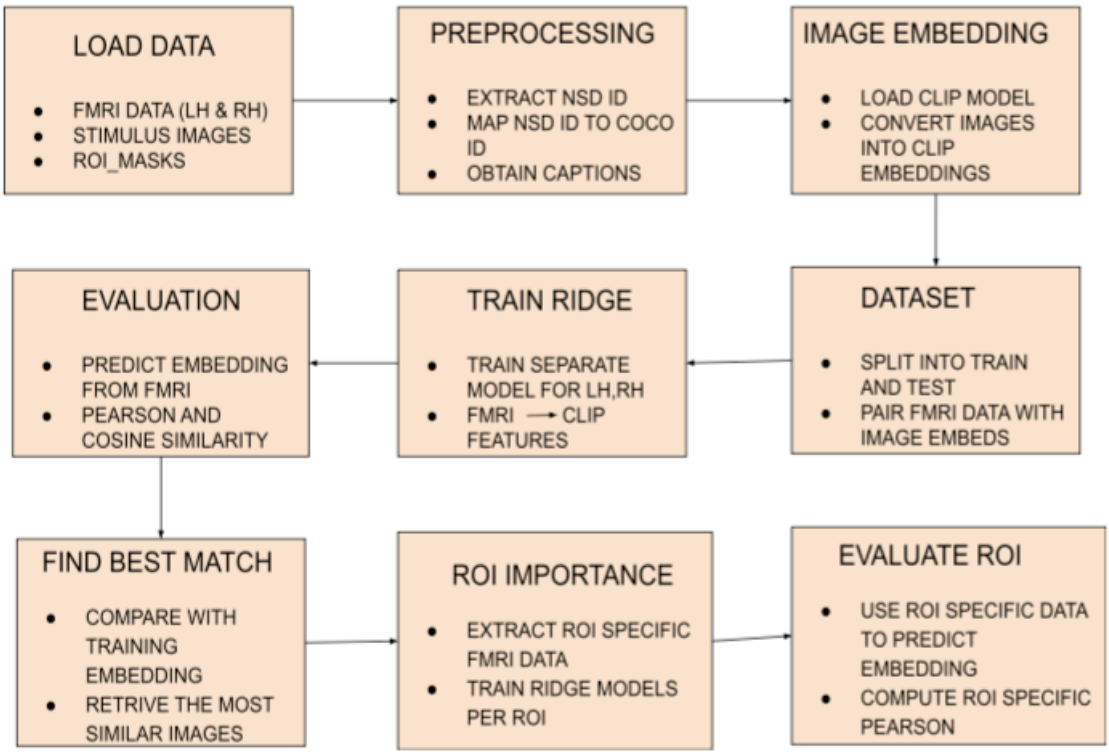
Brain Decoding for Visual Stimuli using fMRI and CLIP

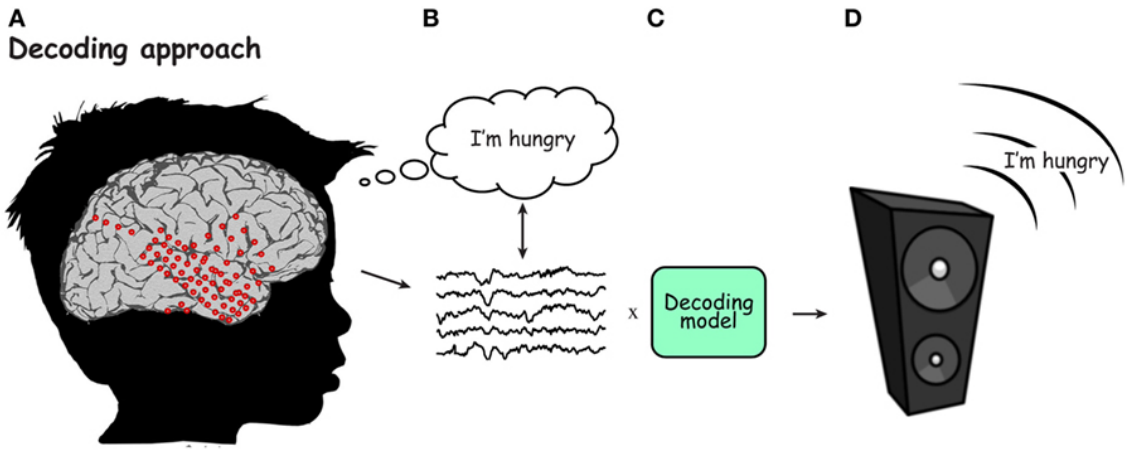
WORKFLOW :



WORKFLOW FOR THE BRAIN DECODING PIPELINE

INTRODUCTION

Brain decoding aims to reconstruct perceived images from brain activity. In this project, we train Ridge Regression models to map fMRI signals to CLIP image embeddings. By analyzing ROI-based feature importance, we identify brain regions crucial for visual processing.



Taken From Frontiers of Young Mind.com to draw analogy

DATA PREPARATION

Loading fMRI and Stimulus Data

We load preprocessed fMRI signals for both hemispheres (LH & RH) and the corresponding stimulus images. Each fMRI sample corresponds to a visual stimulus presented during scanning.

