In [1]:
```
# visualizes the group parcellations (averaged across the subjects) for all
```

In [1]:
```
import os
import numpy as np
import pandas as pd
import nibabel as nib
import nilearn
import json
import datetime
import pickle
import seaborn as sns
import gc
import psutil
import math
import scipy.stats as stats
from matplotlib.patches import Patch
from nilearn import plotting
from nilearn.glm.first_level import FirstLevelModel
from nilearn.glm.second_level import SecondLevelModel
from nilearn.glm import threshold_stats_img
from nilearn.image import concat_imgs, mean_img, index_img
from nilearn.reporting import make_glm_report
from nilearn import masking, image
from nilearn import datasets
from scipy.stats import pearsonr
import matplotlib.pyplot as plt
from collections import defaultdict
from nilearn.maskers import NiftiLabelsMasker
from nilearn.plotting.find_cuts import find_cut_slices
import nibabel as nib
```

In [2]:
```
# general helper functions:
def cleanup_memory():
    """
    Clean up memory between batches
    """
    # Force garbage collection
    gc.collect()

    # Get memory info
    memory = psutil.virtual_memory()
    print(f"Memory after cleanup: {memory.percent:.1f}% used ({memory.availa
```

In [3]:
```
# all tasks and contrasts
TASKS = ["nBack","flanker","directedForgetting","goNogo", "shapeMatching", "
CONTRASTS = {}
CONTRASTS["nBack"] = ["twoBack-oneBack", "match-mismatch","task-baseline","r
```

```
CONTRASTS["flanker"] = ["incongruent-congruent", "task-baseline"]
CONTRASTS["directedForgetting"] = ["neg-con", "task-baseline","response_time
CONTRASTS["goNogo"] = ["nogo_success-go", "nogo_success","task-baseline","re
CONTRASTS["shapeMatching"] = ["DDD", "DDS", "DNN", "DSD", "main_vars", "SDD"
CONTRASTS["stopSignal"] = ["go", "stop_failure-go", "stop_failure", "stop_fa
CONTRASTS["cuedTS"] = ["cue_switch_cost", "task_switch_cost", "task_switch_c
CONTRASTS["spatialTS"] = ["cue_switch_cost", "task_switch_cost", "task_switc

# interested in looking at them all now:
requested_task_contrasts = defaultdict(lambda: defaultdict(list))
requested_task_contrasts['nBack'] = CONTRASTS["nBack"]
requested_task_contrasts['flanker'] = CONTRASTS["flanker"]
requested_task_contrasts['directedForgetting'] = CONTRASTS["directedForgetti
requested_task_contrasts['goNogo'] = CONTRASTS["goNogo"]
requested_task_contrasts['shapeMatching'] = CONTRASTS["shapeMatching"]
requested_task_contrasts['stopSignal'] = CONTRASTS["stopSignal"]
requested_task_contrasts['cuedTS'] = CONTRASTS["cuedTS"]
requested_task_contrasts['spatialTS'] = CONTRASTS["spatialTS"]

# compiled_req_contrasts = ["twoBack-oneBack", 'task-baseline', "incongruent
# compile all requested contrasts into one list
compiled_req_contrasts = []
for task in TASKS:
    for contrast in requested_task_contrasts[task]:
        if (contrast not in compiled_req_contrasts):
            compiled_req_contrasts.append(contrast)


ENCOUNTERS = ['01', '02','03','04','05']
SUBJECTS = ['sub-s03', 'sub-s10', 'sub-s19', 'sub-s29', 'sub-s43']
```

load atlas

In [4]:
```
# schafer stuff
SCHAFER_PARCELLATED_DIR = 'schafer400_dfs'
schafer_files = {'mean':f'discovery_parcel_indiv_mean_updated'}
schafer_date_updated = '1001'
indices = [1,2,3]
# Get schaefer atlas
SCHAEFER = datasets.fetch_atlas_schaefer_2018(n_rois=400)
SCHAEFER_IMG = nib.load(SCHAEFER.maps)
SCHAEFER_DATA = SCHAEFER_IMG.get_fdata()

# smorgasbord stuff
SMORG_PARCELLATED_DIR = 'smor_parcel_dfs'
smor_files = {'mean':f'discovery_parcel_indiv_mean_updated'}
smor_date_updated = '1027'
indices = [1,2,3]
```

```
# get smorgasbord atlas
with open(f'{SMORG_PARCELLATED_DIR}/smorgasbord_atlas_files/smorgasbord_atla
    smorgasbord_atlas = pickle.load(f)
SMORG_IMG = smorgasbord_atlas.maps
SMORG_DATA = SMORG_IMG.get_fdata()
```

```
[get_dataset_dir] Dataset found in /home/users/nklevak/nilearn_data/sc
```

In [5]:
```
req_atlas = "smor"

# Select atlas configuration
if req_atlas == "schafer":
    main_dir = SCHAFER_PARCELLATED_DIR
    main_files = schafer_files
    date_updated = schafer_date_updated
    atlas_obj = SCHAEFER
    atlas_img = SCHAEFER_IMG
    atlas_data = SCHAEFER_DATA
elif req_atlas == "smor":
    main_dir = SMORG_PARCELLATED_DIR
    main_files = smor_files
    date_updated = smor_date_updated
    atlas_obj = smorgasbord_atlas
    atlas_img = SMORG_IMG
    atlas_data = SMORG_DATA
else:
    raise ValueError(f"Unknown atlas: {req_atlas}. Use 'schafer' or 'smor'")
```

loading fixed effect maps

In [6]:
```
# LOADING ALL FIXED EFFECTS:
BASE_DIR = '/oak/stanford/groups/russpold/data/network_grant/discovery_BIDS_
LEVEL = 'output_lev1_mni'
# subjects in the discovery sample
SUBJECTS = ['sub-s03', 'sub-s10', 'sub-s19', 'sub-s29', 'sub-s43']
SESSIONS = ['ses-01', 'ses-02', 'ses-03', 'ses-04', 'ses-05', 'ses-06', 'ses

fe_all_contrast_maps = defaultdict(lambda: defaultdict(list))

for task in TASKS:
    for contrast_name in CONTRASTS[task]:
        for subject in SUBJECTS:
            filename = f'{subject}_task-{task}_contrast-{contrast_name}_rtmc
            contrast_map_path = os.path.join(BASE_DIR, LEVEL, subject, task,

            if os.path.exists(contrast_map_path):
                fe_all_contrast_maps[task][contrast_name].append(contrast_ma
```

```
            else:
                print(f"{contrast_map_path} does not exist.")
fe_session_maps = fe_all_contrast_maps

# load each fixed effects map
specified_maps = []
specified_descriptors = []
for task in TASKS:
    for contrast in CONTRASTS[task]:
        for map in fe_all_contrast_maps[task][contrast]:
            subj_id = map.split("mni/sub-")[1]
            subj = subj_id.split("/")[0]

            descriptor_name = f"{subj}:fixed_effect:{task}|{contrast}"

            if isinstance(map, str):
                # map_data is a file path, need to load it
                try:
                    if os.path.exists(map):
                        loaded_map = nib.load(map)
                        specified_maps.append(loaded_map)
                        specified_descriptors.append(descriptor_name)
                    else:
                        print(f"File not found: {map}")
                        failed_loads.append((descriptor_name, "File not four
                except Exception as e:
                    print(f"Error loading {map}: {str(e)}")
            else:
                print(f"Unexpected data type for {descriptor_name}: {type(ma

def average_across_participants(specified_maps):
    """
    Average the fixed effects maps across participants for a specific task a
    """
    sum_data = None
    count = 0
    affine = None

    for map_path in specified_maps:
        try:
            if os.path.exists(map_path):
                img = nib.load(map_path)

                if sum_data is None:
                    sum_data = img.get_fdata()
                    affine = img.affine
                else:
                    sum_data += img.get_fdata()
                count += 1
```

```
        except Exception as e:
            print(f"Error loading {map_path}: {str(e)}")

    if count > 0:
        avg_data = sum_data / count
        return nib.Nifti1Image(avg_data, affine)
    else:
        raise ValueError("No valid images found for averaging")

# Average the fixed effects maps
averaged_fixed_maps = defaultdict(lambda: defaultdict(dict))
for task in TASKS:
    for contrast_name in CONTRASTS[task]:
        try:
            averaged_fixed_maps[task][contrast_name] = average_across_partic
            print(f"Averaged map created for {task}/{contrast_name}")
        except Exception as e:
            print(f"Error averaging maps for {task}/{contrast_name}: {str(e)

# Print structure of averaged_fixed_maps
for task in averaged_fixed_maps:
    for contrast in averaged_fixed_maps[task]:
        print(f"{task}/{contrast}: {type(averaged_fixed_maps[task][contrast]
```

```
Averaged map created for nBack/twoBack-oneBack
Averaged map created for nBack/match-mismatch
Averaged map created for nBack/task-baseline
Averaged map created for nBack/response_time
Averaged map created for flanker/incongruent-congruent
Averaged map created for flanker/task-baseline
Averaged map created for directedForgetting/neg-con
Averaged map created for directedForgetting/task-baseline
Averaged map created for directedForgetting/response_time
Averaged map created for goNogo/nogo_success-go
Averaged map created for goNogo/nogo_success
Averaged map created for goNogo/task-baseline
Averaged map created for goNogo/response_time
Averaged map created for shapeMatching/DDD
Averaged map created for shapeMatching/DDS
Averaged map created for shapeMatching/DNN
Averaged map created for shapeMatching/DSD
Averaged map created for shapeMatching/main_vars
Averaged map created for shapeMatching/SDD
Averaged map created for shapeMatching/SNN
Averaged map created for shapeMatching/SSS
Averaged map created for shapeMatching/task-baseline
Averaged map created for shapeMatching/response_time
Averaged map created for stopSignal/go
Averaged map created for stopSignal/stop_failure-go
Averaged map created for stopSignal/stop_failure
```

```
Averaged map created for stopSignal/stop_failure-stop_success
Averaged map created for stopSignal/stop_success-go
Averaged map created for stopSignal/stop_success
Averaged map created for stopSignal/stop_success-stop_failure
Averaged map created for stopSignal/task-baseline
Averaged map created for stopSignal/response_time
Averaged map created for cuedTS/cue_switch_cost
Averaged map created for cuedTS/task_switch_cost
Averaged map created for cuedTS/task_switch_cue_switch-task_stay_cue_stay
Averaged map created for cuedTS/task-baseline
Averaged map created for cuedTS/response_time
Averaged map created for spatialTS/cue_switch_cost
Averaged map created for spatialTS/task_switch_cost
Averaged map created for spatialTS/task_switch_cue_switch-task_stay_cue_stay
Averaged map created for spatialTS/task-baseline
Averaged map created for spatialTS/response_time
nBack/twoBack-oneBack: <class 'nibabel.nifti1.Nifti1Image'>
nBack/match-mismatch: <class 'nibabel.nifti1.Nifti1Image'>
nBack/task-baseline: <class 'nibabel.nifti1.Nifti1Image'>
nBack/response_time: <class 'nibabel.nifti1.Nifti1Image'>
flanker/incongruent-congruent: <class 'nibabel.nifti1.Nifti1Image'>
flanker/task-baseline: <class 'nibabel.nifti1.Nifti1Image'>
directedForgetting/neg-con: <class 'nibabel.nifti1.Nifti1Image'>
directedForgetting/task-baseline: <class 'nibabel.nifti1.Nifti1Image'>
directedForgetting/response_time: <class 'nibabel.nifti1.Nifti1Image'>
goNogo/nogo_success-go: <class 'nibabel.nifti1.Nifti1Image'>
goNogo/nogo_success: <class 'nibabel.nifti1.Nifti1Image'>
goNogo/task-baseline: <class 'nibabel.nifti1.Nifti1Image'>
goNogo/response_time: <class 'nibabel.nifti1.Nifti1Image'>
shapeMatching/DDD: <class 'nibabel.nifti1.Nifti1Image'>
shapeMatching/DDS: <class 'nibabel.nifti1.Nifti1Image'>
shapeMatching/DNN: <class 'nibabel.nifti1.Nifti1Image'>
shapeMatching/DSD: <class 'nibabel.nifti1.Nifti1Image'>
shapeMatching/main_vars: <class 'nibabel.nifti1.Nifti1Image'>
shapeMatching/SDD: <class 'nibabel.nifti1.Nifti1Image'>
shapeMatching/SNN: <class 'nibabel.nifti1.Nifti1Image'>
shapeMatching/SSS: <class 'nibabel.nifti1.Nifti1Image'>
shapeMatching/task-baseline: <class 'nibabel.nifti1.Nifti1Image'>
shapeMatching/response_time: <class 'nibabel.nifti1.Nifti1Image'>
stopSignal/go: <class 'nibabel.nifti1.Nifti1Image'>
stopSignal/stop_failure-go: <class 'nibabel.nifti1.Nifti1Image'>
stopSignal/stop_failure: <class 'nibabel.nifti1.Nifti1Image'>
stopSignal/stop_failure-stop_success: <class 'nibabel.nifti1.Nifti1Image'>
stopSignal/stop_success-go: <class 'nibabel.nifti1.Nifti1Image'>
stopSignal/stop_success: <class 'nibabel.nifti1.Nifti1Image'>
stopSignal/stop_success-stop_failure: <class 'nibabel.nifti1.Nifti1Image'>
stopSignal/task-baseline: <class 'nibabel.nifti1.Nifti1Image'>
stopSignal/response_time: <class 'nibabel.nifti1.Nifti1Image'>
cuedTS/cue_switch_cost: <class 'nibabel.nifti1.Nifti1Image'>
cuedTS/task_switch_cost: <class 'nibabel.nifti1.Nifti1Image'>
```

```
cuedTS/task_switch_cue_switch-task_stay_cue_stay: <class 'nibabel.nifti1.Nif
ti1Image'>
cuedTS/task-baseline: <class 'nibabel.nifti1.Nifti1Image'>
cuedTS/response_time: <class 'nibabel.nifti1.Nifti1Image'>
spatialTS/cue_switch_cost: <class 'nibabel.nifti1.Nifti1Image'>
spatialTS/task_switch_cost: <class 'nibabel.nifti1.Nifti1Image'>
spatialTS/task_switch_cue_switch-task_stay_cue_stay: <class 'nibabel.nifti1.
Nifti1Image'>
spatialTS/task-baseline: <class 'nibabel.nifti1.Nifti1Image'>
spatialTS/response_time: <class 'nibabel.nifti1.Nifti1Image'>
```

Load the averaged parcel results

In [7]:
```
avg_parcel_traj_results = {}
mean_filename = f"{main_dir}/{main_files['mean']}_{date_updated}_averaged.pk

with open(mean_filename, 'rb') as f:
    avg_parcel_traj_results = pickle.load(f)
```

Visualizing functions

In [8]:
```
def create_parcel_practice_heatmap(parcel_traj, title, indiv_data = True, n_
    """
    Create a heatmap showing practice effects across all parcels

    input:
    parcel_traj: a df of parcels and activations
    title: the title for this heatmap
    """

    # Prepare data for heatmap
    df = pd.DataFrame(parcel_traj).T

    if (indiv_data):
        # Sort by slope
        df_sorted = df.sort_values('slope', key=abs).head(n_rows)
        # Create trajectory matrix
        trajectory_matrix = np.array([row['trajectory'] for _, row in df_sor
    else:
        # Sort by avg slope
        df_sorted = df.sort_values('slope_mean', key=abs).head(n_rows)
        # Create trajectory matrix
        trajectory_matrix = np.array([row['trajectory_mean'] for _, row in c
```

```
    # Create the heatmap
    plt.figure(figsize=(30, 12))

    # Plot trajectories
    sns.heatmap(trajectory_matrix,
                xticklabels=['Enc 1', 'Enc 2', 'Enc 3', 'Enc 4', 'Enc 5'],
                yticklabels=[row.name for _, row in df_sorted.iterrows()],
                cmap='RdBu_r', center=0,
                cbar_kws={'label': 'Activation'})

    plt.title('Practice Effects Across All Parcels')
    plt.xlabel('Encounter')
    plt.ylabel('Brain Parcel ID')

    plt.title(f"{title}: first {n_rows} rows")
    plt.tight_layout()
    plt.show()
```

In [9]:
```
# Calculate cut slices
x_cuts = find_cut_slices(SCHAEFER_IMG, direction='x', n_cuts=8)
y_cuts = find_cut_slices(SCHAEFER_IMG, direction='y', n_cuts=8)
z_cuts = find_cut_slices(SCHAEFER_IMG, direction='z', n_cuts=8)

# Specify which slices to display
FIXED_X_CUTS = x_cuts[1:7]  # Middle six X slices
FIXED_Y_CUTS = y_cuts[1:7]  # Middle six Y slices
FIXED_Z_CUTS = z_cuts[1:7]  # Middle six Z slices
```

In [10]:
```
def plot_slopes_on_brain(avg_results, task, contrast, n_rois=400, atlas_name
    """
    Plot parcel slope means on brain using atlas labels.
    """
    # Get atlas labels
    atlas_labels = [label.decode('utf-8') if isinstance(label, bytes) else l
                    for label in atlas.labels]

    # Get slope data
    parcel_data = avg_results[task][contrast]

    # Create brain image with slope values
    slope_data = np.zeros_like(atlas_data)

    # Map parcel names to atlas regions
    if hasattr(atlas, 'roi_values'):
        # Smorgasbord atlas - use actual ROI values
        for i, (roi_value, atlas_label) in enumerate(zip(atlas.roi_values, a
            if atlas_label in parcel_data:
                slope_value = parcel_data[atlas_label]['slope_mean']
                slope_data[atlas_data == roi_value] = slope_value
```

```
    else:
        # Schaefer atlas - use consecutive indexing
        for i, atlas_label in enumerate(atlas_labels):
            if atlas_label in parcel_data:
                slope_value = parcel_data[atlas_label]['slope_mean']
                slope_data[atlas_data == (i + 1)] = slope_value

    # Create a NIfTI image
    slope_img = nib.Nifti1Image(slope_data, atlas_img.affine)

    # Calculate vmin and vmax
    nonzero_slopes = slope_data[slope_data != 0]
    vmin, vmax = np.percentile(nonzero_slopes, [2, 98])

    # Ensure vmin and vmax are symmetric for diverging colormap
    abs_max = max(abs(vmin), abs(vmax))
    vmin, vmax = -abs_max, abs_max

    # # Create custom colormap
    # cmap = create_custom_colormap()

    # Set up the layout for plotting
    fig, axes = plt.subplots(3, 6, figsize=(20, 15))
    fig.suptitle(f'{title}: {task}/{contrast} ({atlas_name} atlas)', fontsiz

    display_modes = ['x', 'y', 'z']
    cuts_by_view = [FIXED_X_CUTS, FIXED_Y_CUTS, FIXED_Z_CUTS]

    for idx, ax_row in enumerate(axes):
        for j, ax in enumerate(ax_row):
            coord = cuts_by_view[idx][j]
            plotting.plot_stat_map(slope_img,
                                   colorbar=True,
                                   cmap='seismic',
                                   vmin=vmin,
                                   vmax=vmax,
                                   threshold=threshold,
                                   display_mode=display_modes[idx],
                                   axes=ax,
                                   cut_coords=[coord],
                                   draw_cross=False)

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
```

visualizing the averaged parcel slopes

In [13]:
```
# # heatmap of top slope-changed parcels
```

```
# for task in requested_task_contrasts:
#     if (count > 3):
#         break
#     contrast = requested_task_contrasts[task][0]
#     title = f"{task}:{contrast} heatmap of avg parcel activation (all subj

#     create_parcel_practice_heatmap(avg_parcel_traj_results[task][contrast]
```

In [14]:
```
# # Plot all contrasts
# # top 10% of slope change
# count = 0
# for task in avg_parcel_traj_results.keys():
#     for contrast in avg_parcel_traj_results[task].keys():
#         if (count >2):
#             break
#         if len(avg_parcel_traj_results[task][contrast]) > 0:
#             plot_slopes_on_brain(avg_parcel_traj_results, task, contrast)
#             count += 1
```

In [15]:
```
# do the same thing but with an unthresholded brain
# Plot all contrasts with no threshold
count = 0
for task in avg_parcel_traj_results.keys():
    for contrast in avg_parcel_traj_results[task].keys():
        if (count >= 0):
            break
        if len(avg_parcel_traj_results[task][contrast]) > 0:
            plot_slopes_on_brain(avg_parcel_traj_results, task, contrast, th
            count += 1
```

plotting fixed effects

In [11]:
```
def plot_fixed_effects_maps(subjects, tasks, contrasts, maps, descriptors,
                            x_cuts, y_cuts, z_cuts, threshold=None):
    """
    Plot fixed effects maps on brain slices.

    Parameters:
    -----------
    subjects : list
        List of subject IDs
    tasks : list
        List of task names
    contrasts : dict
        Dictionary mapping tasks to contrasts
    maps : list
        List of NIfTI images to plot
```

```
    descriptors : list
        List of descriptive labels for each map
    x_cuts, y_cuts, z_cuts : list
        Coordinates for slice cuts
    threshold : float or None
        Threshold for stat map (None = show all values)
    """
    count = 0
    display_modes = ['x', 'y', 'z']

    for subj in subjects:
        for task in tasks:
            for contrast in contrasts[task]:
                fig, axes = plt.subplots(3, 6, figsize=(20, 15))

                title = descriptors[count]
                img = maps[count]

                fig.suptitle(f'{title}: {task}/{contrast}', fontsize=20)

                # Define cuts for each view
                cuts_by_view = [x_cuts, y_cuts, z_cuts]

                for idx, ax_row in enumerate(axes):
                    for j, ax in enumerate(ax_row):
                        coord = cuts_by_view[idx][j]

                        plotting.plot_stat_map(
                            img,
                            colorbar=True,
                            cmap='seismic',
                            symmetric_cbar=True,
                            threshold=threshold,
                            display_mode=display_modes[idx],
                            axes=ax,
                            cut_coords=[coord],
                            draw_cross=False
                        )

                plt.tight_layout(rect=[0, 0.03, 1, 0.95])
                plt.show()

                count += 1
```

plotting averaged fixed effects next to the averaged slope maps

In [12]:

```
def plot_slopes_and_fixed_effects(avg_results, fixed_effects_img, task, cont
    """
    Plot parcel slope means and fixed effects maps side by side.
    """
    # Get atlas labels
    atlas_labels = [label.decode('utf-8') if isinstance(label, bytes) else l
                    for label in atlas.labels]

    # Get slope data
    parcel_data = avg_results[task][contrast]

    # Create brain image with slope values
    slope_data = np.zeros_like(atlas_data)

    # Map parcel names to atlas regions
    if hasattr(atlas, 'roi_values'):
        for i, (roi_value, atlas_label) in enumerate(zip(atlas.roi_values, a
            if atlas_label in parcel_data:
                slope_value = parcel_data[atlas_label]['slope_mean']
                slope_data[atlas_data == roi_value] = slope_value
    else:
        for i, atlas_label in enumerate(atlas_labels):
            if atlas_label in parcel_data:
                slope_value = parcel_data[atlas_label]['slope_mean']
                slope_data[atlas_data == (i + 1)] = slope_value

    # Create a NIfTI image for slopes
    slope_img = nib.Nifti1Image(slope_data, atlas_img.affine)

    # Calculate vmin and vmax for slopes
    nonzero_slopes = slope_data[slope_data != 0]
    vmin_slope, vmax_slope = np.percentile(nonzero_slopes, [2, 98])
    abs_max_slope = max(abs(vmin_slope), abs(vmax_slope))
    vmin_slope, vmax_slope = -abs_max_slope, abs_max_slope

    # Set up the layout for plotting
    fig, axes = plt.subplots(6, 6, figsize=(30, 30))
    fig.suptitle(f'{title}: {task}/{contrast} ({atlas_name} atlas)', fontsiz

    display_modes = ['x', 'y', 'z']
    cuts_by_view = [FIXED_X_CUTS, FIXED_Y_CUTS, FIXED_Z_CUTS]

    # Plot slopes (top 3 rows)
    for idx, ax_row in enumerate(axes[:3]):
        for j, ax in enumerate(ax_row):
            coord = cuts_by_view[idx][j]
            plotting.plot_stat_map(slope_img,
                                   colorbar=True,
                                   cmap='RdBu_r',
                                   vmin=vmin_slope,
```

