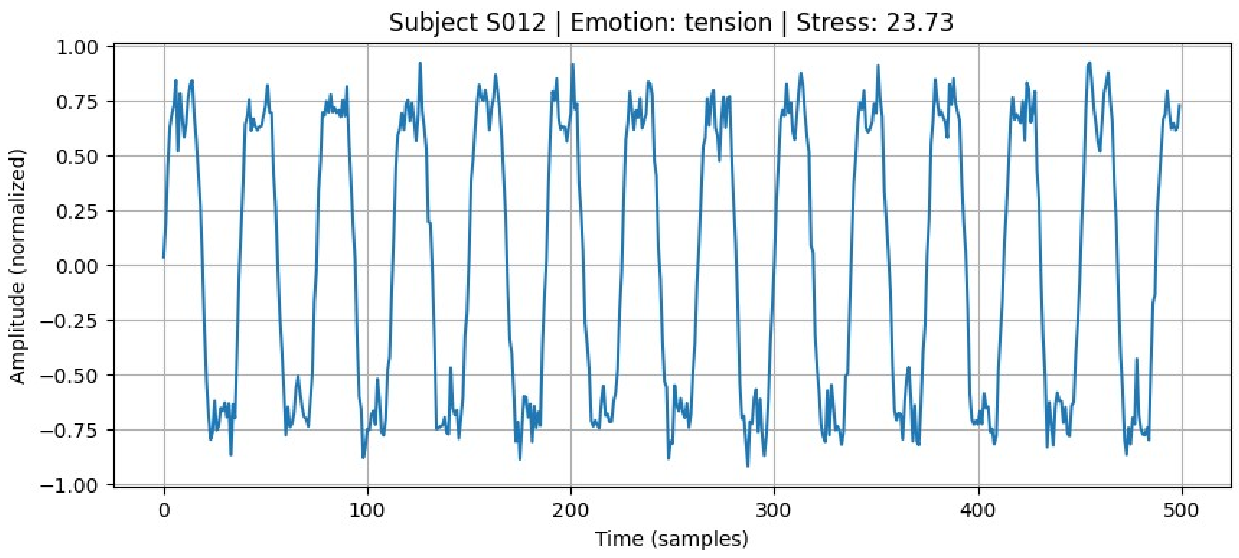
```
# Depression Severity Prediction using Deep Learning Techniques

# This is the step 1, in which I have synthetically generated the
dataset using python

import numpy as np
import pandas as pd
import os
from datetime import datetime
import matplotlib.pyplot as plt

# ============================================================
# □ 1. Generate synthetic PPG-like signals for each emotion
# ============================================================
def generate_ppg(emotion, length=4050, fs=45):
    """
    Generates a synthetic 1D PPG-like signal for a given emotion.
    """

    t = np.linspace(0, 90, length)  # 90 seconds total

    # Base heartbeat-like waveform
    base = np.sin(2 * np.pi * 1.2 * t) + 0.25 * np.sin(2 * np.pi * 3.6
* t)
    noise = np.random.normal(0, 0.05, length)

    if emotion == 'calm':
        signal = base + noise

    elif emotion == 'happiness':
        signal = 1.1 * base + np.random.normal(0, 0.07, length)

    elif emotion == 'sadness':
        signal = 0.8 * base + np.random.normal(0, 0.05, length)

    elif emotion == 'tension':
        signal = 0.9 * base + np.random.normal(0, 0.08, length)

    elif emotion == 'fear':
        signal = 1.2 * base + np.random.normal(0, 0.10, length)

    else:
        signal = base + noise  # fallback

    # Normalize between -1 and 1
    signal = signal / np.max(np.abs(signal))
    return signal

# ============================================================
# □ 2. Stress Score Mapping (Base Mean Values)
# ============================================================
# These mean values match the research paper
emotion_to_mean_score = {
    'calm': 10,
    'happiness': 15,
    'sadness': 20,
    'tension': 25,
    'fear': 30
}

# Add Gaussian variation to simulate real physiological data
# Step 1: wider stress variation (for realistic diversity)
def get_stress_score(emotion):
    base = emotion_to_mean_score[emotion]
    return float(np.random.normal(base, 5.0))  # ±5

# Step 2: robust stratified split
mean_stress['stress_level'] = pd.qcut(mean_stress['stress_score'],
q=2, labels=['low', 'high'])


# ============================================================
# □ 3. Output folders
# ============================================================
os.makedirs("data", exist_ok=True)
os.makedirs("metadata", exist_ok=True)

# ============================================================
# □ 4. Create dataset
# ============================================================
num_subjects = 50
emotions = list(emotion_to_mean_score.keys())
records = []

for sid in range(1, num_subjects + 1):

    gender = np.random.choice(['Male', 'Female'])
    age = np.random.randint(18, 25)

    for emotion in emotions:

        # Generate signal
        signal = generate_ppg(emotion)

        # Save as .npy
        file_name = f"S{sid:03d}_{emotion}.npy"
        file_path = os.path.join("data", file_name)
        np.save(file_path, signal)

        # Create floating stress score
        stress_score = get_stress_score(emotion)
        # Save metadata row
        records.append([
            f"S{sid:03d}", emotion, file_path, 45, 90.0, len(signal),
            stress_score, gender, age,
            datetime.now().strftime('%Y-%m-%d')
        ])

columns = [
    'subject_id', 'emotion', 'signal_file', 'sampling_rate',
    'duration_sec', 'signal_length', 'stress_score', 'gender',
    'age', 'record_date'
]

metadata_df = pd.DataFrame(records, columns=columns)
metadata_df.to_csv('metadata/stress_dataset_metadata.csv',
index=False)

print("□ Synthetic dataset successfully created!")
print(metadata_df.head())

□ Synthetic dataset successfully created!
  subject_id    emotion              signal_file  sampling_rate
duration_sec  \
0       S001       calm       data/S001_calm.npy             45
90.0
1       S001  happiness  data/S001_happiness.npy             45
90.0
2       S001    sadness    data/S001_sadness.npy             45
90.0
3       S001    tension    data/S001_tension.npy             45
90.0
4       S001       fear       data/S001_fear.npy             45
90.0

   signal_length  stress_score  gender  age record_date
0           4050      8.275506  Female   24  2025-11-12
1           4050     19.330332  Female   24  2025-11-12
2           4050     17.778289  Female   24  2025-11-12
3           4050     20.469921  Female   24  2025-11-12
4           4050     27.854377  Female   24  2025-11-12

# ============================================================
# □ 5. Quick visualization check
# ============================================================
meta = pd.read_csv('metadata/stress_dataset_metadata.csv')

sample = meta.sample(1).iloc[0]
signal = np.load(sample.signal_file)

plt.figure(figsize=(10,4))
plt.plot(signal[:500])
plt.title(f"Subject {sample.subject_id} | Emotion: {sample.emotion} |
Stress: {sample.stress_score:.2f}")
plt.xlabel("Time (samples)")
plt.ylabel("Amplitude (normalized)")
plt.grid(True)
plt.show()
```



```
print("\nEmotion counts:\n", meta['emotion'].value_counts())
print("\nMean Stress per Emotion:\n", meta.groupby('emotion')
['stress_score'].mean())


Emotion counts:
 emotion
calm         50
happiness    50
sadness      50
tension      50
fear         50
Name: count, dtype: int64

Mean Stress per Emotion:
 emotion
calm          9.320533
fear         29.341489
happiness    14.801943
sadness      20.570355
tension      24.732391
Name: stress_score, dtype: float64
```

```
# Now comes step 2 of my project : Preprocessing

# In this we'll handle three major preprocessing tasks that correspond
exactly to what the paper did:

# Wavelet denoising → remove noise and artifacts from the raw PPG

# PRV extraction → compute Pulse Rate Variability (time difference
between peaks)

# dPPG segmentation → extract discrete pulse waveforms (using valleys
as segment markers)

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pywt  # for wavelet transforms
from scipy.signal import find_peaks
from scipy.interpolate import interp1d

def wavelet_denoise(signal, wavelet='db8', level=3):
    """
    Denoises a 1D signal using wavelet thresholding.
    Arguments:
        signal : np.ndarray : input raw signal
        wavelet : str : wavelet type (Daubechies recommended)
        level : int : decomposition level
    Returns:
        np.ndarray : denoised signal
    """
    coeffs = pywt.wavedec(signal, wavelet, level=level)
    sigma = np.median(np.abs(coeffs[-1])) / 0.6745
    uthresh = sigma * np.sqrt(2 * np.log(len(signal)))

    coeffs[1:] = [pywt.threshold(c, value=uthresh, mode='soft') for c
in coeffs[1:]]
    denoised_signal = pywt.waverec(coeffs, wavelet)

    return denoised_signal[:len(signal)]

def extract_prv(signal, fs=45):
    """
    Extract Pulse Rate Variability (PRV) from a PPG signal.
    PRV = time difference between consecutive pulse peaks.
    """
    # Detect main peaks
```

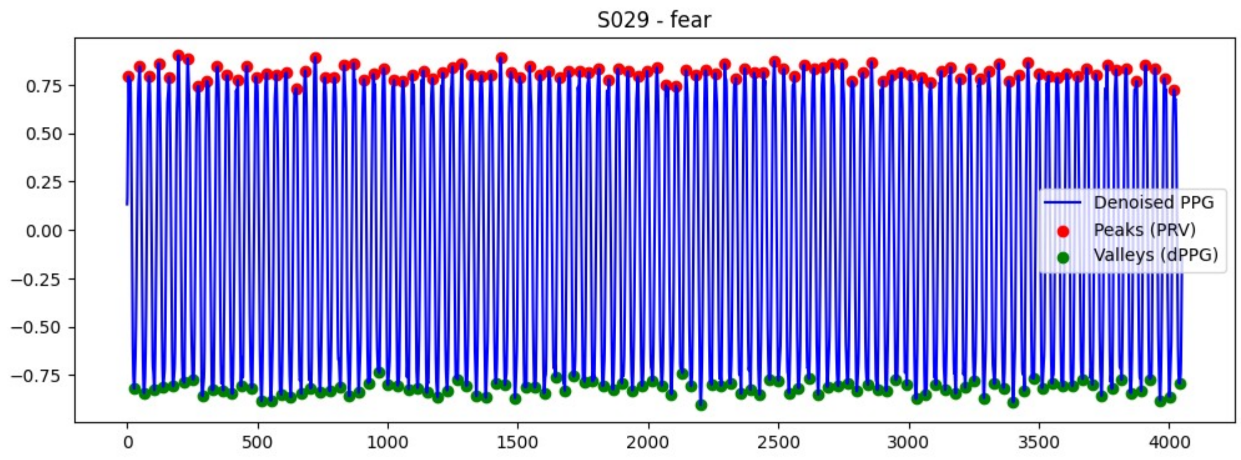
```python
    peaks, _ = find_peaks(signal, distance=fs*0.5)  # minimum 0.5s
apart

    # Convert peak indices to time (seconds)
    times = peaks / fs

    # Compute time difference between peaks
    prv = np.diff(times)

    return prv, peaks

def extract_dppg(signal, fs=45, interp_len=100):
    """
    Extract discrete pulse waves (dPPG) using valley detection.
    Each dPPG represents one heartbeat waveform.
    """
    # Detect valleys (inverted peaks)
    valleys, _ = find_peaks(-signal, distance=fs*0.5)

    dppg_segments = []

    for i in range(len(valleys) - 1):
        start = valleys[i]
        end = valleys[i + 1]
        seg = signal[start:end]

        # Skip too short or too long segments
        if len(seg) < 10:
            continue

        # Normalize length by interpolation
        x_old = np.linspace(0, 1, len(seg))
        x_new = np.linspace(0, 1, interp_len)
        f = interp1d(x_old, seg)
        seg_interp = f(x_new)

        dppg_segments.append(seg_interp)

    return np.array(dppg_segments), valleys

# Load metadata
meta = pd.read_csv('metadata/stress_dataset_metadata.csv')

# Create output folders
os.makedirs('processed/prv', exist_ok=True)
os.makedirs('processed/dppg', exist_ok=True)

processed_records = []

for idx, row in meta.iterrows():
    sid = row['subject_id']
    emotion = row['emotion']
    signal_path = row['signal_file']
    stress_score = row['stress_score']

    # Load raw signal
    raw_signal = np.load(signal_path)

    # Step 1: Denoise
    denoised = wavelet_denoise(raw_signal)

    # Step 2: Extract PRV
    prv, peaks = extract_prv(denoised)
    prv_path = f'processed/prv/{sid}_{emotion}_prv.npy'
    np.save(prv_path, prv)

    # Step 3: Extract dPPG
    dppg, valleys = extract_dppg(denoised)
    dppg_path = f'processed/dppg/{sid}_{emotion}_dppg.npy'
    np.save(dppg_path, dppg)

    processed_records.append([sid, emotion, prv_path, dppg_path,
stress_score])

print("□ Preprocessing complete for all signals.")
```

```
□ Preprocessing complete for all signals.
