Appendix C: Google Colab Setup for NeuroAI

C.1 Getting Started with Google Colab

What is Google Colab?

Google Colaboratory (Colab) is a free cloud-based Jupyter notebook environment that requires no setup and runs entirely in the cloud. It's particularly useful for machine learning and data analysis tasks in the NeuroAI domain because:

- **Zero Configuration**: Run Python code through your browser with no installation required
- **Free GPU/TPU Access**: Accelerate neural network training with free GPU (NVIDIA K80, T4, P100, or V100) and TPU resources
- **Easy Sharing**: Collaborate with colleagues through Google Drive integration
- **Pre-installed Libraries**: Comes with many ML libraries (TensorFlow, PyTorch, scikit-learn) pre-installed

Basic Setup

1. **Accessing Colab**:
    - Go to https://colab.research.google.com/
    - Sign in with your Google account
    - Create a new notebook or open an existing one
2. **Notebook Interface**:
    - **Menu Bar**: File operations, runtime management, and help resources
    - **Toolbar**: Common operations like adding cells and running code
    - **Cells**: Individual code or text blocks (similar to Jupyter)

3. **Cell Types**:
    - **Code Cells**: For Python code execution
    - **Text Cells**: For markdown documentation
    - **Output Cells**: Display results of executed code
4. **Runtime Management**:
    - Select **Runtime > Change runtime type** to choose hardware accelerator (None, GPU, or TPU)
    - Runtimes automatically disconnect after 90 minutes of inactivity or 12 hours of total usage

```python
# Check what type of hardware you're using
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
print("GPU Available:", tf.config.list_physical_devices('GPU'))
print("TPU Available:", tf.config.list_physical_devices('TPU'))
```

C.2 Setting Up NeuroAI-Handbook Environment

Cloning the NeuroAI-Handbook Repository

```
# Clone the repository
!git clone https://github.com/yourusername/NeuroAI-Handbook.git

# Change to the repository directory
%cd NeuroAI-Handbook

# Install required packages
!pip install -r book/requirements.txt
```

Required Package Installation

For the exercises in this handbook, you'll need various neuroscience and ML libraries:

```
# Install essential packages for NeuroAI
!pip install -q torch torchvision tensorflow matplotlib numpy pandas scipy scikit

# Install neuroscience-specific packages
!pip install -q nibabel nilearn mne seaborn allensdk neurom pynwb

# Install JupyterBook for building the handbook (optional)
!pip install -q jupyter-book

# Verify key installations
import sys
import torch
import tensorflow as tf
import mne
import nibabel as nib
import allensdk

print(f"Python version: {sys.version}")
print(f"PyTorch version: {torch.__version__}")
print(f"TensorFlow version: {tf.__version__}")
print(f"MNE version: {mne.__version__}")
print(f"NiBabel version: {nib.__version__}")
print(f"AllenSDK version: {allensdk.__version__}")
```

GPU Configuration and Management

```
# Check GPU details
!nvidia-smi

# Configure TensorFlow for GPU memory growth (prevents memory errors)
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        print(f"Memory growth enabled for {len(gpus)} GPUs")
    except RuntimeError as e:
        print(f"Error setting memory growth: {e}")

# Configure PyTorch to use GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"PyTorch is using: {device}")
```

Directory Structure Setup

```
# Create standard directory structure for NeuroAI projects
!mkdir -p ./data/raw ./data/processed ./models ./results ./figures

# Define and export path variables
import os

# Define key paths
RAW_DATA_PATH = './data/raw'
PROCESSED_DATA_PATH = './data/processed'
MODEL_PATH = './models'
RESULTS_PATH = './results'
FIGURES_PATH = './figures'

# Create a function to create needed directories
def ensure_directories(paths):
    """Create directories if they don't exist."""
    for path in paths:
        os.makedirs(path, exist_ok=True)
        print(f"Directory available: {path}")

ensure_directories([RAW_DATA_PATH, PROCESSED_DATA_PATH, MODEL_PATH, RESULTS_PATH,
```

C.3 Data Management in Colab

Mounting Google Drive

For persistent storage, mount your Google Drive:

```
from google.colab import drive
drive.mount('/content/drive')

# Create a dedicated folder for NeuroAI work
DRIVE_PATH = '/content/drive/MyDrive/NeuroAI-Handbook'
!mkdir -p $DRIVE_PATH

# Create symbolic links for convenient access
!ln -s $DRIVE_PATH/data ./drive_data
!ln -s $DRIVE_PATH/models ./drive_models

print(f"Google Drive mounted at: {DRIVE_PATH}")
```

Data Transfer Methods

From Google Drive to Colab VM

```
# Copy datasets from Drive to Colab (faster processing)
!cp -r /content/drive/MyDrive/NeuroAI-Handbook/data/raw/* ./data/raw/
```