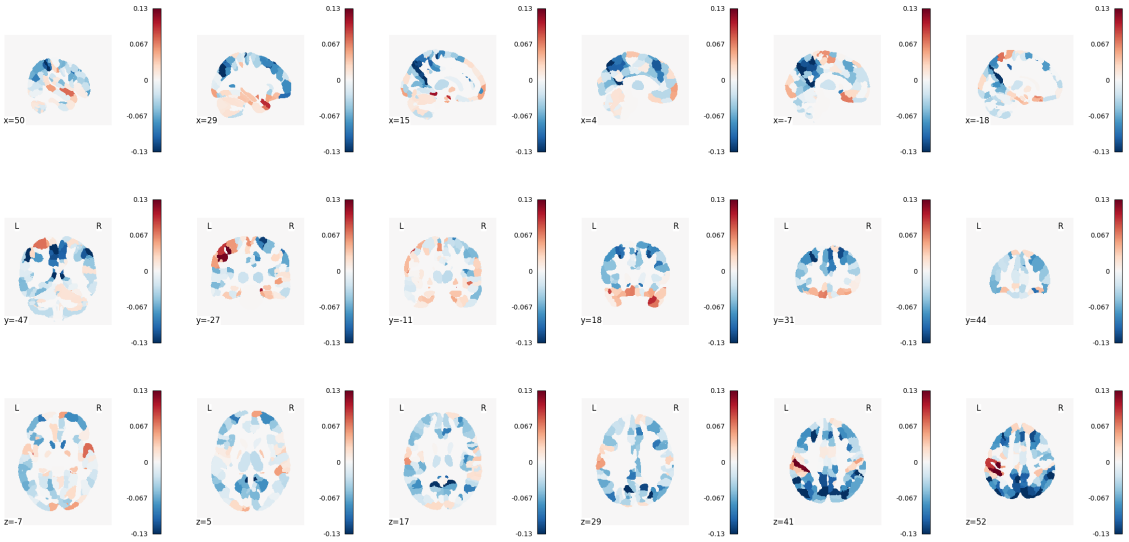
```
                                    vmax=vmax_slope,
                                    threshold=threshold,
                                    display_mode=display_modes[idx],
                                    axes=ax,
                                    cut_coords=[coord],
                                    draw_cross=False)
            if j == 0:
                ax.set_ylabel(display_modes[idx].upper(), rotation=0, labelp

    # Plot fixed effects (bottom 3 rows)
    vmin_fe, vmax_fe = np.percentile(fixed_effects_img.get_fdata(), [2, 98])
    abs_max_fe = max(abs(vmin_fe), abs(vmax_fe))

    for idx, ax_row in enumerate(axes[3:]):
        for j, ax in enumerate(ax_row):
            coord = cuts_by_view[idx][j]
            plotting.plot_stat_map(fixed_effects_img,
                                    colorbar=True,
                                    cmap='RdBu_r',
                                    vmin=-abs_max_fe,
                                    vmax=abs_max_fe,
                                    symmetric_cbar=True,
                                    threshold=threshold,
                                    display_mode=display_modes[idx],
                                    axes=ax,
                                    cut_coords=[coord],
                                    draw_cross=False)
            if j == 0:
                ax.set_ylabel(display_modes[idx].upper(), rotation=0, labelp

    # Add labels for the two sections
    fig.text(0.5, 1.0, 'Parcel Slopes', ha='center', va='center', fontsize=2
    fig.text(0.5, 0.50, 'Fixed Effects', ha='center', va='center', fontsize=

    plt.tight_layout(rect=[0, 0.03, 0.9, 0.95])
    plt.show()
```

In [ ]:
```
for task in avg_parcel_traj_results.keys():
    for contrast in avg_parcel_traj_results[task].keys():
        if len(avg_parcel_traj_results[task][contrast]) > 0:
            plot_slopes_and_fixed_effects(avg_parcel_traj_results, averaged_
```

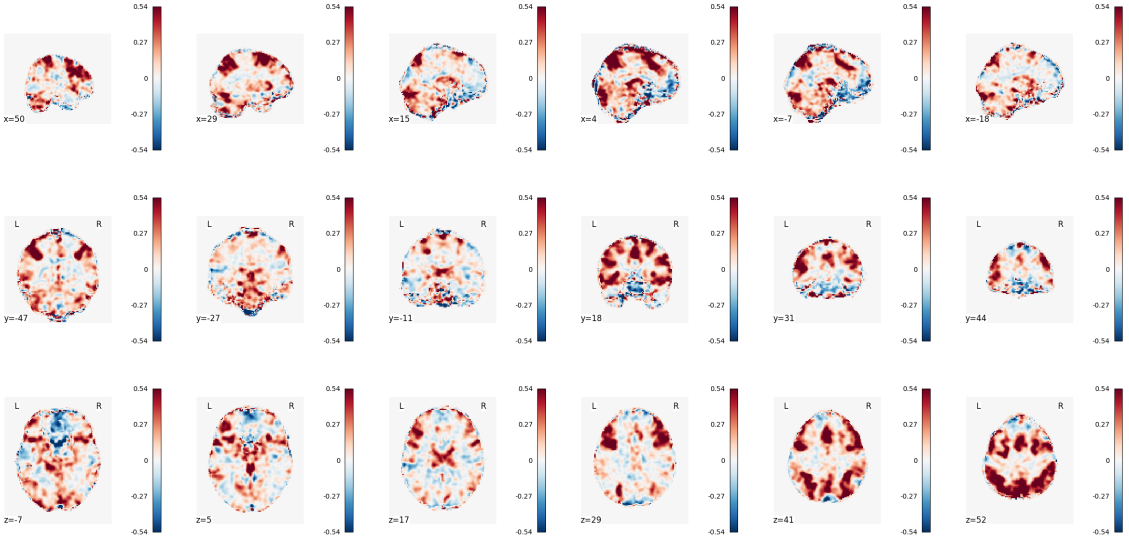
```
/tmp/ipykernel_3075/1305435263.py:85: UserWarning: This figure includes Axes
that are not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout(rect=[0, 0.03, 0.9, 0.95])
```

Parcel Slopes

Average parcel slopes: nBack/twoBack-oneBack (smor atlas)

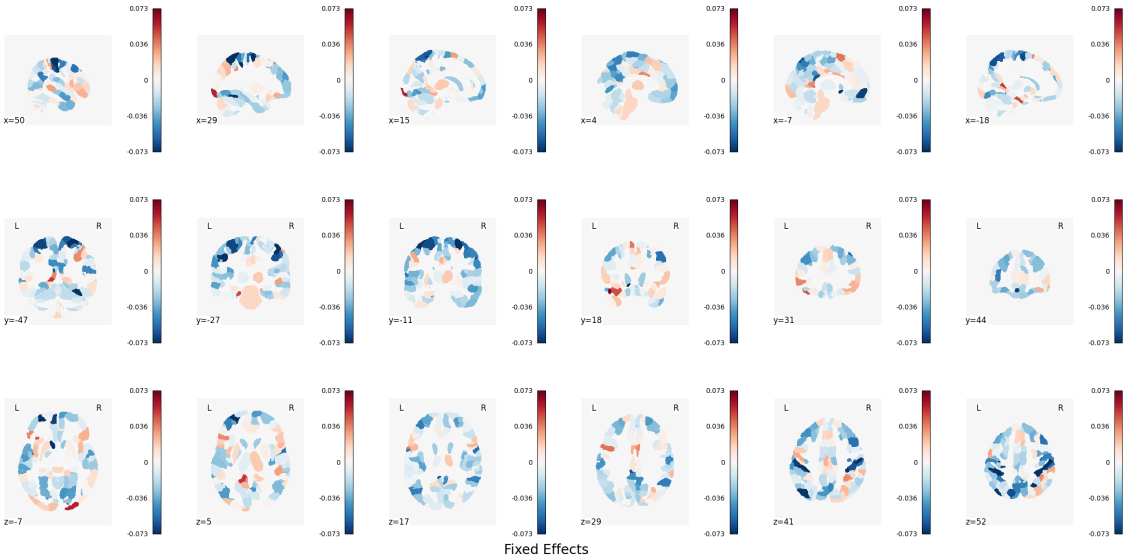


Fixed Effects

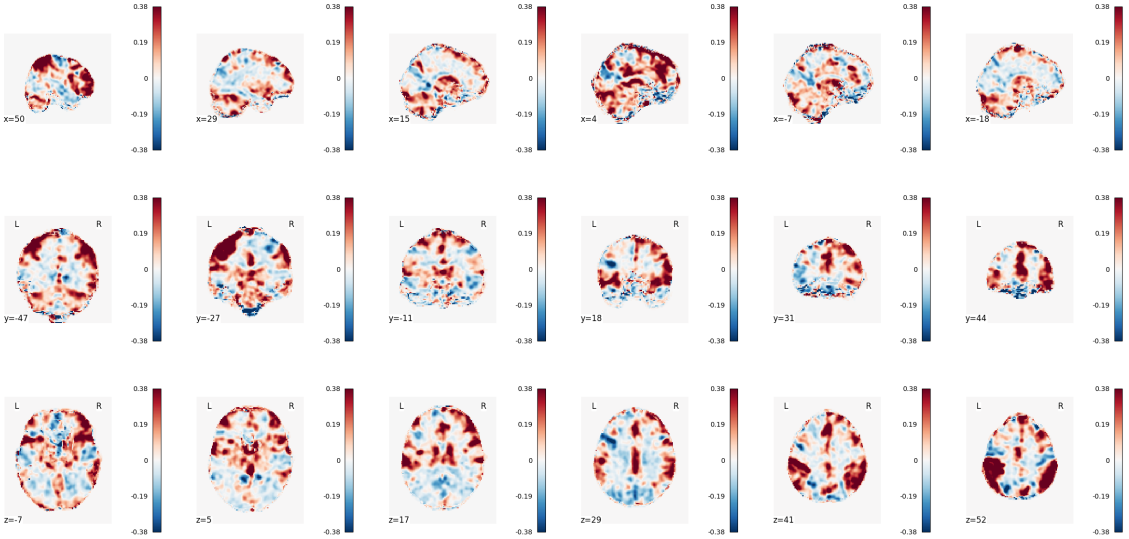


Parcel Slopes

Average parcel slopes: nBack/match-mismatch (smor atlas)

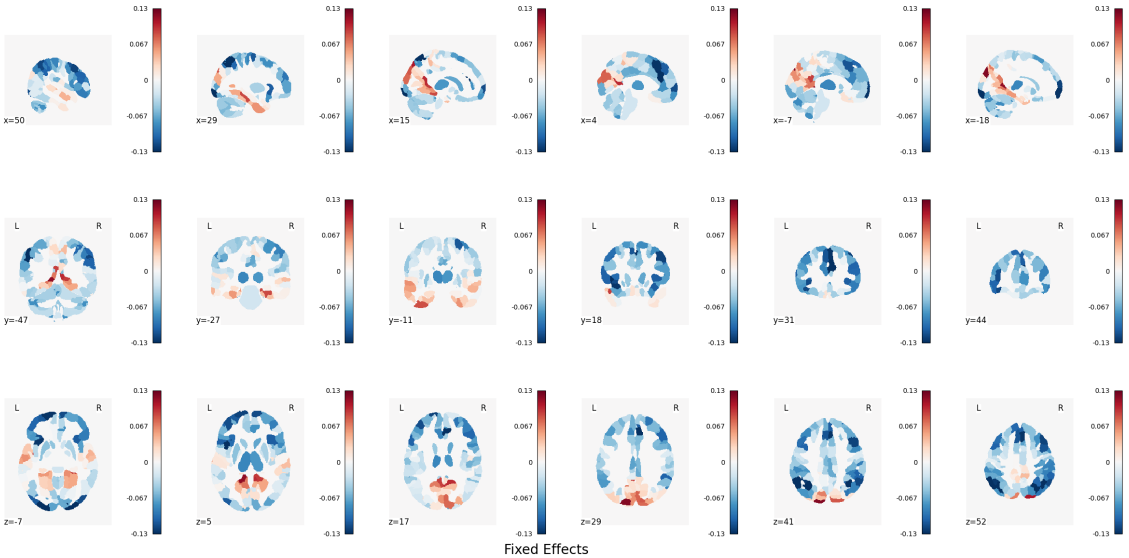


Fixed Effects

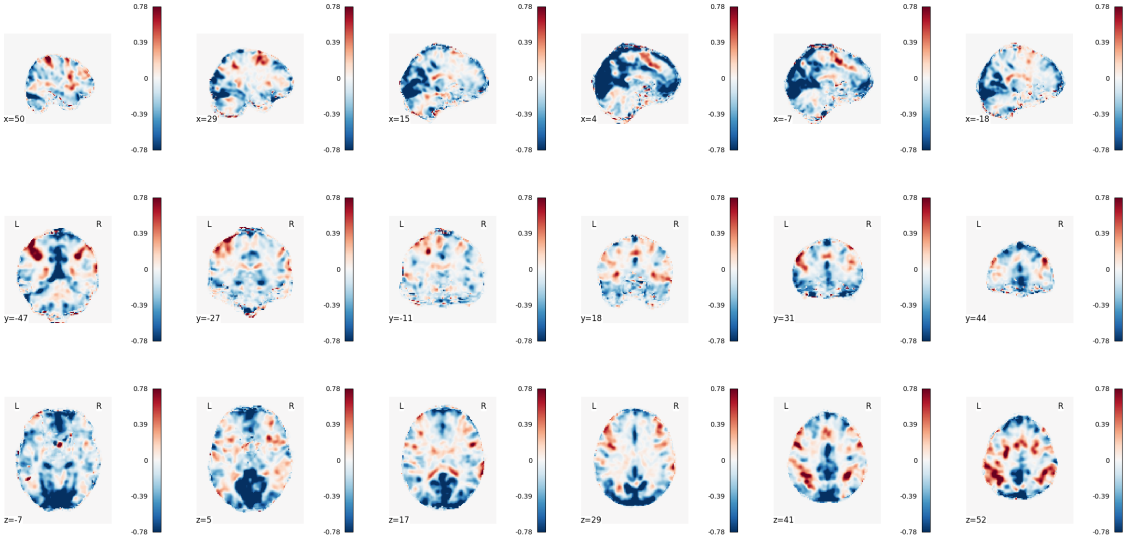


Parcel Slopes

Average parcel slopes: nBack/task-baseline (smor atlas)

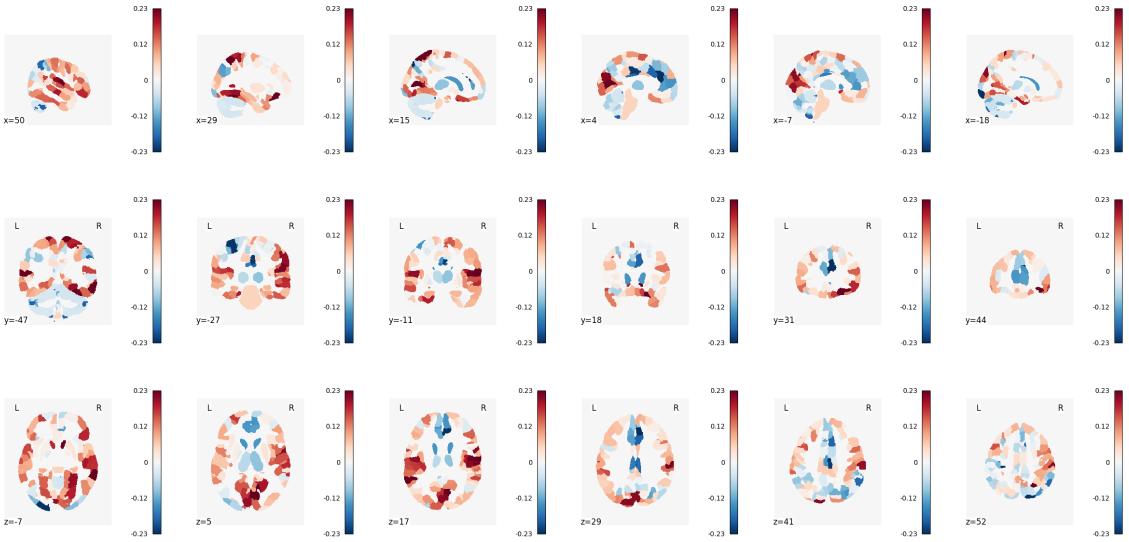


Fixed Effects

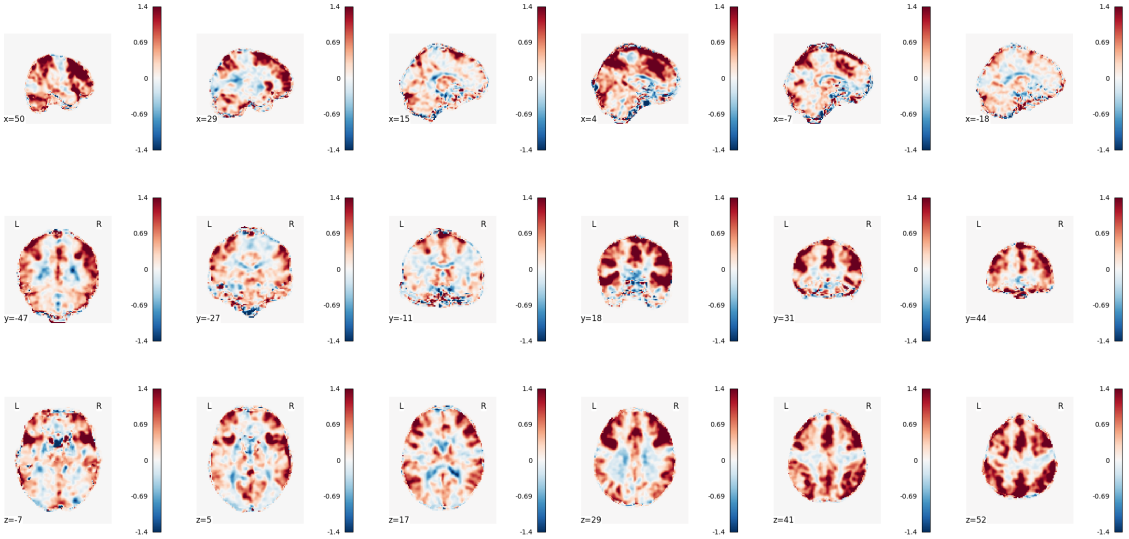


Parcel Slopes

Average parcel slopes: nBack/response_time (smor atlas)

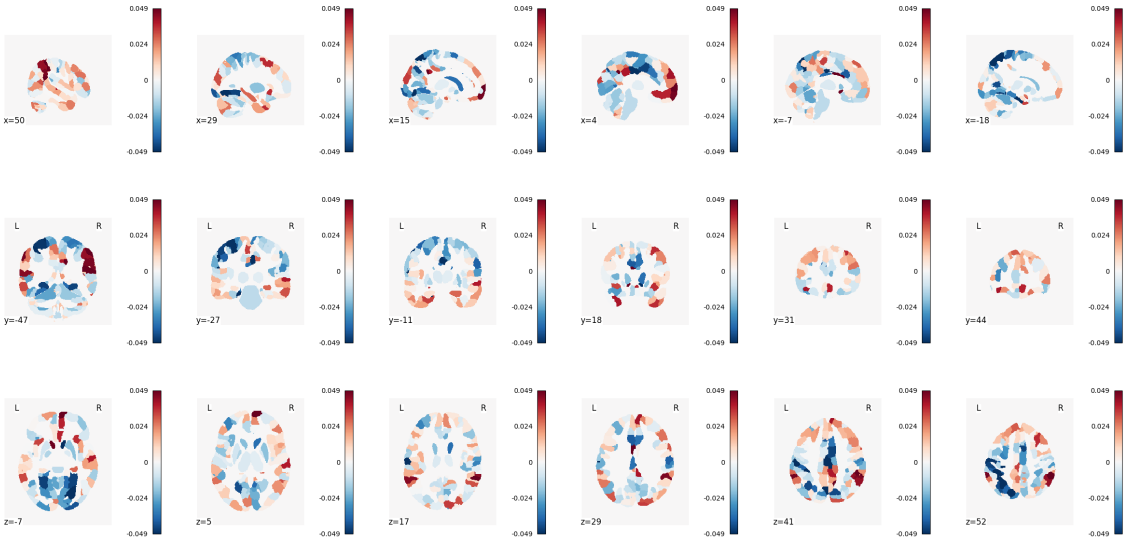


Fixed Effects

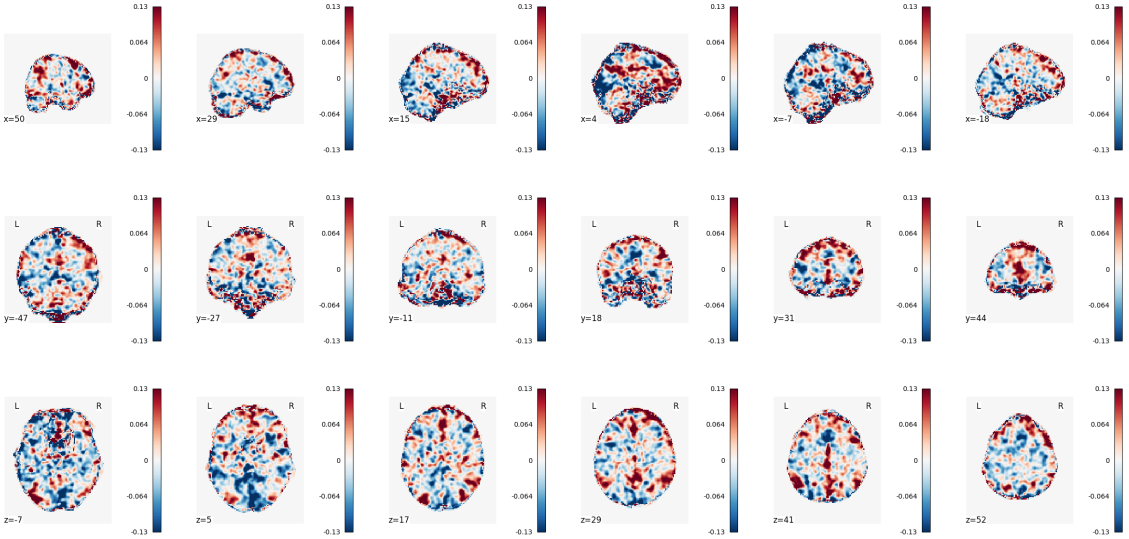


Parcel Slopes

Average parcel slopes: flanker/incongruent-congruent (smor atlas)