LOAD THE DATA

```python
# Set random seed for reproducibility
np.random.seed(42)
torch.manual_seed(42)

# Constants and paths
SUBJ = "subj04"
TRAINING_PATH = f"{SUBJ}/training_split"
ROI_PATH = f"{SUBJ}/roi_masks"
IMAGES_PATH = f"{TRAINING_PATH}/training_images"
FMRI_PATH = f"{TRAINING_PATH}/training_fmri"
LH_FMRI_PATH = os.path.join(FMRI_PATH, "lh_training_fmri.npy")
RH_FMRI_PATH = os.path.join(FMRI_PATH, "rh_training_fmri.npy")
STIM_INFO_PATH = "nsd_stim_info_merged.csv"

def load_fmri_data():
    """
    Load fMRI data for both hemispheres
    """
    print("Loading fMRI data...")
    lh_fmri = np.load(LH_FMRI_PATH)
    rh_fmri = np.load(RH_FMRI_PATH)

    print(f"Left hemisphere data shape: {lh_fmri.shape}")
    print(f"Right hemisphere data shape: {rh_fmri.shape}")

    return lh_fmri, rh_fmri
```

[3] ✓ 0.0s

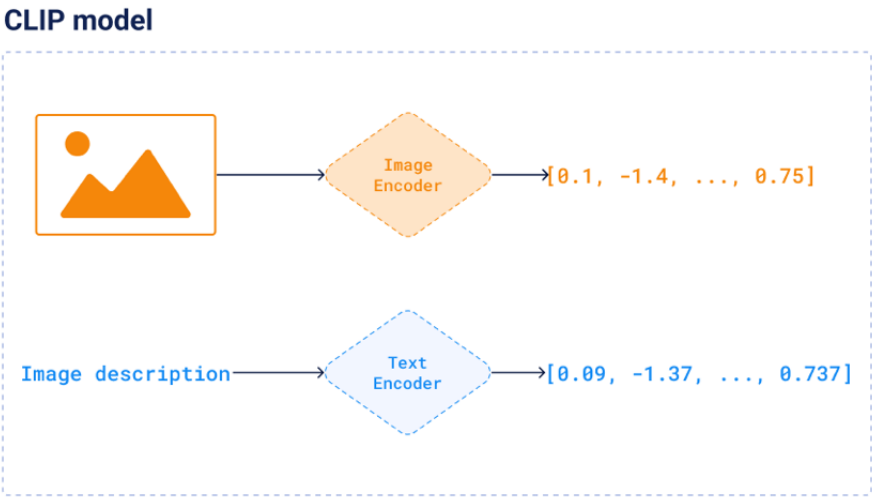
Mapping NSD IDs to COCO Captions

To retrieve textual descriptions of images, we map NSD dataset image IDs to COCO dataset captions.

```python
def load_mapping_info():
    """
    Load and preprocess the mapping information for the NSD dataset.
    """
    stim_info = pd.read_csv(STIM_INFO_PATH, index_col=0)
    important_columns = ['cocoId', 'cocoSplit', 'nsdId']
    stim_info = stim_info[important_columns]
    return stim_info

def extract_nsd_id(image_path):
    """
    Extract nsd_id from the given image path.
    """
    match = re.search(r'nsd-(\d+)', image_path)
    if match:
        return int(match.group(1))
    else:
        raise ValueError("nsd_id not found in the image path")


# Cache COCO data (to avoid reloading multiple times)
COCO_CACHE = {}

def get_coco_captions(nsdId, stim_info):
    """
    Get COCO captions for a given NSD image ID efficiently (without reloading COCO).
    """
    try:
        # Get COCO ID and split
        coco_id = stim_info[stim_info['nsdId'] == nsdId]['cocoId'].values[0]
        coco_split = stim_info[stim_info['nsdId'] == nsdId]['cocoSplit'].values[0]

        # Load COCO dataset only once per split
        if coco_split not in COCO_CACHE:
            coco_annotation_file = f"captions_{coco_split}.json"
            COCO_CACHE[coco_split] = COCO(coco_annotation_file)  # Load once

        coco_data = COCO_CACHE[coco_split]
        coco_ann_ids = coco_data.getAnnIds(coco_id)
        coco_annotations = coco_data.loadAnns(coco_ann_ids)

        # Extract captions
        captions = [anno['caption'] for anno in coco_annotations]
        return captions
    except Exception as e:
        print(f"Error getting captions for nsdId {nsdId}: {e}")
        return []
```

FEATURE EXTRACTION

We use the CLIP model to generate high-dimensional embeddings for stimulus images. These embeddings serve as the target outputs for our regression model.




```python
def extract_image_features(image_paths, batch_size=16):
    """
    Extract CLIP image features in batches.
    """
    all_features = []

    for i in tqdm(range(0, len(image_paths), batch_size), desc="Extracting image features"):
        batch_paths = image_paths[i: i + batch_size]
        images = [Image.open(p).convert("RGB") for p in batch_paths]

        # Process batch
        inputs = clip_processor(images=images, return_tensors="pt").to(device)

        with torch.no_grad():
            image_features = clip_model.get_image_features(**inputs)
            image_features = image_features / image_features.norm(dim=-1, keepdim=True)
            image_features = image_features.cpu().numpy()

        all_features.append(image_features)

    print("Shape of image features:", np.vstack(all_features).shape)
    return np.vstack(all_features)
```

✓ 0.0s

DATA PREPROCESSING AND SPLITTING

We split the dataset into **training (80%)** and **testing (20%)** sets, ensuring a balanced representation of visual stimuli. Each sample consists of **fMRI features as input** and **CLIP embeddings as output**.To optimize model training, we structure the data using PyTorch DataLoaders. This allows batch-