From External Sources

```
# Method 1: Download using wget
!wget -P $RAW_DATA_PATH https://openneuro.org/crn/datasets/ds003031/snapshots/2.0

# Method 2: From GitHub
!curl -L -o $RAW_DATA_PATH/sample_data.zip https://github.com/username/repo/raw/m
!unzip -q $RAW_DATA_PATH/sample_data.zip -d $RAW_DATA_PATH

# Method 3: From Kaggle (requires API key setup)
!pip install -q kaggle
!mkdir -p ~/.kaggle

# Create Kaggle API token (replace with your token details)
# Get your token from https://www.kaggle.com/account
!echo '{"username":"YOUR_USERNAME","key":"YOUR_API_KEY"}' > ~/.kaggle/kaggle.json
!chmod 600 ~/.kaggle/kaggle.json

# Download dataset from Kaggle
!kaggle datasets download -d codingdisciple/brain-mri-dataset -p $RAW_DATA_PATH
!unzip -q $RAW_DATA_PATH/brain-mri-dataset.zip -d $RAW_DATA_PATH
```

Saving Results Back to Drive

```python
# Save processed data and models to Google Drive for persistence
def save_to_drive(source_path, drive_destination):
    """Save files to Google Drive with logging."""
    import shutil
    import os

    # Create destination directory if it doesn't exist
    os.makedirs(os.path.dirname(drive_destination), exist_ok=True)

    # Copy data
    try:
        if os.path.isdir(source_path):
            shutil.copytree(source_path, drive_destination, dirs_exist_ok=True)
        else:
            shutil.copy2(source_path, drive_destination)
        return True
    except Exception as e:
        print(f"Error saving to Drive: {e}")
        return False

# Example usage
model_path = './models/trained_cnn.h5'
drive_model_path = f'{DRIVE_PATH}/models/trained_cnn.h5'

if os.path.exists(model_path):
    if save_to_drive(model_path, drive_model_path):
        print(f"Model saved to Drive: {drive_model_path}")
```

C.4 Optimizing Colab for NeuroAI Computations

Memory Management

```python
# Check and monitor memory usage
!pip install -q psutil
import psutil
import gc

def print_memory_usage():
    """Print current memory usage of the Colab instance."""
    mem = psutil.virtual_memory()
    print(f"MEMORY STATUS:")
    print(f"  Total:     {mem.total / 1e9:.1f} GB")
    print(f"  Available: {mem.available / 1e9:.1f} GB")
    print(f"  Used:      {mem.used / 1e9:.1f} GB ({mem.percent}%)")
    print(f"  Free:      {mem.free / 1e9:.1f} GB")

print_memory_usage()

# Memory optimization function
def optimize_memory():
    """Perform garbage collection and clear GPU memory if available."""
    # Clear memory
    gc.collect()

    # Clear PyTorch cache if using GPU
    import torch
    if torch.cuda.is_available():
        torch.cuda.empty_cache()
        print("PyTorch GPU cache cleared")

    # Clear TensorFlow GPU memory
    import tensorflow as tf
    if tf.config.list_physical_devices('GPU'):
        tf.keras.backend.clear_session()
        print("TensorFlow session cleared")

    # Report memory after optimization
    print("\nMemory after optimization:")
    print_memory_usage()

# Use after heavy computations
# optimize_memory()
```

Session Management

```python
# Function to prevent Colab from disconnecting (prevents idle timeouts)
from IPython.display import display, Javascript
import time

def keep_alive(delay_minutes=55):
    """
    Prevents Colab from disconnecting due to inactivity.
    Note: Use responsibly and only when needed for long computations.
    """
    delay_seconds = delay_minutes * 60
    display(Javascript('''
        function click_connect(){
            console.log("Clicking connect button");
            document.querySelector("colab-connect-button").click()
        }
        setInterval(click_connect, ''' + str(delay_seconds * 1000) + ''');
    '''))
    print(f"Keep-alive service started. Will refresh every {delay_minutes} minute

# Uncomment to use (be considerate of resources)
# keep_alive(delay_minutes=55)
```