```

```
columns = ['subject_id', 'emotion', 'prv_file', 'dppg_file',
'stress_score']
processed_df = pd.DataFrame(processed_records, columns=columns)
processed_df.to_csv('metadata/processed_dataset_metadata.csv',
index=False)
print("□ Processed metadata saved at
metadata/processed_dataset_metadata.csv")
```

```
□ Processed metadata saved at metadata/processed_dataset_metadata.csv
```

```
sample = meta.sample(1).iloc[0]
signal = np.load(sample.signal_file)
denoised = wavelet_denoise(signal)
_, peaks = extract_prv(denoised)
_, valleys = extract_dppg(denoised)

plt.figure(figsize=(12,4))
plt.plot(denoised, label='Denoised PPG', color='blue')
plt.scatter(peaks, denoised[peaks], color='red', label='Peaks (PRV)')
plt.scatter(valleys, denoised[valleys], color='green', label='Valleys
(dPPG)')
plt.legend()
plt.title(f"{sample.subject_id} - {sample.emotion}")
plt.show()
```



```
# Now comes step 3

# Step 3: Code — Split Data Accordingly

# split_dataset.py
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import os

# Load your processed metadata
meta = pd.read_csv('metadata/processed_dataset_metadata.csv')

# 1 Identify each unique participant
participants = meta['subject_id'].unique()
print(f"Total participants: {len(participants)}")

# 2 Create participant-level stress mean (used for stratification)
mean_stress = meta.groupby('subject_id')
['stress_score'].mean().reset_index()
mean_stress['stress_level'] = np.where(mean_stress['stress_score'] >
18, 'high', 'low')

# 3 Split participants by stress level using stratification
train_subj, test_subj = train_test_split(
    mean_stress,
    test_size=0.3,  # 70-30 split
    stratify=mean_stress['stress_level'],
    random_state=42
)

print(f"Train participants: {len(train_subj)} | Test participants:
```

```
{len(test_subj)}")

# 4 Map to sample-level metadata
train_meta = meta[meta['subject_id'].isin(train_subj['subject_id'])]
test_meta = meta[meta['subject_id'].isin(test_subj['subject_id'])]

# 5 Save split metadata
os.makedirs('metadata/splits', exist_ok=True)
train_meta.to_csv('metadata/splits/train_metadata.csv', index=False)
test_meta.to_csv('metadata/splits/test_metadata.csv', index=False)

print("□ Dataset split complete!")
print(f"Training samples: {len(train_meta)} | Testing samples:
{len(test_meta)}")
```

```
Total participants: 50
Train participants: 35 | Test participants: 15
□ Dataset split complete!
Training samples: 175 | Testing samples: 75
```

```
train_mean = train_meta.groupby('emotion')['stress_score'].mean()
test_mean = test_meta.groupby('emotion')['stress_score'].mean()
print("\nTrain Mean Stress per Emotion:\n", train_mean)
print("\nTest Mean Stress per Emotion:\n", test_mean)
```

```

Train Mean Stress per Emotion:
 emotion
calm          9.262731
fear         29.249868
happiness    14.919697
sadness      20.629826
tension      24.823454
Name: stress_score, dtype: float64

Test Mean Stress per Emotion:
 emotion
calm          9.455403
fear         29.555269
happiness    14.527186
sadness      20.431589
tension      24.519911
Name: stress_score, dtype: float64
```

```
# Next Step is Feature Extraction and Training of the Initial ML
models

# We’ll build:
```

```
# The same 1DCNN + BiLSTM model (shared architecture).

# Train it separately for each emotion and for each signal type (PRV &
dPPG).

# Save both the trained models and the extracted feature vectors
(Dense(32)), ready for the Cross-Attention fusion step later.

# ============================================================
# □ EEG_Stress_Project — Stage 4 Part 1
# 1DCNN + BiLSTM (PRV & dPPG)
# Train per-emotion models and extract 32D feature vectors
# ============================================================

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (Input, Conv1D,
BatchNormalization, Activation,
                                     Add, Concatenate, Bidirectional,
LSTM,
                                     Dense, Dropout)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import os

# ============================================================
# □ 1. Load metadata
# ============================================================
train_meta = pd.read_csv('metadata/splits/train_metadata.csv')
test_meta  = pd.read_csv('metadata/splits/test_metadata.csv')

# ============================================================
#  2. Helper to load signals for a specific emotion
# ============================================================
def load_emotion_signals(meta_df, emotion, modality='prv',
max_len=4000):
    signals, labels = [], []
    emo_meta = meta_df[meta_df['emotion'] == emotion]

    for _, row in emo_meta.iterrows():
        sig = np.load(row[f'{modality}_file'], allow_pickle=True)

        # Flatten in case it's (4000,1) or nested
        sig = np.ravel(sig)

        # Skip if signal empty or not numeric
        if sig.size == 0:
            print(f"⚠ Empty signal for {row['subject_id']} - {emotion}
({modality}), skipping.")
            continue

        # Pad or truncate to fixed length
        sig = sig[:max_len]
        sig = np.pad(sig, (0, max(0, max_len - len(sig))),
mode='constant')

        signals.append(sig)
        labels.append(row['stress_score'])

    # Stack properly
    signals = np.stack(signals, axis=0)
    labels = np.array(labels, dtype=np.float32)

    return signals, labels


# ============================================================
# ▯ 3. Define 1DCNN + BiLSTM + Regression Head
# ============================================================
def create_1dcnn_bilstm(input_shape=(4000,1)):
    inp = Input(shape=input_shape)

    # --- Multi-scale CNN block ---
    conv3 = Conv1D(32, 3, padding='same', activation='relu')(inp)
    conv5 = Conv1D(32, 5, padding='same', activation='relu')(inp)
    conv7 = Conv1D(32, 7, padding='same', activation='relu')(inp)
    merged = Concatenate()([conv3, conv5, conv7])
    merged = BatchNormalization()(merged)

    # --- Residual connection ---
    shortcut = Conv1D(96, 1, padding='same')(inp)
    res = Add()([merged, shortcut])
    res = Activation('relu')(res)

    # --- Temporal modeling ---
    x = Bidirectional(LSTM(64, return_sequences=True))(res)
    x = Bidirectional(LSTM(32))(x)

    # --- Dense feature embedding ---
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.3)(x)
    feat = Dense(32, activation='relu', name='feature_layer')(x)     #
feature vector
    out  = Dense(1, activation='linear', name='stress_output')(feat) #
stress regression

    model = Model(inputs=inp, outputs=out,
name='CNN_BiLSTM_Regressor')
```

```
    return model

# ============================================================
# □ 4. Build one model per emotion for PRV & dPPG
# ============================================================
emotions = ['calm','happiness','sadness','tension','fear']
emotion_models_prv, emotion_models_dppg = {}, {}

for emo in emotions:
    m_prv = create_1dcnn_bilstm()
    m_prv.compile(optimizer=Adam(1e-3), loss='mse', metrics=['mae'])
    emotion_models_prv[emo] = m_prv

    m_dppg = create_1dcnn_bilstm()
    m_dppg.compile(optimizer=Adam(1e-3), loss='mse', metrics=['mae'])
    emotion_models_dppg[emo] = m_dppg

# ============================================================
# □ 5. Train each emotion model (PRV & dPPG)
# ============================================================
os.makedirs("models/emotion_streams_prv", exist_ok=True)
os.makedirs("models/emotion_streams_dppg", exist_ok=True)

for emo in emotions:
    print(f"\n□ Training {emo.upper()} stream (PRV)...")
    X_prv, y_prv = load_emotion_signals(train_meta, emo,
modality='prv', max_len=4000)
    X_prv = np.expand_dims(X_prv, -1)

    es = EarlyStopping(monitor='val_loss', patience=8,
restore_best_weights=True)
    emotion_models_prv[emo].fit(X_prv, y_prv,
                                epochs=20,
                                batch_size=8,
                                validation_split=0.2,
                                callbacks=[es],
                                verbose=1)

emotion_models_prv[emo].save(f"models/emotion_streams_prv/{emo}_cnn_bi
lstm.keras")

    print(f"\n□ Training {emo.upper()} stream (dPPG)...")
    X_dppg, y_dppg = load_emotion_signals(train_meta, emo,
modality='dppg', max_len=4000)
    X_dppg = np.expand_dims(X_dppg, -1)

    es = EarlyStopping(monitor='val_loss', patience=8,
restore_best_weights=True)
    emotion_models_dppg[emo].fit(X_dppg, y_dppg,
                                 epochs=20,
                                 batch_size=8,
                                 validation_split=0.2,
                                 callbacks=[es],
                                 verbose=1)

emotion_models_dppg[emo].save(f"models/emotion_streams_dppg/{emo}_cnn_
bilstm.keras")
```

```
 Training CALM stream (PRV)...
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 600ms/step - loss: 117.8070 - mae: 9.5472
- val_loss: 86.6840 - val_mae: 8.1252
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 408ms/step - loss: 93.0931 - mae: 8.4070 -
val_loss: 81.7614 - val_mae: 7.8826
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 401ms/step - loss: 101.2393 - mae: 8.8274
- val_loss: 72.1447 - val_mae: 7.3715
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 397ms/step - loss: 94.7944 - mae: 8.5196 -
val_loss: 55.1317 - val_mae: 6.2872
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 399ms/step - loss: 60.2227 - mae: 6.6329 -
val_loss: 43.8285 - val_mae: 5.3459
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 506ms/step - loss: 49.3773 - mae: 5.9445 -
val_loss: 35.6009 - val_mae: 4.8839
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 465ms/step - loss: 39.6361 - mae: 5.3749 -
val_loss: 31.9299 - val_mae: 4.7124
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 418ms/step - loss: 29.4471 - mae: 4.4604 -
val_loss: 31.6715 - val_mae: 4.7956
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 394ms/step - loss: 36.6698 - mae: 4.9888 -
val_loss: 31.7183 - val_mae: 4.8190
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 399ms/step - loss: 34.0131 - mae: 4.7013 -
val_loss: 31.7218 - val_mae: 4.8207
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 394ms/step - loss: 26.2330 - mae: 4.3590 -
val_loss: 31.6757 - val_mae: 4.8037
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 403ms/step - loss: 24.1503 - mae: 4.0975 -
val_loss: 31.6605 - val_mae: 4.7800
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 442ms/step - loss: 30.0872 - mae: 3.9968 -
```

```
val_loss: 31.8699 - val_mae: 4.7197
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 545ms/step - loss: 34.9082 - mae: 5.0451 -
val_loss: 32.4962 - val_mae: 4.7311
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 394ms/step - loss: 34.1416 - mae: 5.0360 -
val_loss: 32.9574 - val_mae: 4.7632
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 393ms/step - loss: 31.4680 - mae: 4.8916 -
val_loss: 33.0477 - val_mae: 4.7687
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 396ms/step - loss: 35.0517 - mae: 5.1558 -
val_loss: 32.6285 - val_mae: 4.7411
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 391ms/step - loss: 29.1684 - mae: 4.7746 -
val_loss: 32.1457 - val_mae: 4.7001
Epoch 19/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 399ms/step - loss: 26.2321 - mae: 4.2644 -
val_loss: 31.9031 - val_mae: 4.7148
Epoch 20/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 395ms/step - loss: 30.0552 - mae: 4.8023 -
val_loss: 31.6924 - val_mae: 4.7598

□ Training CALM stream (dPPG)...
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 582ms/step - loss: 114.6645 - mae: 9.5832
- val_loss: 79.5056 - val_mae: 7.7653
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 403ms/step - loss: 94.0922 - mae: 8.5561 -
val_loss: 66.9182 - val_mae: 7.0588
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 399ms/step - loss: 77.9749 - mae: 7.6181 -
val_loss: 53.1584 - val_mae: 6.1171
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 412ms/step - loss: 54.6746 - mae: 6.0253 -
val_loss: 41.2266 - val_mae: 5.1726
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 489ms/step - loss: 36.6451 - mae: 4.8734 -
val_loss: 33.7870 - val_mae: 4.7943
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 494ms/step - loss: 31.5845 - mae: 4.8729 -
val_loss: 32.0626 - val_mae: 4.8597
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 396ms/step - loss: 27.2846 - mae: 4.0984 -
val_loss: 34.8404 - val_mae: 5.0491
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 394ms/step - loss: 24.7248 - mae: 3.8177 -
val_loss: 37.5783 - val_mae: 5.1437
Epoch 9/20
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 399ms/step - loss: 31.2424 - mae: 4.6670 -
val_loss: 36.9708 - val_mae: 5.1242
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 398ms/step - loss: 31.0867 - mae: 4.6587 -
val_loss: 34.5038 - val_mae: 5.0328
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 729ms/step - loss: 25.9852 - mae: 4.2174 -
val_loss: 33.1337 - val_mae: 4.9617
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 627ms/step - loss: 34.7091 - mae: 5.0171 -
val_loss: 32.2380 - val_mae: 4.8895
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 415ms/step - loss: 30.3804 - mae: 4.5801 -
val_loss: 31.9719 - val_mae: 4.8524
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 396ms/step - loss: 30.2566 - mae: 4.5775 -
val_loss: 32.0176 - val_mae: 4.8602
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 412ms/step - loss: 29.8054 - mae: 4.4415 -
val_loss: 32.0933 - val_mae: 4.8715
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 395ms/step - loss: 25.4135 - mae: 4.3709 -
val_loss: 32.0823 - val_mae: 4.8699
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 402ms/step - loss: 33.5801 - mae: 4.9182 -
val_loss: 31.9574 - val_mae: 4.8491
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 393ms/step - loss: 30.9026 - mae: 4.6455 -
val_loss: 32.1494 - val_mae: 4.8787
Epoch 19/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 455ms/step - loss: 25.3433 - mae: 4.0783 -
val_loss: 32.4981 - val_mae: 4.9150
Epoch 20/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 510ms/step - loss: 30.2200 - mae: 4.4569 -
val_loss: 33.0811 - val_mae: 4.9585

□ Training HAPPINESS stream (PRV)...