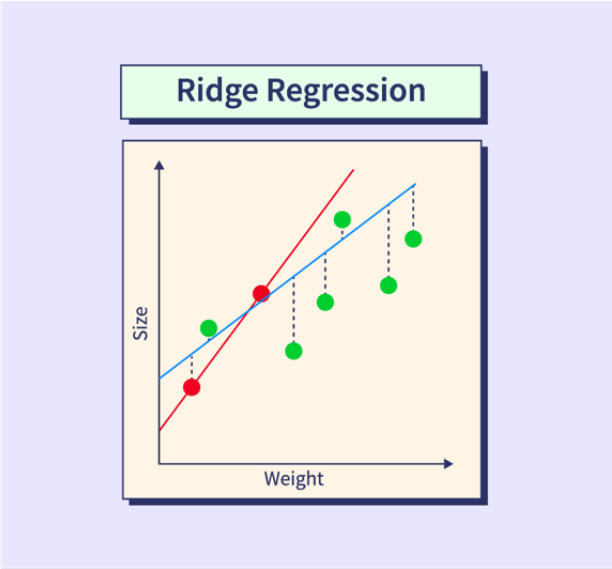
wise training and testing.

```python
def prepare_dataset():
    """
    Prepare dataset for training and testing
    """
    # Load stimulus info for caption mapping
    stim_info = load_mapping_info()

    # Get all image paths
    image_paths = sorted(glob.glob(os.path.join(IMAGES_PATH, "*.png")))
    print(f"Found {len(image_paths)} images")

    # Extract image features
    image_features = extract_image_features(image_paths)
    print(f"Image features shape: {image_features.shape}")

    # Load fMRI data
    lh_fmri, rh_fmri = load_fmri_data()

    # Create image info with nsd_ids
    image_info = []
    for path in image_paths:
        nsd_id = extract_nsd_id(path)
        image_info.append({
            'path': path,
            'nsd_id': nsd_id
        })

    # Split data into train and test sets
    train_indices, test_indices = train_test_split(
        range(len(image_paths)),
        test_size=0.2,
        random_state=42
    )

    # Prepare train and test datasets - INPUT : FMRI, OUTPUT : IMAGE FEATURES
    X_train_lh = lh_fmri[train_indices]
    X_train_rh = rh_fmri[train_indices]
    X_test_lh = lh_fmri[test_indices]
    X_test_rh = rh_fmri[test_indices]

    y_train = image_features[train_indices]
    y_test = image_features[test_indices]

    train_info = [image_info[i] for i in train_indices]
    test_info = [image_info[i] for i in test_indices]

    return (X_train_lh, X_train_rh, y_train, train_info,
            X_test_lh, X_test_rh, y_test, test_info,
            stim_info)
```

TRAINING THE RIDGE REGRESSION MODEL

We train **separate Ridge regression models** for the left and right hemispheres, mapping fMRI activity to image embeddings. The models are

optimized using **L2-regularization** to prevent overfitting.




```python
def train_decoding_model(train_loader, hemisphere, alpha=100.0):
    """
    Train Ridge Regression using sklearn's .fit() function with DataLoader.
    """
    print(f"Training Ridge regression model for {hemisphere} hemisphere")

    # Collect all training data from DataLoader
    X_train_list, y_train_list = [], []

    for fmri, embedding in train_loader:
        X_train_list.append(fmri.numpy())  # Convert tensor to NumPy
        y_train_list.append(embedding.numpy())

    # Stack all batches into final train arrays
    X_train = np.vstack(X_train_list)
    y_train = np.vstack(y_train_list)

    # Train Ridge Regression model
    model = Ridge(alpha=alpha)
    model.fit(X_train, y_train)

    print(f"Training complete for {hemisphere} hemisphere.")
    return model
```

0.0s

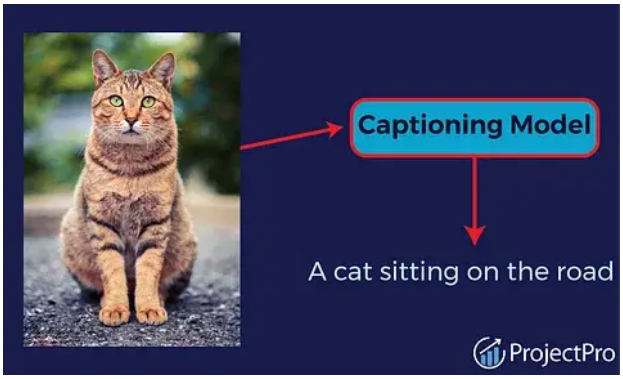
MODEL EVALUATION

After training, we use the models to predict embeddings from test fMRI data.We evaluate model performance using:

- **Pearson Correlation**: Measures how well predicted embeddings align with ground truth embeddings.
- **Cosine Similarity**: Assesses the similarity between predicted and actual embeddings in high-dimensional space.

```python
def evaluate_model(model, test_loader, y_train,hemisphere):
    """
    Evaluate Ridge Regression model using Pearson correlation, Cosine similarity, and Word Mover's Distance.
    """
    print(f"Evaluating Ridge regression model for {hemisphere} hemisphere...")

    # Collect all test data
    X_test_list, y_test_list = [], []

    for fmri, embedding in test_loader:
        X_test_list.append(fmri.numpy())  # Convert tensors to NumPy
        y_test_list.append(embedding.numpy())

    # Stack all batches into final test arrays
    X_test = np.vstack(X_test_list)
    y_test = np.vstack(y_test_list)

    # Predict embeddings
    y_pred = model.predict(X_test)

    # Compute Pearson correlation
    pearson_corrs = np.array([pearsonr(y_pred[i], y_test[i])[0] for i in range(len(y_test))])
    avg_corr = np.mean(pearson_corrs)

    # Compute Cosine similarity
    cosine_similarities = cosine_similarity(y_pred, y_train)
    avg_cosine_sim = np.mean(np.max(cosine_similarities, axis=1))  # Max similarity per test sample

    print(f"Average Pearson correlation for {hemisphere}: {avg_corr:.4f}")
    print(f"Average Cosine Similarity for {hemisphere}: {avg_cosine_sim:.4f}")

    return y_pred, pearson_corrs, cosine_similarities
```