Visualization Configuration for Neuroscience

```python
# Standardized visualization setup for NeuroAI plots
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

def setup_neuro_visualization(style='whitegrid', context='talk', palette='viridis
    """Set up standardized visualization environment for neuroscience."""
    sns.set_theme(style=style, context=context, palette=palette)

    # Configure matplotlib
    plt.rcParams['figure.figsize'] = (12, 8)
    plt.rcParams['figure.dpi'] = 100
    plt.rcParams['savefig.dpi'] = 300
    plt.rcParams['font.family'] = 'sans-serif'
    plt.rcParams['axes.titlesize'] = 18
    plt.rcParams['axes.labelsize'] = 14
    plt.rcParams['xtick.labelsize'] = 12
    plt.rcParams['ytick.labelsize'] = 12

    # Scientific notation settings
    plt.rcParams['axes.formatter.use_mathtext'] = True
    plt.rcParams['font.sans-serif'] = ['Arial', 'Helvetica', 'DejaVu Sans']

    # Default colormap options
    cmap_options = {
        'brain': 'gray',            # Structural brain images
        'activation': 'hot',        # Activation maps
        'connectivity': 'coolwarm', # Connectivity matrices
        'category': 'tab10',        # Categorical data
        'continuous': 'viridis'     # Continuous data
    }

    return cmap_options

# Set up visualization
cmap_options = setup_neuro_visualization()
print("Visualization environment configured for NeuroAI")
```

C.5 Working with Specific NeuroAI Datasets

Example: Loading and Processing MRI Data

```
# Install necessary libraries
!pip install -q nibabel nilearn matplotlib

# Sample code to download a test MRI dataset
!mkdir -p $RAW_DATA_PATH/mri_sample
!curl -L -o $RAW_DATA_PATH/mri_sample/brain.nii.gz https://github.com/nilearn/nil

# Load and visualize MRI data
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
from nilearn import plotting

def load_and_view_mri(filepath):
    """Load and display a 3D brain image."""
    # Load the NIfTI file
    img = nib.load(filepath)
    print(f"Image shape: {img.shape}")
    print(f"Image affine:\n{img.affine}")

    # Extract data array
    data = img.get_fdata()

    # Plot using nilearn
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    # Calculate middle slices
    x_mid, y_mid, z_mid = np.array(data.shape) // 2

    # Display middle slices
    axes[0].imshow(data[x_mid, :, :].T, cmap='gray', origin='lower')
    axes[0].set_title(f'Sagittal (x={x_mid})')

    axes[1].imshow(data[:, y_mid, :].T, cmap='gray', origin='lower')
    axes[1].set_title(f'Coronal (y={y_mid})')

    axes[2].imshow(data[:, :, z_mid].T, cmap='gray', origin='lower')
    axes[2].set_title(f'Axial (z={z_mid})')

    plt.tight_layout()

    # Also show a nilearn 3D plot
    plotting.plot_anat(img, title="3D MRI Visualization")

    return img, data
```

```
# Load and view the sample MRI
mri_path = f"{RAW_DATA_PATH}/mri_sample/brain.nii.gz"
brain_img, brain_data = load_and_view_mri(mri_path)
```

Example: Working with EEG Data

```
# Install MNE for EEG/MEG data analysis
!pip install -q mne

# Download sample EEG data
!mkdir -p $RAW_DATA_PATH/eeg_sample
!curl -L -o $RAW_DATA_PATH/eeg_sample/sample_eeg.fif.gz https://mne.tools/stable/

# Process and visualize EEG data
import mne
import numpy as np
import matplotlib.pyplot as plt

def explore_eeg_data(filepath):
    """Load and explore EEG data using MNE-Python."""
    # Load the data
    raw = mne.io.read_raw_fif(filepath, preload=True)

    # Print basic information
    print(f"EEG data loaded: {filepath}")
    print(f"Number of channels: {len(raw.ch_names)}")
    print(f"Sampling frequency: {raw.info['sfreq']} Hz")
    print(f"Duration: {raw.times.max():.1f} seconds")

    # Basic preprocessing
    raw.filter(1, 40)  # Bandpass filter between 1-40 Hz

    # Plot the data
    raw.plot(duration=5, n_channels=10, scalings='auto')

    # Plot the power spectral density
    raw.plot_psd(fmax=50)

    # Plot channel locations
    raw.plot_sensors(show_names=True)

    return raw

# Load and explore the sample EEG data
eeg_path = f"{RAW_DATA_PATH}/eeg_sample/sample_eeg.fif.gz"
eeg_data = explore_eeg_data(eeg_path)
```

C.6 Collaboration and Sharing

Sharing Your Notebooks

```
# Generate a link to share your notebook
from google.colab import files
from IPython.display import HTML, display

def get_share_link():
    """Display instructions for sharing the current notebook."""
    display(HTML("""
    <div style="background-color:#f8f9fa; padding:12px; border-radius:5px; margin
        <h3 style="margin-top:0">Share this Notebook</h3>
        <p>To share this notebook:</p>
        <ol>
            <li>Click <b>File → Share</b> in the menu</li>
            <li>Adjust sharing settings (View only, Comment, or Edit)</li>
            <li>Copy the link and share with collaborators</li>
        </ol>
        <p>For version control, consider saving to GitHub:</p>
        <ol>
            <li>Click <b>File → Save a copy in GitHub</b></li>
            <li>Connect to your GitHub account if prompted</li>
            <li>Select repository and file path</li>
        </ol>
    </div>
    """))

# Show sharing instructions
get_share_link()
```

