the random forest classifier is poorer than the logistic regression classifier also trained on fluorescent intensity data. Furthermore, it also have lower peformance than that of the random forest model trained on LFP data.

In [233…
```python
# Train a Random Forest Classifier
ci_rf_clf = RandomForestClassifier(n_estimators=26, class_weight='balanced', random_state=1)
ci_rf_clf.fit(ci_X_train, ci_y_train)

# Make predictions
ci_y_pred = ci_rf_clf.predict(ci_X_test)

# Evaluate
print("Accuracy:", accuracy_score(ci_y_test, ci_y_pred))
print(classification_report(ci_y_test, ci_y_pred))
```

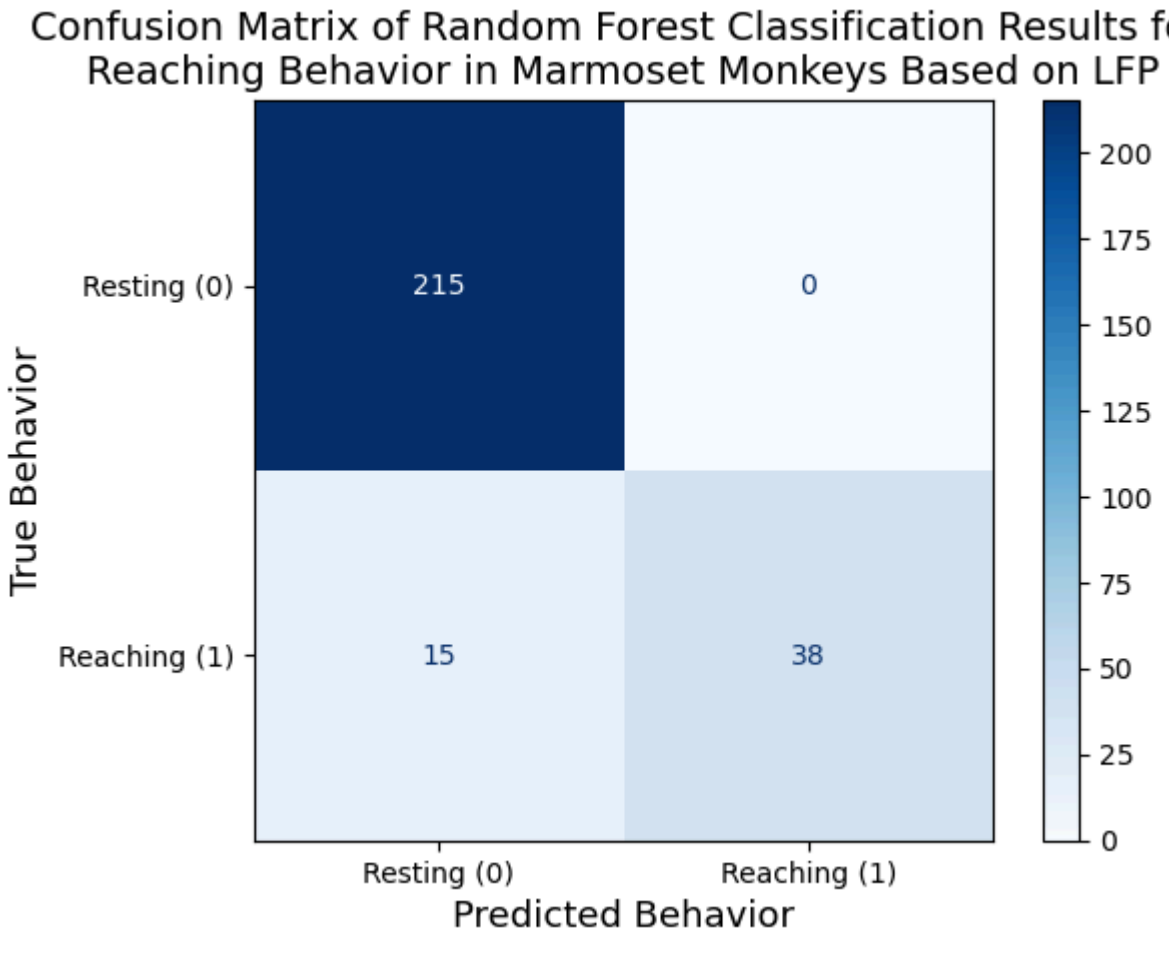
```
Accuracy: 0.7932203389830509
              precision    recall  f1-score   support

         0.0       0.82      0.85      0.83       182
         1.0       0.74      0.71      0.72       113

    accuracy                           0.79       295
   macro avg       0.78      0.78      0.78       295
weighted avg       0.79      0.79      0.79       295
```