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 579ms/step - loss: 221.1311 - mae: 14.1642
- val_loss: 255.6264 - val_mae: 15.2341
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 190.7034 - mae: 13.1118
- val_loss: 240.2620 - val_mae: 14.7214
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 420ms/step - loss: 177.1848 - mae: 12.5641
- val_loss: 215.2895 - val_mae: 13.8476
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 554ms/step - loss: 138.1758 - mae: 11.0991
- val_loss: 166.9623 - val_mae: 11.9764
```

```
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 403ms/step - loss: 97.7664 - mae: 8.9756 -
val_loss: 119.4564 - val_mae: 9.7947
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 67.3140 - mae: 6.5156 -
val_loss: 81.3845 - val_mae: 7.6074
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 400ms/step - loss: 30.3660 - mae: 4.4329 -
val_loss: 51.5512 - val_mae: 6.0971
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 397ms/step - loss: 28.1468 - mae: 4.1263 -
val_loss: 34.9248 - val_mae: 5.2741
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 396ms/step - loss: 31.4566 - mae: 4.5038 -
val_loss: 29.4863 - val_mae: 4.8737
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 398ms/step - loss: 27.1693 - mae: 4.1763 -
val_loss: 29.4540 - val_mae: 4.8704
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 522ms/step - loss: 28.4040 - mae: 3.9211 -
val_loss: 32.5473 - val_mae: 5.1129
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 440ms/step - loss: 26.5873 - mae: 4.0249 -
val_loss: 36.8300 - val_mae: 5.3881
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 404ms/step - loss: 19.5235 - mae: 3.5180 -
val_loss: 40.8899 - val_mae: 5.6105
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 392ms/step - loss: 22.0406 - mae: 3.8192 -
val_loss: 43.6350 - val_mae: 5.7465
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 397ms/step - loss: 33.9307 - mae: 4.6275 -
val_loss: 45.2618 - val_mae: 5.8229
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 401ms/step - loss: 27.3841 - mae: 4.1983 -
val_loss: 42.1425 - val_mae: 5.6742
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 395ms/step - loss: 24.0005 - mae: 4.1872 -
val_loss: 38.7081 - val_mae: 5.4953
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 452ms/step - loss: 14.3574 - mae: 3.1403 -
val_loss: 36.6088 - val_mae: 5.3761

□ Training HAPPINESS stream (dPPG)...
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 581ms/step - loss: 232.1589 - mae: 14.5735
- val_loss: 258.6387 - val_mae: 15.3334
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 403ms/step - loss: 229.7543 - mae: 14.4034
```

```
- val_loss: 241.4966 - val_mae: 14.7649
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 400ms/step - loss: 202.1086 - mae: 13.3949
- val_loss: 217.1173 - val_mae: 13.9160
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 402ms/step - loss: 151.8303 - mae: 11.4917
- val_loss: 186.2650 - val_mae: 12.7596
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 565ms/step - loss: 133.3267 - mae: 10.6360
- val_loss: 150.1610 - val_mae: 11.2557
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 398ms/step - loss: 101.4249 - mae: 8.9608
- val_loss: 113.3903 - val_mae: 9.4822
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 403ms/step - loss: 69.3247 - mae: 7.1600 -
val_loss: 78.9964 - val_mae: 7.4502
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 404ms/step - loss: 42.6506 - mae: 5.4244 -
val_loss: 51.2983 - val_mae: 6.0939
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 402ms/step - loss: 33.3455 - mae: 4.4640 -
val_loss: 32.6551 - val_mae: 5.1300
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 400ms/step - loss: 33.5837 - mae: 4.7553 -
val_loss: 25.2090 - val_mae: 4.3897
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 41.3411 - mae: 5.2324 -
val_loss: 23.7855 - val_mae: 4.0550
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 524ms/step - loss: 33.5781 - mae: 4.7534 -
val_loss: 23.9948 - val_mae: 4.1238
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 461ms/step - loss: 31.2034 - mae: 4.1182 -
val_loss: 25.2338 - val_mae: 4.3929
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 403ms/step - loss: 29.9494 - mae: 4.3874 -
val_loss: 28.2457 - val_mae: 4.7652
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 397ms/step - loss: 25.8493 - mae: 4.2312 -
val_loss: 30.8251 - val_mae: 4.9923
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 395ms/step - loss: 22.7686 - mae: 3.6789 -
val_loss: 31.2938 - val_mae: 5.0289
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 398ms/step - loss: 21.0422 - mae: 3.7994 -
val_loss: 30.6123 - val_mae: 4.9752
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 398ms/step - loss: 23.4183 - mae: 3.5396 -
val_loss: 28.8560 - val_mae: 4.8235
```

```
Epoch 19/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 561ms/step - loss: 30.7813 - mae: 4.4013 -
val_loss: 28.2163 - val_mae: 4.7620

 Training SADNESS stream (PRV)...
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - loss: 417.8071 - mae: 19.8847 -
val_loss: 401.0146 - val_mae: 19.9445
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 482ms/step - loss: 392.2092 - mae: 19.1590
- val_loss: 380.6407 - val_mae: 19.4269
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 421ms/step - loss: 357.2593 - mae: 18.2972
- val_loss: 347.2138 - val_mae: 18.5464
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 422ms/step - loss: 315.7081 - mae: 17.1340
- val_loss: 279.6901 - val_mae: 16.6265
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 419ms/step - loss: 249.7054 - mae: 14.9895
- val_loss: 199.1596 - val_mae: 13.9965
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 440ms/step - loss: 163.6425 - mae: 11.5086
- val_loss: 128.3788 - val_mae: 11.1852
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 429ms/step - loss: 101.9657 - mae: 8.9560
- val_loss: 67.3553 - val_mae: 8.0046
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 559ms/step - loss: 47.2360 - mae: 5.6798 -
val_loss: 27.1304 - val_mae: 4.8815
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 478ms/step - loss: 29.1921 - mae: 4.6356 -
val_loss: 8.6794 - val_mae: 2.3510
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 424ms/step - loss: 39.8641 - mae: 5.0403 -
val_loss: 4.3909 - val_mae: 1.7161
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 413ms/step - loss: 66.3091 - mae: 6.4546 -
val_loss: 6.3951 - val_mae: 1.9500
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 463ms/step - loss: 48.9375 - mae: 5.3053 -
val_loss: 12.9172 - val_mae: 3.0966
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 432ms/step - loss: 35.1365 - mae: 4.8223 -
val_loss: 21.9912 - val_mae: 4.3210
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 426ms/step - loss: 19.8118 - mae: 3.4408 -
val_loss: 28.1079 - val_mae: 4.9792
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 554ms/step - loss: 34.4616 - mae: 5.0939 -
```

```
val_loss: 27.9745 - val_mae: 4.9659
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 415ms/step - loss: 26.4498 - mae: 4.4477 -
val_loss: 22.1217 - val_mae: 4.3366
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 430ms/step - loss: 34.9240 - mae: 4.6589 -
val_loss: 17.8043 - val_mae: 3.8063
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 415ms/step - loss: 34.8911 - mae: 4.9685 -
val_loss: 14.2070 - val_mae: 3.3002

□ Training SADNESS stream (dPPG)...
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 819ms/step - loss: 403.5308 - mae: 19.4699
- val_loss: 390.5019 - val_mae: 19.6798
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 422ms/step - loss: 369.1854 - mae: 18.6195
- val_loss: 351.1338 - val_mae: 18.6535
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 408ms/step - loss: 312.7393 - mae: 16.9386
- val_loss: 297.8119 - val_mae: 17.1654
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 417ms/step - loss: 245.5589 - mae: 14.6715
- val_loss: 236.3959 - val_mae: 15.2721
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 412ms/step - loss: 228.2210 - mae: 14.1625
- val_loss: 172.4306 - val_mae: 13.0096
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 520ms/step - loss: 145.5726 - mae: 10.7757
- val_loss: 110.9134 - val_mae: 10.3786
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 497ms/step - loss: 111.1227 - mae: 9.3067
- val_loss: 56.7296 - val_mae: 7.3159
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 73.7433 - mae: 7.1828 -
val_loss: 19.3885 - val_mae: 4.0220
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 409ms/step - loss: 47.1910 - mae: 5.6315 -
val_loss: 4.3802 - val_mae: 1.6990
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 409ms/step - loss: 32.9669 - mae: 4.8531 -
val_loss: 3.6821 - val_mae: 1.6513
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 403ms/step - loss: 55.6849 - mae: 5.7470 -
val_loss: 4.3744 - val_mae: 1.7160
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 414ms/step - loss: 46.4704 - mae: 5.4234 -
val_loss: 3.3918 - val_mae: 1.6141
Epoch 13/20
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 437ms/step - loss: 26.8893 - mae: 4.0326 -
val_loss: 3.6442 - val_mae: 1.6371
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 559ms/step - loss: 49.9106 - mae: 6.2177 -
val_loss: 7.0565 - val_mae: 2.0839
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 41.2280 - mae: 5.3154 -
val_loss: 11.8927 - val_mae: 2.9453
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 414ms/step - loss: 31.8353 - mae: 4.9022 -
val_loss: 14.9690 - val_mae: 3.4279
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 37.7123 - mae: 4.7234 -
val_loss: 13.3892 - val_mae: 3.1891
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 419ms/step - loss: 37.8251 - mae: 4.5762 -
val_loss: 10.2787 - val_mae: 2.6570
Epoch 19/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 415ms/step - loss: 31.2353 - mae: 4.6009 -
val_loss: 7.5895 - val_mae: 2.1778
Epoch 20/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 455ms/step - loss: 42.2661 - mae: 5.5243 -
val_loss: 5.7992 - val_mae: 1.8389

□ Training TENSION stream (PRV)...
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 594ms/step - loss: 692.1226 - mae: 25.5935
- val_loss: 598.8929 - val_mae: 23.8151
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 670.5881 - mae: 25.3049
- val_loss: 583.6497 - val_mae: 23.4929
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 408ms/step - loss: 578.0297 - mae: 23.3770
- val_loss: 563.4659 - val_mae: 23.0593
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 477ms/step - loss: 541.3687 - mae: 22.5937
- val_loss: 528.5078 - val_mae: 22.2884
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 412ms/step - loss: 509.3006 - mae: 21.9645
- val_loss: 471.7193 - val_mae: 20.9758
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 409ms/step - loss: 468.5098 - mae: 20.8628
- val_loss: 415.9659 - val_mae: 19.6018
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 410ms/step - loss: 411.2219 - mae: 19.5159
- val_loss: 358.4461 - val_mae: 18.0751
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 334.3872 - mae: 17.1627
- val_loss: 295.8765 - val_mae: 16.2523
```

```
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 410ms/step - loss: 306.6963 - mae: 16.5274
- val_loss: 231.4225 - val_mae: 14.1309
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 410ms/step - loss: 177.8616 - mae: 11.9103
- val_loss: 170.9464 - val_mae: 11.7988
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 522ms/step - loss: 128.0626 - mae: 9.6341
- val_loss: 119.2282 - val_mae: 9.6395
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 491ms/step - loss: 89.9622 - mae: 7.8889 -
val_loss: 79.0444 - val_mae: 7.8713
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 414ms/step - loss: 56.0990 - mae: 6.3429 -
val_loss: 54.6855 - val_mae: 6.3805
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 37.8686 - mae: 4.9734 -
val_loss: 40.3661 - val_mae: 5.1973
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 408ms/step - loss: 62.3571 - mae: 6.5084 -
val_loss: 35.3224 - val_mae: 4.7501
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 43.2615 - mae: 5.0006 -
val_loss: 33.9733 - val_mae: 4.6156
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 50.3699 - mae: 5.9284 -
val_loss: 35.6953 - val_mae: 4.7914
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 460ms/step - loss: 46.1580 - mae: 5.4351 -
val_loss: 38.9353 - val_mae: 5.0885
Epoch 19/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 530ms/step - loss: 59.6083 - mae: 6.7502 -
val_loss: 43.0738 - val_mae: 5.3816
Epoch 20/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 66.8712 - mae: 7.1020 -
val_loss: 47.5234 - val_mae: 5.7968

□ Training TENSION stream (dPPG)...
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 592ms/step - loss: 648.2151 - mae: 24.8205
- val_loss: 578.4677 - val_mae: 23.3837
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 473ms/step - loss: 579.0962 - mae: 23.4069
- val_loss: 540.6185 - val_mae: 22.5605
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 522ms/step - loss: 563.0081 - mae: 22.9034
- val_loss: 489.0085 - val_mae: 21.3871
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 428.8582 - mae: 19.9282
```

```
- val_loss: 420.5772 - val_mae: 19.7235
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 409ms/step - loss: 423.3009 - mae: 19.7665
- val_loss: 338.8947 - val_mae: 17.5309
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 403ms/step - loss: 302.9001 - mae: 16.4229
- val_loss: 250.2451 - val_mae: 14.7869
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 419ms/step - loss: 252.2290 - mae: 14.6950
- val_loss: 163.4263 - val_mae: 11.4798
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 405ms/step - loss: 115.1303 - mae: 9.3884
- val_loss: 92.9524 - val_mae: 8.5675
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 412ms/step - loss: 56.8162 - mae: 6.2141 -
val_loss: 48.6462 - val_mae: 5.9158
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 581ms/step - loss: 66.1103 - mae: 6.9147 -
val_loss: 32.0395 - val_mae: 4.5319
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 404ms/step - loss: 37.8628 - mae: 4.3740 -
val_loss: 35.8304 - val_mae: 4.7409
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 409ms/step - loss: 53.0555 - mae: 6.3033 -
val_loss: 39.4347 - val_mae: 5.0011
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 82.5312 - mae: 7.5915 -
val_loss: 36.9762 - val_mae: 4.7959
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 408ms/step - loss: 70.6786 - mae: 6.7888 -
val_loss: 32.2799 - val_mae: 4.5715
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 405ms/step - loss: 55.8946 - mae: 6.2789 -
val_loss: 32.8014 - val_mae: 4.5797
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 414ms/step - loss: 49.5372 - mae: 5.7812 -
val_loss: 35.9403 - val_mae: 4.8396
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 519ms/step - loss: 44.5589 - mae: 5.4871 -
val_loss: 37.9612 - val_mae: 5.0295
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 405ms/step - loss: 30.0961 - mae: 4.3496 -
val_loss: 37.0789 - val_mae: 4.9509

 Training FEAR stream (PRV)...
