Sample code to proceed :

```
# NeuroSentiment3: EEG + MRI from LEMON and Text from IMDB

# STEP 1: Load EEG, MRI from LEMON and Text from IMDB

lemon_eeg_dir = 'datasets/LEMON/EEG/'  # Replace with your actual path
lemon_mri_dir = 'datasets/LEMON/MRI/'  # Replace with your actual path

from datasets import load_dataset

# Load IMDB text data
imdb_dataset = load_dataset("imdb")
text_data = imdb_dataset['train']['text'][:1000]  # Subset for efficiency
text_labels = imdb_dataset['train']['label'][:1000]  # 0=neg, 1=pos

# STEP 2: Preprocessing EEG using MNE
import mne
import numpy as np
import os

def preprocess_eeg(file_path):
    raw = mne.io.read_raw_fif(file_path, preload=True)
    raw.filter(1., 40., fir_design='firwin')
    raw.set_eeg_reference('average', projection=True)
    epochs = mne.make_fixed_length_epochs(raw, duration=2.0, preload=True)
    return epochs.get_data()

# STEP 3: Preprocessing MRI using NiBabel
import nibabel as nib
from nilearn.image import resample_img

def preprocess_mri(mri_path):
    img = nib.load(mri_path)
    target_shape = (64, 64, 64)
    resampled = resample_img(img, target_affine=np.eye(3)*4, target_shape=target_shape)
    return resampled.get_fdata()

# STEP 4: Tokenize and Encode Text using BERT
from transformers import BertTokenizer, BertModel
import torch

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = BertModel.from_pretrained('bert-base-uncased')

def encode_text(text_list):
    inputs = tokenizer(text_list, return_tensors="pt", padding=True, truncation=True,
max_length=128)
    with torch.no_grad():
        outputs = bert_model(**inputs)
    return outputs.last_hidden_state.mean(dim=1)  # [B, 768]
```

```
# STEP 5: Build Modality Encoders
import torch.nn as nn

class EEGEncoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.lstm = nn.LSTM(input_size=64, hidden_size=128, batch_first=True)
    def forward(self, x):
        _, (hn, _) = self.lstm(x)
        return hn[-1]

class MRIEncoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv3d(1, 8, 3, padding=1), nn.ReLU(), nn.MaxPool3d(2),
            nn.Conv3d(8, 16, 3, padding=1), nn.ReLU(), nn.AdaptiveAvgPool3d(1)
        )
    def forward(self, x):
        x = x.unsqueeze(1)  # [B, 1, 64, 64, 64]
        x = self.conv(x)
        return x.view(x.size(0), -1)

# STEP 6: Fusion Model with Cross-Attention
class FusionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.eeg_encoder = EEGEncoder()
        self.mri_encoder = MRIEncoder()
        self.text_fc = nn.Linear(768, 128)
        self.attn = nn.MultiheadAttention(embed_dim=128, num_heads=4, batch_first=True)
        self.classifier = nn.Sequential(
            nn.Linear(384, 128), nn.ReLU(), nn.Linear(128, 3)  # 3 classes: pos, neutral, neg
        )

    def forward(self, eeg, mri, text):
        eeg_feat = self.eeg_encoder(eeg)  # [B, 128]
        mri_feat = self.mri_encoder(mri)  # [B, 128]
        text_feat = self.text_fc(text)    # [B, 128]

        eeg_attn, _ = self.attn(eeg_feat.unsqueeze(1), text_feat.unsqueeze(1), text_feat.unsqueeze(1))
        mri_attn, _ = self.attn(mri_feat.unsqueeze(1), text_feat.unsqueeze(1), text_feat.unsqueeze(1))

        combined = torch.cat([eeg_attn.squeeze(1), mri_attn.squeeze(1), text_feat], dim=1)
        return self.classifier(combined)

# STEP 7: Training and Evaluation

def train(model, data_loader, optimizer, loss_fn):
    model.train()
    for batch in data_loader:
        eeg, mri, text, labels = batch
```

```
        preds = model(eeg, mri, text)
        loss = loss_fn(preds, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

# STEP 8: Usage Example (pseudo-code)
# eeg_data = [preprocess_eeg(os.path.join(lemon_eeg_dir, f)) for f in eeg_file_list]
# mri_data = [preprocess_mri(os.path.join(lemon_mri_dir, f)) for f in mri_file_list]
# text_encoded = encode_text(text_data)
# model = FusionModel()
# train(model, data_loader, optimizer, loss_fn)

# Note: You need to implement a DataLoader that returns (eeg_tensor, mri_tensor, text_tensor,
label_tensor)
# where eeg_tensor.shape = [B, T, C], mri_tensor.shape = [B, 64, 64, 64], etc.
```