Plots for Comparison

Confusion Matrices:

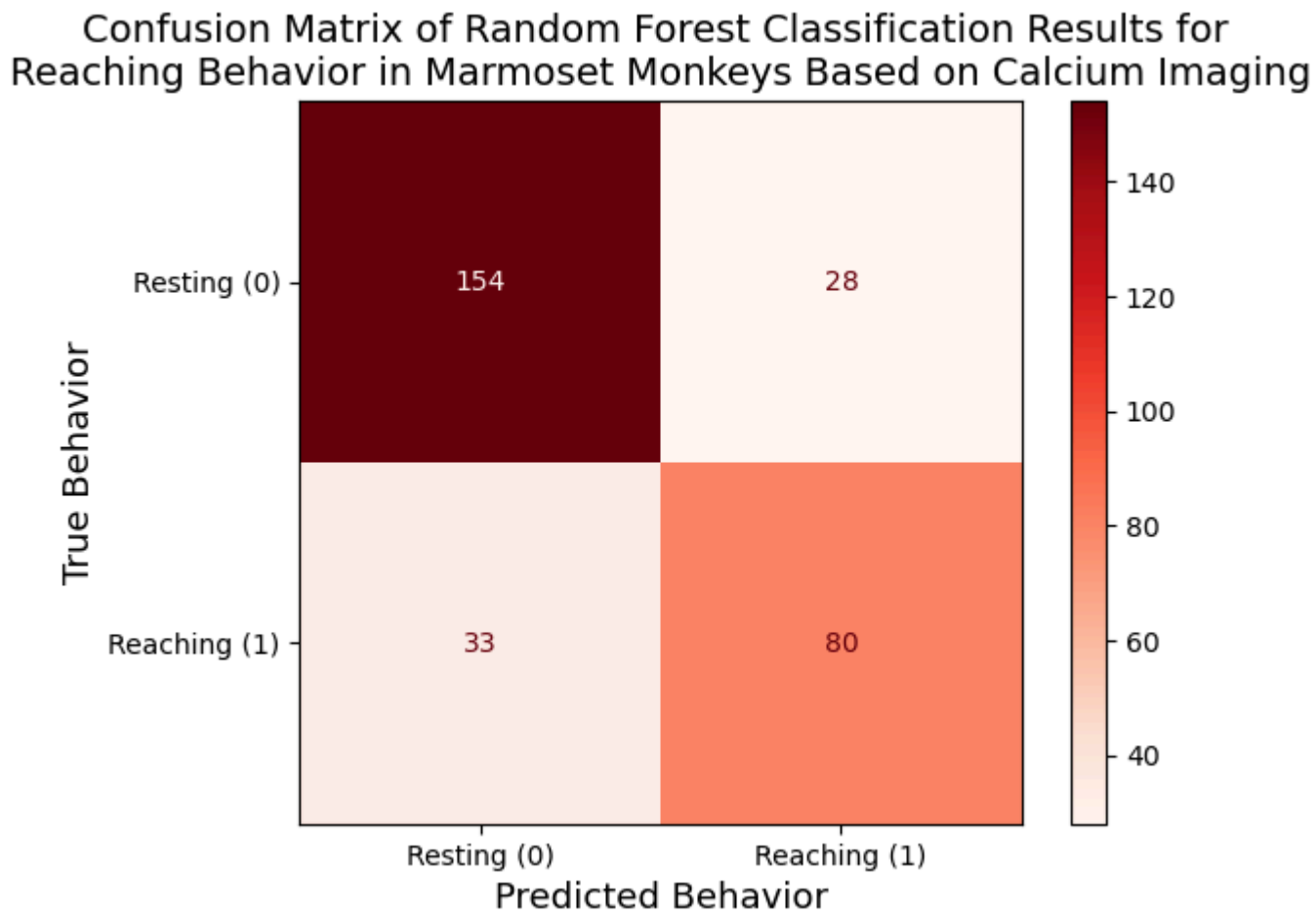
We first plot confusion matrices for both logistic regression and random forest model trained on LFP and calcium imaging (fluorescent intensity) data.