IMAGE MATCHING AND VISUALISATION



For each test sample, we retrieve the most similar training images using cosine similarity. This allows us to visualize how well our model reconstructs perceived stimuli. The test image is displayed alongside its **top-N closest matches**, with similarity scores indicating the degree of alignment.

```python
def visualize_results(test_idx, top_train_indices, top_similarities, test_info, train_info, stim_info, n=5, images_per_row=3):
    """
    Displays the test image separately, then shows the top N matched images in multiple rows.
    Captions are neatly aligned under each image.
    """
    test_path = test_info[test_idx]['path']
    test_nsd_id = test_info[test_idx]['nsd_id']
    test_image = Image.open(test_path)
    test_captions = get_coco_captions(test_nsd_id, stim_info)
    test_caption = test_captions[0] if test_captions else "No caption available"

    num_rows = (n + images_per_row - 1) // images_per_row  # Calculate rows needed for matches

    fig, axes = plt.subplots(num_rows * 2 + 2, images_per_row, figsize=(15, (num_rows + 1) * 5),
                             gridspec_kw={'height_ratios': [3, 0.5] + [3, 0.5] * num_rows})

    # Show test image in a separate row
    axes[0, 1].imshow(test_image)  # Center test image
    axes[0, 1].set_title(f"Test Image\n(NSD ID: {test_nsd_id})", fontsize=12)
    axes[0, 1].axis('off')

    # Show test caption centered below it
    wrapped_test_caption = "\n".join(textwrap.wrap(test_caption, width=40))
    axes[1, 1].text(0.5, 0.5, wrapped_test_caption, fontsize=10, color='blue', ha='center', va='center')
    axes[1, 1].axis('off')

    # Hide extra test row axes
    for i in range(images_per_row):
        if i != 1:
            axes[0, i].axis('off')
            axes[1, i].axis('off')

    # Display top matches with captions
    for i in range(n):
        row_idx = (i // images_per_row) * 2 + 2  # Offset due to test image row
        col_idx = (i % images_per_row)

        train_idx = top_train_indices[test_idx][i]
        train_path = train_info[train_idx]['path']
        train_nsd_id = train_info[train_idx]['nsd_id']
        similarity = top_similarities[test_idx][i]

        train_image = Image.open(train_path)
        train_captions = get_coco_captions(train_nsd_id, stim_info)
        best_caption = train_captions[0] if train_captions else "No caption available"

        axes[row_idx, col_idx].imshow(train_image)
        axes[row_idx, col_idx].set_title(f"Match {i+1}\nSim: {similarity:.4f}", fontsize=12)
        axes[row_idx, col_idx].axis('off')

        # Wrap caption for better readability
        wrapped_caption = "\n".join(textwrap.wrap(best_caption, width=40))
        axes[row_idx + 1, col_idx].text(0.5, 0.5, wrapped_caption, fontsize=10, color='green', ha='center', va='center')
        axes[row_idx + 1, col_idx].axis('off')

    # Hide any unused axes
    for i in range(n, images_per_row * num_rows):
        row_idx = (i // images_per_row) * 2 + 2
        col_idx = i % images_per_row
        axes[row_idx, col_idx].axis('off')
        axes[row_idx + 1, col_idx].axis('off')

    plt.tight_layout()
    plt.show()
```

ROI-BASED FEATURE IMPORTANCE

Using predefined brain surface maps, we extract fMRI signals corresponding to specific **visual and cognitive brain regions**.We train Ridge Regression models for each ROI separately to determine **which brain regions contribute most to decoding accuracy**. ROIs are ranked based on their **Pearson correlation scores**, identifying key areas involved in visual

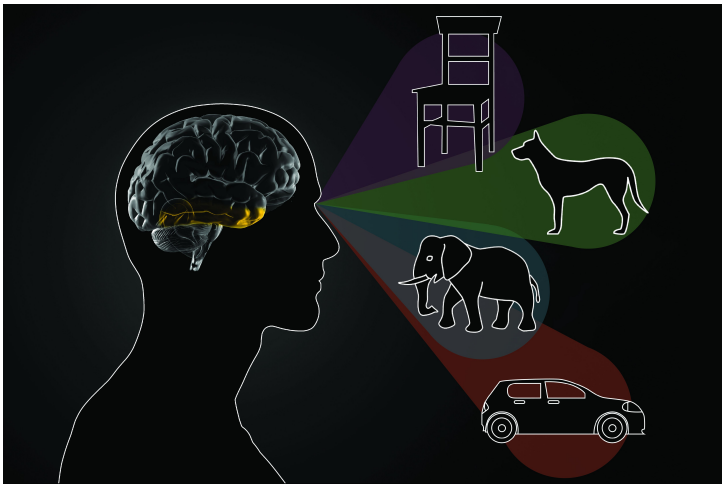
perception.

```python
roi_mapping_files = ['mapping_prf-visualrois.npy', 'mapping_floc-bodies.npy',
    'mapping_floc-faces.npy', 'mapping_floc-places.npy',
    'mapping_floc-words.npy', 'mapping_streams.npy']
roi_name_maps = []
for r in roi_mapping_files:
    roi_name_maps.append(np.load(os.path.join(ROI_PATH, '', r),
        allow_pickle=True).item())

# Load the ROI brain surface maps
lh_challenge_roi_files = ['lh.prf-visualrois_challenge_space.npy',
    'lh.floc-bodies_challenge_space.npy', 'lh.floc-faces_challenge_space.npy',
    'lh.floc-places_challenge_space.npy', 'lh.floc-words_challenge_space.npy',
    'lh.streams_challenge_space.npy']
rh_challenge_roi_files = ['rh.prf-visualrois_challenge_space.npy',
    'rh.floc-bodies_challenge_space.npy', 'rh.floc-faces_challenge_space.npy',
    'rh.floc-places_challenge_space.npy', 'rh.floc-words_challenge_space.npy',
    'rh.streams_challenge_space.npy']
lh_challenge_rois = []
rh_challenge_rois = []
for r in range(len(lh_challenge_roi_files)):
    lh_challenge_rois.append(np.load(os.path.join(ROI_PATH, '',
        lh_challenge_roi_files[r])))
    rh_challenge_rois.append(np.load(os.path.join(ROI_PATH, '',
        rh_challenge_roi_files[r])))
```