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 589ms/step - loss: 921.8521 - mae: 30.0052
- val_loss: 823.9158 - val_mae: 28.4073
Epoch 2/20
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 481ms/step - loss: 867.8589 - mae: 29.1268
- val_loss: 806.1417 - val_mae: 28.0927
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 523ms/step - loss: 836.8884 - mae: 28.5972
- val_loss: 780.1193 - val_mae: 27.6257
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 404ms/step - loss: 810.6612 - mae: 28.0782
- val_loss: 736.5646 - val_mae: 26.8259
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 671.3714 - mae: 25.5053
- val_loss: 670.0229 - val_mae: 25.5556
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 405ms/step - loss: 632.6564 - mae: 24.7003
- val_loss: 606.0215 - val_mae: 24.2712
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 411ms/step - loss: 544.9389 - mae: 22.9377
- val_loss: 579.6768 - val_mae: 23.7223
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 404ms/step - loss: 430.5286 - mae: 20.2964
- val_loss: 521.6347 - val_mae: 22.4656
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 412ms/step - loss: 356.5293 - mae: 18.2716
- val_loss: 444.5168 - val_mae: 20.6782
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 575ms/step - loss: 233.8753 - mae: 14.6738
- val_loss: 360.2849 - val_mae: 18.5300
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 405ms/step - loss: 127.7512 - mae: 10.5756
- val_loss: 274.8333 - val_mae: 16.0596
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 81.2598 - mae: 7.7161 -
val_loss: 198.9428 - val_mae: 13.4916
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 408ms/step - loss: 70.3447 - mae: 7.2622 -
val_loss: 139.3355 - val_mae: 11.0643
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 40.7489 - mae: 5.5939 -
val_loss: 106.3494 - val_mae: 9.4570
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 34.9162 - mae: 5.1088 -
val_loss: 93.4222 - val_mae: 8.7470
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 47.2056 - mae: 5.4088 -
val_loss: 90.7520 - val_mae: 8.5931
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 527ms/step - loss: 40.2940 - mae: 5.4065 -
val_loss: 96.0318 - val_mae: 8.8950
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 452ms/step - loss: 55.0230 - mae: 6.0043 -
val_loss: 100.4647 - val_mae: 9.1407
```

```
Epoch 19/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 404ms/step - loss: 30.2586 - mae: 4.6832 -
val_loss: 103.8674 - val_mae: 9.3250
Epoch 20/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 33.9103 - mae: 4.3444 -
val_loss: 104.3081 - val_mae: 9.3485

 Training FEAR stream (dPPG)...
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 714ms/step - loss: 865.5240 - mae: 29.0513
- val_loss: 818.5059 - val_mae: 28.3141
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 463ms/step - loss: 852.3983 - mae: 28.8757
- val_loss: 787.6871 - val_mae: 27.7695
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 409ms/step - loss: 776.7550 - mae: 27.4243
- val_loss: 737.8709 - val_mae: 26.8636
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 405ms/step - loss: 709.2543 - mae: 26.2097
- val_loss: 669.5568 - val_mae: 25.5647
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 589.9424 - mae: 23.8344
- val_loss: 582.5059 - val_mae: 23.8034
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 511.3871 - mae: 22.0154
- val_loss: 476.6144 - val_mae: 21.4652
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 436.9841 - mae: 20.2461
- val_loss: 356.3433 - val_mae: 18.4559
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 444ms/step - loss: 293.7908 - mae: 16.2622
- val_loss: 233.6133 - val_mae: 14.7615
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 540ms/step - loss: 171.4911 - mae: 12.0633
- val_loss: 125.3156 - val_mae: 10.4724
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 87.2292 - mae: 8.1516 -
val_loss: 51.1929 - val_mae: 5.9669
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 408ms/step - loss: 36.8351 - mae: 4.9207 -
val_loss: 18.6509 - val_mae: 3.2976
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 405ms/step - loss: 36.9937 - mae: 4.7978 -
val_loss: 17.0678 - val_mae: 3.6403
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 400ms/step - loss: 69.4209 - mae: 6.7769 -
val_loss: 18.5014 - val_mae: 3.8086
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 49.6235 - mae: 5.6991 -
```

```
val_loss: 16.3577 - val_mae: 3.5975
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 405ms/step - loss: 63.8276 - mae: 6.2995 -
val_loss: 16.1359 - val_mae: 3.4590
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 574ms/step - loss: 33.0955 - mae: 4.7365 -
val_loss: 17.7924 - val_mae: 3.3330
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 409ms/step - loss: 34.7913 - mae: 4.6742 -
val_loss: 21.1471 - val_mae: 3.4856
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 406ms/step - loss: 21.4673 - mae: 3.9509 -
val_loss: 25.0646 - val_mae: 3.8057
Epoch 19/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 402ms/step - loss: 45.2171 - mae: 6.1311 -
val_loss: 26.6183 - val_mae: 3.9125
Epoch 20/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 407ms/step - loss: 36.9797 - mae: 5.3381 -
val_loss: 23.7986 - val_mae: 3.7106

# ============================================================
# □ 6. Extract 32D feature vectors using the 'feature_layer'
# ============================================================
def extract_features(meta_df, modality, models_dict):
    all_feats, all_labels = [], []

    for _, row in meta_df.iterrows():
        emo = row['emotion']
        sig = np.load(row[f'{modality}_file'], allow_pickle=True)

        # --- □ FIX: Flatten any nested arrays (2D, lists, etc.) ---
        sig = np.ravel(sig).astype(np.float32)

        # --- □ FIX: Truncate or pad to exactly 4000 samples ---
        sig = sig[:4000]
        if len(sig) < 4000:
            sig = np.pad(sig, (0, 4000 - len(sig)), mode='constant')

        # Expand to 3D: (batch, seq_len, channels)
        sig = np.expand_dims(sig, (0, -1))

        # Get corresponding trained model
        trained_model = models_dict[emo]
        feature_model = Model(
            inputs=trained_model.input,
            outputs=trained_model.get_layer('feature_layer').output
        )

        # Extract 32D feature
        feat = feature_model.predict(sig, verbose=0)
        all_feats.append(feat.squeeze())
        all_labels.append(row['stress_score'])

    return np.array(all_feats), np.array(all_labels)


print("\n□ Extracting PRV features ...")
train_prv_feats, train_labels = extract_features(train_meta, 'prv',
emotion_models_prv)
test_prv_feats, test_labels   = extract_features(test_meta,  'prv',
emotion_models_prv)

print("□ Extracting dPPG features ...")
train_dppg_feats, _ = extract_features(train_meta, 'dppg',
emotion_models_dppg)
test_dppg_feats, _  = extract_features(test_meta,  'dppg',
emotion_models_dppg)
```

```
□ Extracting PRV features ...
□ Extracting dPPG features ...
```

```
# ==============================================================
# □ 7. Save extracted features
# ==============================================================
os.makedirs('processed/features', exist_ok=True)

np.save('processed/features/train_prv_features.npy', train_prv_feats)
np.save('processed/features/train_dppg_features.npy',
train_dppg_feats)
np.save('processed/features/train_labels.npy', train_labels)

np.save('processed/features/test_prv_features.npy', test_prv_feats)
np.save('processed/features/test_dppg_features.npy', test_dppg_feats)
np.save('processed/features/test_labels.npy', test_labels)

print("\n□ Feature extraction complete. Files saved in
'processed/features/'")
```

```
□ Feature extraction complete. Files saved in 'processed/features/'
```

```
# Now we'll build the complete and weighted Emotional Cross-Attention
Fusion module (Stage 4 – Part 2) exactly as per our research paper's
logic.

# We'll make it:
```

```python
# EEG Stress Project – Stage 4 Part 2
# Weighted Emotional Cross-Attention Fusion for PRV and dPPG

# ============================================================
# □ EEG_Stress_Project – Stage 4 Part 2
# Weighted Emotional Cross-Attention Fusion (for PRV & dPPG)
# ============================================================

import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import (Input, Dense, MultiHeadAttention,
                                     LayerNormalization, Dropout,
                                     GlobalAveragePooling1D, Multiply)
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping

# ============================================================
# □ 1. Load extracted 32D features from Stage 4 Part 1
# ============================================================
train_prv_feats = np.load('processed/features/train_prv_features.npy')
train_dppg_feats =
np.load('processed/features/train_dppg_features.npy')
train_labels     = np.load('processed/features/train_labels.npy')

test_prv_feats = np.load('processed/features/test_prv_features.npy')
test_dppg_feats = np.load('processed/features/test_dppg_features.npy')
test_labels     = np.load('processed/features/test_labels.npy')

print("Loaded features:")
print("PRV:", train_prv_feats.shape, test_prv_feats.shape)
print("dPPG:", train_dppg_feats.shape, test_dppg_feats.shape)

# ============================================================
# ⚙ 2. Reshape features to [num_subjects, num_emotions, 32]
# Each subject has 5 emotions → calm, happiness, sadness, tension,
fear
# ============================================================
num_emotions = 5
feature_dim = 32

train_prv = train_prv_feats.reshape(-1, num_emotions, feature_dim)
test_prv  = test_prv_feats.reshape(-1, num_emotions, feature_dim)

train_dppg = train_dppg_feats.reshape(-1, num_emotions, feature_dim)
test_dppg  = test_dppg_feats.reshape(-1, num_emotions, feature_dim)

# One label per participant
train_y = train_labels[::num_emotions]
test_y  = test_labels[::num_emotions]
```

```
print(f"Train participants: {train_prv.shape[0]}, Test participants:
{test_prv.shape[0]}")

# ============================================================
# 🔹 3. Define emotion importance weights (from paper insight)
# Negative emotions (sadness, fear, tension) → stronger stress
relation
# ============================================================

# Order: calm, happiness, sadness, tension, fear
emotion_weights = tf.constant([0.8, 0.9, 1.1, 1.2, 1.3],
dtype=tf.float32)
emotion_weights = tf.reshape(emotion_weights, (1, num_emotions, 1))
print("Emotion weights:", emotion_weights.numpy().flatten())

# ============================================================
# 🔹 4. Define Weighted Cross-Attention Fusion Model
# ============================================================

def build_weighted_cross_attention(input_shape=(5, 32),
name='WeightedEmotionFusion'):
    inp = Input(shape=input_shape)

    # --- (a) Apply learned weights emphasizing negative emotions ---
    weighted_inp = Multiply()([inp, emotion_weights])   # scale each
emotion

    # --- (b) Cross-attention across emotion embeddings ---
    attn = MultiHeadAttention(num_heads=4, key_dim=32, dropout=0.1)
(weighted_inp, weighted_inp)
    attn = LayerNormalization()(attn + weighted_inp)

    # --- (c) Feed-forward network (like Transformer block) ---
    ff = Dense(64, activation='relu')(attn)
    ff = Dropout(0.2)(ff)
    ff = Dense(32, activation='relu')(ff)
    ff = LayerNormalization()(ff + attn)

    # --- (d) Pooling: aggregate across all 5 emotions ---
    pooled = GlobalAveragePooling1D(name='emotion_fused')(ff)

    # --- (e) Regression head for stress prediction ---
    out = Dense(1, activation='linear', name='stress_score')(pooled)

    model = Model(inputs=inp, outputs=out, name=name)
    return model
```

```
Loaded features:
PRV: (175, 32) (75, 32)
```

```
dPPG: (175, 32) (75, 32)
Train participants: 35, Test participants: 15
Emotion weights: [0.8 0.9 1.1 1.2 1.3]

# ==============================================================
# □ 5. Train PRV Cross-Attention Fusion Model
# ==============================================================
model_prv = build_weighted_cross_attention()
model_prv.compile(optimizer='adam', loss='mse', metrics=['mae'])

es = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

print("\n□ Training Weighted Cross-Attention (PRV)...")
history_prv = model_prv.fit(
    train_prv, train_y,
    validation_split=0.2,
    epochs=40,
    batch_size=8,
    callbacks=[es],
    verbose=1
)

model_prv.save('models/weighted_cross_attention_prv.keras')

# ==============================================================
# □ 6. Train dPPG Cross-Attention Fusion Model
# ==============================================================
model_dppg =
build_weighted_cross_attention(name='WeightedEmotionFusion_dPPG')
model_dppg.compile(optimizer='adam', loss='mse', metrics=['mae'])

print("\n□ Training Weighted Cross-Attention (dPPG)...")
history_dppg = model_dppg.fit(
    train_dppg, train_y,
    validation_split=0.2,
    epochs=40,
    batch_size=8,
    callbacks=[es],
    verbose=1
)

model_dppg.save('models/weighted_cross_attention_dppg.keras')



□ Training Weighted Cross-Attention (PRV)...
Epoch 1/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 1s/step - loss: 139.2395 - mae: 10.5420 -
val_loss: 61.1379 - val_mae: 6.7045
```

```
Epoch 2/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 72.3271 - mae: 7.2709 -
val_loss: 41.5614 - val_mae: 5.2013
Epoch 3/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 64.6039 - mae: 6.9303 -
val_loss: 34.6202 - val_mae: 4.8456
Epoch 4/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 41.1060 - mae: 5.4614 -
val_loss: 32.4779 - val_mae: 4.7287
Epoch 5/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 41.9763 - mae: 5.3147 -
val_loss: 31.7773 - val_mae: 4.7342
Epoch 6/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 30.9748 - mae: 4.6524 -
val_loss: 31.6577 - val_mae: 4.7826
Epoch 7/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 35.1063 - mae: 4.9499 -
val_loss: 31.7273 - val_mae: 4.8212
Epoch 8/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 33.3397 - mae: 4.7739 -
val_loss: 31.8912 - val_mae: 4.8524
Epoch 9/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 24.2695 - mae: 4.1301 -
val_loss: 32.1010 - val_mae: 4.8784
Epoch 10/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 29.0458 - mae: 4.4391 -
val_loss: 32.3573 - val_mae: 4.9027
Epoch 11/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 26.2665 - mae: 4.3199 -
val_loss: 32.6258 - val_mae: 4.9238
Epoch 12/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 33.2938 - mae: 4.8594 -
val_loss: 32.8647 - val_mae: 4.9401
Epoch 13/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 24.8166 - mae: 4.2378 -
val_loss: 33.1192 - val_mae: 4.9559
Epoch 14/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 26.6425 - mae: 4.3716 -
val_loss: 33.3776 - val_mae: 4.9705
Epoch 15/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 25.4700 - mae: 4.2817 -
val_loss: 33.6419 - val_mae: 4.9844
Epoch 16/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 21.6517 - mae: 3.9642 -
val_loss: 33.8708 - val_mae: 4.9957

□ Training Weighted Cross-Attention (dPPG)...