In [234…
```python
# Import confusion matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Plot confusion matrix for random forest classifier trained on LFP data
cm_1 = confusion_matrix(lfp_y_test, lfp_y_pred)
display_cm_1 = ConfusionMatrixDisplay(confusion_matrix=cm_1)
display_cm_1.plot(cmap='Blues')
plt.xlabel('Predicted Behavior', fontsize=13)
plt.ylabel('True Behavior', fontsize=13)
plt.xticks([0, 1], ['Resting (0)', 'Reaching (1)'], fontsize=10)
plt.yticks([0, 1], ['Resting (0)', 'Reaching (1)'], fontsize=10)
plt.title('Confusion Matrix of Random Forest Classification Results for \nReaching Behavior in Marmoset Monkeys Based on LFP',
plt.show()
```



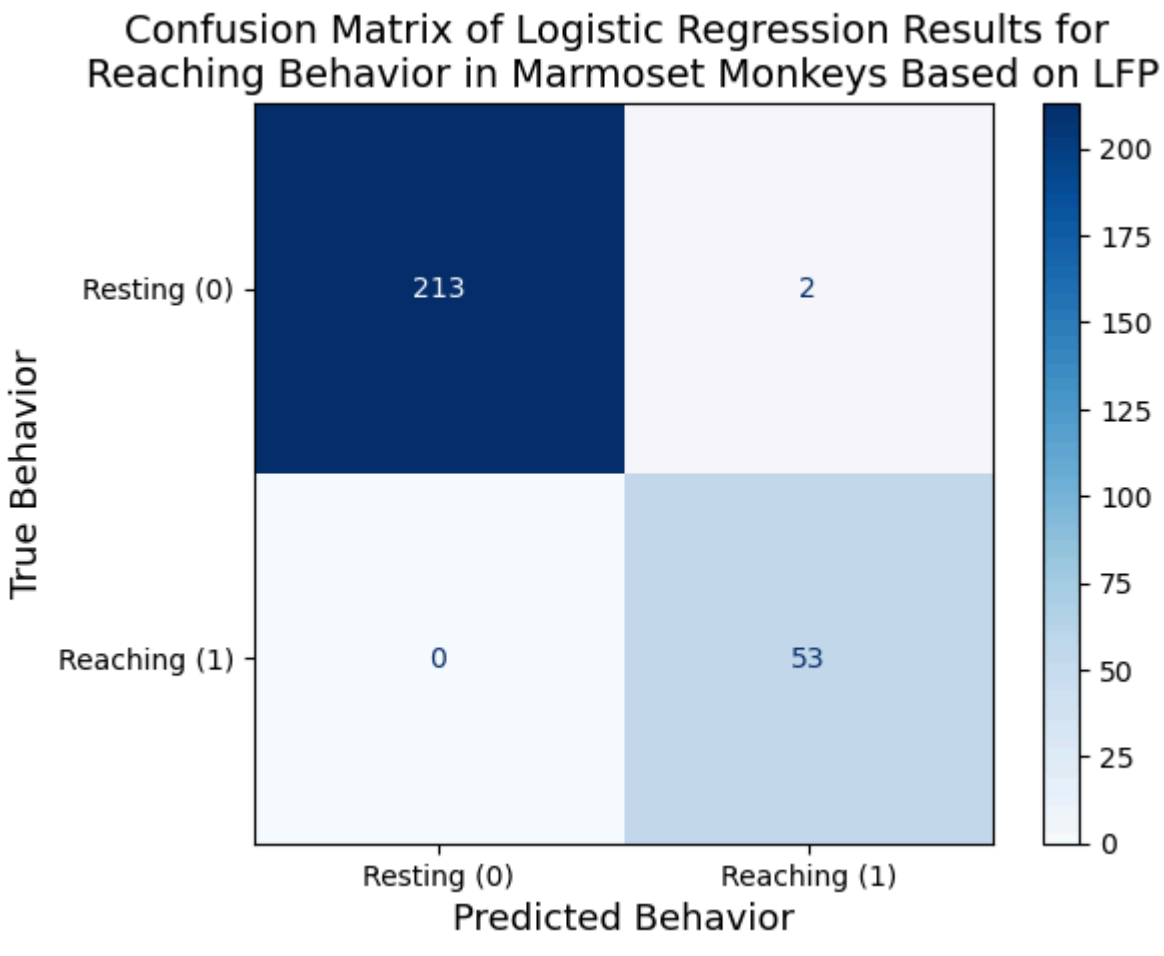
In [235…

```
# Plot confusion matrix of random forest classifier trained on calcium imaging (fluorescent intensity) data
cm_2 = confusion_matrix(ci_y_test, ci_y_pred)
display_cm_2 = ConfusionMatrixDisplay(confusion_matrix=cm_2)
display_cm_2.plot(cmap='Reds')
plt.xlabel('Predicted Behavior', fontsize=13)
plt.ylabel('True Behavior', fontsize=13)
plt.xticks([0, 1], ['Resting (0)', 'Reaching (1)'], fontsize=10)
plt.yticks([0, 1], ['Resting (0)', 'Reaching (1)'], fontsize=10)
plt.title('Confusion Matrix of Random Forest Classification Results for \nReaching Behavior in Marmoset Monkeys Based on Calci
plt.show()
```