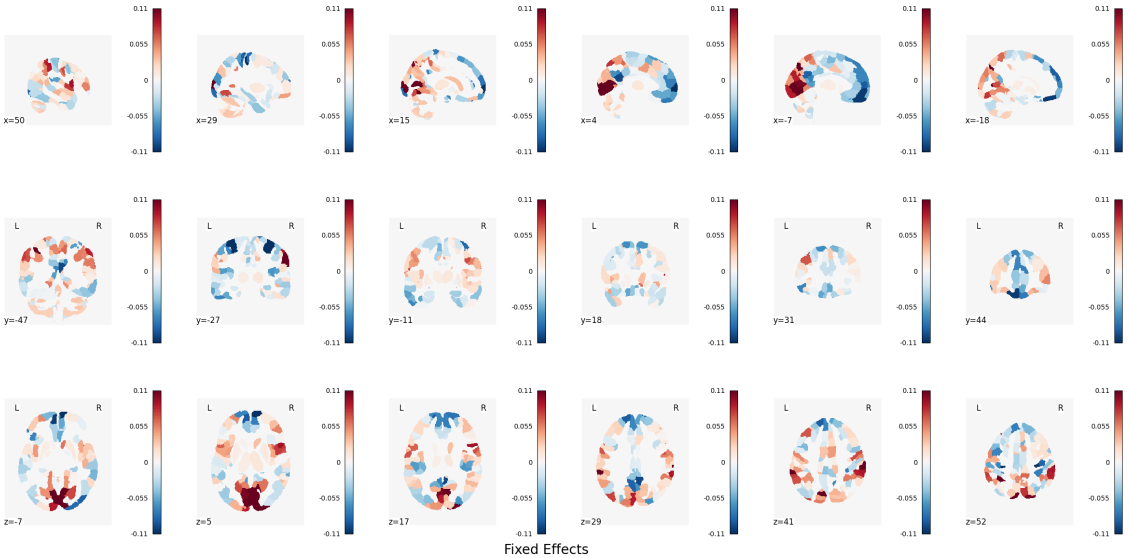


Fixed Effects

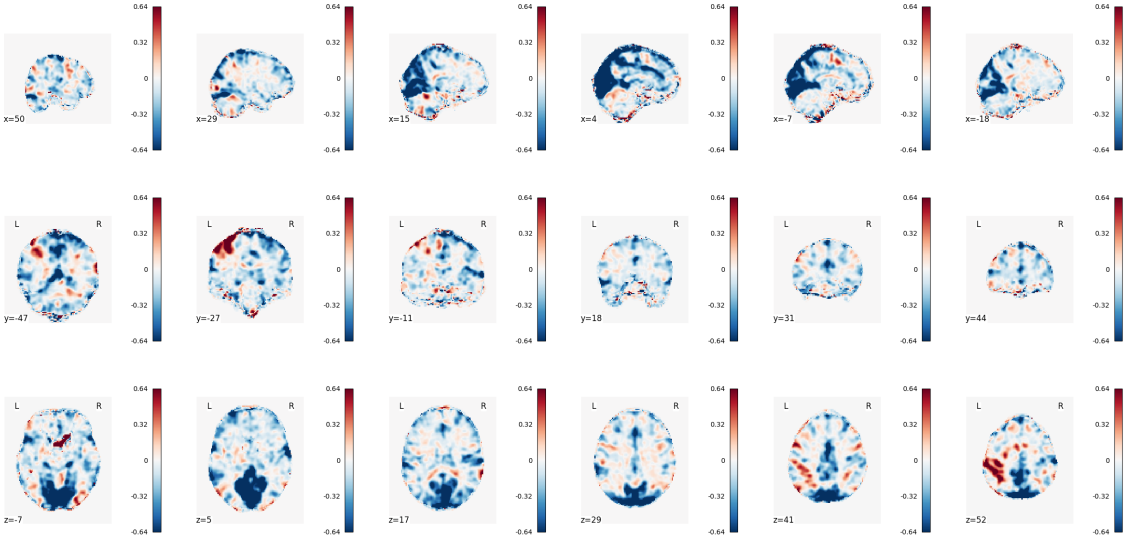


Parcel Slopes

Average parcel slopes: flanker/task-baseline (smor atlas)

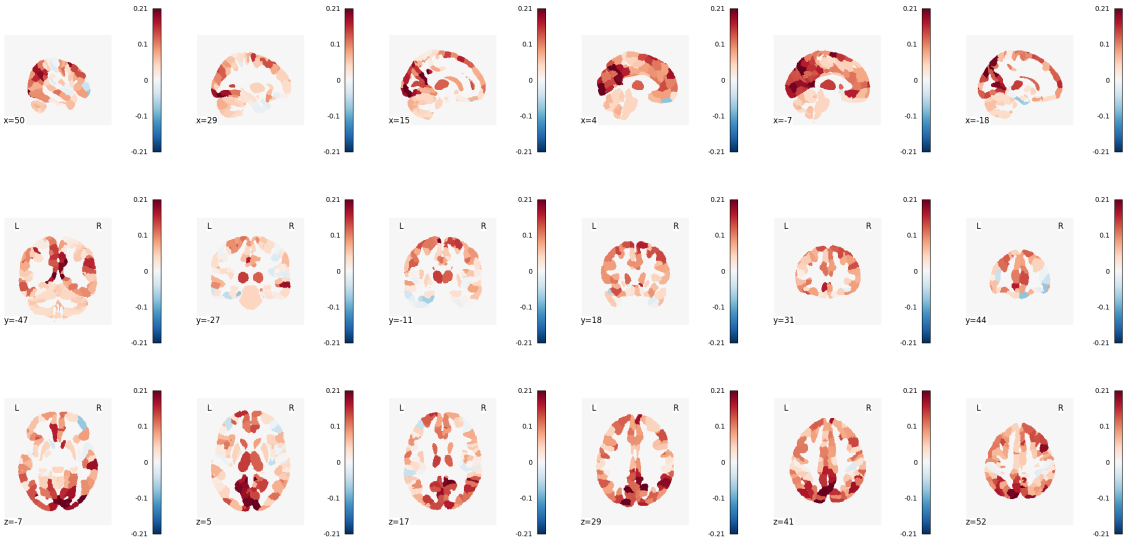


Fixed Effects

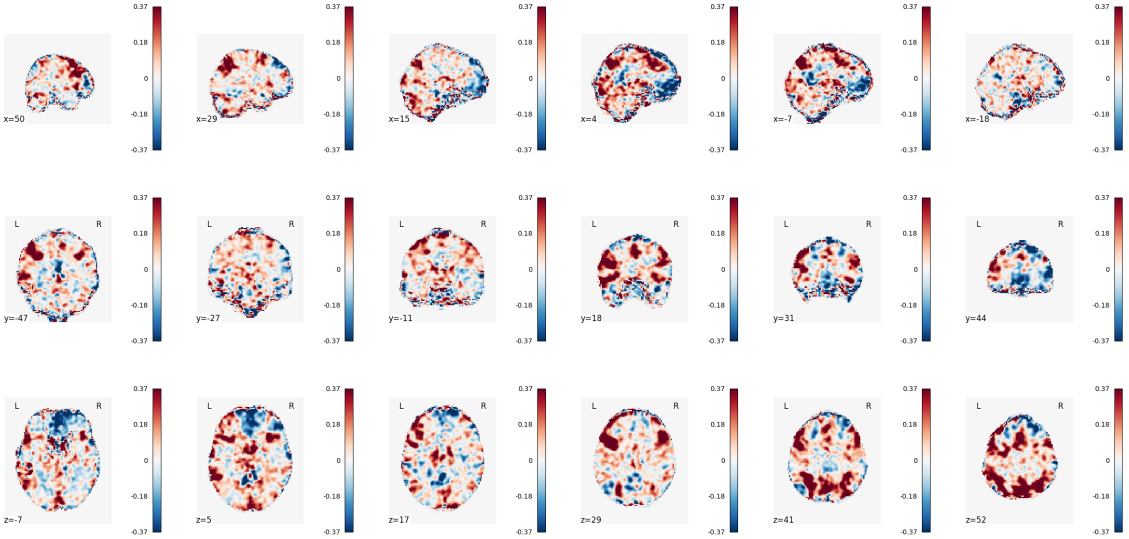


Parcel Slopes

Average parcel slopes: directedForgetting/neg-con (smor atlas)

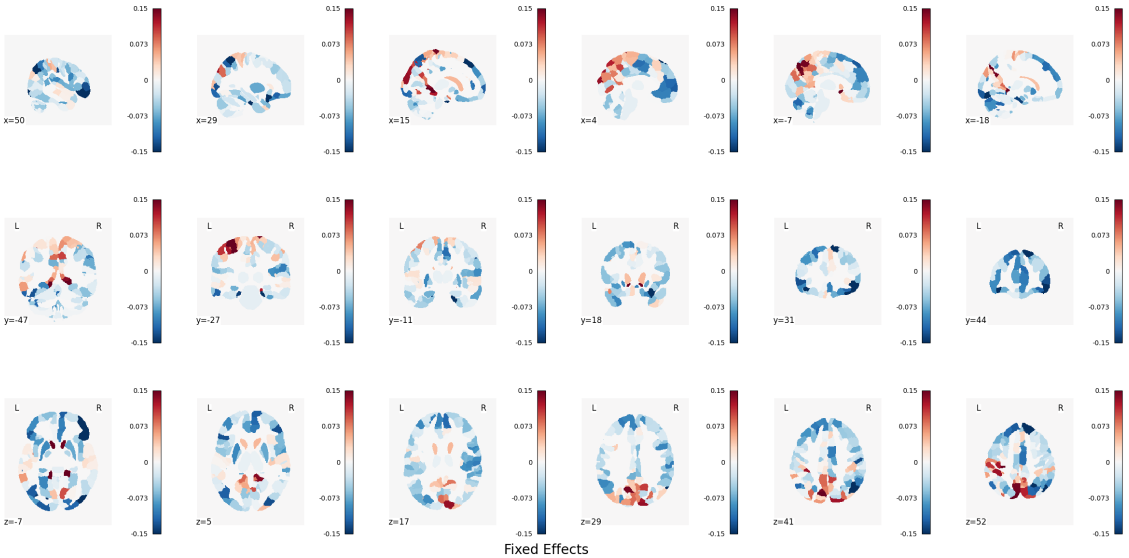


Fixed Effects

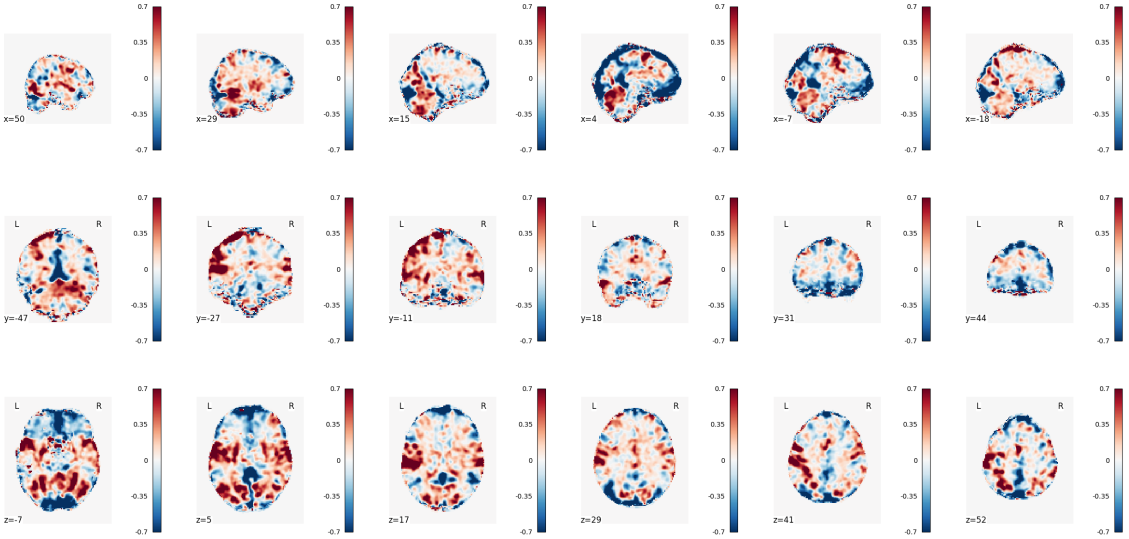


Parcel Slopes

Average parcel slopes: directedForgetting/task-baseline (smor atlas)

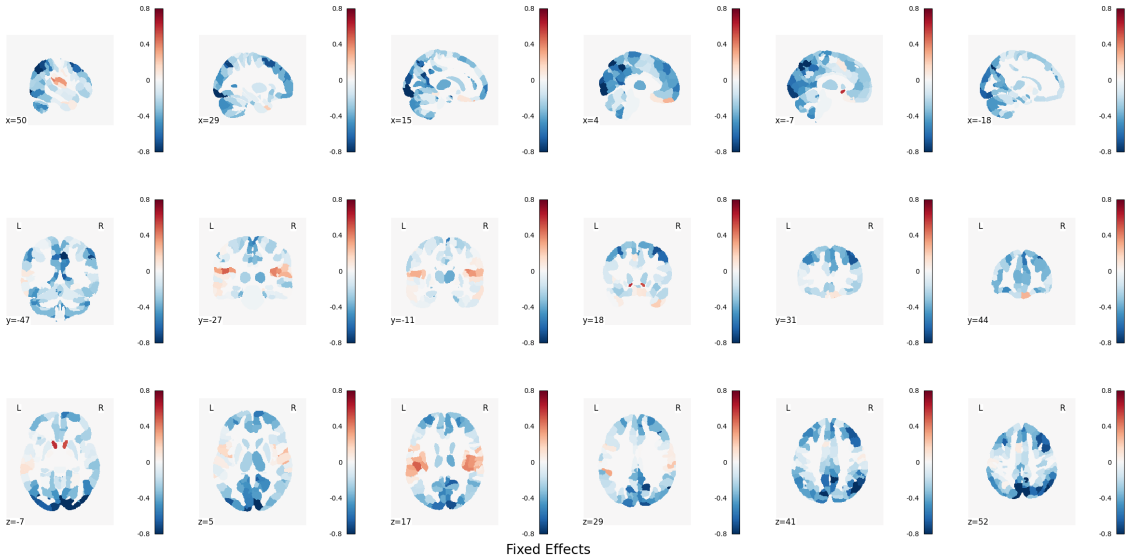


Fixed Effects

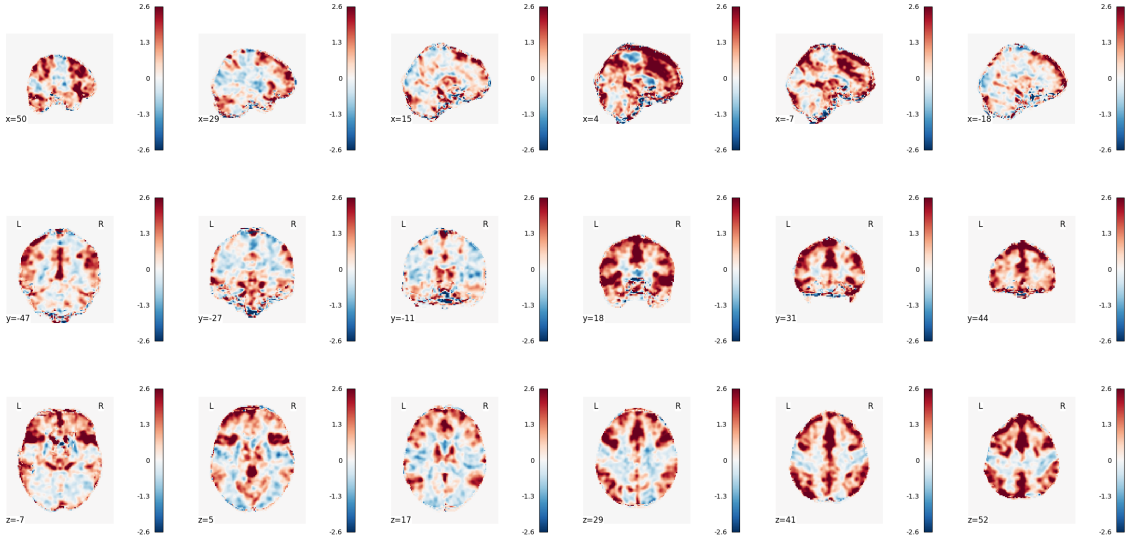


Parcel Slopes

Average parcel slopes: directedForgetting/response_time (smor atlas)

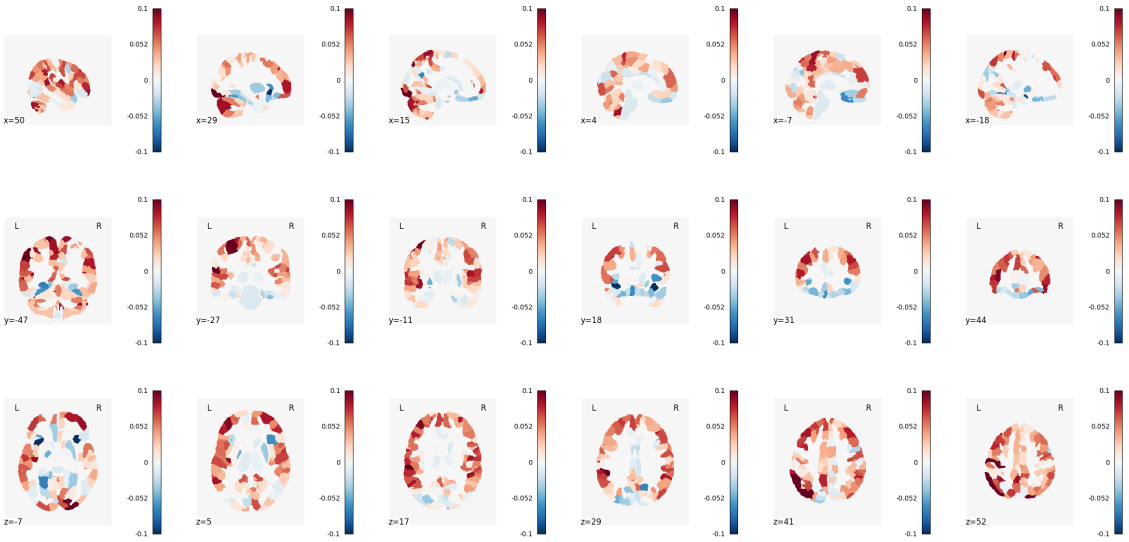


Fixed Effects

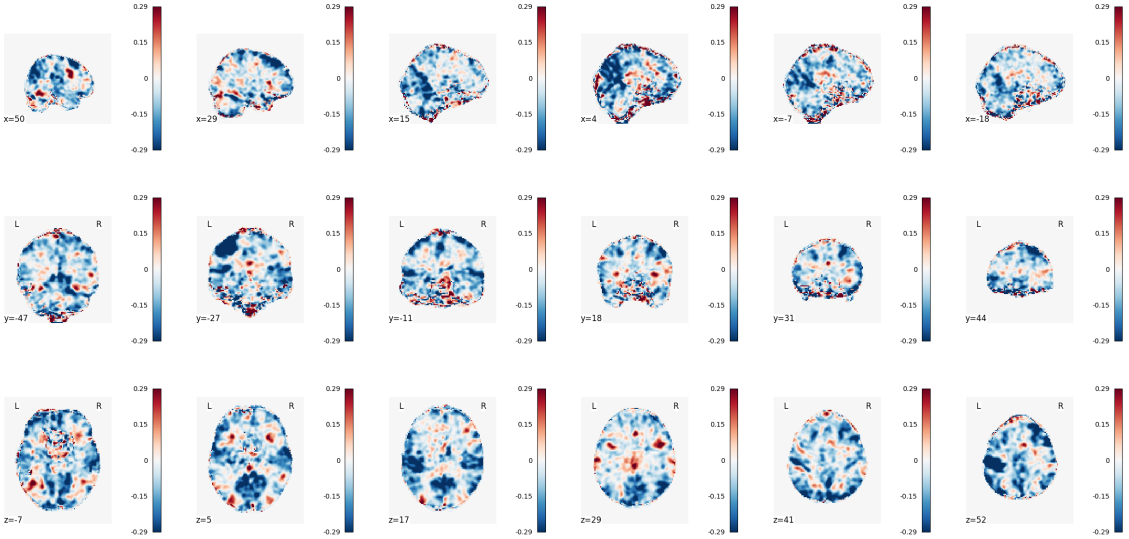


Parcel Slopes

Average parcel slopes: goNogo/nogo_success-go (smor atlas)

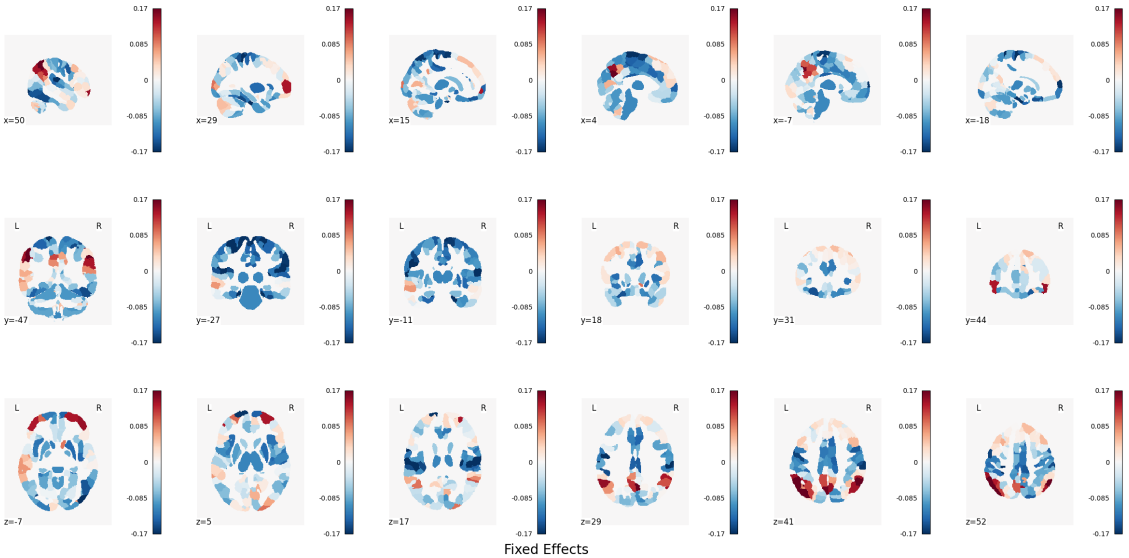


Fixed Effects

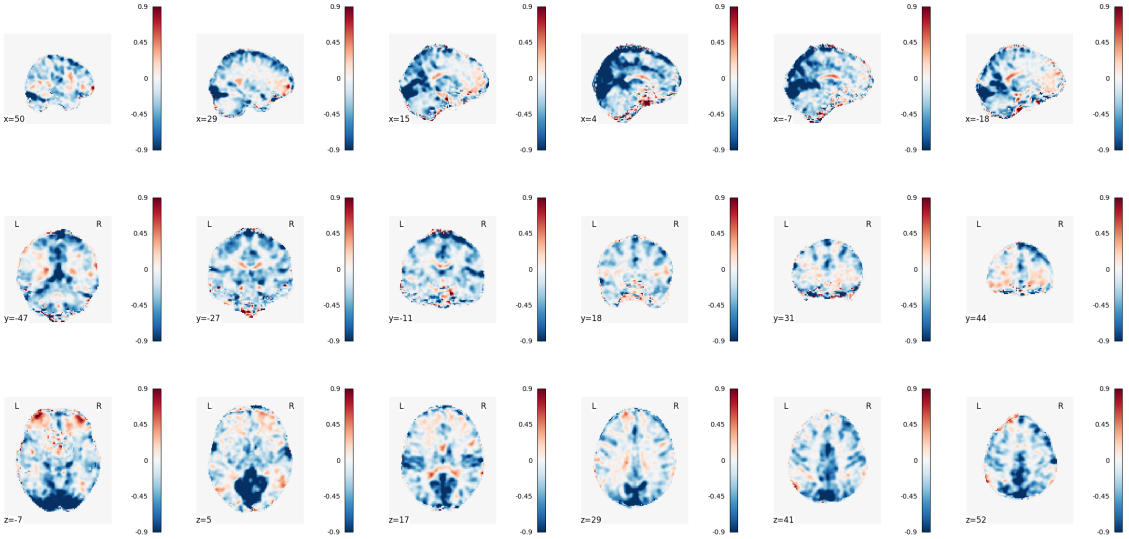


Parcel Slopes

Average parcel slopes: goNogo/nogo_success (smor atlas)