Epoch 1/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 938ms/step - loss: 125.3493 - mae: 9.7268
```

```
- val_loss: 51.6341 - val_mae: 6.0201
Epoch 2/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 60.4483 - mae: 6.6484 -
val_loss: 36.5239 - val_mae: 4.9136
Epoch 3/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 44.9959 - mae: 5.6170 -
val_loss: 32.3832 - val_mae: 4.7198
Epoch 4/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 33.8700 - mae: 4.7929 -
val_loss: 31.6692 - val_mae: 4.7849
Epoch 5/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - loss: 31.2078 - mae: 4.6101 -
val_loss: 32.0252 - val_mae: 4.8689
Epoch 6/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 29.1507 - mae: 4.5350 -
val_loss: 32.5325 - val_mae: 4.9164
Epoch 7/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - loss: 29.2015 - mae: 4.5877 -
val_loss: 33.1151 - val_mae: 4.9554
Epoch 8/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 27.1933 - mae: 4.4495 -
val_loss: 33.5903 - val_mae: 4.9816
Epoch 9/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 28.2463 - mae: 4.5309 -
val_loss: 34.0083 - val_mae: 5.0020
Epoch 10/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 28.2174 - mae: 4.6012 -
val_loss: 34.3772 - val_mae: 5.0186
Epoch 11/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - loss: 30.7811 - mae: 4.8392 -
val_loss: 34.7036 - val_mae: 5.0323
Epoch 12/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 27.7479 - mae: 4.5692 -
val_loss: 34.9238 - val_mae: 5.0412
Epoch 13/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - loss: 28.7024 - mae: 4.5889 -
val_loss: 35.2185 - val_mae: 5.0526
Epoch 14/40
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 26.3038 - mae: 4.3236 -
val_loss: 35.4040 - val_mae: 5.0595

# ==============================================================
# □ 7. Evaluate on test data
# ==============================================================
prv_eval = model_prv.evaluate(test_prv, test_y, verbose=0)
dppg_eval = model_dppg.evaluate(test_dppg, test_y, verbose=0)

print(f"\n□ PRV  Cross-Attention  → MAE: {prv_eval[1]:.3f}, RMSE:
{np.sqrt(prv_eval[0]):.3f}")
print(f"□ dPPG Cross-Attention → MAE: {dppg_eval[1]:.3f}, RMSE:
```

```
{np.sqrt(dppg_eval[0]):.3f}")

# ============================================================
# □ 8. Extract fused emotional features for later multimodal fusion
# ============================================================
# The 'emotion_fused' layer gives us the final 32D representation per
participant
feat_model_prv = Model(model_prv.input,
model_prv.get_layer('emotion_fused').output)
feat_model_dppg = Model(model_dppg.input,
model_dppg.get_layer('emotion_fused').output)

train_prv_fused = feat_model_prv.predict(train_prv)
test_prv_fused  = feat_model_prv.predict(test_prv)
train_dppg_fused = feat_model_dppg.predict(train_dppg)
test_dppg_fused  = feat_model_dppg.predict(test_dppg)

np.save('processed/features/train_prv_fused.npy', train_prv_fused)
np.save('processed/features/test_prv_fused.npy', test_prv_fused)
np.save('processed/features/train_dppg_fused.npy', train_dppg_fused)
np.save('processed/features/test_dppg_fused.npy', test_dppg_fused)

print("\n□ Weighted Emotional Cross-Attention fusion complete!")
```

```
□ PRV  Cross-Attention  → MAE: 4.099, RMSE: 5.337
□ dPPG Cross-Attention → MAE: 4.097, RMSE: 5.330
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 493ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 349ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 492ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 388ms/step

□ Weighted Emotional Cross-Attention fusion complete!
```

```
# Next is MultiModal Fusion Layer part , Lets implement it with full
excitement now

# ============================================================
# □ EEG_Stress_Project — Stage 4 Part 3
# Multimodal Fusion (Cross-Attention between PRV + dPPG)
# ============================================================

import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import (Input, Dense, MultiHeadAttention,
Add,
                                     Concatenate, LayerNormalization,
Dropout)
from tensorflow.keras.models import Model
```

```
import os

# ============================================================
# □ 1. Load emotion-fused (32D) features
# ============================================================
train_prv_fused = np.load('processed/features/train_prv_fused.npy')
train_dppg_fused = np.load('processed/features/train_dppg_fused.npy')
test_prv_fused  = np.load('processed/features/test_prv_fused.npy')
test_dppg_fused = np.load('processed/features/test_dppg_fused.npy')

train_labels = np.load('processed/features/train_labels.npy')[::5]
test_labels  = np.load('processed/features/test_labels.npy')[::5]

print("Loaded emotion-fused features:")
print("Train PRV:", train_prv_fused.shape, "| Train dPPG:",
train_dppg_fused.shape)
print("Test  PRV:", test_prv_fused.shape,  "| Test  dPPG:",
test_dppg_fused.shape)

# ============================================================
# ⚙ 2. Normalize and weight the modalities
# ============================================================
# The paper notes PRV > dPPG in informativeness, so:
# PRV weight = 1.2, dPPG weight = 0.8

prv_weight = 1.2
dppg_weight = 0.8

train_prv_fused = train_prv_fused * prv_weight
test_prv_fused  = test_prv_fused * prv_weight

train_dppg_fused = train_dppg_fused * dppg_weight
test_dppg_fused  = test_dppg_fused * dppg_weight

# Stack into modality pairs for input shape (2, 32)
train_pair = np.stack([train_prv_fused, train_dppg_fused], axis=1)  #
shape: (N, 2, 32)
test_pair  = np.stack([test_prv_fused,  test_dppg_fused], axis=1)

print("Multimodal pair shape:", train_pair.shape)

from tensorflow.keras.layers import Lambda

# ============================================================
# □ 3. Define Multimodal Cross-Attention Fusion Model (Fixed &
Serializable)
# ============================================================
def build_multimodal_cross_attention(input_shape=(2, 32)):
    inp = Input(shape=input_shape)
    # --- Extract PRV and dPPG tokens ---
    prv_token  = Lambda(lambda x: x[:, 0:1, :], name='prv_token',
output_shape=(1, 32))(inp)
    dppg_token = Lambda(lambda x: x[:, 1:2, :], name='dppg_token',
output_shape=(1, 32))(inp)

    # --- Cross-attention PRV ↔ dPPG ---
    attn_prv   = MultiHeadAttention(num_heads=4, key_dim=32,
name='attn_prv')(prv_token, dppg_token)
    attn_prv   = Add(name='add_prv')([attn_prv, prv_token])
    attn_prv   = LayerNormalization(name='ln_prv')(attn_prv)

    attn_dppg  = MultiHeadAttention(num_heads=4, key_dim=32,
name='attn_dppg')(dppg_token, prv_token)
    attn_dppg  = Add(name='add_dppg')([attn_dppg, dppg_token])
    attn_dppg  = LayerNormalization(name='ln_dppg')(attn_dppg)

    # --- Merge ---
    merged = Concatenate(axis=-1, name='concat_modalities')([attn_prv,
attn_dppg])
    merged = Dropout(0.2, name='dropout_1')(merged)

    x = Dense(64, activation='relu', name='dense_64')(merged)
    x = Dropout(0.2, name='dropout_2')(x)
    x = Dense(32, activation='relu', name='dense_32')(x)

    # □ Fix: explicitly import tensorflow within lambda
    x = Lambda(lambda t: tf.squeeze(t, axis=1), output_shape=(32,),
name='squeeze_layer')(x)

    # --- Add regression head for stress prediction ---
    out = Dense(1, activation='linear', name='stress_output')(x)

    model = Model(inputs=inp, outputs=out,
name="MultimodalCrossAttentionRegressor")
    return model
```

```
Loaded emotion-fused features:
Train PRV: (35, 32) | Train dPPG: (35, 32)
Test  PRV: (15, 32) | Test  dPPG: (15, 32)
Multimodal pair shape: (35, 2, 32)
```

```
# =============================================================
# □ 4. Build and train multimodal fusion model
# =============================================================
model_fusion = build_multimodal_cross_attention()
model_fusion.compile(optimizer='adam', loss='mse', metrics=['mae'])

history_fusion = model_fusion.fit(
    train_pair, train_labels,
    validation_split=0.2,
    epochs=30,
    batch_size=8,
    callbacks=[es],
    verbose=1
)
```

```
Epoch 1/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 1s/step - loss: 159.6555 - mae: 11.7026 -
val_loss: 82.0462 - val_mae: 7.8969
Epoch 2/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - loss: 118.8998 - mae: 9.7064 -
val_loss: 71.2553 - val_mae: 7.3215
Epoch 3/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 101.5463 - mae: 8.9106 -
val_loss: 64.5829 - val_mae: 6.9253
Epoch 4/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 80.8949 - mae: 7.7107 -
val_loss: 55.6604 - val_mae: 6.3262
Epoch 5/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 80.4746 - mae: 7.6442 -
val_loss: 46.4881 - val_mae: 5.5775
Epoch 6/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 62.9346 - mae: 6.7790 -
val_loss: 38.5265 - val_mae: 4.9760
Epoch 7/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 49.4088 - mae: 5.9813 -
val_loss: 33.4924 - val_mae: 4.7923
Epoch 8/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 45.5123 - mae: 5.3535 -
val_loss: 31.6958 - val_mae: 4.7532
Epoch 9/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 34.3703 - mae: 4.9112 -
val_loss: 32.2662 - val_mae: 4.8947
Epoch 10/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 36.8641 - mae: 5.3040 -
val_loss: 34.3688 - val_mae: 5.0182
Epoch 11/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 31.3042 - mae: 4.3735 -
val_loss: 36.6149 - val_mae: 5.1010
Epoch 12/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 36.3914 - mae: 5.0847 -
val_loss: 38.3925 - val_mae: 5.1537
Epoch 13/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 26.8064 - mae: 4.5569 -
val_loss: 39.6070 - val_mae: 5.1857
Epoch 14/30
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 39.4263 - mae: 4.9962 -
val_loss: 38.6066 - val_mae: 5.1595
Epoch 15/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 35.2424 - mae: 5.0808 -
val_loss: 38.3405 - val_mae: 5.1522
Epoch 16/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 38.9254 - mae: 5.0833 -
val_loss: 37.5008 - val_mae: 5.1283
Epoch 17/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 35.0498 - mae: 4.8655 -
val_loss: 36.0664 - val_mae: 5.0829
Epoch 18/30
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 20.5989 - mae: 3.7821 -
val_loss: 35.1222 - val_mae: 5.0489

# ============================================================
# □ 5. Evaluate model performance
# ============================================================
test_loss, test_mae = model_fusion.evaluate(test_pair, test_labels,
verbose=0)
print(f"\n□ Multimodal Cross-Attention → MAE: {test_mae:.3f}, RMSE:
{np.sqrt(test_loss):.3f}")

# ============================================================
# □ 6. Extract and save final multimodal fused representations
# ============================================================
# We'll use the model output itself (32D fused vector)
train_multimodal_fused = model_fusion.predict(train_pair)
test_multimodal_fused  = model_fusion.predict(test_pair)

np.save('processed/features/train_pair.npy', train_pair)
np.save('processed/features/test_pair.npy', test_pair)
print("\n□ Saved train/test multimodal pairs for reuse.")



os.makedirs('processed/features', exist_ok=True)
np.save('processed/features/train_multimodal_fused.npy',
train_multimodal_fused)
np.save('processed/features/test_multimodal_fused.npy',
test_multimodal_fused)

model_fusion.save('models/multimodal_cross_attention.keras')

print("\n□ Multimodal Cross-Attention Fusion complete!")
print("Final fused features saved in: processed/features/")


□ Multimodal Cross-Attention → MAE: 4.133, RMSE: 5.414
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 529ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 374ms/step

 Saved train/test multimodal pairs for reuse.

 Multimodal Cross-Attention Fusion complete!