In [236…

```
# Plot confusion matrix for logistic regression classifier trained on LFP data
cm_3 = confusion_matrix(lfp_y_test, lfp_lr_y_pred)
display_cm_3 = ConfusionMatrixDisplay(confusion_matrix=cm_3)
display_cm_3.plot(cmap='Blues')
plt.xlabel('Predicted Behavior', fontsize=13)
plt.ylabel('True Behavior', fontsize=13)
plt.xticks([0, 1], ['Resting (0)', 'Reaching (1)'], fontsize=10)
plt.yticks([0, 1], ['Resting (0)', 'Reaching (1)'], fontsize=10)
plt.title('Confusion Matrix of Logistic Regression Results for \nReaching Behavior in Marmoset Monkeys Based on LFP', fontsize
plt.show()
```



In [237…

```
# Plot confusion matrix for logistic regression classifier trained on fluorescent intensity data
cm_4 = confusion_matrix(ci_y_test, ci_lr_y_pred)
display_cm_4 = ConfusionMatrixDisplay(confusion_matrix=cm_4)
display_cm_4.plot(cmap='Reds')
plt.xlabel('Predicted Behavior', fontsize=13)
plt.ylabel('True Behavior', fontsize=13)
plt.xticks([0, 1], ['Resting (0)', 'Reaching (1)'], fontsize=10)
plt.yticks([0, 1], ['Resting (0)', 'Reaching (1)'], fontsize=10)
plt.title('Confusion Matrix of Logistic Regression Results for \nReaching Behavior in Marmoset Monkeys Based on Calcium Imagin
plt.show()
```

ROC-AUC

We plot the roc-auc curve for both logistic regression and random forest classifiers trained on LFP and calcium imaging data.