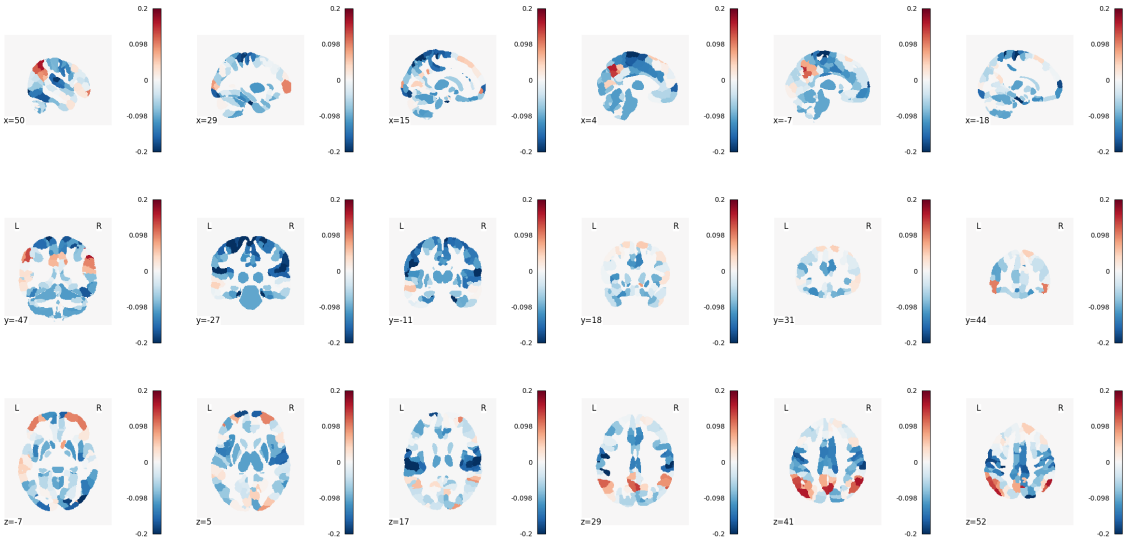


Fixed Effects

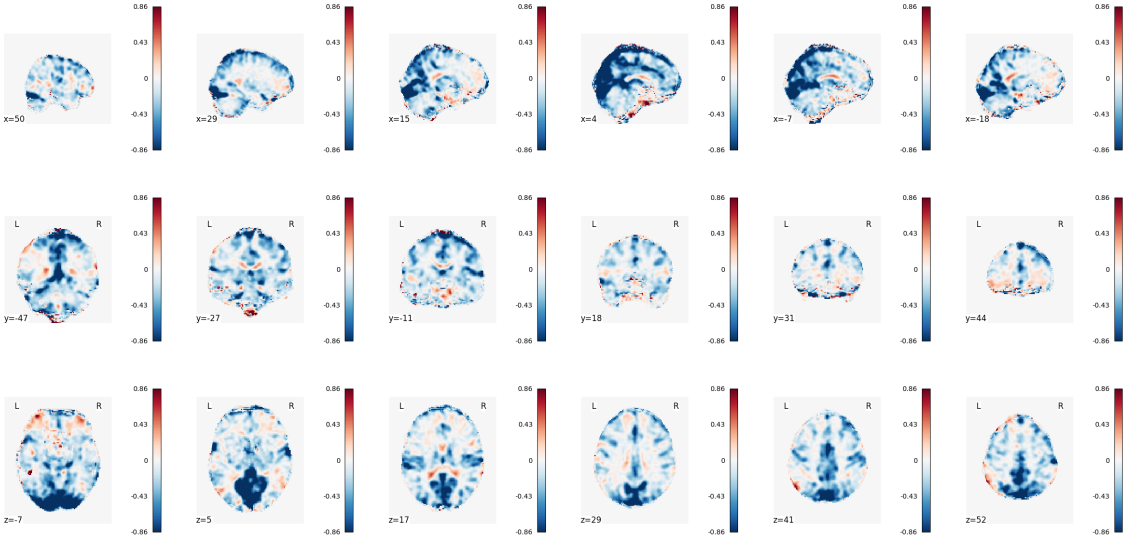


Parcel Slopes

Average parcel slopes: goNogo/task-baseline (smor atlas)

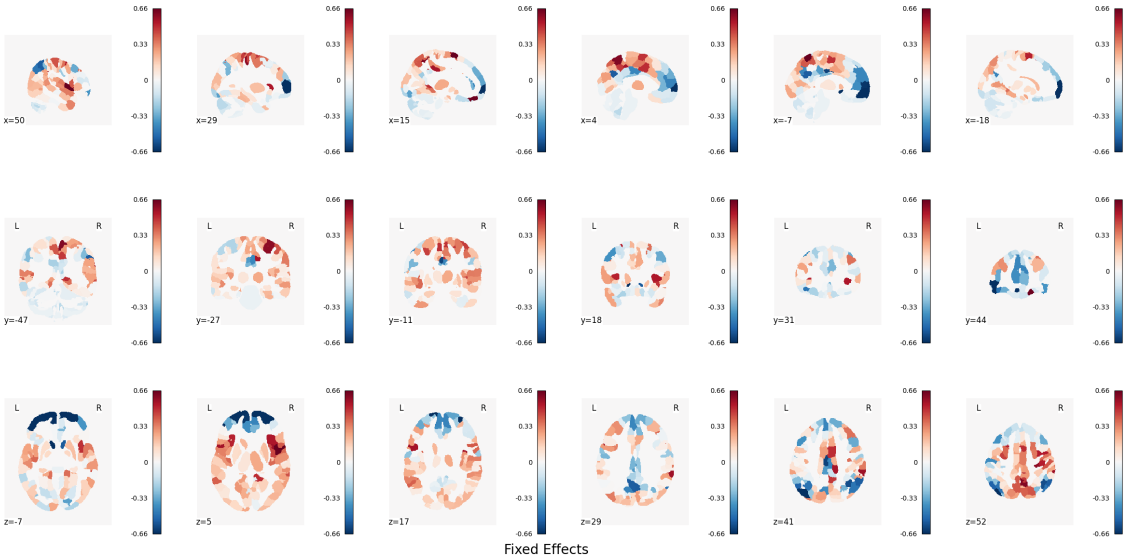


Fixed Effects

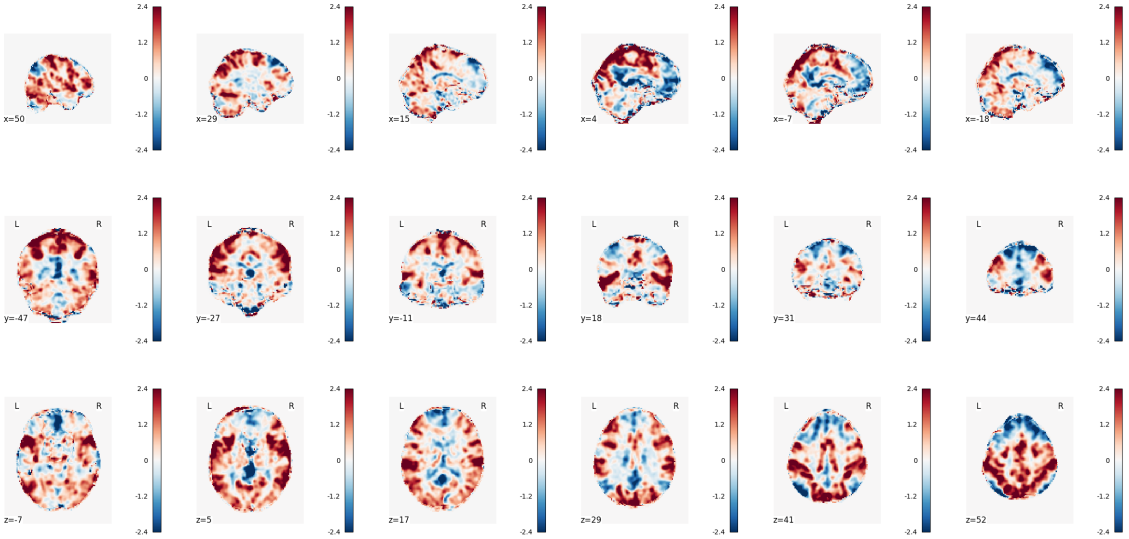


Parcel Slopes

Average parcel slopes: goNogo/response_time (smor atlas)

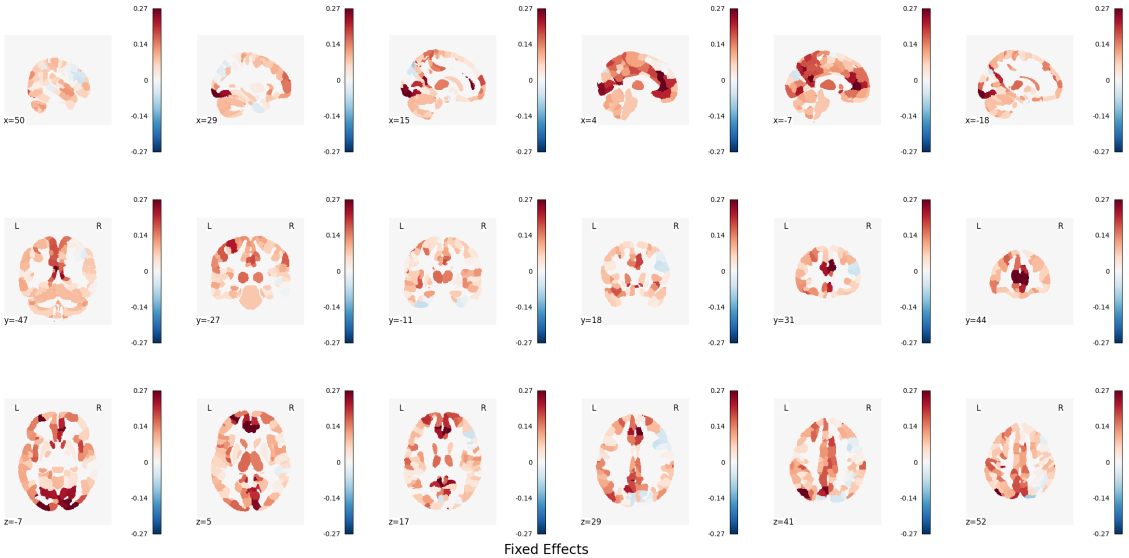


Fixed Effects

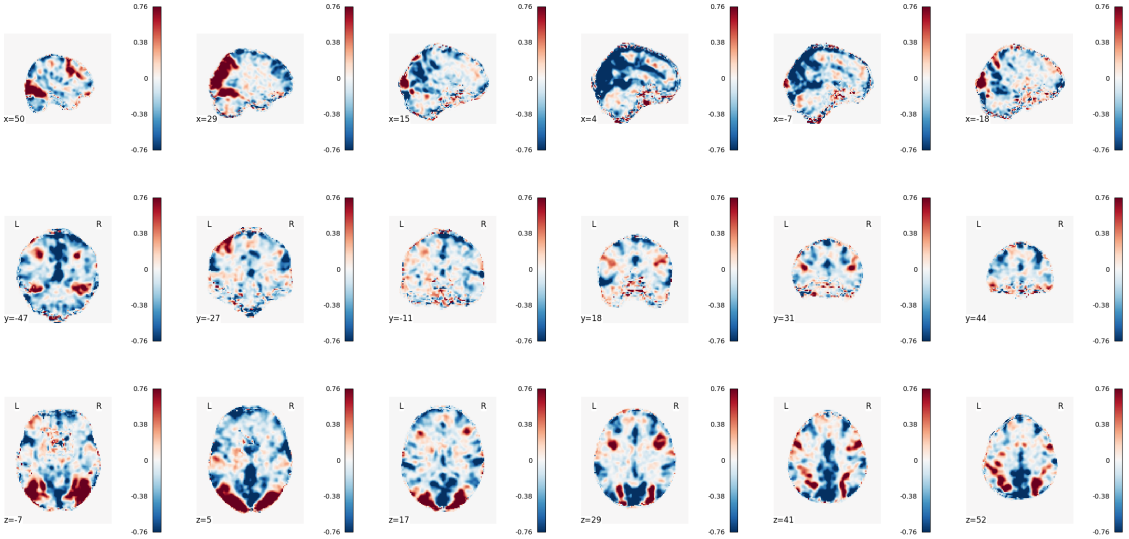


Parcel Slopes

Average parcel slopes: shapeMatching/DDD (smor atlas)

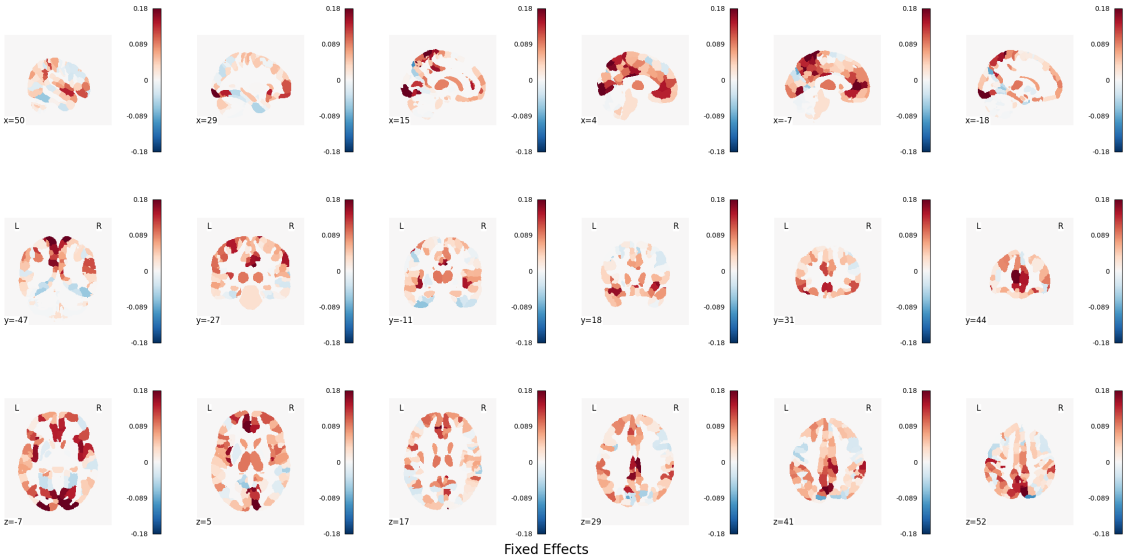


Fixed Effects

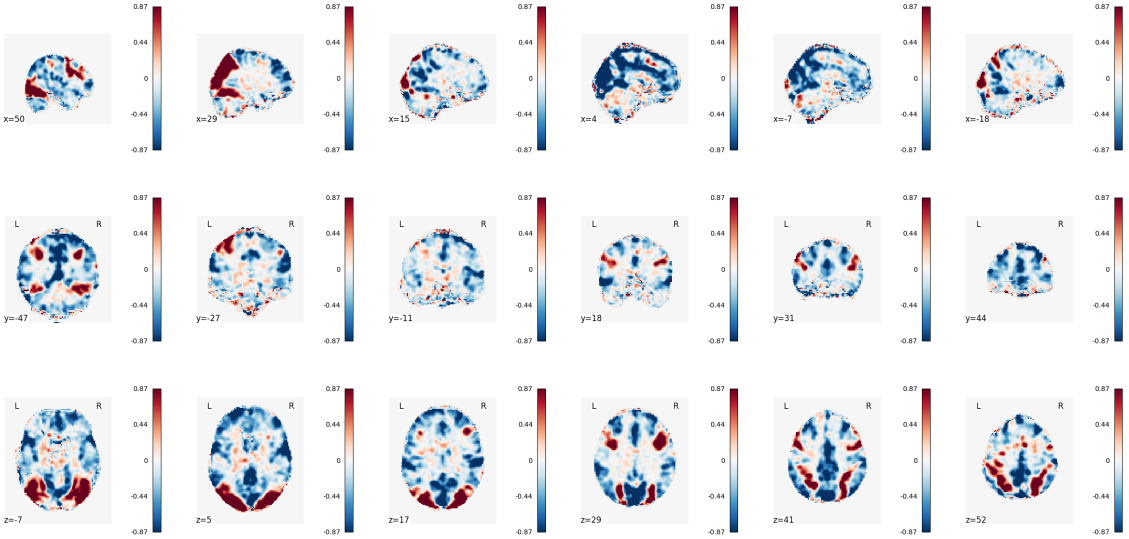


Parcel Slopes

Average parcel slopes: shapeMatching/DDS (smor atlas)

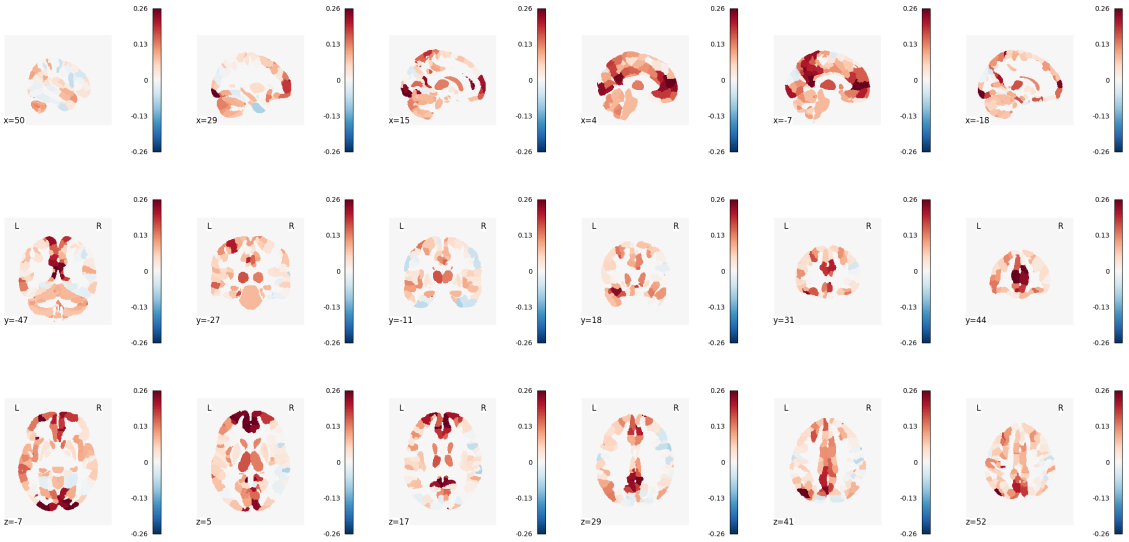


Fixed Effects

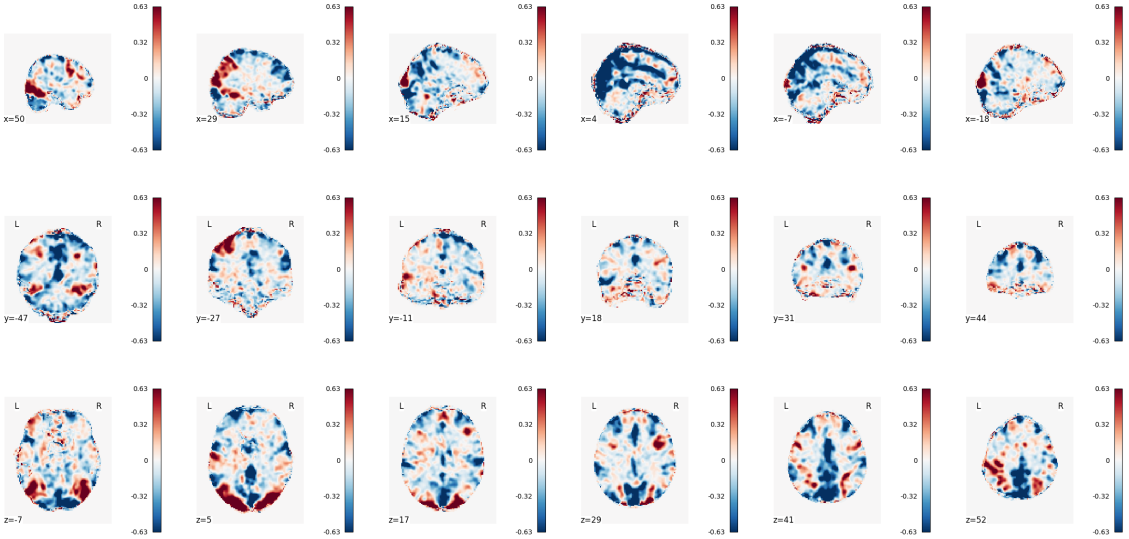


Parcel Slopes

Average parcel slopes: shapeMatching/DNN (smor atlas)

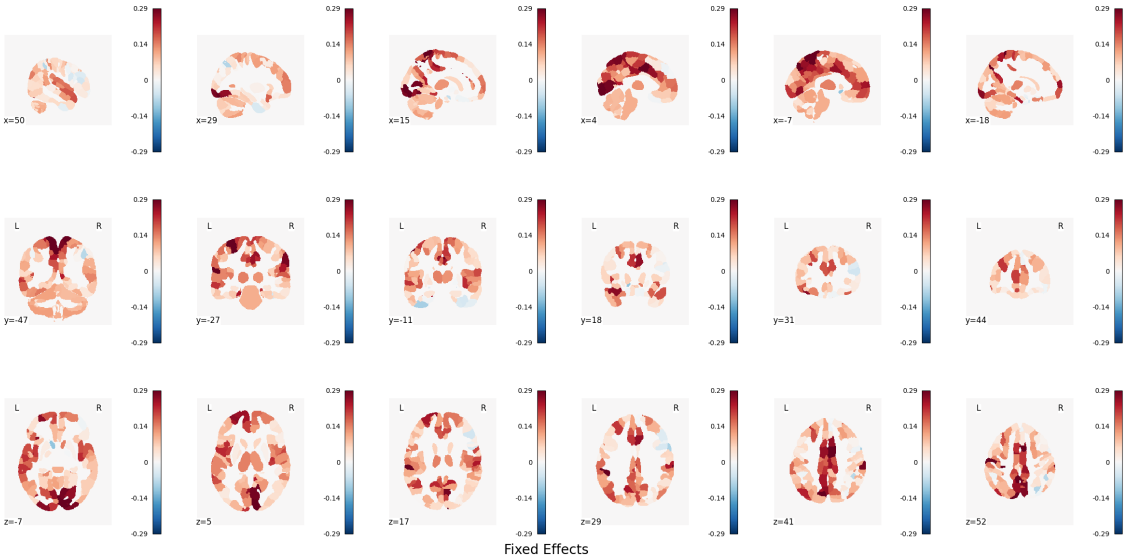


Fixed Effects

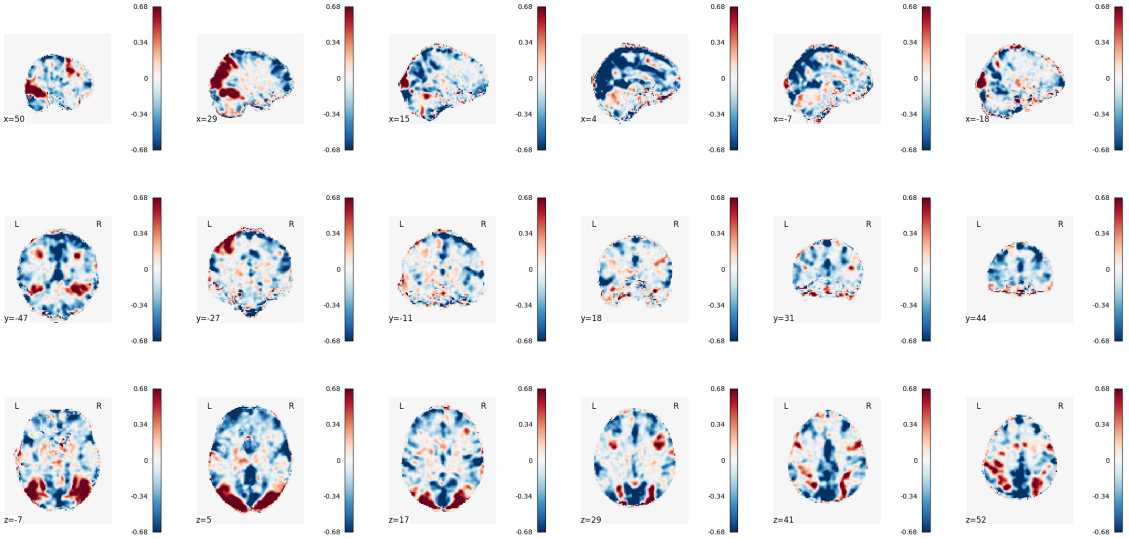


Parcel Slopes

Average parcel slopes: shapeMatching/DSD (smor atlas)