✓ 0.0s

```python
def load_fmri_data_by_roi():
    """
    Load fMRI data and filter voxels based on predefined ROIs.
    """
    print("Loading fMRI data...")
    lh_fmri = X_train_lh
    rh_fmri = X_train_rh

    # Dictionary to store fMRI data per ROI
    lh_roi_data = {}
    rh_roi_data = {}

    for i, roi_name_map in enumerate(roi_name_maps):
        for roi_label, roi_name in roi_name_map.items():
            if roi_label == 0:
                continue  # Skip background

            # Get voxel indices for the current ROI
            lh_roi_idx = np.where(lh_challenge_rois[i] == roi_label)[0]
            rh_roi_idx = np.where(rh_challenge_rois[i] == roi_label)[0]
```

FUNCTIONAL BRAIN AREA ANALYSIS



Grouping ROIs by Function

To gain deeper insights, we categorize ROIs into **functional groups**, such as:

- **Early Visual Cortex (V1-V4)**
- **Body-Selective Regions (EBA, FBA)**
- **Face-Selective Regions (FFA, OFA)**
- **Place-Selective Regions (PPA, OPA, RSC)**
- **Word-Selective Regions (VWFA, OWFA)**

Hemisphere Comparison

We compare **left vs. right hemisphere** performance to identify asymmetries in visual processing.

```python
def evaluate_roi_models(roi_models, lh_roi_data, rh_roi_data, y_test):
    """
    Evaluate Ridge Regression models for each ROI using Pearson correlation.
    """
    roi_scores = {}

    for roi_name, model in roi_models.items():
        if roi_name.startswith("lh_"):
            roi_key = roi_name.replace("lh_", "")
            roi_data = lh_roi_data.get(roi_key, None)
        else:
            roi_key = roi_name.replace("rh_", "")
            roi_data = rh_roi_data.get(roi_key, None)

        # Ensure ROI data exists and has valid dimensions
        if roi_data is None or roi_data.shape[1] == 0:
            continue

        # Predict embeddings
        y_pred = model.predict(roi_data)

        # Compute Pearson correlation
        pearson_corrs = np.array([
            pearsonr(y_pred[i], y_test[i])[0] if not np.isnan(pearsonr(y_pred[i], y_test[i])[0]) else 0
            for i in range(len(y_test))
        ])
        avg_corr = np.mean(pearson_corrs)

        roi_scores[roi_name] = avg_corr
        print(f"Avg Pearson Correlation for {roi_name}: {avg_corr:.4f}")

    return roi_scores
```

✓ 0.0s

```python
roi_scores = evaluate_roi_models(roi_models, lh_roi_data, rh_roi_data, y_test)
```

✓ 48.8s

```
Avg Pearson Correlation for lh_V1v: 0.6717
Avg Pearson Correlation for lh_V1d: 0.6733
Avg Pearson Correlation for lh_V2v: 0.6678
Avg Pearson Correlation for lh_V2d: 0.6746
Avg Pearson Correlation for lh_V3v: 0.6658
Avg Pearson Correlation for lh_V3d: 0.6692
Avg Pearson Correlation for lh_hV4: 0.6620
```

Whole-Brain Voxel-Wise Analysis

Instead of training separate models per ROI, we extract **voxel-wise importance** directly from the Ridge model weights, identifying **which individual voxels contribute most** to decoding accuracy.

By mapping high-importance voxels back to their corresponding ROIs, we obtain a global ranking of the **most influential brain regions** for image decoding.

```python
import numpy as np

def compute_voxel_importance(model):
    """
    Compute voxel importance scores from Ridge regression model weights.

    Args:
        model (sklearn Ridge object): Trained Ridge regression model.

    Returns:
        importance (np.array): Importance scores for each voxel.
    """
    W = model.coef_  # Shape: (embedding_dim, num_voxels)
    importance = np.linalg.norm(W, axis=0)  # L2 norm per voxel
    return importance

def map_voxel_importance_to_roi(importance, lh_challenge_rois, rh_challenge_rois, roi_name_maps):
    """
    Map voxel importance scores to their respective ROIs.

    Args:
        importance (tuple): (lh_importance, rh_importance) - importance scores for LH and RH.
        lh_challenge_rois (list of np.array): LH voxel maps for each ROI category.
        rh_challenge_rois (list of np.array): RH voxel maps for each ROI category.
        roi_name_maps (list of dict): ROI label-to-name mappings.

    Returns:
        roi_importance (dict): Aggregated importance scores per ROI.
    """
    lh_importance, rh_importance = importance
    roi_importance = {}

    for i, roi_name_map in enumerate(roi_name_maps):
        for roi_label, roi_name in roi_name_map.items():
            if roi_label == 0:
                continue  # Skip background

            # Get voxel indices for the current ROI
            lh_voxels = np.where(lh_challenge_rois[i] == roi_label)[0]
            rh_voxels = np.where(rh_challenge_rois[i] == roi_label)[0]

            # Sum voxel importance within each ROI
            roi_importance[f"lh_{roi_name}"] = np.sum(lh_importance[lh_voxels])
            roi_importance[f"rh_{roi_name}"] = np.sum(rh_importance[rh_voxels])

    return roi_importance

# Compute voxel importance for LH & RH
lh_importance = compute_voxel_importance(model_lh)  # Trained whole-brain model for LH
rh_importance = compute_voxel_importance(model_rh)  # Trained whole-brain model for RH
```