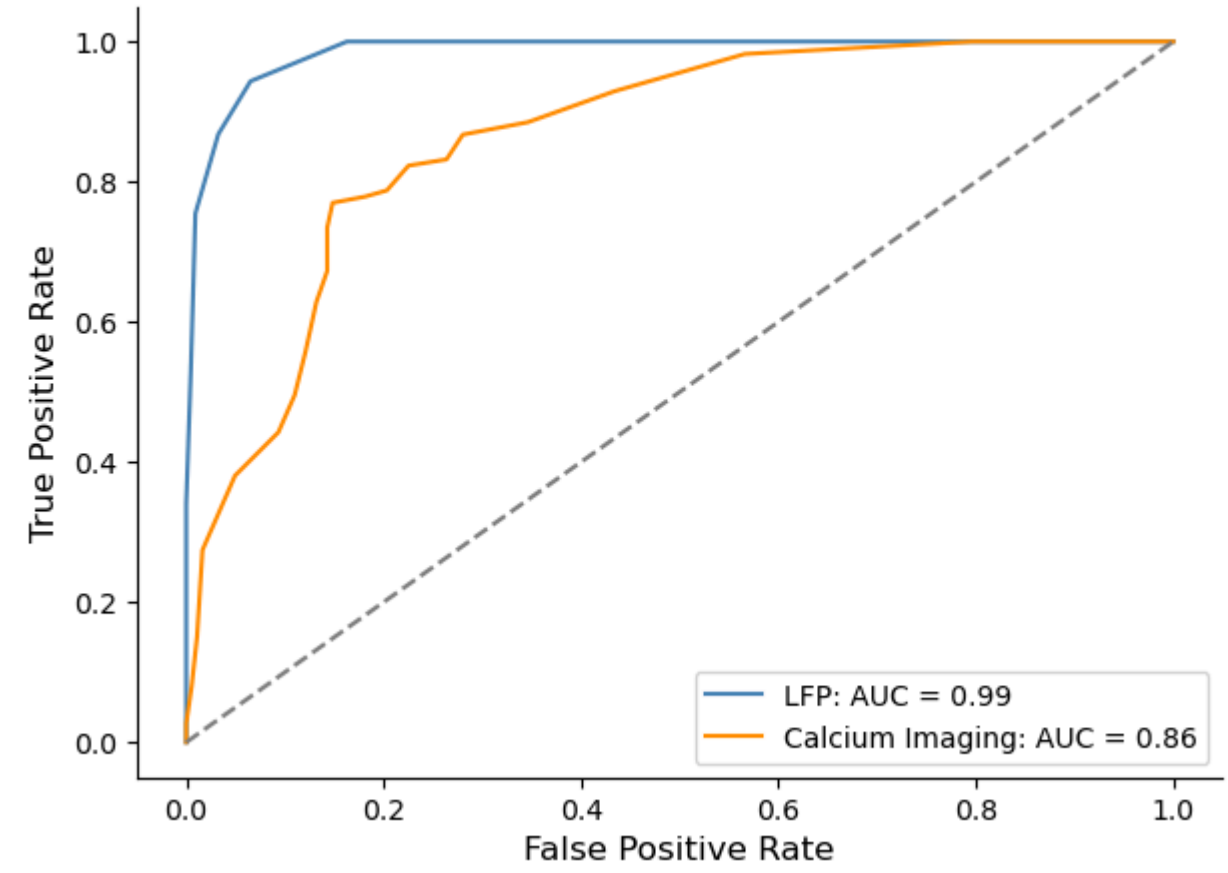
In [238…

```
# Calculate prediction probabilities
lfp_y_pred_prob = lfp_rf_clf.predict_proba(lfp_X_test)[:, 1]
ci_y_pred_prob = ci_rf_clf.predict_proba(ci_X_test)[:, 1]
lfp_lr_y_pred_prob = lfp_lr_clf.predict_proba(lfp_X_test)[:, 1]
ci_lr_y_pred_prob = ci_lr_clf.predict_proba(ci_X_test)[:, 1]
print(lfp_y_pred_prob.shape)
print(ci_y_pred_prob.shape)
print(lfp_lr_y_pred_prob.shape)
print(ci_lr_y_pred_prob.shape)
```

```
(268,)
(295,)
(268,)
(295,)
```