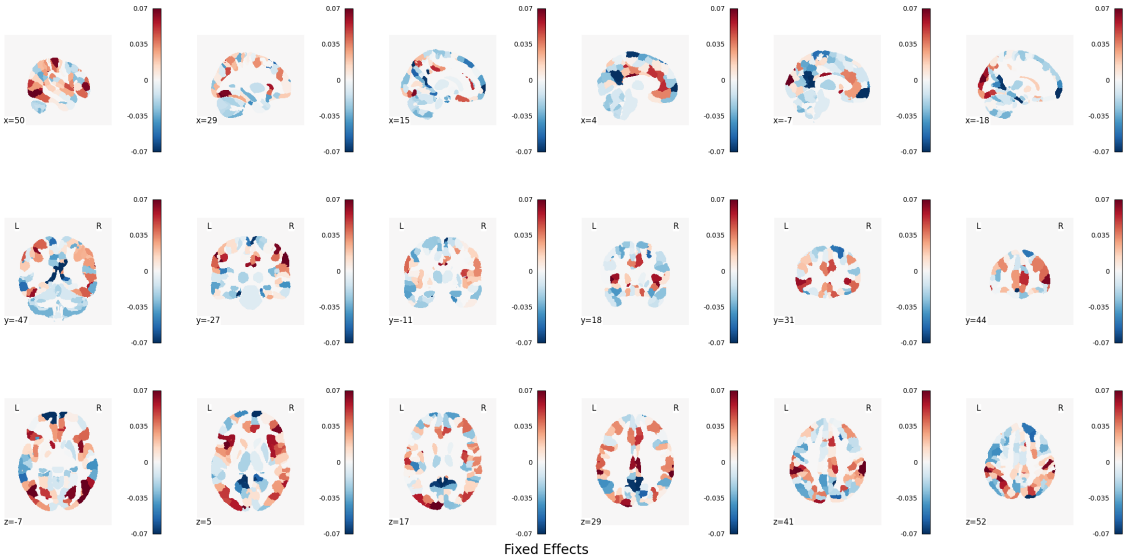


Fixed Effects

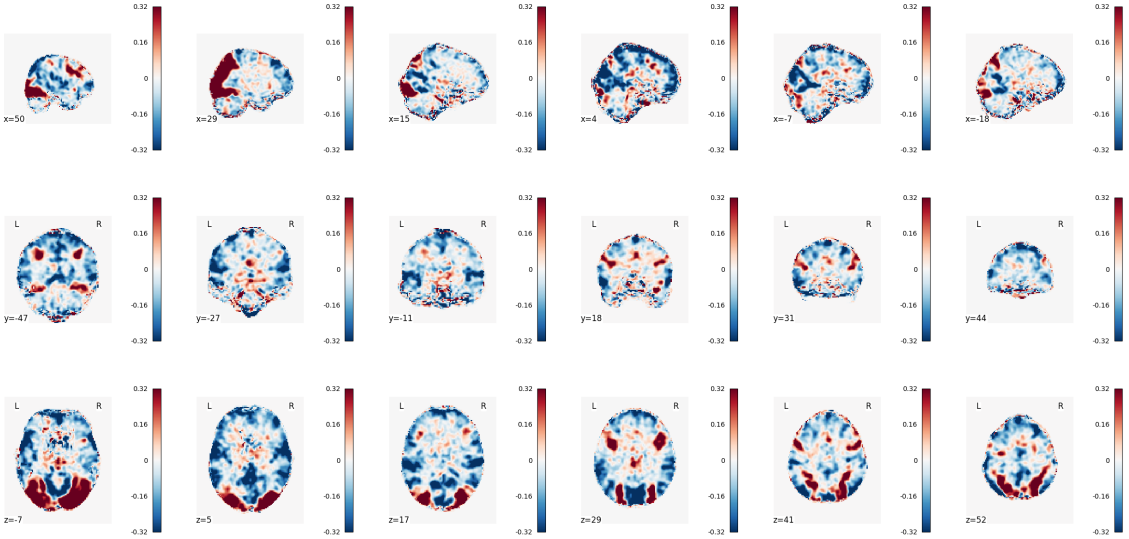


Parcel Slopes

Average parcel slopes: shapeMatching/main_vars (smor atlas)

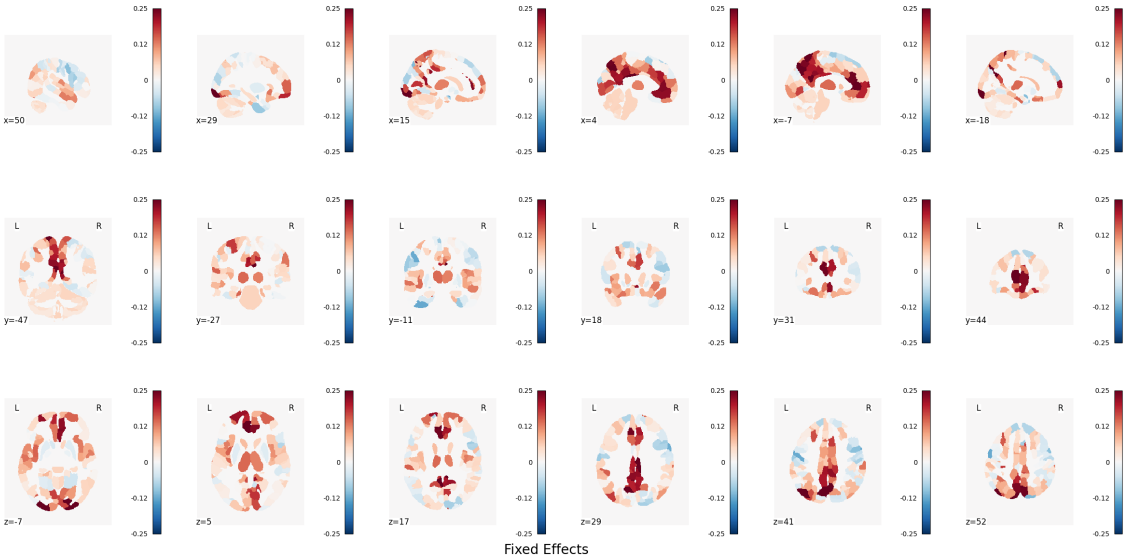


Fixed Effects

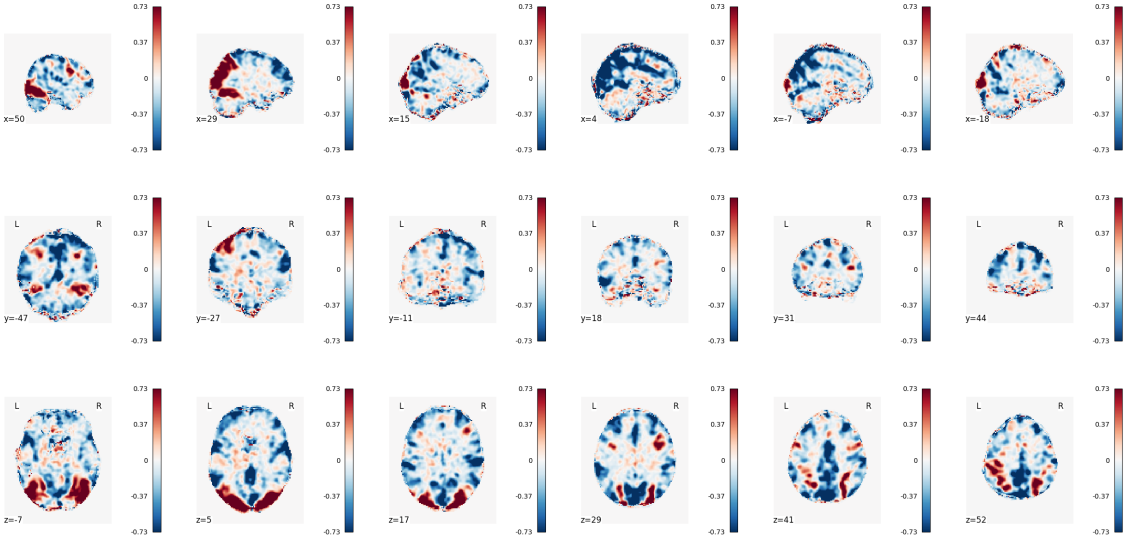


Parcel Slopes

Average parcel slopes: shapeMatching/SDD (smor atlas)

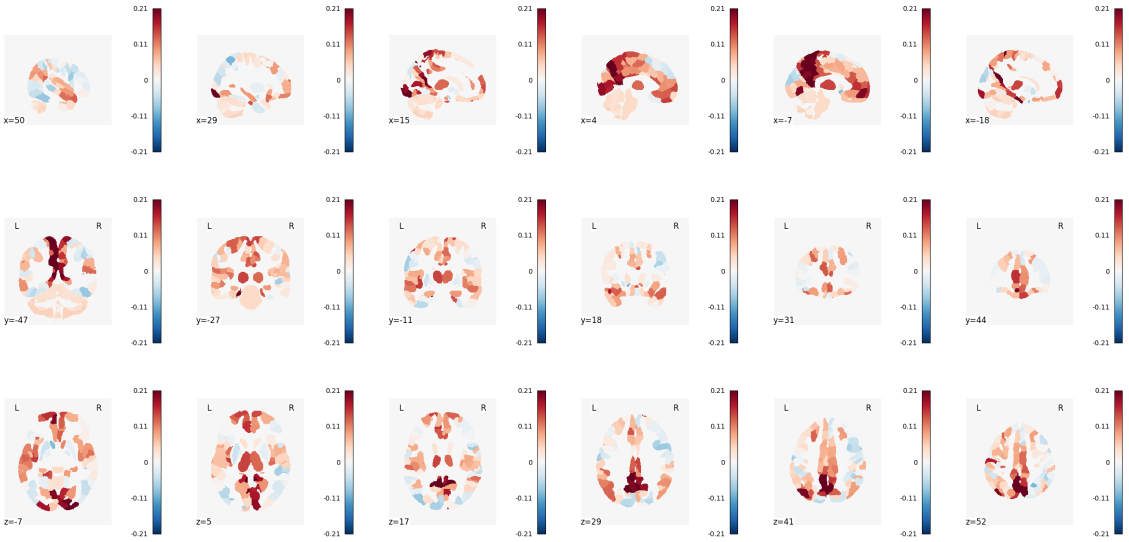


Fixed Effects

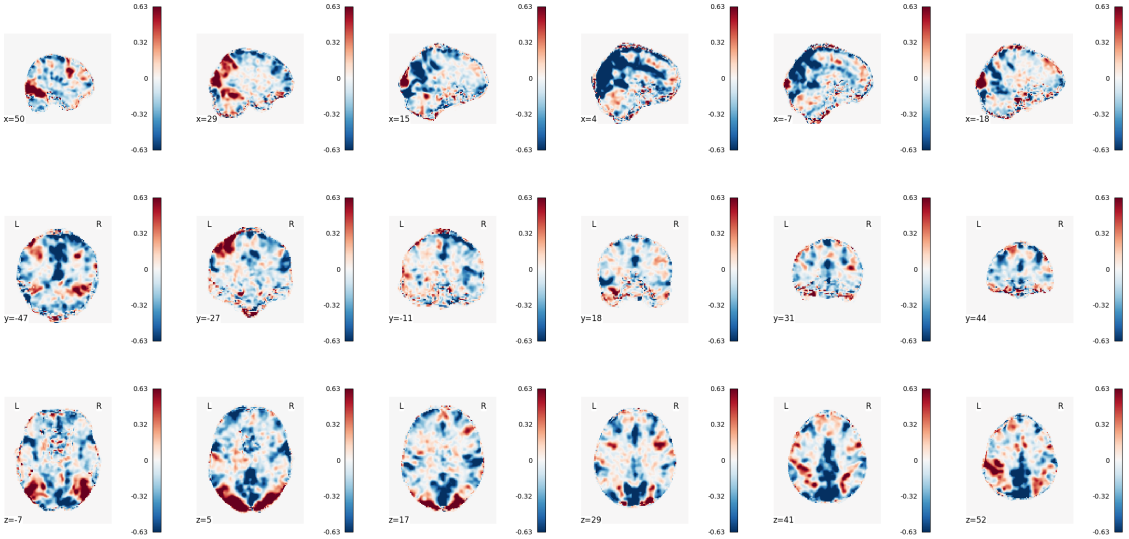


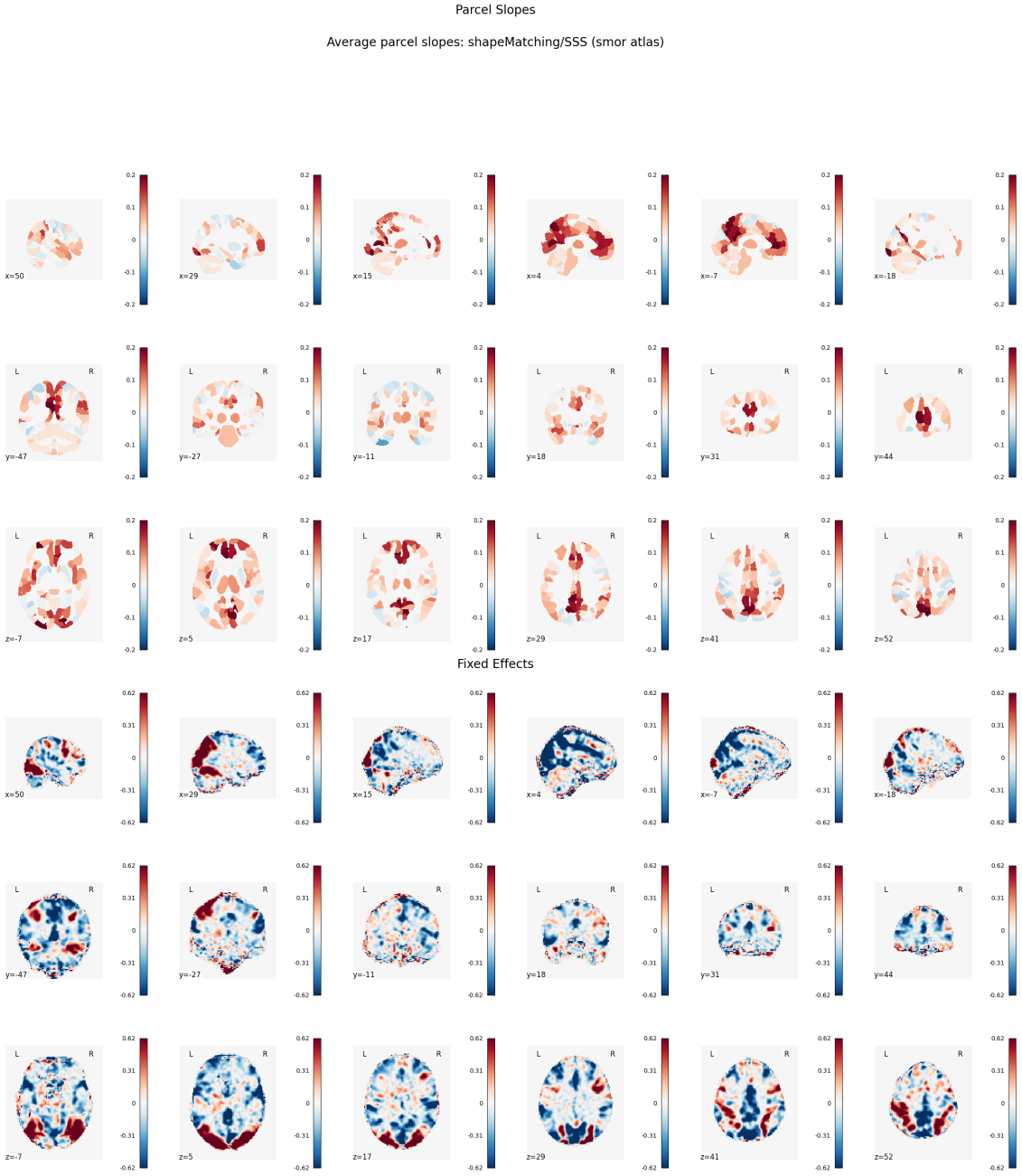
Parcel Slopes

Average parcel slopes: shapeMatching/SNN (smor atlas)



Fixed Effects



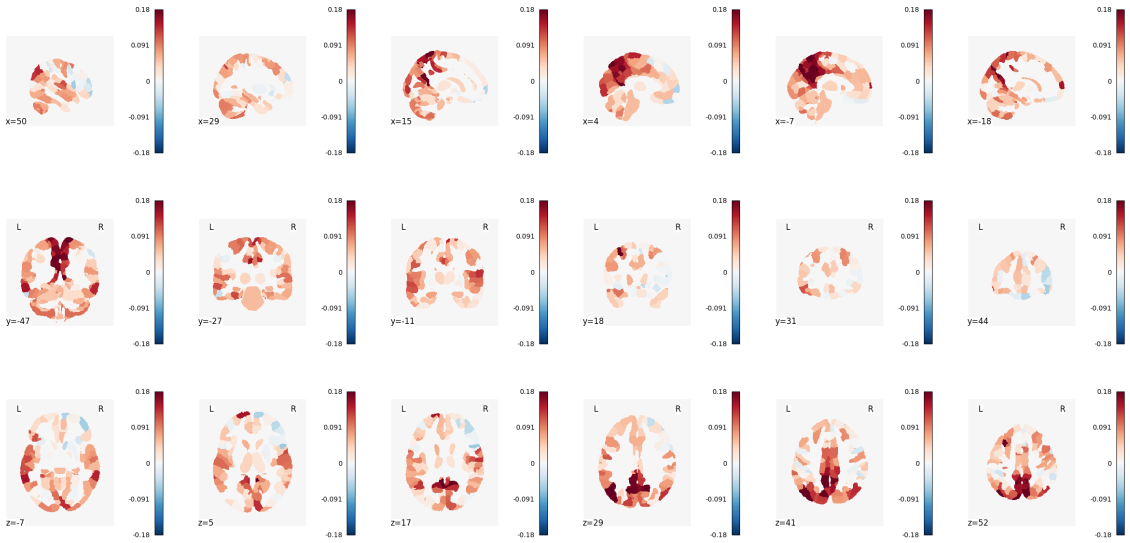


In [13]:
```
# plotting stop signal and rest of the tasks that didn't plot above:
rest_of_tasks = ["stopSignal", "cuedTS", "spatialTS"]

for task in rest_of_tasks:
    for contrast in avg_parcel_traj_results[task].keys():
        if len(avg_parcel_traj_results[task][contrast]) > 0:
            plot_slopes_and_fixed_effects(avg_parcel_traj_results, averaged_
```

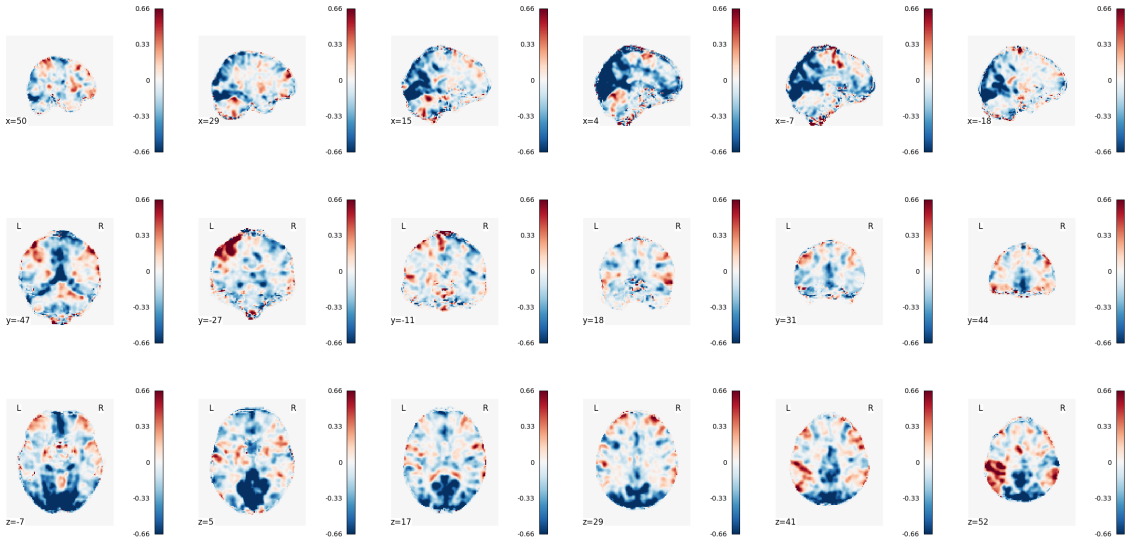
```
/tmp/ipykernel_18957/1305435263.py:85: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout(rect=[0, 0.03, 0.9, 0.95])
```

Parcel Slopes

Average parcel slopes: stopSignal/go (smor atlas)