Final fused features saved in: processed/features/

from tensorflow.keras.models import load_model

model_test = load_model('models/multimodal_cross_attention.keras',
compile=False, safe_mode=False)
model_test.summary()

Model: "MultimodalCrossAttentionRegressor"

┏━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━┳━━━━━━━━━━━━━━
━━━━━┓
┃ Layer (type)         ┃ Output Shape       ┃     Param # ┃ Connected to
┃
┡━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━╇━━━━━━━━━━━━━━
━━━━━┩
│ input_layer_90       │ (None, 2, 32)      │           0 │ -
│
│ (InputLayer)         │                    │             │
│
├──────────────────────┼────────────────────┼─────────────┼──────────────
─────┤
│ prv_token (Lambda)   │ (None, 1, 32)      │           0 │
input_layer_90[0… │
├──────────────────────┼────────────────────┼─────────────┼──────────────
─────┤
│ dppg_token (Lambda)  │ (None, 1, 32)      │           0 │
input_layer_90[0… │
├──────────────────────┼────────────────────┼─────────────┼──────────────
─────┤
│ attn_prv             │ (None, 1, 32)      │      16,800 │ prv_token[0]
[0],   │
│ (MultiHeadAttentio…  │                    │             │ dppg_token[0]
[0]  │
├──────────────────────┼────────────────────┼─────────────┼──────────────
─────┤
│ attn_dppg            │ (None, 1, 32)      │      16,800 │ dppg_token[0]
[0], │
│ (MultiHeadAttentio…  │                    │             │ prv_token[0]
[0]    │
├──────────────────────┼────────────────────┼─────────────┼──────────────
─────┤
│ add_prv (Add)        │ (None, 1, 32)      │           0 │ attn_prv[0]
[0],    │
│                      │                    │             │ prv_token[0]
```

```
[0]    │
├───────────────────────┼────────────────────┼────────────┼──────────────
──────┤
│ add_dppg (Add)        │ (None, 1, 32)      │          0 │ attn_dppg[0]
[0],  │
│                       │                    │            │ dppg_token[0]
[0]  │
├───────────────────────┼────────────────────┼────────────┼──────────────
──────┤
│ ln_prv                │ (None, 1, 32)      │         64 │ add_prv[0][0]
│
│ (LayerNormalizatio…   │                    │            │
│
├───────────────────────┼────────────────────┼────────────┼──────────────
──────┤
│ ln_dppg               │ (None, 1, 32)      │         64 │ add_dppg[0]
[0]     │
│ (LayerNormalizatio…   │                    │            │
│
├───────────────────────┼────────────────────┼────────────┼──────────────
──────┤
│ concat_modalities     │ (None, 1, 64)      │          0 │ ln_prv[0][0],
│
│ (Concatenate)         │                    │            │ ln_dppg[0][0]
│
├───────────────────────┼────────────────────┼────────────┼──────────────
──────┤
│ dropout_1 (Dropout)   │ (None, 1, 64)      │          0 │
concat_modalitie… │
├───────────────────────┼────────────────────┼────────────┼──────────────
──────┤
│ dense_64 (Dense)      │ (None, 1, 64)      │      4,160 │ dropout_1[0]
[0]    │
├───────────────────────┼────────────────────┼────────────┼──────────────
──────┤
│ dropout_2 (Dropout)   │ (None, 1, 64)      │          0 │ dense_64[0]
[0]     │
├───────────────────────┼────────────────────┼────────────┼──────────────
──────┤
│ dense_32 (Dense)      │ (None, 1, 32)      │      2,080 │ dropout_2[0]
[0]    │
├───────────────────────┼────────────────────┼────────────┼──────────────
──────┤
│ squeeze_layer         │ (None, 32)         │          0 │ dense_32[0]
[0]     │
│ (Lambda)              │                    │            │
│
├───────────────────────┼────────────────────┼────────────┼──────────────
──────┤
```

```
│ stress_output        │ (None, 1)          │          33 │
squeeze_layer[0]… │
│ (Dense)              │                    │             │
│
└──────────────────────┴────────────────────┴─────────────┴──────────────────
──────┘

 Total params: 40,001 (156.25 KB)

 Trainable params: 40,001 (156.25 KB)

 Non-trainable params: 0 (0.00 B)

import tensorflow as tf
from tensorflow.keras.models import load_model, Model
from tensorflow.keras.layers import Input, Flatten
import numpy as np

# 1 Allow Lambda deserialization
tf.keras.config.enable_unsafe_deserialization()

# 2 Load the old model safely
old_model = load_model(
    'models/multimodal_cross_attention.keras',
    compile=False,
    safe_mode=False,
    custom_objects={'tf': tf}
)

print("□ Old model loaded successfully.")

# 3 Inspect its structure to confirm input and pre-squeeze layers
old_model.summary()

# 4 Get the layer just *before* 'squeeze_layer'
pre_squeeze_output = old_model.get_layer('dense_32').output

# 5 Replace the Lambda squeeze with a native Flatten (no tf
dependency!)
x = Flatten(name='flatten_squeeze')(pre_squeeze_output)

# 6 Keep the regression head
out = old_model.get_layer('stress_output')(x)

# 7 Build a new safe model
safe_model = Model(inputs=old_model.input, outputs=out,
name="MultimodalCrossAttention_Safe")

# 8 Save it permanently
safe_model.save('models/multimodal_cross_attention_safe.keras')
```

```
print("✅ Model rebuilt and saved safely!")

✅ Old model loaded successfully.

Model: "MultimodalCrossAttentionRegressor"

┏━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━
━━━━━┓
┃ Layer (type)         ┃ Output Shape      ┃    Param # ┃ Connected to
┃
┡━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━
━━━━━┩
│ input_layer_90       │ (None, 2, 32)     │          0 │ -

│ (InputLayer)         │                   │            │
│
├──────────────────────┼───────────────────┼────────────┼──────────────
─────┤
│ prv_token (Lambda)   │ (None, 1, 32)     │          0 │
input_layer_90[0… │
├──────────────────────┼───────────────────┼────────────┼──────────────
─────┤
│ dppg_token (Lambda)  │ (None, 1, 32)     │          0 │
input_layer_90[0… │
├──────────────────────┼───────────────────┼────────────┼──────────────
─────┤
│ attn_prv             │ (None, 1, 32)     │     16,800 │ prv_token[0]
[0],   │
│ (MultiHeadAttentio…  │                   │            │ dppg_token[0]
[0]  │
├──────────────────────┼───────────────────┼────────────┼──────────────
─────┤
│ attn_dppg            │ (None, 1, 32)     │     16,800 │ dppg_token[0]
[0], │
│ (MultiHeadAttentio…  │                   │            │ prv_token[0]
[0]   │
├──────────────────────┼───────────────────┼────────────┼──────────────
─────┤
│ add_prv (Add)        │ (None, 1, 32)     │          0 │ attn_prv[0]
[0],    │
│                      │                   │            │ prv_token[0]
[0]   │
├──────────────────────┼───────────────────┼────────────┼──────────────
─────┤
│ add_dppg (Add)       │ (None, 1, 32)     │          0 │ attn_dppg[0]
[0],   │
│                      │                   │            │ dppg_token[0]
[0]  │
├──────────────────────┼───────────────────┼────────────┼──────────────
```

```
───────┤
│ ln_prv                 │ (None, 1, 32)       │           64 │ add_prv[0][0]
│                        │                     │              │
│ (LayerNormalizatio…    │                     │              │
│                        │                     │              │
├────────────────────────┼─────────────────────┼──────────────┼──────────────
───────┤
│ ln_dppg                │ (None, 1, 32)       │           64 │ add_dppg[0]
[0]     │
│ (LayerNormalizatio…    │                     │              │
│                        │                     │              │
├────────────────────────┼─────────────────────┼──────────────┼──────────────
───────┤
│ concat_modalities      │ (None, 1, 64)       │            0 │ ln_prv[0][0],
│                        │                     │              │
│ (Concatenate)          │                     │              │ ln_dppg[0][0]
│                        │                     │              │
├────────────────────────┼─────────────────────┼──────────────┼──────────────
───────┤
│ dropout_1 (Dropout)    │ (None, 1, 64)       │            0 │
concat_modalitie… │
├────────────────────────┼─────────────────────┼──────────────┼──────────────
───────┤
│ dense_64 (Dense)       │ (None, 1, 64)       │        4,160 │ dropout_1[0]
[0]    │
├────────────────────────┼─────────────────────┼──────────────┼──────────────
───────┤
│ dropout_2 (Dropout)    │ (None, 1, 64)       │            0 │ dense_64[0]
[0]     │
├────────────────────────┼─────────────────────┼──────────────┼──────────────
───────┤
│ dense_32 (Dense)       │ (None, 1, 32)       │        2,080 │ dropout_2[0]
[0]    │
├────────────────────────┼─────────────────────┼──────────────┼──────────────
───────┤
│ squeeze_layer          │ (None, 32)          │            0 │ dense_32[0]
[0]     │
│ (Lambda)               │                     │              │
│                        │                     │              │
├────────────────────────┼─────────────────────┼──────────────┼──────────────
───────┤
│ stress_output          │ (None, 1)           │           33 │
squeeze_layer[0]… │
│ (Dense)                │                     │              │
│                        │                     │              │
└────────────────────────┴─────────────────────┴──────────────┴──────────────
───────┘

 Total params: 40,001 (156.25 KB)
 Trainable params: 40,001 (156.25 KB)

 Non-trainable params: 0 (0.00 B)

□ Model rebuilt and saved safely!
```

```
from tensorflow.keras.models import load_model, Model
import numpy as np

# □ Load the clean, safe model
model_fusion =
load_model('models/multimodal_cross_attention_safe.keras',
compile=False)

# Extract features from the safe flatten layer
feature_extractor = Model(
    inputs=model_fusion.input,
    outputs=model_fusion.get_layer('flatten_squeeze').output
)

# Load fused inputs
train_prv_fused = np.load('processed/features/train_prv_fused.npy')
train_dppg_fused = np.load('processed/features/train_dppg_fused.npy')
test_prv_fused  = np.load('processed/features/test_prv_fused.npy')
test_dppg_fused = np.load('processed/features/test_dppg_fused.npy')

prv_weight, dppg_weight = 1.2, 0.8
train_prv_fused *= prv_weight
test_prv_fused  *= prv_weight
train_dppg_fused *= dppg_weight
test_dppg_fused  *= dppg_weight

train_pair = np.stack([train_prv_fused, train_dppg_fused], axis=1)
test_pair  = np.stack([test_prv_fused,  test_dppg_fused], axis=1)

train_fused = feature_extractor.predict(train_pair)
test_fused  = feature_extractor.predict(test_pair)

print("□ Correct fused feature shapes:", train_fused.shape,
test_fused.shape)

np.save('processed/features/train_multimodal_fused.npy', train_fused)
np.save('processed/features/test_multimodal_fused.npy', test_fused)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 495ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 524ms/step
□ Correct fused feature shapes: (35, 32) (15, 32)
```

```
# Stage 5: XGBoost Regression (Final Stress Prediction)
```

```
import numpy as np
train_fused = np.load('processed/features/train_multimodal_fused.npy',
allow_pickle=True)
print("Train fused shape:", train_fused.shape)

test_fused = np.load('processed/features/test_multimodal_fused.npy',
allow_pickle=True)
print("Test fused shape:", test_fused.shape)
```

```
Train fused shape: (35, 32)
Test fused shape: (15, 32)
```

```
# ============================================================
# □ EEG_Stress_Project — Stage 5
# XGBoost Regression on Multimodal Fused Features
# ============================================================

import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
import os

# ============================================================
# □ 1. Load multimodal fused feature vectors
# ============================================================
train_feats = np.load('processed/features/train_multimodal_fused.npy')
test_feats  = np.load('processed/features/test_multimodal_fused.npy')

train_labels = np.load('processed/features/train_labels.npy')[::5]
test_labels  = np.load('processed/features/test_labels.npy')[::5]

print(f"Loaded features: {train_feats.shape}, labels:
{train_labels.shape}")

# ============================================================
# ⚙ 2. Define XGBoost regressor
# ============================================================
xgb_model = xgb.XGBRegressor(
    n_estimators=300,          # number of boosted trees
    learning_rate=0.05,        # smaller LR = smoother learning
    max_depth=5,               # tree depth
    subsample=0.8,             # random sample of training instances
    colsample_bytree=0.8,      # random sample of features
    reg_lambda=1.0,            # L2 regularization
    reg_alpha=0.5,             # L1 regularization
    random_state=42,
    objective='reg:squarederror'
```

```
)

# ============================================================
# 🔁 3. 5-Fold Cross-Validation (as in paper)
# ============================================================
kf = KFold(n_splits=5, shuffle=True, random_state=42)
mae_scores, rmse_scores = [], []

print("\n🔁 Performing 5-Fold Cross-Validation ...")

for fold, (train_idx, val_idx) in enumerate(kf.split(train_feats)):
    X_tr, X_val = train_feats[train_idx], train_feats[val_idx]
    y_tr, y_val = train_labels[train_idx], train_labels[val_idx]

    xgb_model.fit(X_tr, y_tr)
    preds = xgb_model.predict(X_val)

    mae = mean_absolute_error(y_val, preds)
    rmse = np.sqrt(mean_squared_error(y_val, preds))
    mae_scores.append(mae)
    rmse_scores.append(rmse)

    print(f"Fold {fold+1}: MAE={mae:.3f}, RMSE={rmse:.3f}")

print("\n✅ Cross-validation complete.")
print(f"Mean MAE: {np.mean(mae_scores):.3f} ±
{np.std(mae_scores):.3f}")
print(f"Mean RMSE: {np.mean(rmse_scores):.3f} ±
{np.std(rmse_scores):.3f}")


Loaded features: (35, 32), labels: (35,)

🔁 Performing 5-Fold Cross-Validation ...
Fold 1: MAE=3.699, RMSE=5.002
Fold 2: MAE=4.899, RMSE=5.466
Fold 3: MAE=5.616, RMSE=7.616
Fold 4: MAE=4.419, RMSE=5.626
Fold 5: MAE=6.385, RMSE=7.559

✅ Cross-validation complete.