OUTPUTS / RESULTS

- The shape of the input image embeddings is **(8779, 512)**, with each of the 8779 images having a 512-dimensional embedding. For fMRI data, the shape is **(8779, 19004)** for the left hemisphere (LH) and **(8779, 20544)** for the right hemisphere (RH). This data is then split into train and test sets and used as input for the regression model by creating a dataset.

```
(X_train_lh, X_train_rh, y_train, train_info,
    X_test_lh, X_test_rh, y_test, test_info,
    stim_info) = prepare_dataset()
```

[39]

```
Found 8779 images
Extracting image features: 100%|██████████| 549/549 [04:05<00:00,  2.24it/s]
Shape of image features: (8779, 512)
Image features shape: (8779, 512)
Loading fMRI data...
Left hemisphere data shape: (8779, 19004)
Right hemisphere data shape: (8779, 20544)
```

- We can see that when considering all voxels in the linear regression model, the **right hemisphere performs slightly better** than the left hemisphere, though the difference is not statistically significant. The **correlation and similarity values** are **reasonably good**, with both **Pearson and Cosine similarity exceeding 0.5**. Note that these are **average values across all images**.

```
y_pred_lh, pearson_lh, cosine_lh = evaluate_model(model_lh, test_loader_lh, y_train,  "Left")
y_pred_rh, pearson_rh, cosine_rh = evaluate_model(model_rh, test_loader_rh, y_train,  "Right")
```

[19] ✓ 1.7s

```
Evaluating Ridge regression model for Left hemisphere...
Average Pearson correlation for Left: 0.5769
Average Cosine Similarity for Left: 0.6853
Evaluating Ridge regression model for Right hemisphere...
Average Pearson correlation for Right: 0.5827
Average Cosine Similarity for Right: 0.6867
```

- Now, we have computed the **cosine similarity** between the predicted embeddings and all embeddings from the training set. We have stored **only the top 5 matching indices** along with their corresponding **similarity values**. As seen in the output, the shape is **(1756, 5)**—meaning **1756 images**, each with their **top 5 matching indices** and similarity scores.

```python
print("Shape of top indices:", top_indices.shape)
print("Shape of top similarities:", top_similarities.shape)

print("0th Image (Sorted by Similarity):", top_indices[0])
print("0th Similarity (Sorted):", top_similarities[0])
```

✓ 0.0s

```
Shape of top indices: (1756, 5)
Shape of top similarities: (1756, 5)
0th Image (Sorted by Similarity): [4313 2214 4338 2943 2860]
0th Similarity (Sorted): [0.7434315  0.7403176  0.7311648  0.7309804  0.72946155]
```

- Here are some of the **output visualizations** showcasing our results. **Note that for the test image, we only used the fMRI input**—the pasted image and caption are included **only for comparison purposes** to ensure the results are accurate. These were **not used** in the actual process.

Test Image
(NSD ID: 4397)

A woman in stockings looks at herself in the mirror on the bed.

Match 1
Sim: 0.7434

A woman lays on a bed with a cat and a dog with a restraining collar.

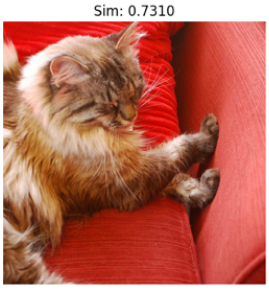
Match 2
Sim: 0.7403

This is an image of a cat on a chair.

Match 3
Sim: 0.7312

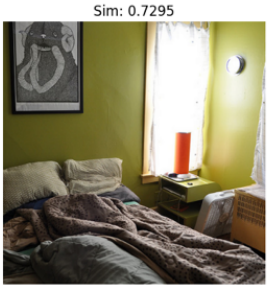
A little boy lays in bed covered by a blanket.

Match 4
Sim: 0.7310



A cat playing with a red couch, with its eyes closed.

Match 5
Sim: 0.7295



A bed and pillows in a small room.

Test Image
(NSD ID: 69722)

A male surfer dressed in black riding a wave.

Match 1
Sim: 0.8522

A shirtless man riding a wave on a white surfboard.

Match 2
Sim: 0.8450

a water skier in a wet suit is doing a trick

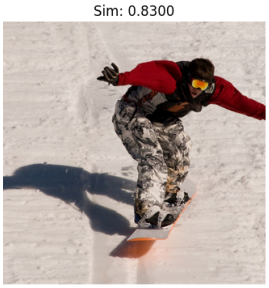
Match 3
Sim: 0.8334

A man surfing on a big wave bare chested.