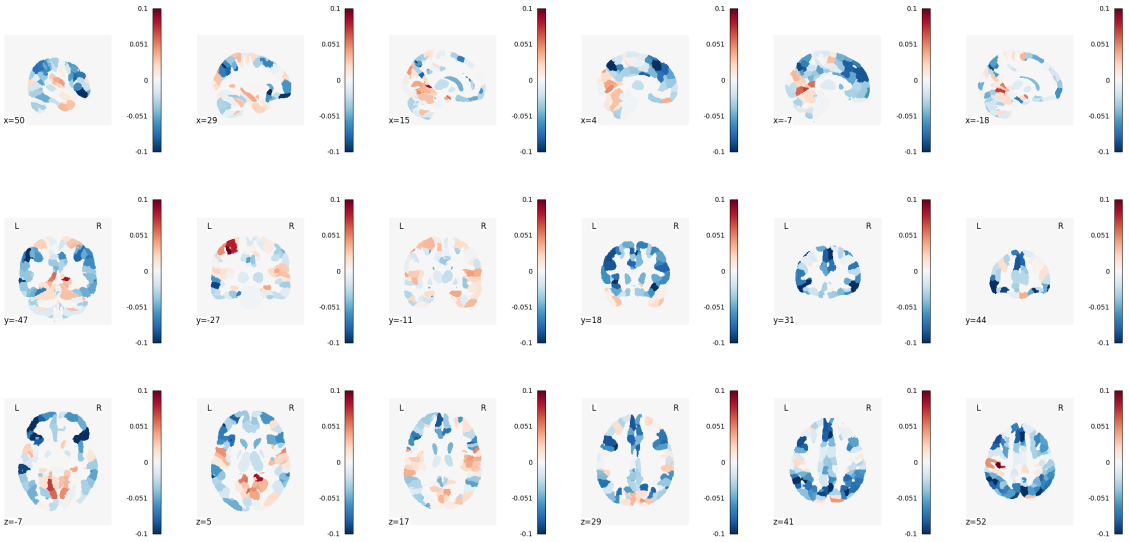


Fixed Effects

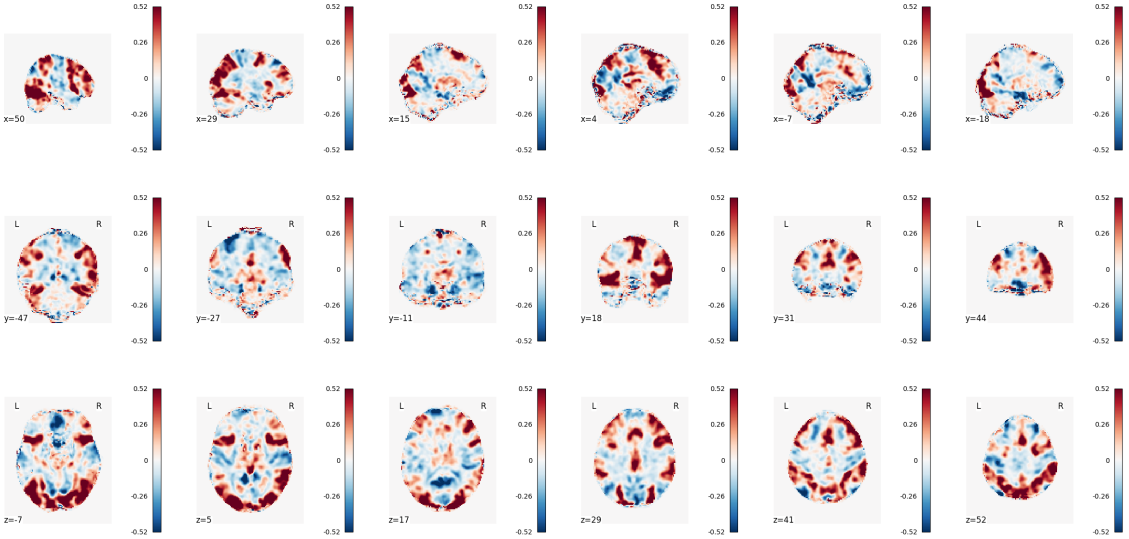


Parcel Slopes

Average parcel slopes: stopSignal/stop_failure-go (smor atlas)

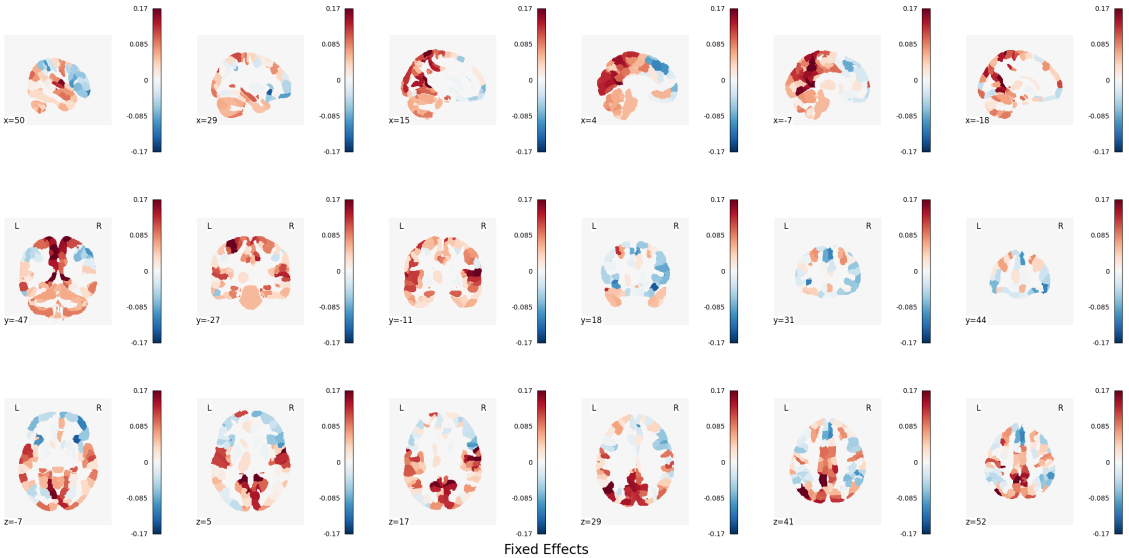


Fixed Effects

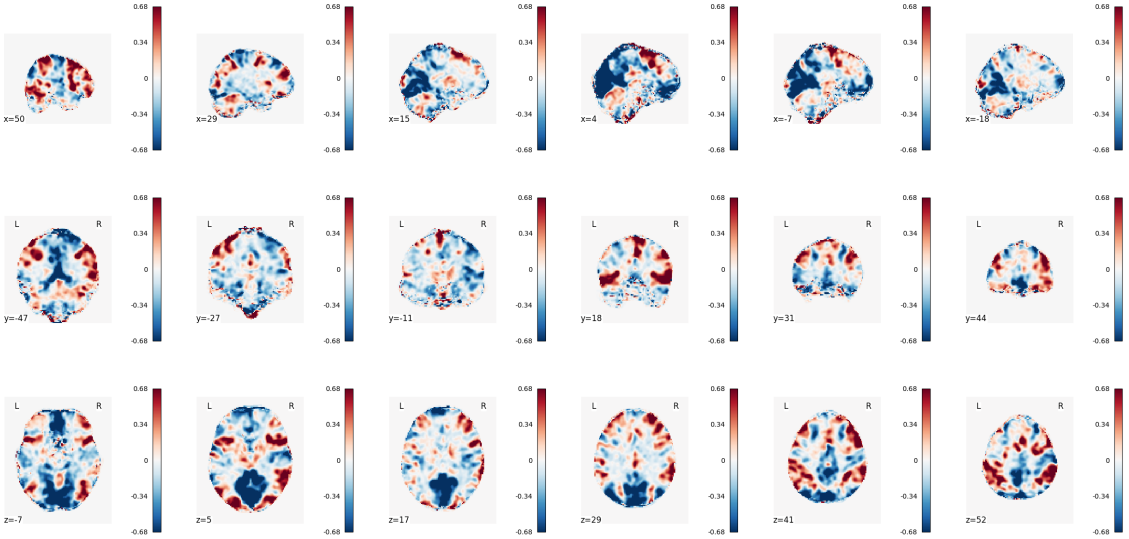


Parcel Slopes

Average parcel slopes: stopSignal/stop_failure (smor atlas)

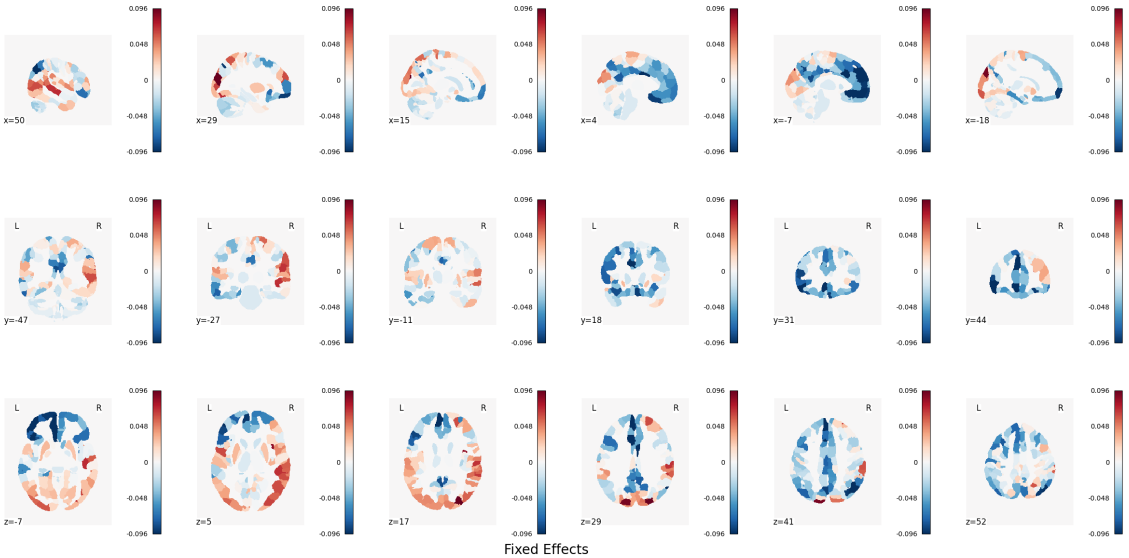


Fixed Effects

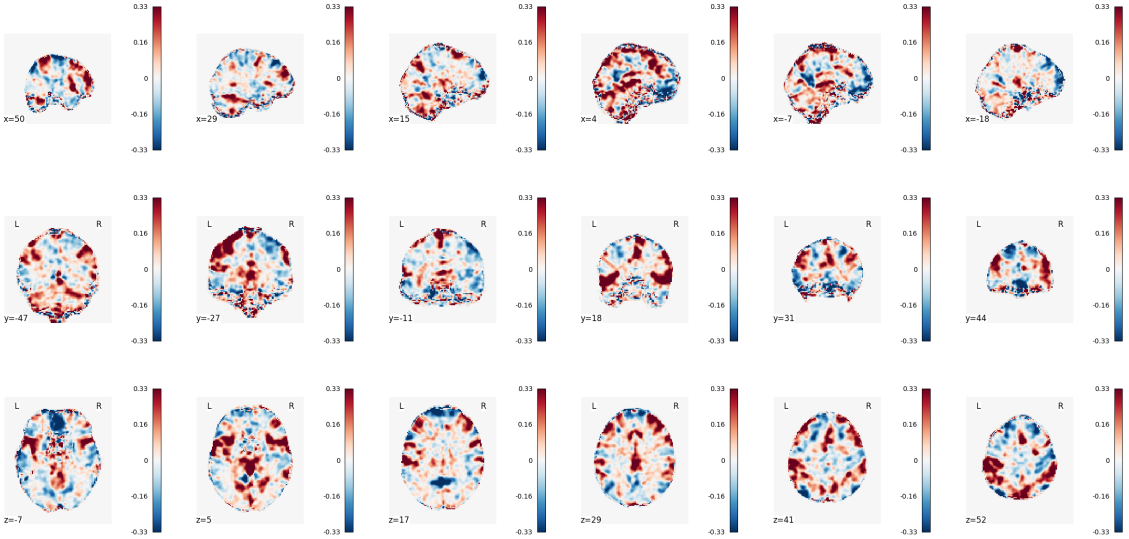


Parcel Slopes

Average parcel slopes: stopSignal/stop_failure-stop_success (smor atlas)

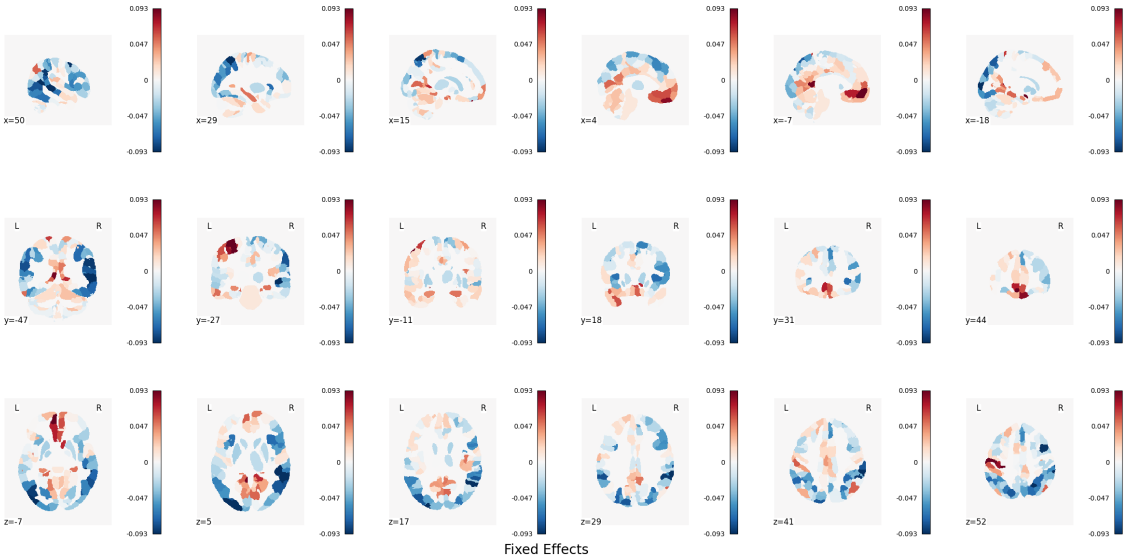


Fixed Effects

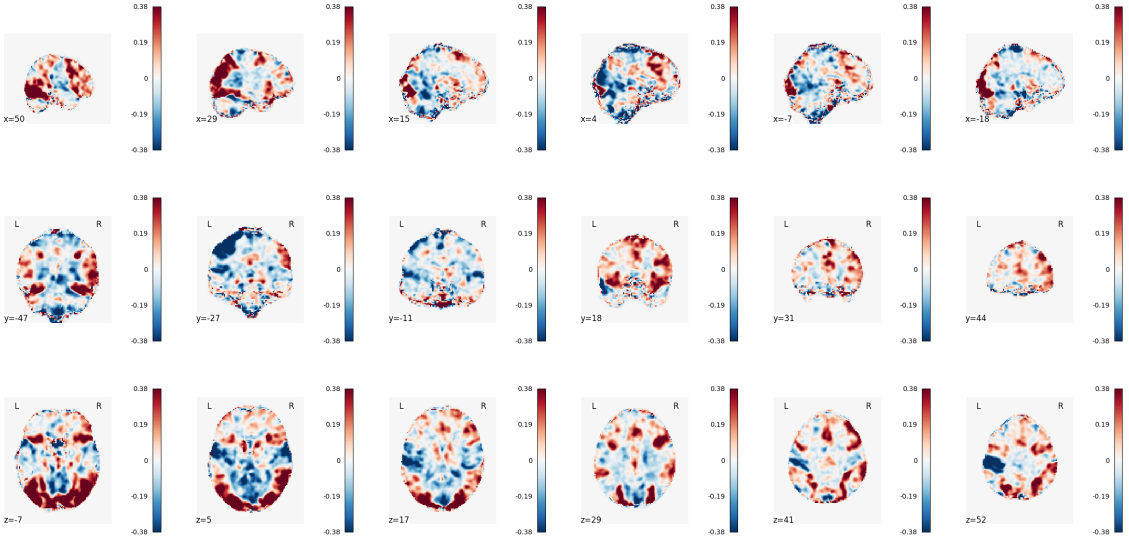


Parcel Slopes

Average parcel slopes: stopSignal/stop_success-go (smor atlas)

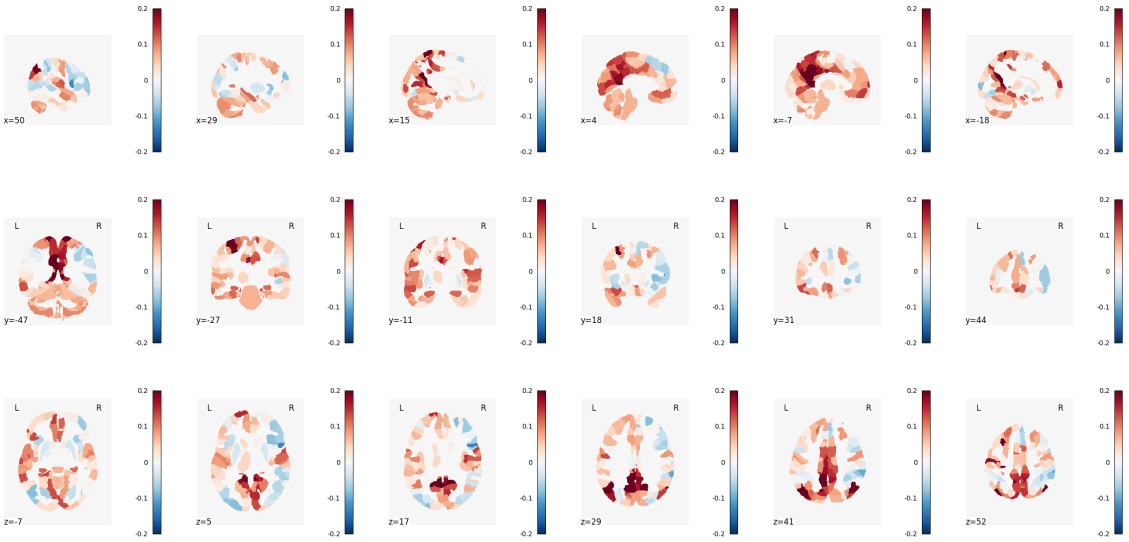


Fixed Effects

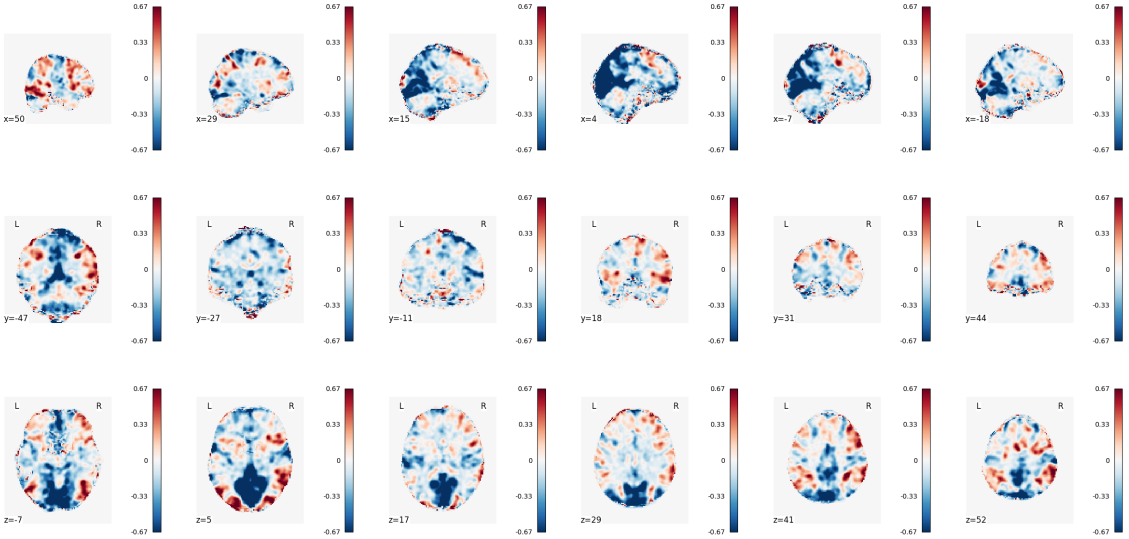


Parcel Slopes

Average parcel slopes: stopSignal/stop_success (smor atlas)

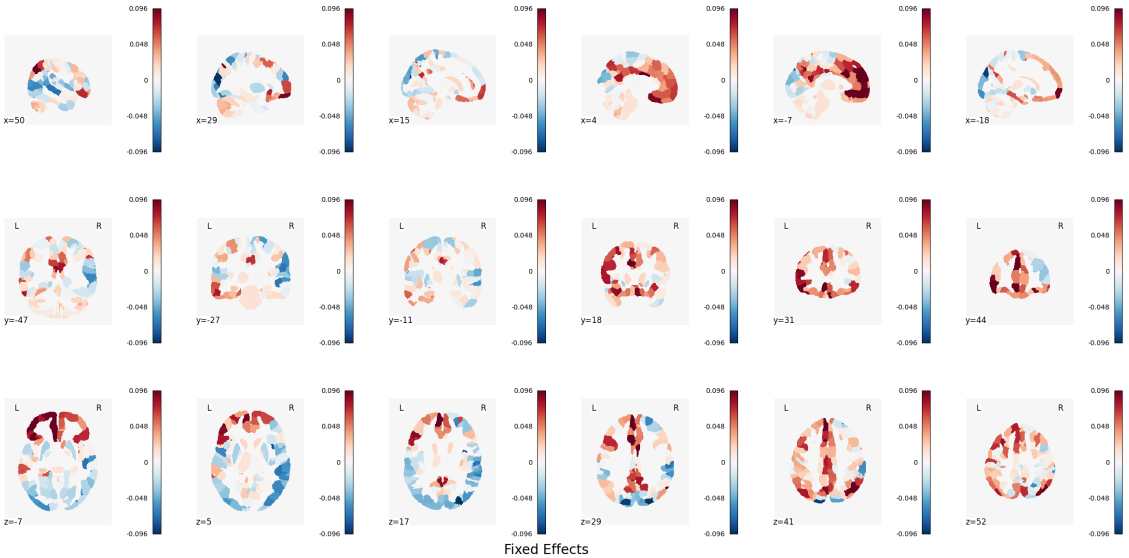


Fixed Effects

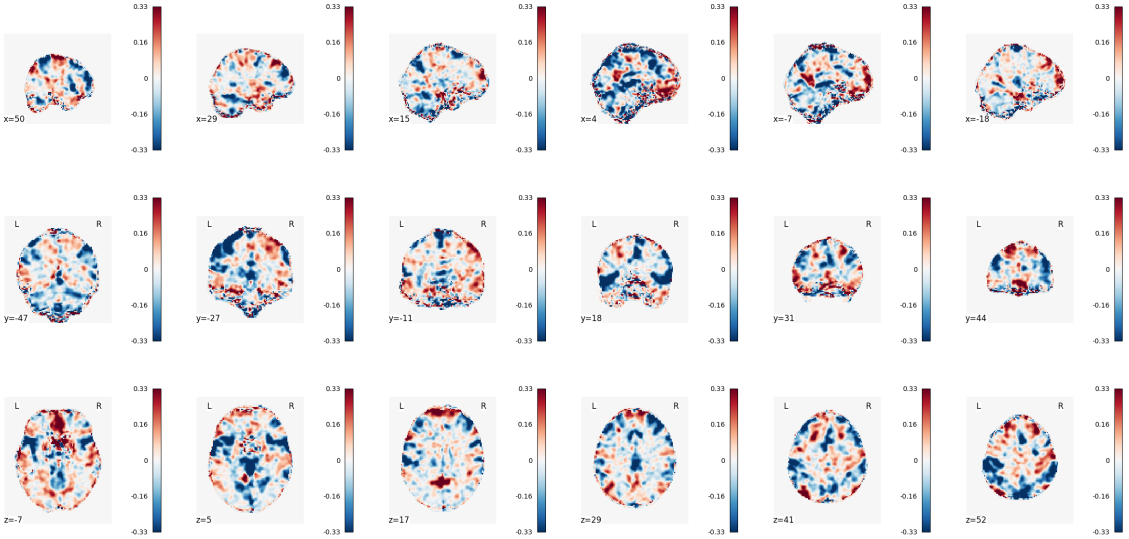


Parcel Slopes

Average parcel slopes: stopSignal/stop_success-stop_failure (smor atlas)