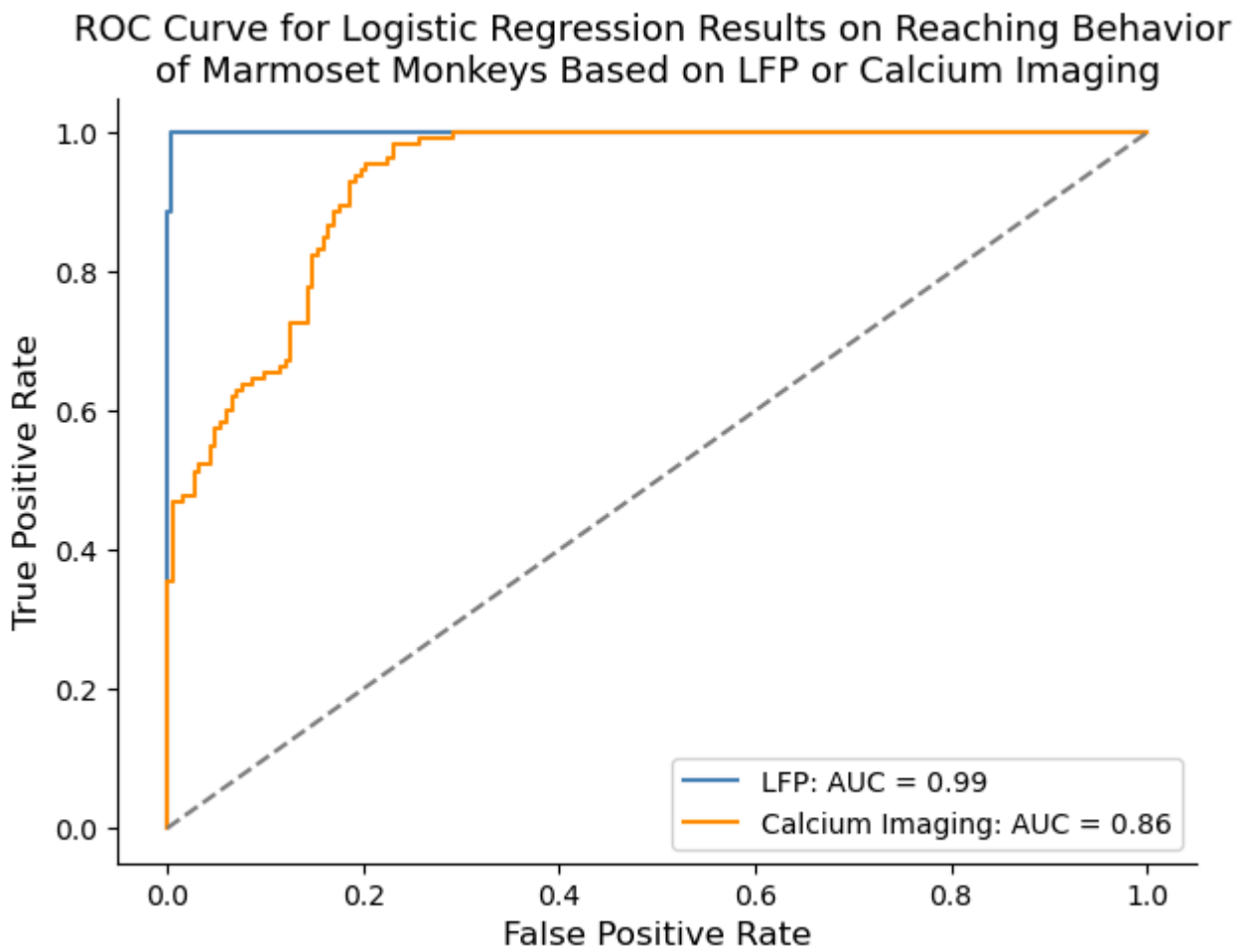
In [198…

```
# Import packages for roc-auc
from sklearn.metrics import roc_curve, auc

# Plot the roc-auc curves for random forest classifier
fpr_1, tpr_1, _ = roc_curve(lfp_y_test, lfp_y_pred_prob)
fpr_2, tpr_2, _ = roc_curve(ci_y_test, ci_y_pred_prob)
roc_auc_1 = auc(fpr_1, tpr_1)
roc_auc_2 = auc(fpr_2, tpr_2)

fig, ax = plt.subplots(figsize=(7, 5))
plt.plot(fpr_1, tpr_1, label=f'LFP: AUC = {roc_auc_1:.2f}', color='#4682B4')
plt.plot(fpr_2, tpr_2, label=f'Calcium Imaging: AUC = {roc_auc_2:.2f}', color='#FF8C00')
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")  # Random model
plt.xlabel("False Positive Rate", fontsize=12)
plt.ylabel("True Positive Rate", fontsize=12)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.title("ROC Curve for Random Forest Classification Results on Reaching Behavior \nof Marmoset Monkeys Based on LFP or Calci
plt.legend()
plt.show()
```



In [239…

```
# Plot the roc-auc curves for logistic regression classifier
fpr_3, tpr_3, _ = roc_curve(lfp_y_test, lfp_lr_y_pred_prob)
fpr_4, tpr_4, _ = roc_curve(ci_y_test, ci_lr_y_pred_prob)
roc_auc_3 = auc(fpr_3, tpr_3)
roc_auc_4 = auc(fpr_4, tpr_4)

fig, ax = plt.subplots(figsize=(7, 5))
plt.plot(fpr_3, tpr_3, label=f'LFP: AUC = {roc_auc_1:.2f}', color='#4682B4')
plt.plot(fpr_4, tpr_4, label=f'Calcium Imaging: AUC = {roc_auc_2:.2f}', color='#FF8C00')
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")  # Random model
plt.xlabel("False Positive Rate", fontsize=12)
plt.ylabel("True Positive Rate", fontsize=12)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
plt.title("ROC Curve for Logistic Regression Results on Reaching Behavior \nof Marmoset Monkeys Based on LFP or Calcium Imagin
plt.legend()
plt.show()
```


ROC Curve for Logistic Regression Results on Reaching Behavior
of Marmoset Monkeys Based on LFP or Calcium Imaging
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
LFP: AUC = 0.99
Calcium Imaging: AUC = 0.86