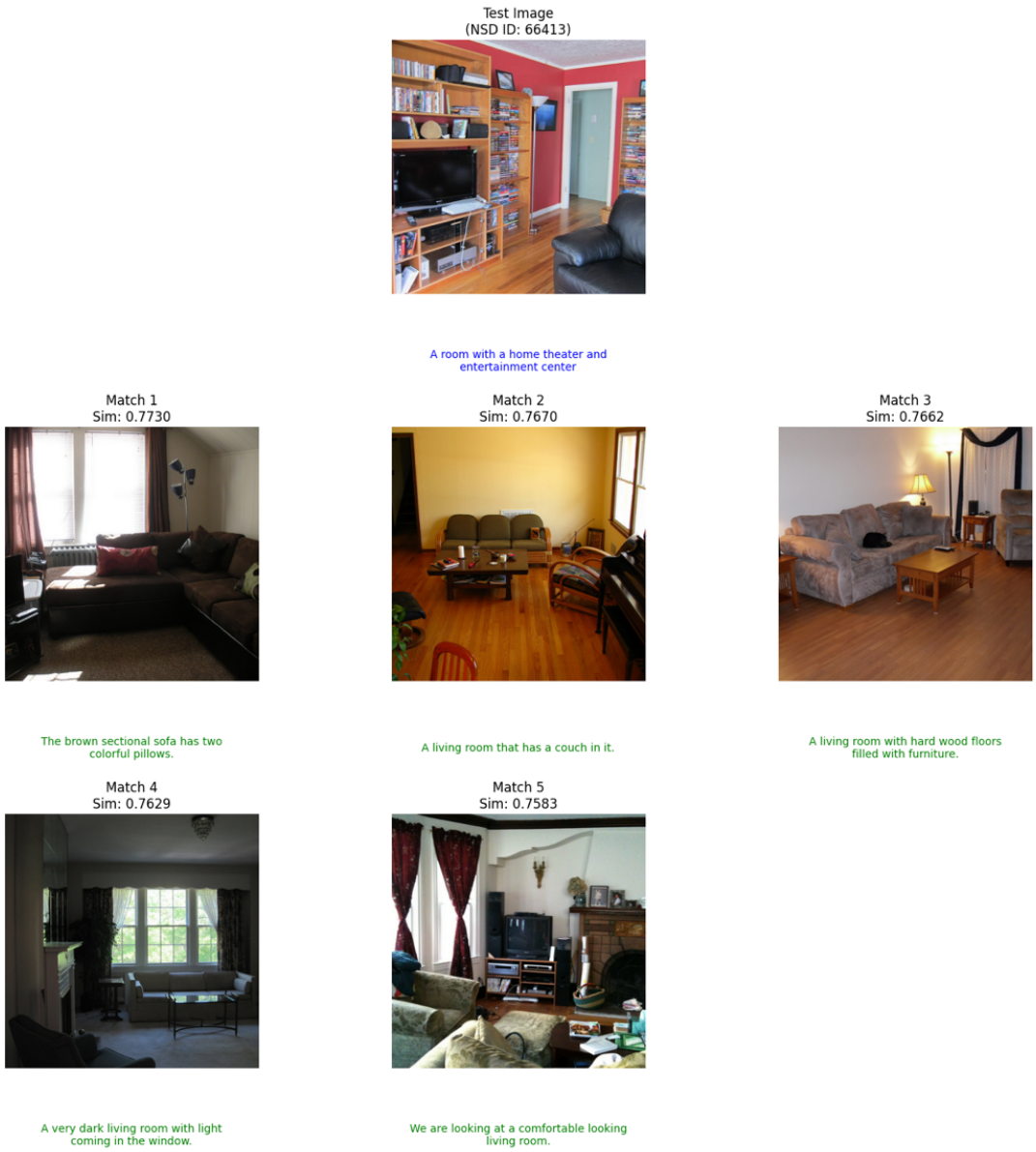
Match 4
Sim: 0.8311

A shirtless man riding a white surfboard on wave.

Match 5
Sim: 0.8300



A man riding a snowboard down a snow covered slope.



- After extracting the voxels corresponding to each **Region of Interest (ROI)**, we trained **separate regression models**using these voxel features and image embeddings. We then computed the **Pearson correlation coefficient** for each model to compare the contribution of different ROIs to decoding accuracy. The assumption here is that **Pearson correlation provides a clear measure of performance** in mapping fMRI responses to image features.

  Top 5 ROIs Contributing to Decoding:

  1. **lh_VWFA-2 (0.6852)** – Key for visual word recognition, aiding

structured visual processing.

2. **rh_mfs-words (0.6784)** – Supports word form recognition, contributing to decoding accuracy.

3. **lh_V2d (0.6746)** – Early visual area refining spatial and feature-based representations.

These ROIs are **more specialized in structured visual processing**, making them more effective for decoding.

Bottom 5 ROIs with Lowest Contribution:

1. **rh_EBA (0.5995)** – More involved in body perception than general visual decoding.

2. **lh_early (0.5958)** – Basic visual feature extraction, less useful for complex decoding.

3. **rh_early (0.5954)** – Similar to lh_early, contributing minimally.

Conclusion:

Higher-order visual and language-related areas (VWFA, fusiform) contribute **more effectively** to decoding, while early visual and body-related regions play a **less significant role** due to their specialized functions.

```python
# Sort ROIs by Pearson correlation (descending)
sorted_rois = sorted(roi_scores.items(), key=lambda x: x[1], reverse=True)

# Get top 5 ROIs
top_5 = sorted_rois[:5]
print("Top 5 ROIs contributing to decoding:")
for roi, score in top_5:
    print(f"{roi}: {score:.4f}")
```

[36] ✓ 0.0s

```
Top 5 ROIs contributing to decoding:
lh_VWFA-2: 0.6852
rh_mfs-words: 0.6784
lh_V2d: 0.6746
lh_mfs-words: 0.6739
rh_V1d: 0.6737
```

```python
# Get bottom 5 ROIs
bottom_5 = sorted_rois[-5:]
print("\n Bottom 5 ROIs with lowest contribution:")
for roi, score in bottom_5:
    print(f"{roi}: {score:.4f}")
```

[37] ✓ 0.0s

```
 Bottom 5 ROIs with lowest contribution:
rh_EBA: 0.5995
lh_early: 0.5958
rh_early: 0.5954
rh_ventral: 0.5911
lh_ventral: 0.5900
```