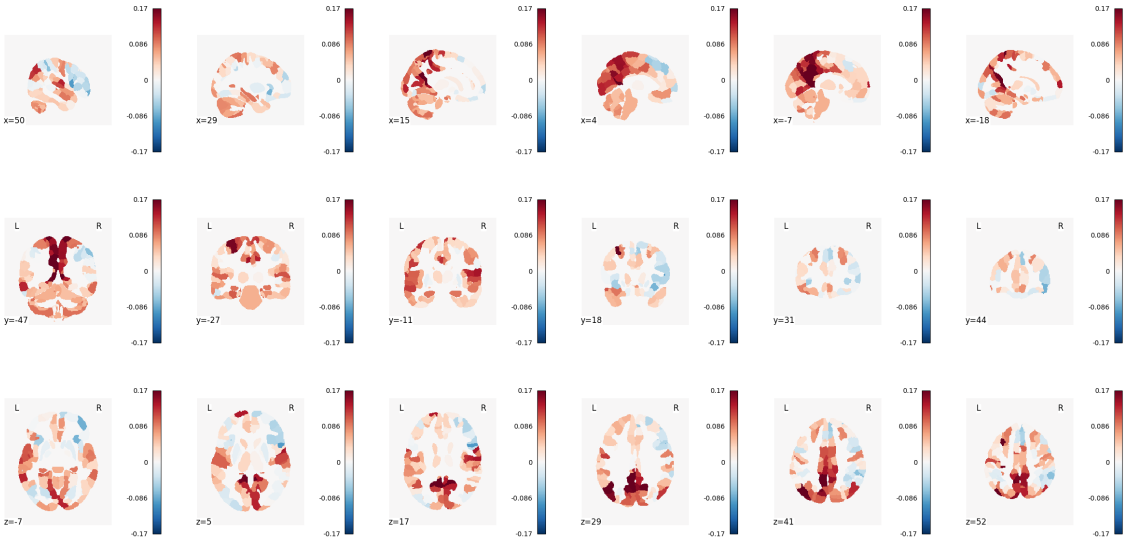


Fixed Effects

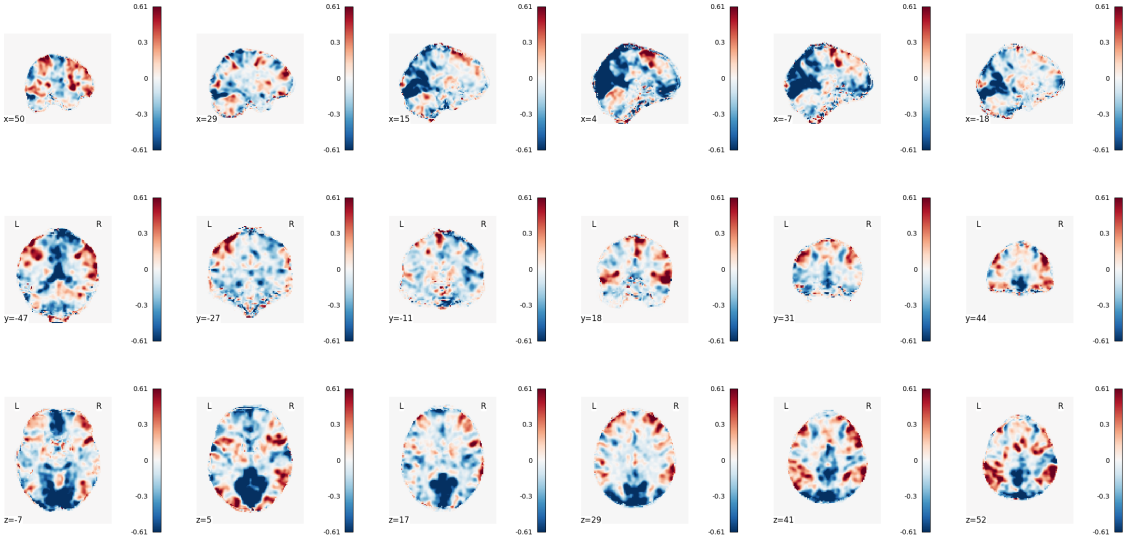


Parcel Slopes

Average parcel slopes: stopSignal/task-baseline (smor atlas)

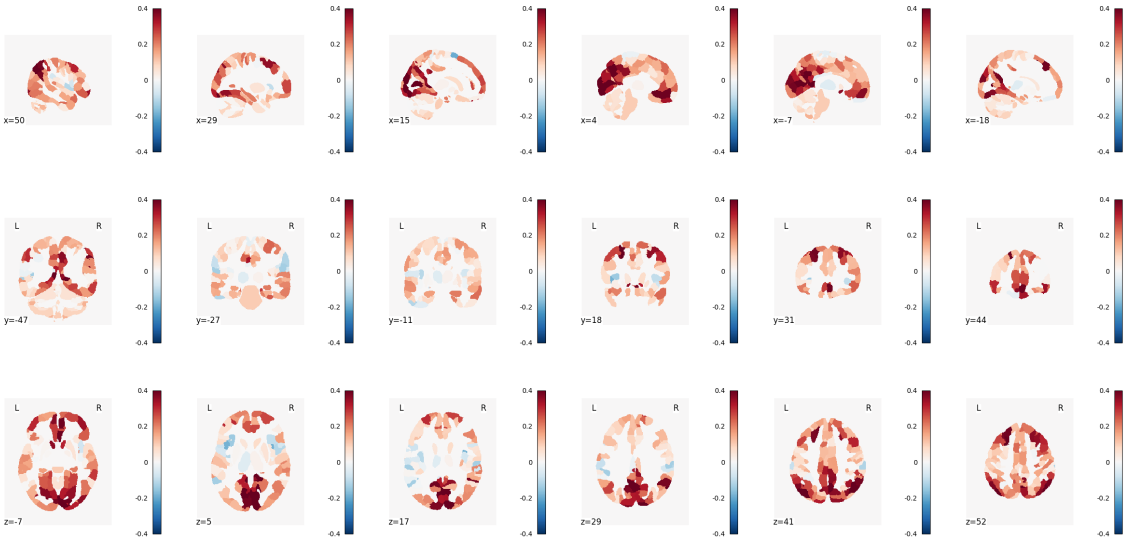


Fixed Effects

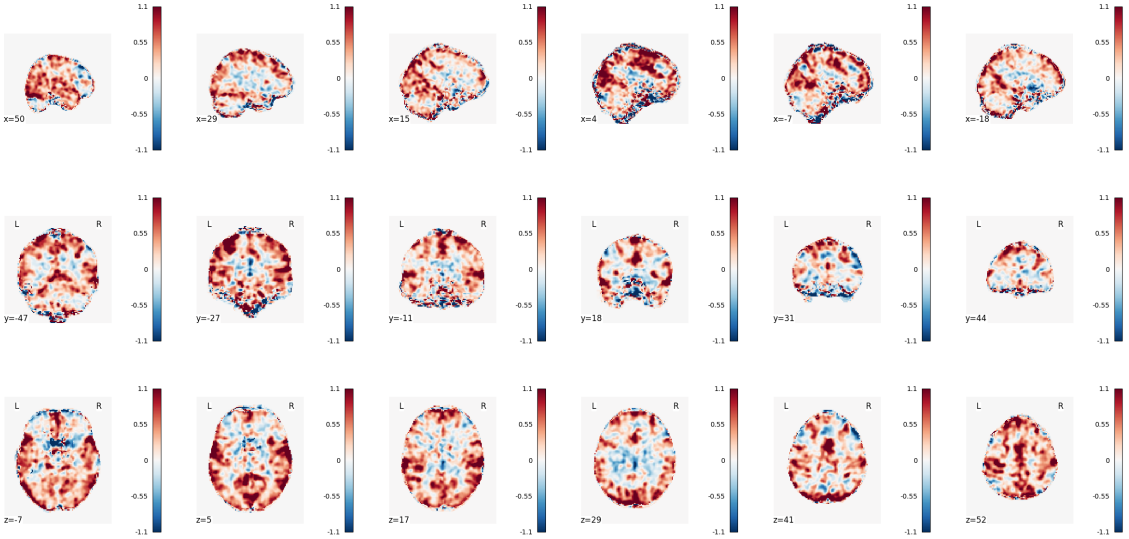


Parcel Slopes

Average parcel slopes: stopSignal/response_time (smor atlas)

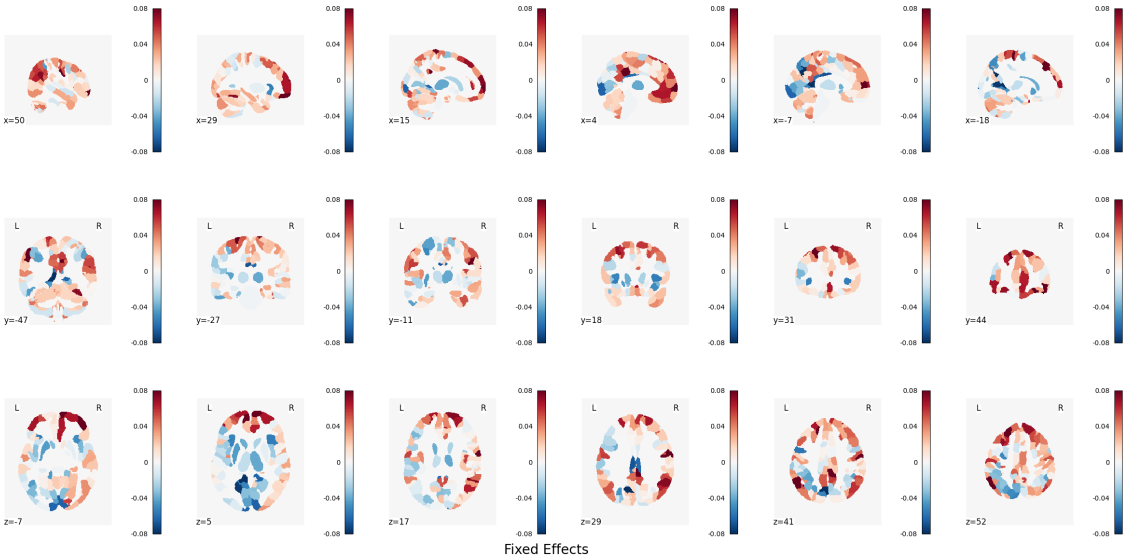


Fixed Effects

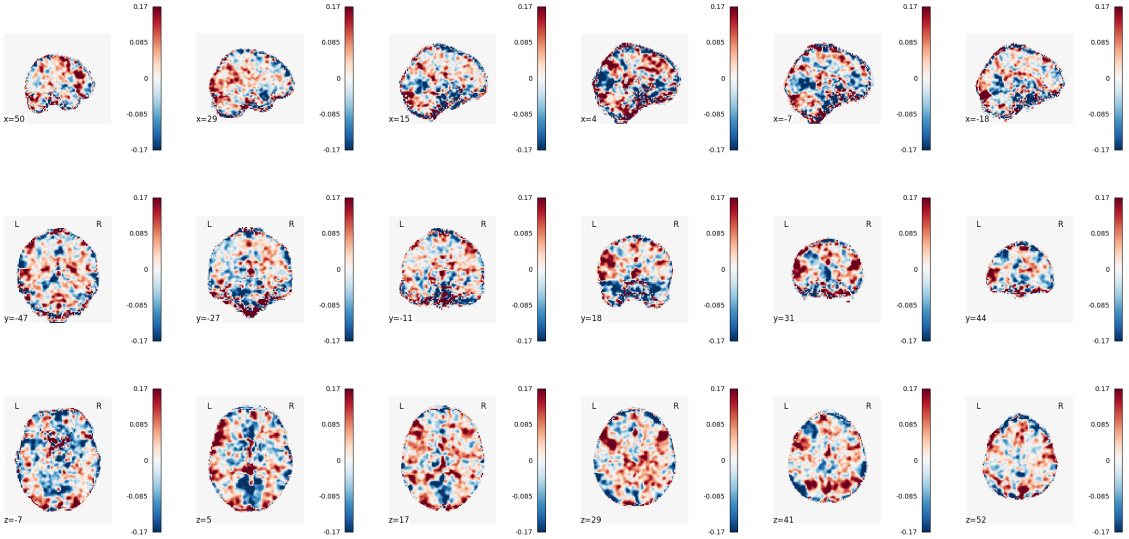


Parcel Slopes

Average parcel slopes: cuedTS/cue_switch_cost (smor atlas)

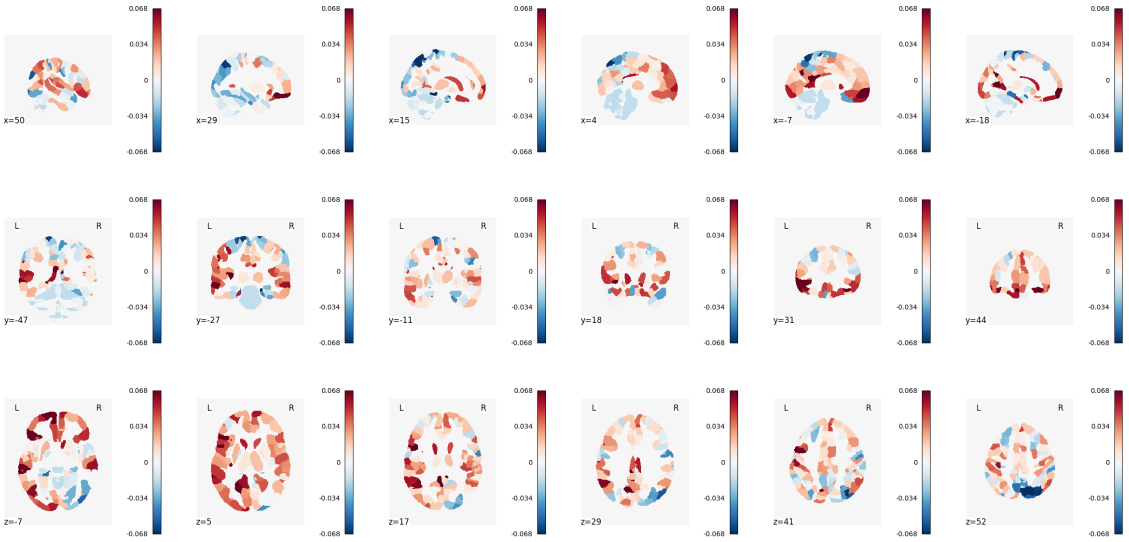


Fixed Effects

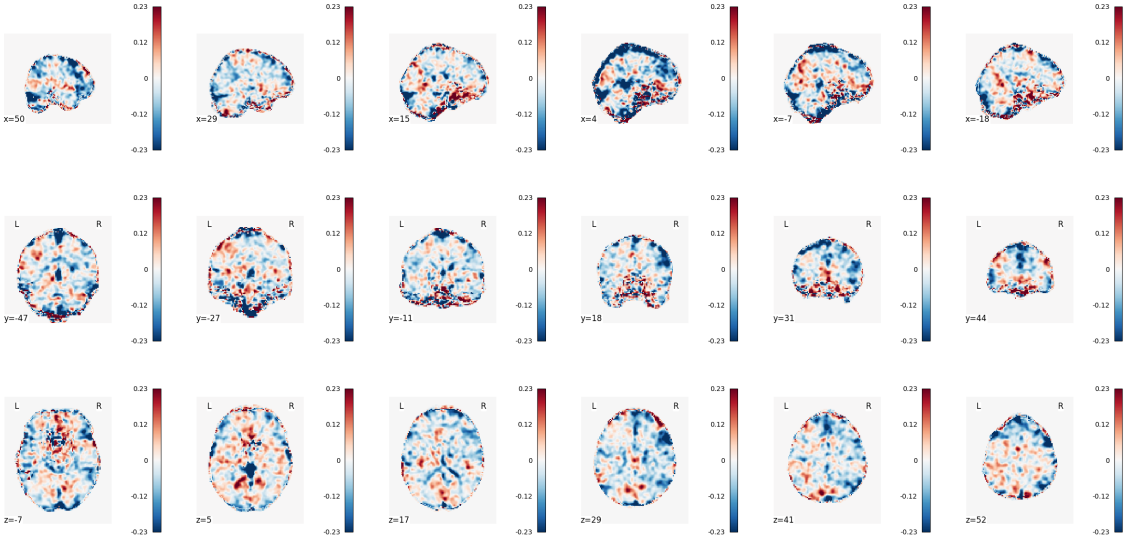


Parcel Slopes

Average parcel slopes: cuedTS/task_switch_cost (smor atlas)

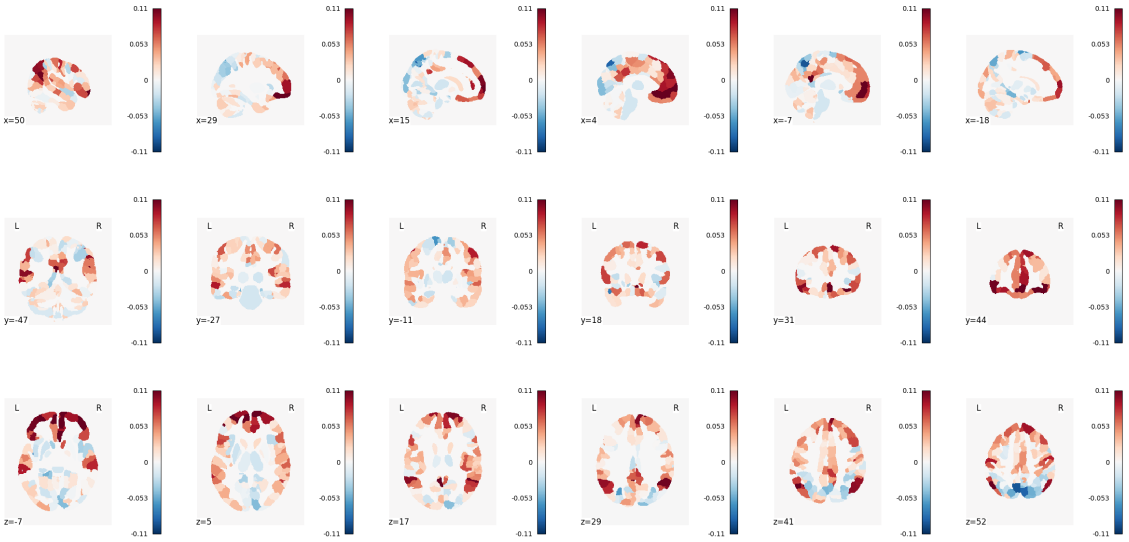


Fixed Effects

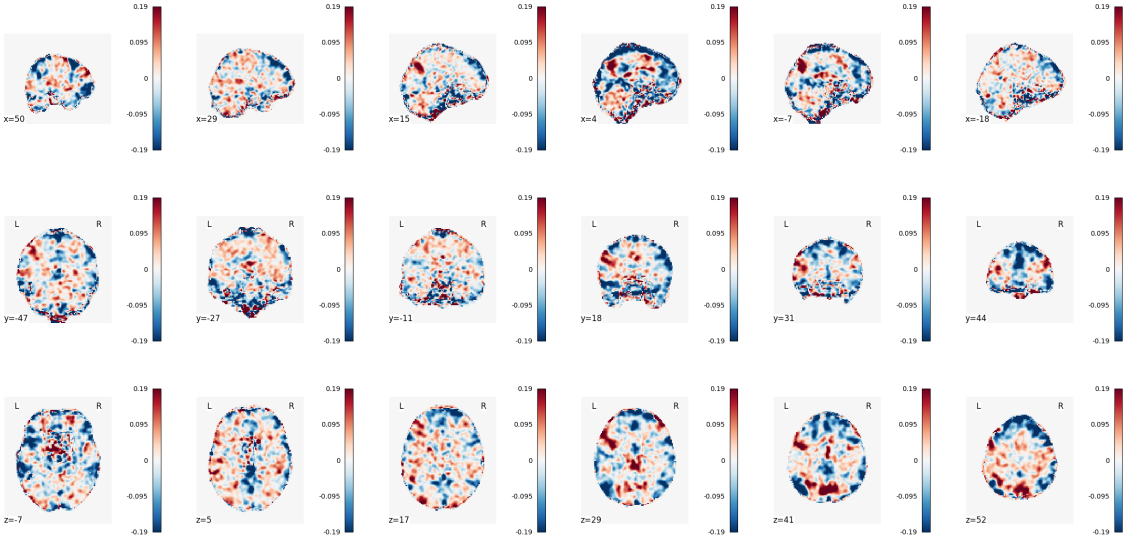


Parcel Slopes

Average parcel slopes: cuedTS/task_switch_cue_switch-task_stay_cue_stay (smor atlas)

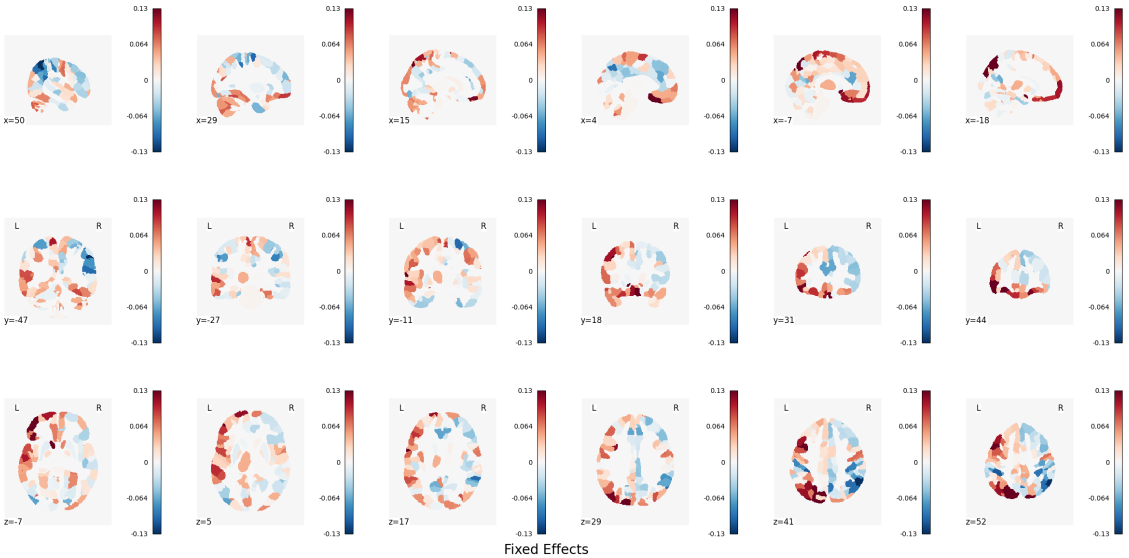


Fixed Effects

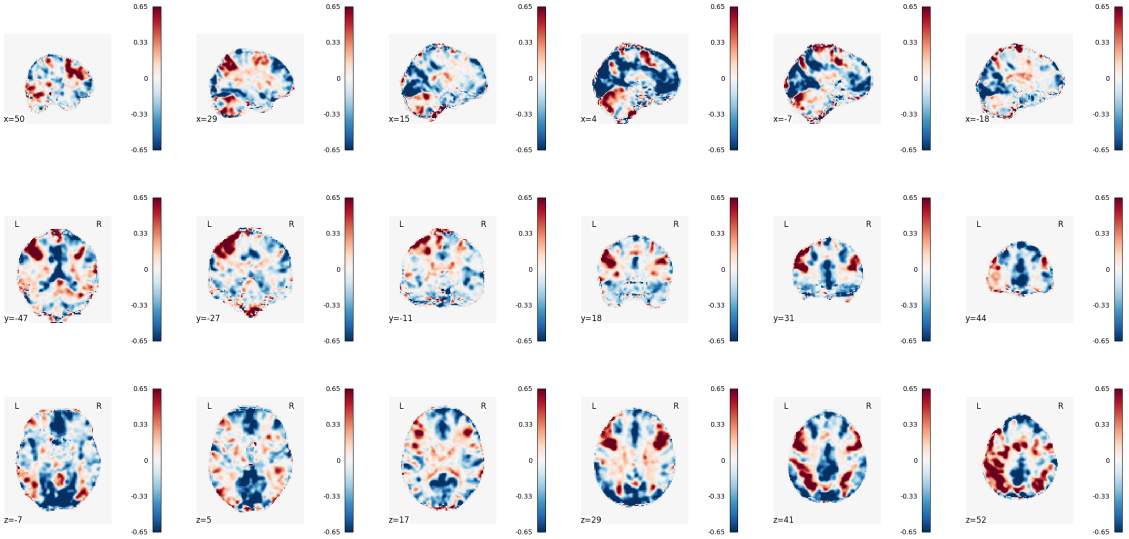


Parcel Slopes

Average parcel slopes: cuedTS/task-baseline (smor atlas)