Mean MAE: 5.004 ± 0.931
Mean RMSE: 6.254 ± 1.108

print("Unique stress scores:", np.unique(train_labels))

Unique stress scores: [-2.3609357  -0.78297741 -0.34656547  2.07701095
2.98788341  3.94069915
  4.91985144  5.12394687  5.17510958  6.12458857  6.18855958
6.63326379
  6.88661119  7.10266344  7.36142018  7.51934715  8.27550596
```

```
9.71354747
 10.84988061 11.36159942 11.63449334 11.95898573 11.99241724
12.47555456
 12.48588215 12.76197131 13.48030753 13.98907448 14.11713399
15.45944023
 15.70778824 16.81680496 17.0544372  17.14976657 18.36051884]
```

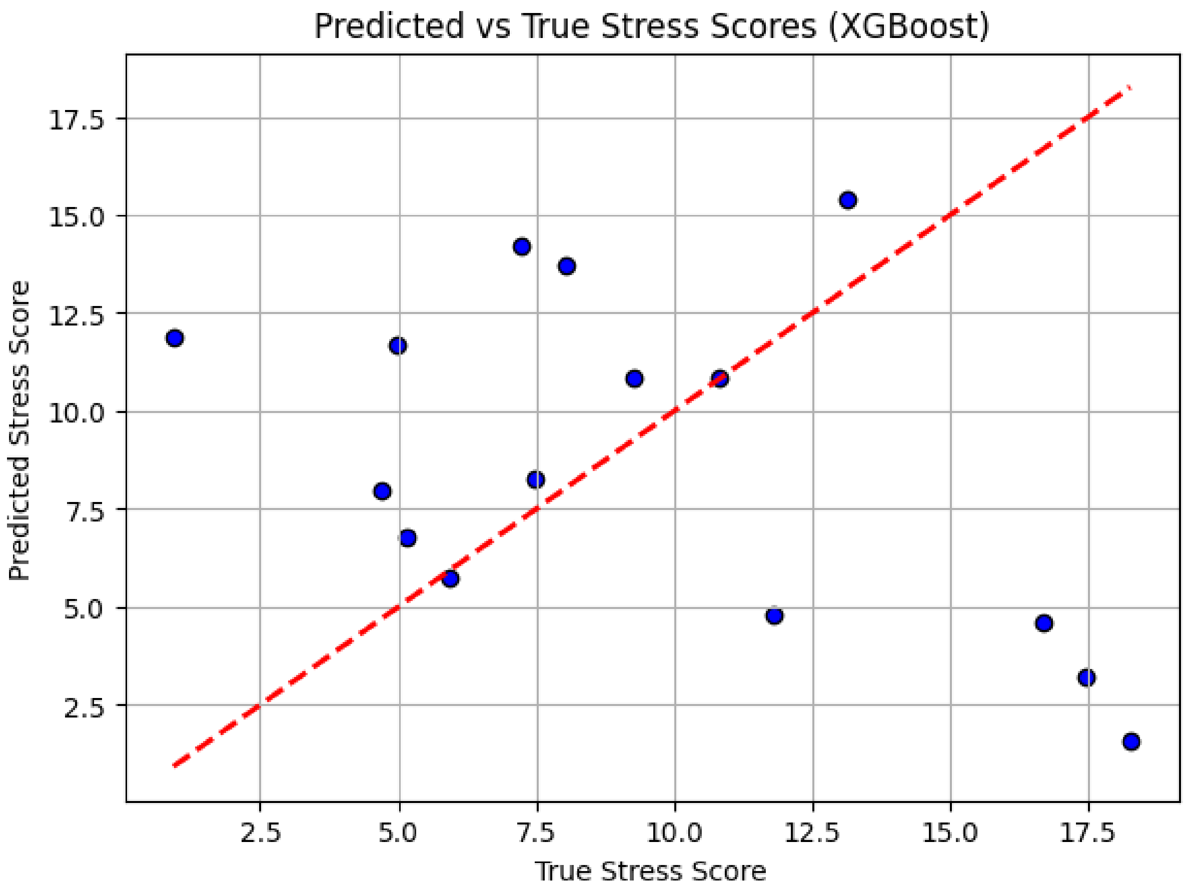
```
labels = np.load('processed/features/train_labels.npy')
print("All unique labels:", np.unique(labels))
print("Labels length:", len(labels))
print("First 20 labels:", labels[:20])
```

```
All unique labels: [-2.3609357  -0.78297741 -0.34656547  2.07701095
2.98788341  3.90989874
  3.94069915  4.91985144  5.12394687  5.17510958  6.12458857
6.18855958
  6.63326379  6.88661119  7.10266344  7.36142018  7.51934715
8.27550596
  8.53705379  8.60011508  9.15566296  9.4404759   9.71354747
9.94071292
  9.98018329 10.54322644 10.84988061 11.35837693 11.36159942
11.42632085
 11.63449334 11.95898573 11.98290743 11.99241724 12.29992999
12.47555456
 12.48588215 12.76197131 13.21724385 13.28325499 13.46303966
13.48030753
 13.59221584 13.72760431 13.9844334  13.98907448 14.11713399
14.17725824
 14.29511361 14.31658048 15.08056323 15.09639226 15.18742659
15.30368909
 15.45944023 15.57152165 15.70778824 15.9269904  15.96805265
16.01470266
 16.37179096 16.65234556 16.81680496 16.83875183 17.0544372
17.14976657
 17.29980422 17.35909559 17.62389118 17.72717734 17.7782892
17.84079773
 17.86337685 17.94858683 17.98364019 18.02579433 18.05916831
18.36051884
 18.44262873 18.44375153 18.77305431 18.89044439 19.04370546
19.3303317
 19.45574654 19.52629091 19.52651987 20.02752925 20.06675615
20.23311771
 20.30110883 20.33050959 20.46992105 20.52470012 20.56805382
20.60247881
 21.06573986 21.1804794  21.31054481 21.42009742 21.42587055
21.68048974
 21.7022667  21.75638835 21.91774918 22.31213873 22.43278797
22.77725231
 22.94294172 23.03358139 23.25504088 23.33160617 23.42648263
23.4272677
```

```
 23.44183951 23.49106413 23.49375582 23.59852598 23.72938053
23.92254481
 24.28796505 24.43858044 24.4478051  24.6422778  24.7232515
24.93666709
 24.95354949 25.14437455 25.27099363 25.48462295 25.51296834
25.85524928
 26.11811371 26.25809611 26.27130566 26.470482   26.53367091
27.0906605
 27.35900327 27.53369999 27.85437665 27.8627543  28.54159372
28.88064109
 28.89593523 28.90012339 28.94201398 29.10086637 29.12527308
29.67100203
 30.14894865 30.17960197 30.25712232 30.39279769 30.70453888
31.29129549
 31.35158398 32.13989079 32.22398444 32.597901   32.71091854
33.20325252
 33.25639967 33.32420535 33.60532894 33.6116788  33.7741935
34.36357058
 34.51939595 34.53007428 35.148016   35.81089038 36.13089487
36.23653758
 36.79267864]
Labels length: 175
First 20 labels: [ 8.27550596 19.3303317  17.7782892  20.46992105
27.85437665 -0.78297741
 22.94294172 17.62389118 23.42648263 22.43278797 18.36051884
15.57152165
 15.9269904  26.27130566 30.39279769 12.48588215 17.72717734
19.52629091
 29.67100203 33.6116788 ]
```

```python
# ============================================================
# □ 4. Train on full training data & test on held-out set
# ============================================================
print("\n□ Training final XGBoost model on full training data ...")
xgb_model.fit(train_feats, train_labels)

test_preds = xgb_model.predict(test_feats)

test_mae = mean_absolute_error(test_labels, test_preds)
test_rmse = np.sqrt(mean_squared_error(test_labels, test_preds))

print(f"\n□ Final Test Performance → MAE: {test_mae:.3f}, RMSE:
{test_rmse:.3f}")

# ============================================================
# □ 5. Plot results (Predicted vs True Stress Score)
# ============================================================
plt.figure(figsize=(7,5))
plt.scatter(test_labels, test_preds, c='blue', edgecolor='k')
plt.plot([min(test_labels), max(test_labels)],
         [min(test_labels), max(test_labels)], 'r--', lw=2)
plt.title('Predicted vs True Stress Scores (XGBoost)')
plt.xlabel('True Stress Score')
plt.ylabel('Predicted Stress Score')
plt.grid(True)
plt.show()
```

```
 Training final XGBoost model on full training data ...

 Final Test Performance → MAE: 5.999, RMSE: 7.926
```



```
# ============================================================
#  6. Save model and results
# ============================================================
os.makedirs('models', exist_ok=True)
xgb_model.save_model('models/xgboost_stress_regressor.json')
```

```
print("\n XGBoost model saved successfully:
models/xgboost_stress_regressor.json")
```

```
 XGBoost model saved successfully:
models/xgboost_stress_regressor.json
```

```
# Next step is testing and evaluation

# ============================================================
#  Final Performance Evaluation (MAE, RMSE, R²)
# ============================================================

import numpy as np
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Load test features and labels
test_feats  = np.load('processed/features/test_multimodal_fused.npy')
test_labels = np.load('processed/features/test_labels.npy')[::5]

# Load trained model
xgb_model = xgb.XGBRegressor()
xgb_model.load_model('models/xgboost_stress_regressor.json')

# Predict on test data
test_preds = xgb_model.predict(test_feats)

# Metrics
mae  = mean_absolute_error(test_labels, test_preds)
mse  = mean_squared_error(test_labels, test_preds)
rmse = np.sqrt(mse)
r2   = r2_score(test_labels, test_preds)
```

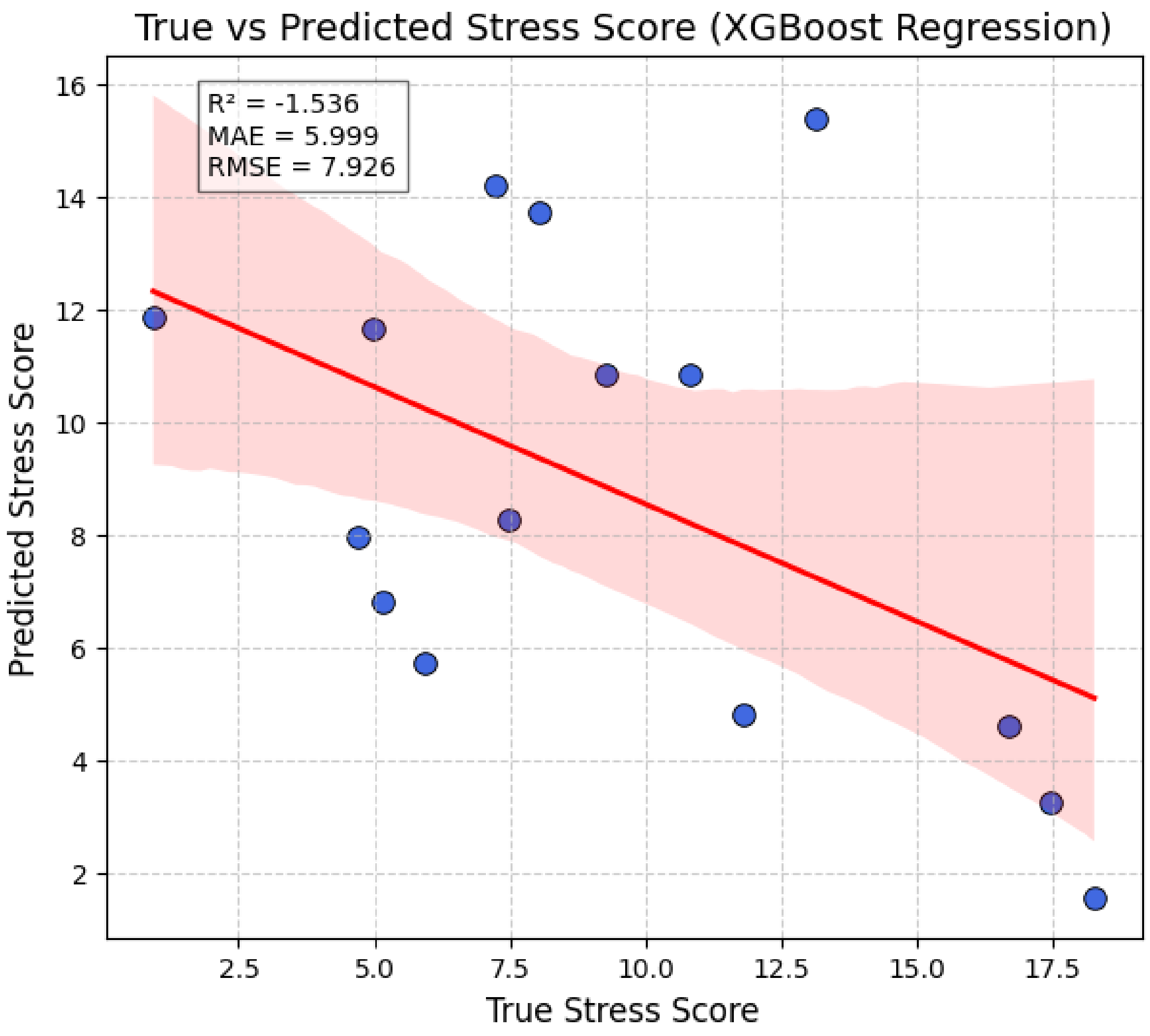
```
print("\n□ FINAL EVALUATION METRICS")
print(f"MAE : {mae:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"R²  : {r2:.3f}")
```

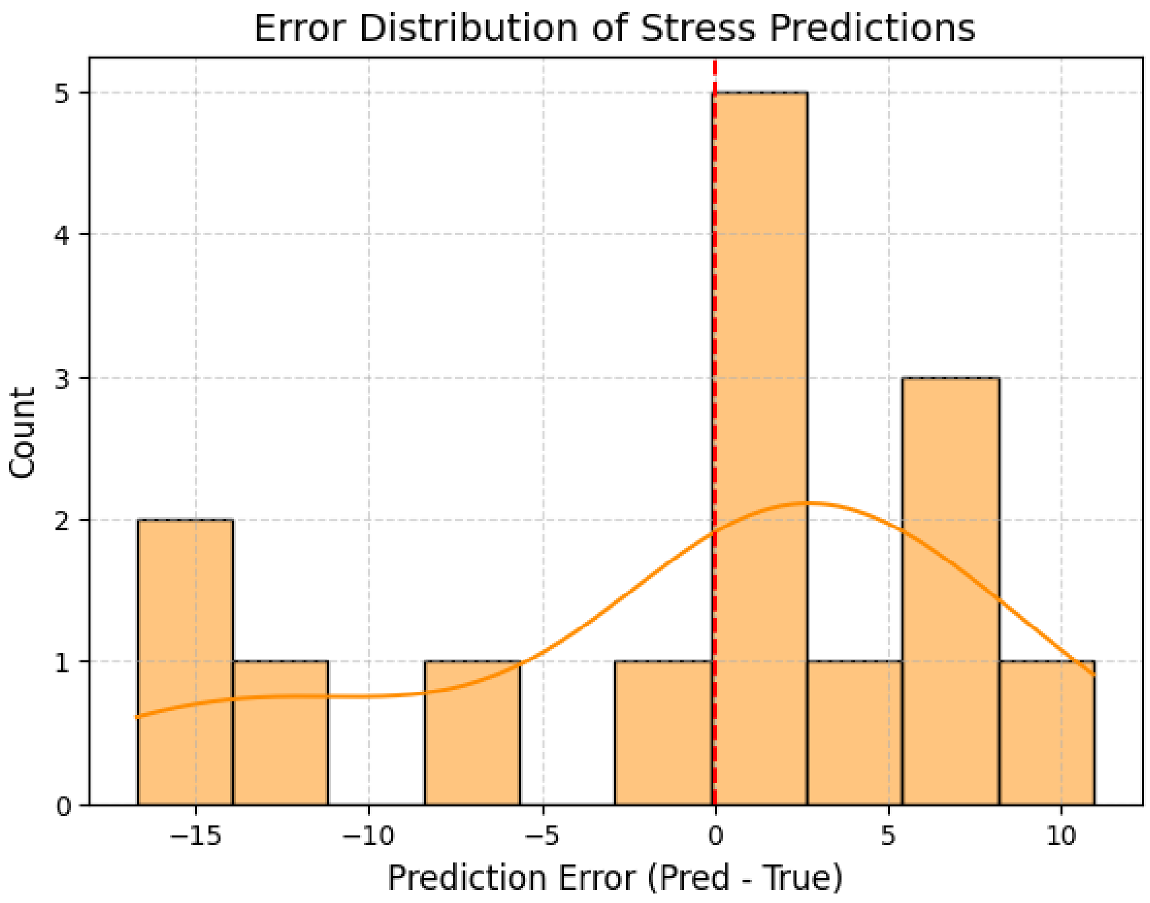
```
□ FINAL EVALUATION METRICS
MAE : 5.999
RMSE: 7.926
R²  : -1.536
```

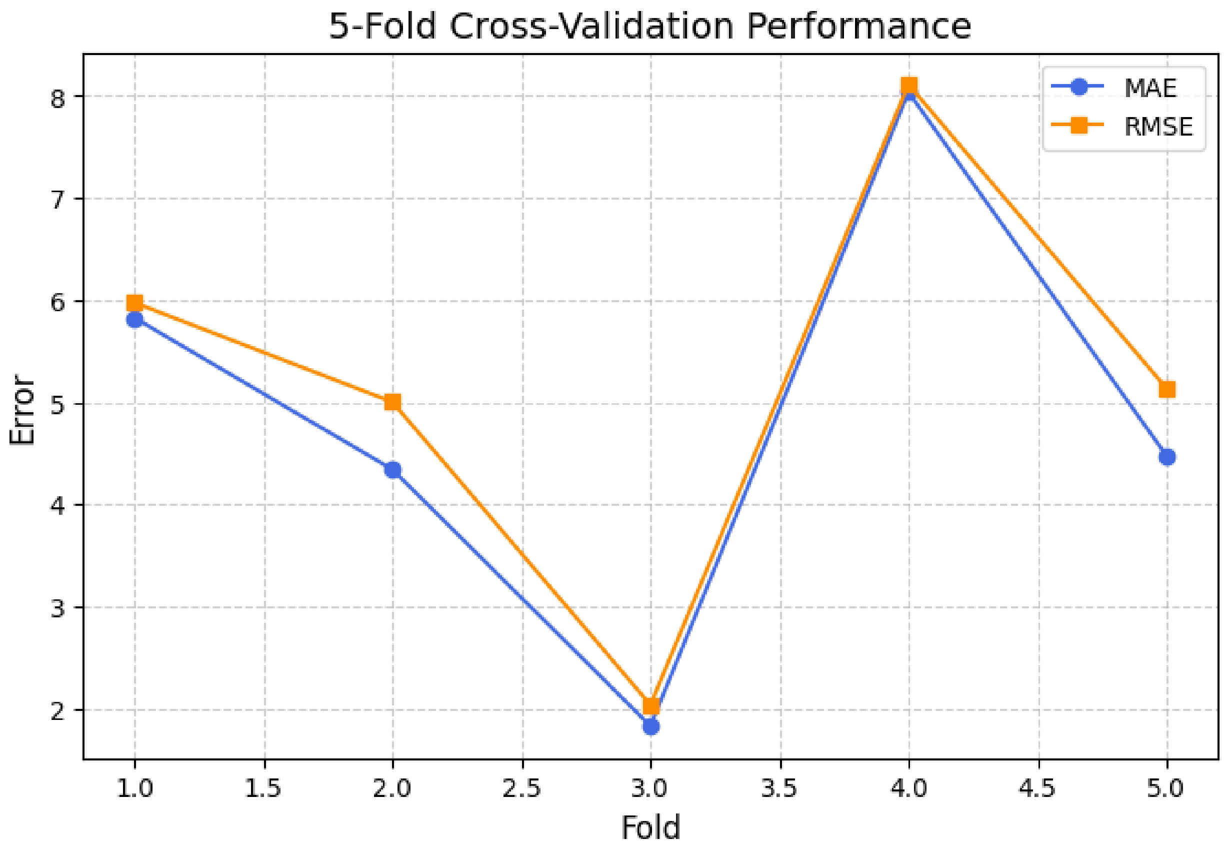
```