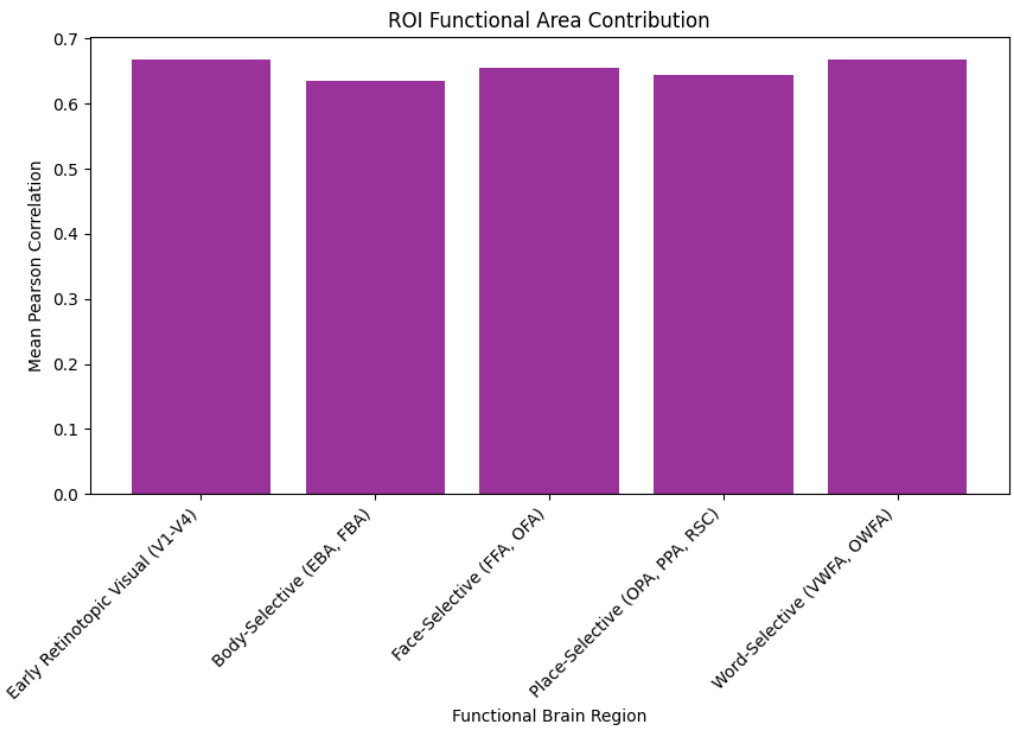
- To further understand ROI contributions, we grouped them into **functional categories** and analyzed their average **Pearson correlation coefficients**. This helps in identifying which **types of brain regions** are more effective in decoding.

  Early **visual processing (V1-V4) and word-selective regions (VWFA, OWFA)** contribute the most, highlighting the importance of structured visual representations. **Body and place-selective regions** play a smaller role, aligning with their specialized functions.

```python
def compute_functional_group_scores(roi_scores, roi_groups):
    """
    Compute the mean Pearson correlation for each functional brain region.
    """
    group_scores = {}

    for group, rois in roi_groups.items():
        group_values = [roi_scores[roi] for roi in roi_scores if any(roi.endswith(r) for r in rois)]
        group_scores[group] = np.mean(group_values) if group_values else 0

    return group_scores

# Compute functional group-wise scores
group_scores = compute_functional_group_scores(roi_scores, roi_groups)

# Print results
print("\n Functional Area Analysis:")
for group, avg_score in group_scores.items():
    print(f"{group}: {avg_score:.4f}")
```

[40] ✓ 0.0s

```
 Functional Area Analysis:
Early Retinotopic Visual (V1-V4): 0.6685
Body-Selective (EBA, FBA): 0.6353
Face-Selective (FFA, OFA): 0.6553
Place-Selective (OPA, PPA, RSC): 0.6443
Word-Selective (VWFA, OWFA): 0.6673
```



- We computed **voxel importance scores** using the **L2 norm** of Ridge

regression model weights, identifying the **most influential ROIs** for decoding. The top-performing ROIs include **early visual (V1/V2), ventral stream, and EBA regions**, which align with previous findings but differ in ranking and magnitude.

Reason for Differences

1. **Whole-Brain Model vs. Individual ROI Models**
   - Here, we directly used **model coefficients** from a **single whole-brain regression model**, whereas earlier results were obtained by training **separate models for each ROI**.
   - In the whole-brain model, weights are optimized **collectively**, leading to competition between regions, whereas independent models focus on local ROI-level performance.
2. **Normalization & Scaling Effects**
   - Using **L2 norm** emphasizes **absolute weight magnitude**, which might **overrepresent** regions with more voxels.
   - The previous Pearson correlation results directly assessed **model performance**, which is more **task-specific** rather than purely based on weight magnitude.
3. **Task-Specific Contributions**
   - Early visual areas (V1/V2) and **ventral regions** consistently rank high, suggesting that **low-level visual processing and object recognition** are fundamental for decoding.

Conclusion

While both analyses highlight the **importance of early visual and ventral stream areas**, using **model coefficients (L2 norm)** gives a different perspective compared to **direct performance evaluation** via correlation. The whole-brain model reflects **global competition between ROIs**, while individual models reveal **localized contributions** more clearly.

```python
# Print ROI importance ranking
sorted_importance = sorted(roi_importance.items(), key=lambda x: x[1], reverse=True
print("Top ROIs by Importance:")
for roi, score in sorted_importance[:10]:  # Show top 10 ROIs
    print(f"{roi}: {score:.4f}")
```

[24] ✓ 0.0s

```
Top ROIs by Importance:
lh_early: 69.9030
rh_early: 69.4260
lh_ventral: 60.2971
rh_ventral: 57.9063
rh_EBA: 52.2973
lh_EBA: 46.2679
lh_lateral: 42.4265
rh_lateral: 40.8674
lh_parietal: 28.8307
rh_parietal: 27.4907
```

- **Pearson-based evaluation**: Early visual areas (V1/V2), VWFA, and mfs-words perform well, as these regions are known to be crucial for visual and word-related processing.
- **Weight-based importance (L2 norm of coefficients)**: Early visual areas dominate, but EBA and ventral regions rank higher, likely due to **larger voxel counts and stronger absolute weight magnitudes** rather than direct decoding performance.

Thus, **Pearson correlation highlights functional decoding ability**, while **weight-based analysis emphasizes overall influence in the regression model**.