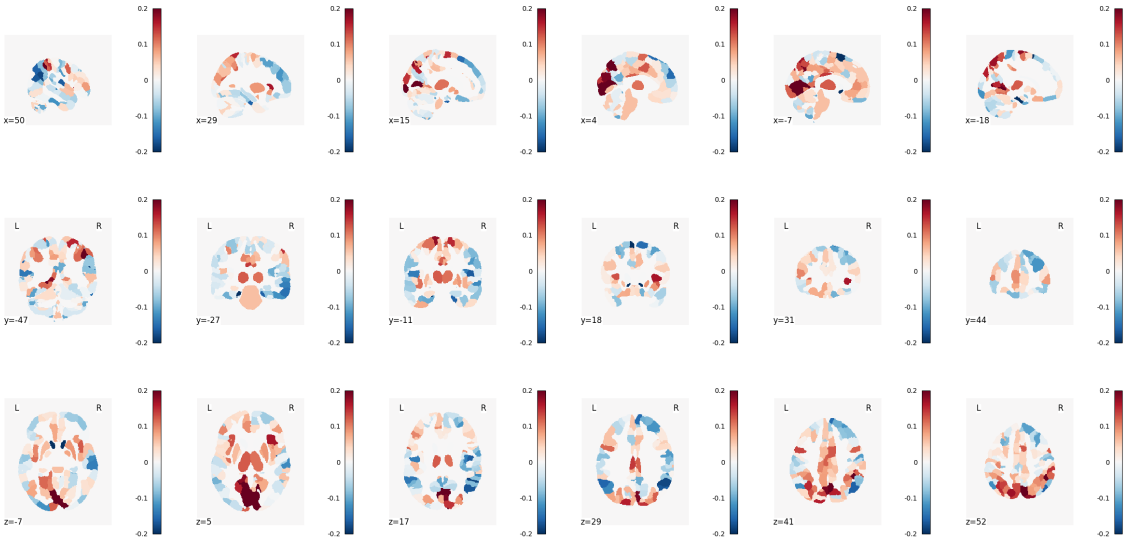


Fixed Effects

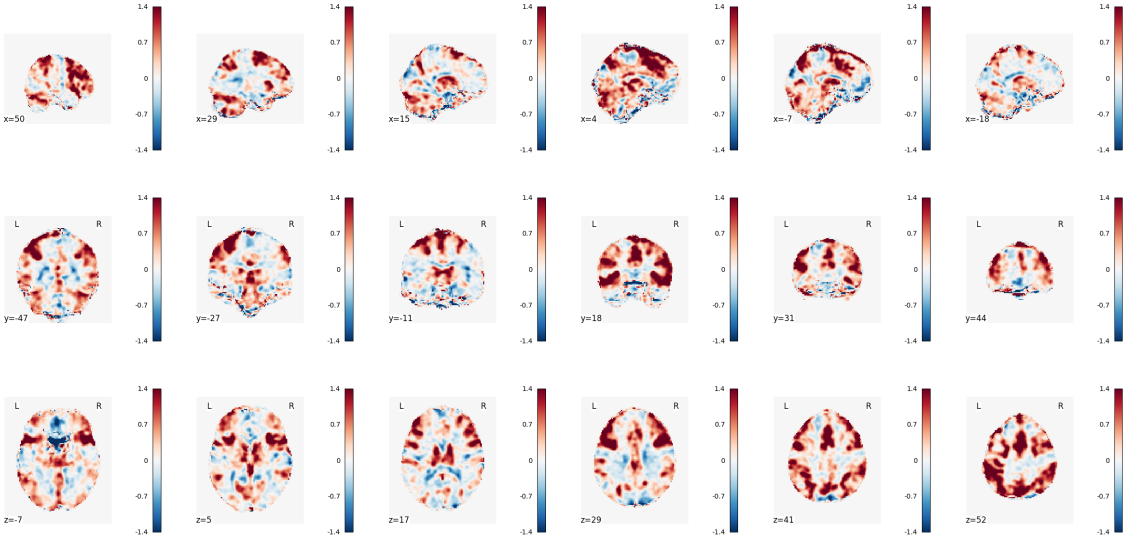


Parcel Slopes

Average parcel slopes: cuedTS/response_time (smor atlas)

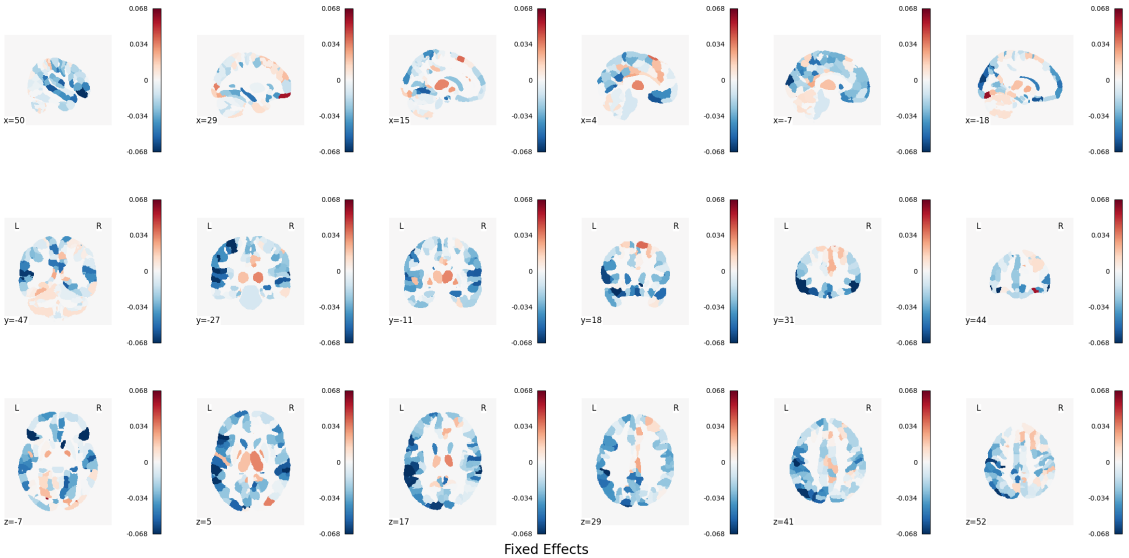


Fixed Effects

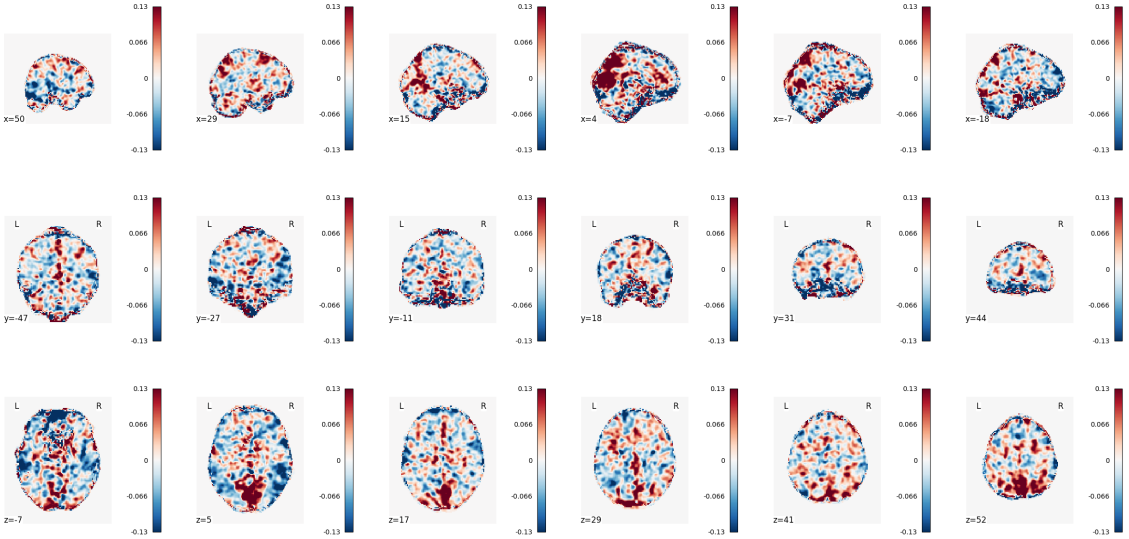


Parcel Slopes

Average parcel slopes: spatialTS/cue_switch_cost (smor atlas)

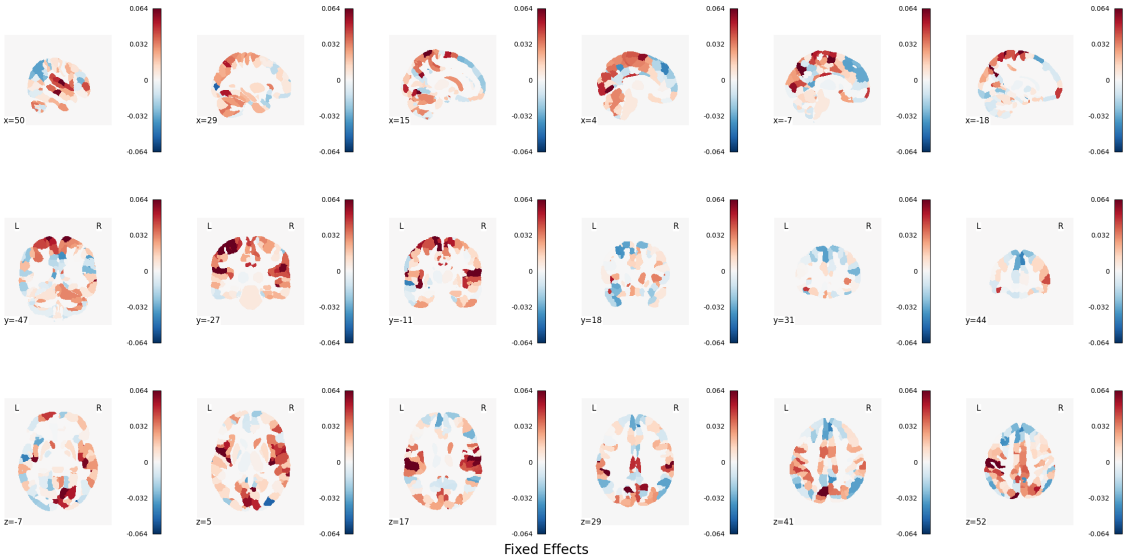


Fixed Effects

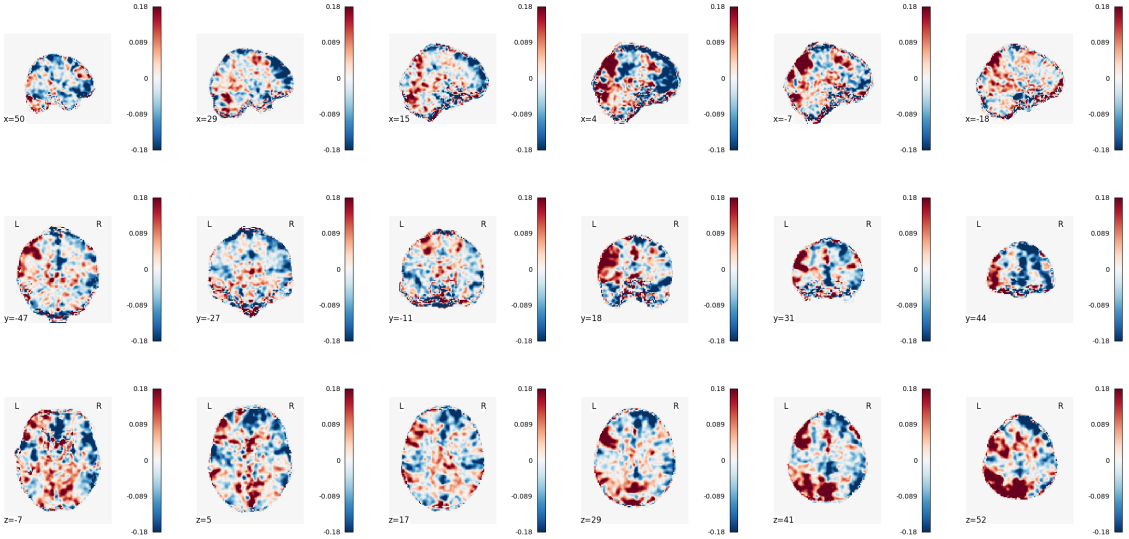


Parcel Slopes

Average parcel slopes: spatialTS/task_switch_cost (smor atlas)

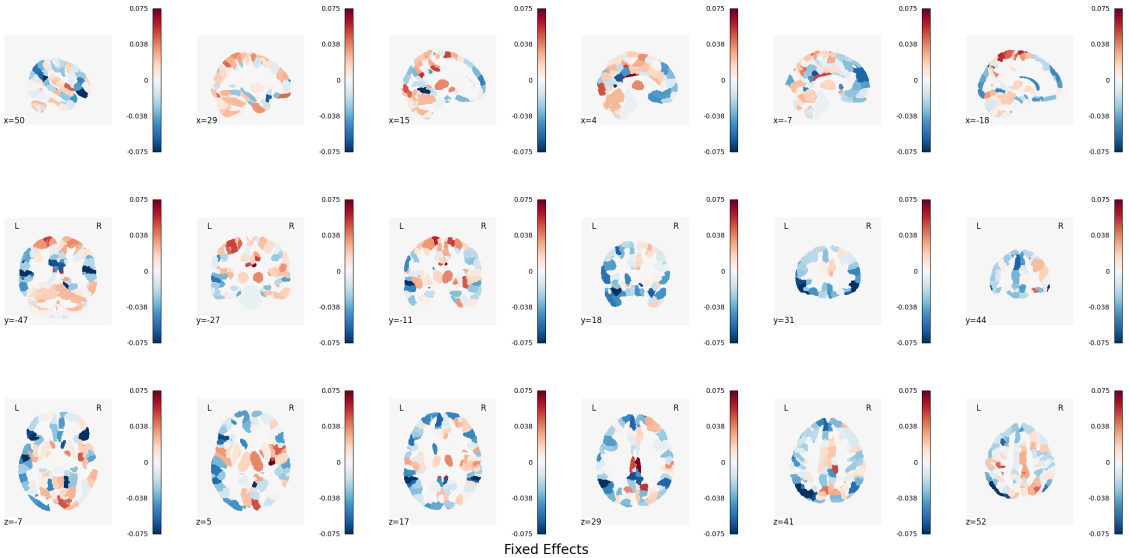


Fixed Effects

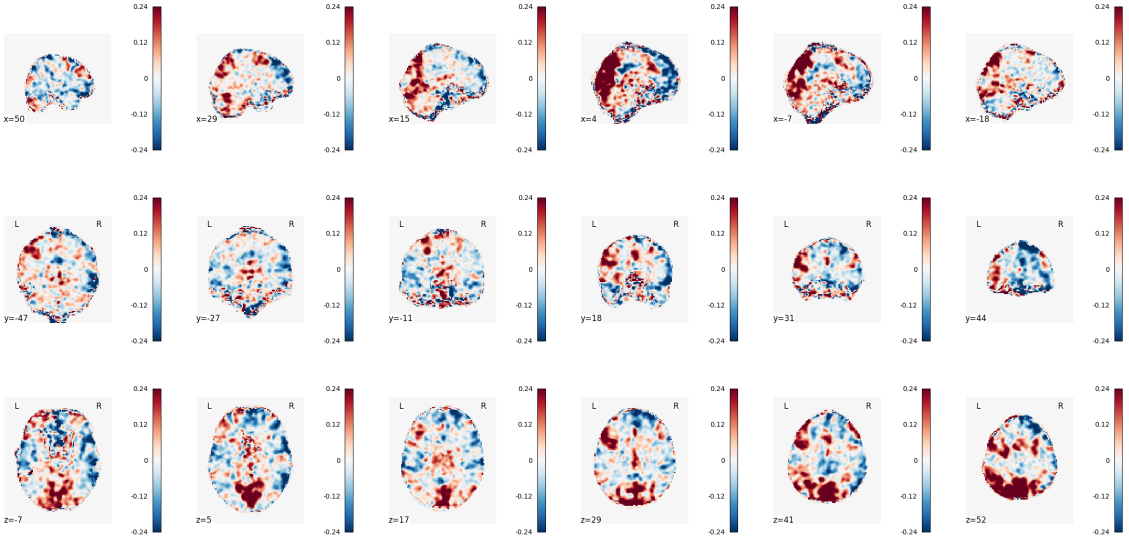


Parcel Slopes

Average parcel slopes: spatialTS/task_switch_cue_switch-task_stay_cue_stay (smor atlas)

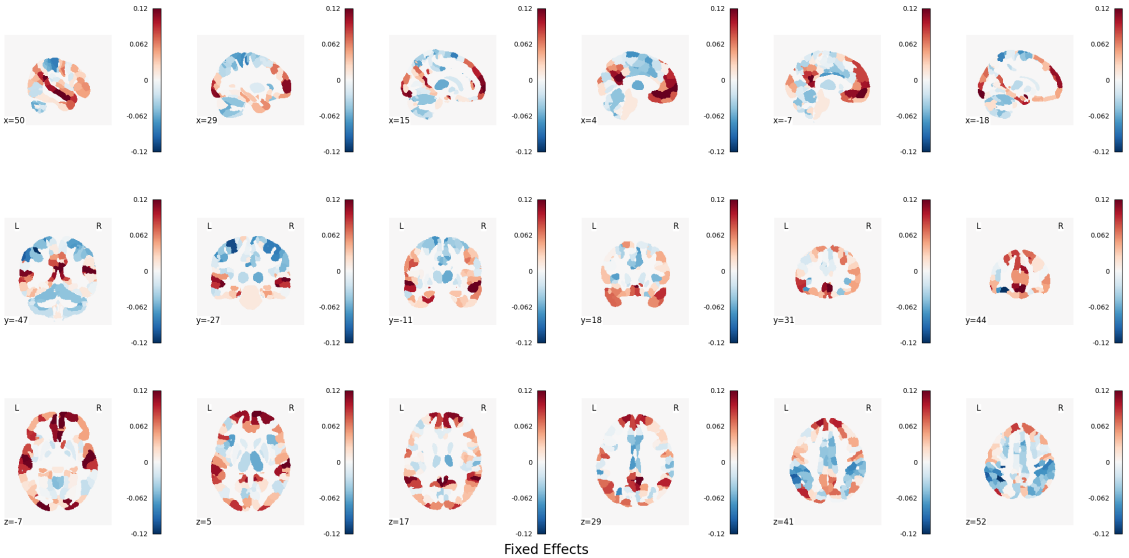


Fixed Effects

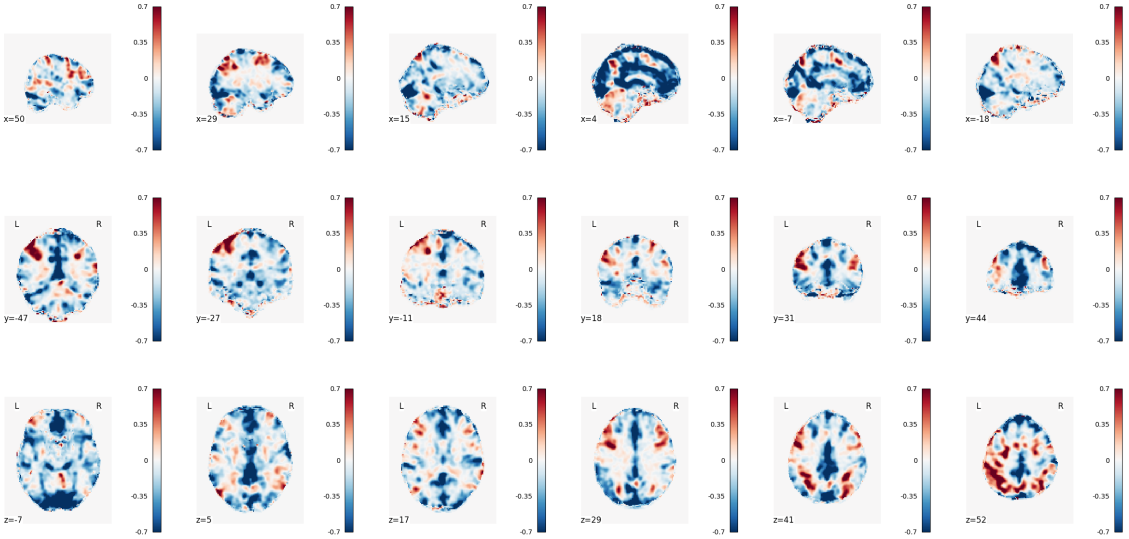


Parcel Slopes

Average parcel slopes: spatialTS/task-baseline (smor atlas)

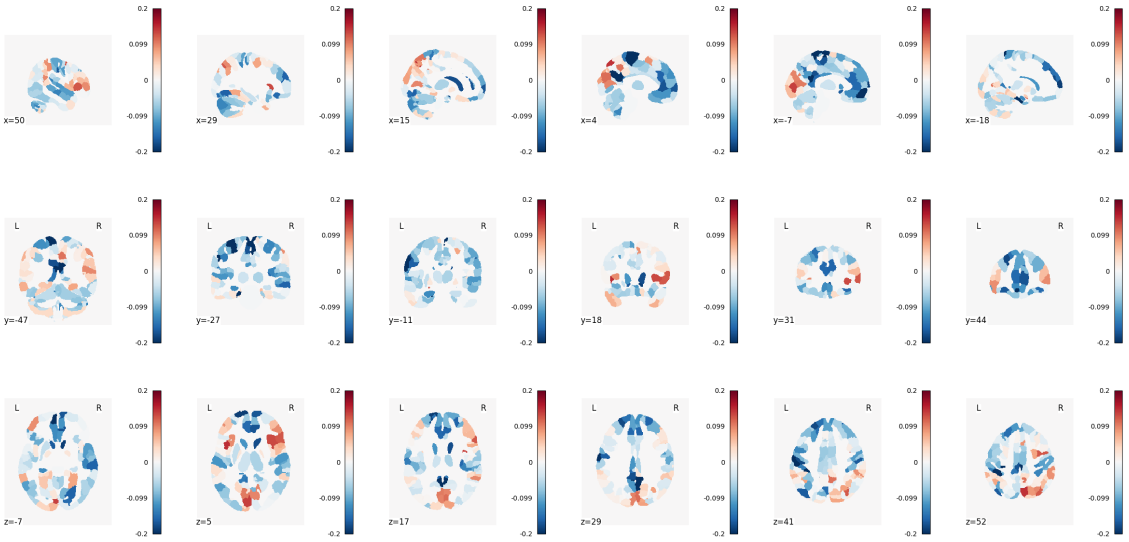


Fixed Effects



Parcel Slopes

Average parcel slopes: spatialTS/response_time (smor atlas)



Fixed Effects