Exporting Your Work

```python
# Export notebook to different formats
from google.colab import files

def export_notebook(format_type='ipynb'):
    """
    Export the current notebook in the specified format.

    Parameters:
    -----------
    format_type : str
        Format to export ('ipynb', 'py', or 'html')
    """
    import os
    import json
    from IPython import get_ipython

    # Get the notebook filename
    notebook_path = '/content/drive/MyDrive/temp_notebook.ipynb'

    if format_type == 'ipynb':
        # Just download the notebook directly
        files.download(os.path.basename('/content/notebook.ipynb'))

    elif format_type == 'py':
        # Convert to Python script using nbconvert
        !pip install -q nbconvert
        !jupyter nbconvert --to python /content/notebook.ipynb
        py_filename = os.path.basename('/content/notebook.py')
        files.download(py_filename)

    elif format_type == 'html':
        # Convert to HTML using nbconvert
        !pip install -q nbconvert
        !jupyter nbconvert --to html /content/notebook.ipynb
        html_filename = os.path.basename('/content/notebook.html')
        files.download(html_filename)

    else:
        print(f"Unsupported format: {format_type}")
        print("Supported formats: 'ipynb', 'py', 'html'")

# Export as Python script
# export_notebook(format_type='py')
```

C.7 Troubleshooting Common Issues

GPU Memory Issues

If you encounter GPU memory errors:

```
# Check current GPU memory usage
!nvidia-smi

# Reset runtime environment
import tensorflow as tf
import torch
import gc

def reset_gpu_memory():
    """Reset GPU memory when you hit Out of Memory errors."""
    # Clear memory
    gc.collect()

    # Clear PyTorch cache
    if torch.cuda.is_available():
        torch.cuda.empty_cache()
        print("PyTorch CUDA memory cleared")

    # Reset TensorFlow session
    tf.keras.backend.clear_session()
    print("TensorFlow session reset")

    # Check GPU memory after clearing
    !nvidia-smi

# Call when you get OOM errors
reset_gpu_memory()
```

Connection Issues

To address connection issues:

```python
# Save your progress frequently
# 1. Auto-backup function for important variables
import pickle
import os
from datetime import datetime

def backup_variables(variables_dict, backup_dir='./backup'):
    """
    Save important variables to Google Drive as a backup.

    Parameters:
    -----------
    variables_dict : dict
        Dictionary of variable names and their values to backup
    backup_dir : str
        Directory to save backups
    """
    os.makedirs(backup_dir, exist_ok=True)

    # Generate timestamp
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    backup_file = f"{backup_dir}/backup_{timestamp}.pkl"

    # Save variables
    with open(backup_file, 'wb') as f:
        pickle.dump(variables_dict, f)

    print(f"Backup saved to {backup_file}")

    # Also save to Drive if mounted
    if os.path.exists('/content/drive'):
        drive_backup_dir = '/content/drive/MyDrive/NeuroAI-Handbook/backups'
        os.makedirs(drive_backup_dir, exist_ok=True)
        drive_backup_file = f"{drive_backup_dir}/backup_{timestamp}.pkl"

        with open(drive_backup_file, 'wb') as f:
            pickle.dump(variables_dict, f)

        print(f"Backup also saved to Google Drive: {drive_backup_file}")

    return backup_file

# Example usage
# important_vars = {
#     'trained_model': model,
#     'history': training_history,
#     'test_results': test_results
# }
# backup_path = backup_variables(important_vars)

# Function to restore from backup
def restore_from_backup(backup_file):
    """Restore variables from backup file."""
    with open(backup_file, 'rb') as f:
        variables = pickle.load(f)

    print(f"Restored variables from {backup_file}")
    print(f"Available variables: {list(variables.keys())}")

    return variables

# Example: Restore from most recent backup
# latest_backup = max(glob.glob('./backup/backup_*.pkl'), key=os.path.getctime)
# restored_vars = restore_from_backup(latest_backup)
```

C.8 Resources and References

Useful Links

- Google Colab Documentation
- Google Colab Pro Features
- Colab Keyboard Shortcuts

NeuroAI-Specific Resources

- Allen Brain Atlas
- Human Connectome Project
- NeuralEnsemble
- OpenNeuro
- NeuroImaging Tools & Resources Collaboratory

Citation

```
@book{neuroai_handbook_2024,
  title={NeuroAI Handbook: Bridging Neuroscience and Artificial Intelligence},
  author={NeuroAI Contributors},
  year={2024},
  publisher={Online Resource},
  url={https://github.com/yourusername/NeuroAI-Handbook}
}
```