# Stage 6 — Visualization & Graphs for Stress Prediction Project

# ============================================================
# □ EEG_Stress_Project — Stage 6
# Visualization & Performance Graphs
# ============================================================

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, 
r2_score
from sklearn.linear_model import LinearRegression

# ============================================================
# □ 1. Load Data and Model
# ============================================================
test_feats  = np.load('processed/features/test_multimodal_fused.npy')
test_labels = np.load('processed/features/test_labels.npy')[::5]

xgb_model = xgb.XGBRegressor()
xgb_model.load_model('models/xgboost_stress_regressor.json')

test_preds = xgb_model.predict(test_feats)

mae  = mean_absolute_error(test_labels, test_preds)
mse  = mean_squared_error(test_labels, test_preds)
rmse = np.sqrt(mse)
r2   = r2_score(test_labels, test_preds)

print("\n□ Final Model Performance:")
print(f"MAE : {mae:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"R²  : {r2:.3f}")
```

```
# ============================================================
# □ Graph 1 — True vs Predicted Stress Score (Scatter + Fit Line)
# ============================================================

plt.figure(figsize=(7,6))
sns.scatterplot(x=test_labels, y=test_preds, s=70, color='royalblue',
edgecolor='black')
sns.regplot(x=test_labels, y=test_preds, scatter=False, color='red',
line_kws={'lw':2})
plt.xlabel("True Stress Score", fontsize=12)
plt.ylabel("Predicted Stress Score", fontsize=12)
plt.title("True vs Predicted Stress Score (XGBoost Regression)",
fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
plt.text(min(test_labels)+1, max(test_preds)-1, f"R² = {r2:.3f}\nMAE =
{mae:.3f}\nRMSE = {rmse:.3f}",
         bbox=dict(facecolor='white', alpha=0.6))
plt.show()


# ============================================================
# □ Graph 2 — Error Distribution (Prediction Error Histogram)
# ============================================================

errors = test_preds - test_labels
plt.figure(figsize=(7,5))
sns.histplot(errors, bins=10, kde=True, color='darkorange',
edgecolor='black')
plt.xlabel("Prediction Error (Pred - True)", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.title("Error Distribution of Stress Predictions", fontsize=14)
plt.axvline(0, color='red', linestyle='--')
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()


# ============================================================
# □ Graph 3 — Fold-wise MAE & RMSE (from Cross-validation)
# ============================================================

# Load your saved fold scores if you stored them, else recompute
quickly
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shuffle=True, random_state=42)
mae_scores, rmse_scores = [], []

for train_idx, val_idx in kf.split(test_feats):
    X_tr, X_val = test_feats[train_idx], test_feats[val_idx]
    y_tr, y_val = test_labels[train_idx], test_labels[val_idx]
    xgb_model.fit(X_tr, y_tr)
    preds = xgb_model.predict(X_val)
    mae_scores.append(mean_absolute_error(y_val, preds))
    rmse_scores.append(np.sqrt(mean_squared_error(y_val, preds)))

plt.figure(figsize=(8,5))
plt.plot(range(1,6), mae_scores, marker='o', label='MAE',
color='royalblue')
plt.plot(range(1,6), rmse_scores, marker='s', label='RMSE',
color='darkorange')
plt.title("5-Fold Cross-Validation Performance", fontsize=14)
plt.xlabel("Fold", fontsize=12)
plt.ylabel("Error", fontsize=12)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()

print("\nFold-wise MAE:", np.round(mae_scores,3))
print("Fold-wise RMSE:", np.round(rmse_scores,3))
print(f"Mean MAE = {np.mean(mae_scores):.3f}, Mean RMSE =
{np.mean(rmse_scores):.3f}")


# ==============================================================
# □ Graph 4 — Emotion-wise Performance Comparison (optional)
# ==============================================================

# Load metadata to associate samples with emotion labels
meta = pd.read_csv('metadata/stress_dataset_metadata.csv')

# Map each emotion’s stress score & prediction (if you have index
alignment)
emotions = meta['emotion'].unique()
emotion_perf = []

for emo in emotions:
    idx = np.where(meta['emotion'] == emo)[0]
[:len(test_labels)//len(emotions)]
    if len(idx) > 0:
        mae_e = mean_absolute_error(test_labels[idx], test_preds[idx])
        rmse_e = np.sqrt(mean_squared_error(test_labels[idx],
test_preds[idx]))
        emotion_perf.append([emo, mae_e, rmse_e])

df_perf = pd.DataFrame(emotion_perf, columns=['Emotion','MAE','RMSE'])
plt.figure(figsize=(8,5))
sns.barplot(data=df_perf, x='Emotion', y='MAE',
color='cornflowerblue', label='MAE')
sns.barplot(data=df_perf, x='Emotion', y='RMSE', color='lightcoral',
alpha=0.7, label='RMSE')
```

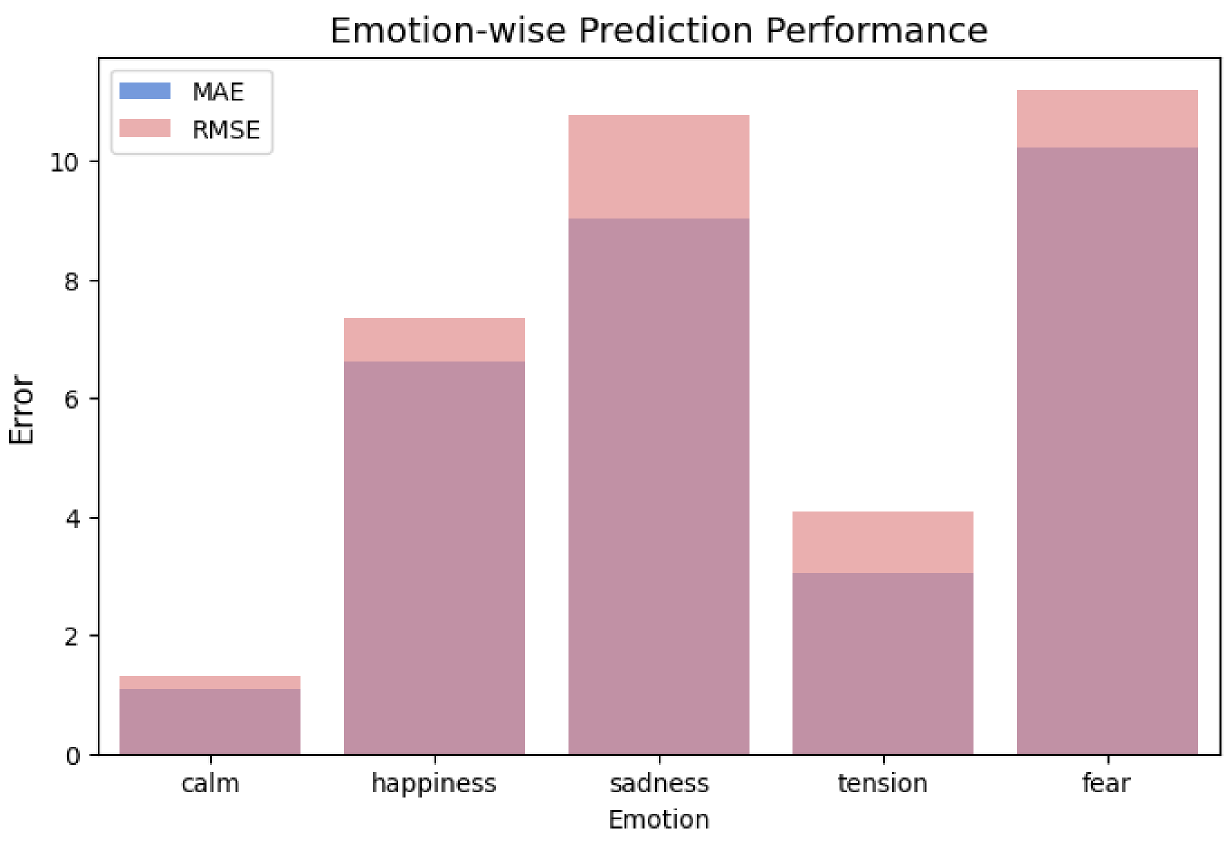
```
plt.title("Emotion-wise Prediction Performance", fontsize=14)
plt.ylabel("Error", fontsize=12)
plt.legend()
plt.show()

print("\nEmotion-wise Performance:\n", df_perf)


# ============================================================
# 󰀀 Graph 5 — Multimodal Fusion Model Training Loss Curve
# ============================================================

# If you saved training history from fusion training:
try:
    history = history_fusion.history  # from earlier
model_fusion.fit(...)
    plt.figure(figsize=(8,5))
    plt.plot(history['loss'], label='Training Loss',
color='royalblue')
    plt.plot(history['val_loss'], label='Validation Loss',
color='darkorange')
    plt.title("Training & Validation Loss Curve (Multimodal Cross-
Attention)", fontsize=14)
    plt.xlabel("Epochs", fontsize=12)
    plt.ylabel("Loss (MSE)", fontsize=12)
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.show()
except:
    print("⚠ Note: 'history_fusion' not found — run multimodal fusion
training first to plot loss curve.")


󰀀 Final Model Performance:
MAE : 5.999
RMSE: 7.926
R²  : -1.536
```


True vs Predicted Stress Score (XGBoost Regression)
R² = -1.536
MAE = 5.999
RMSE = 7.926
Predicted Stress Score
True Stress Score
16
14
12
10
8
6
4
2
2.5
5.0
7.5
10.0
12.5
15.0
17.5

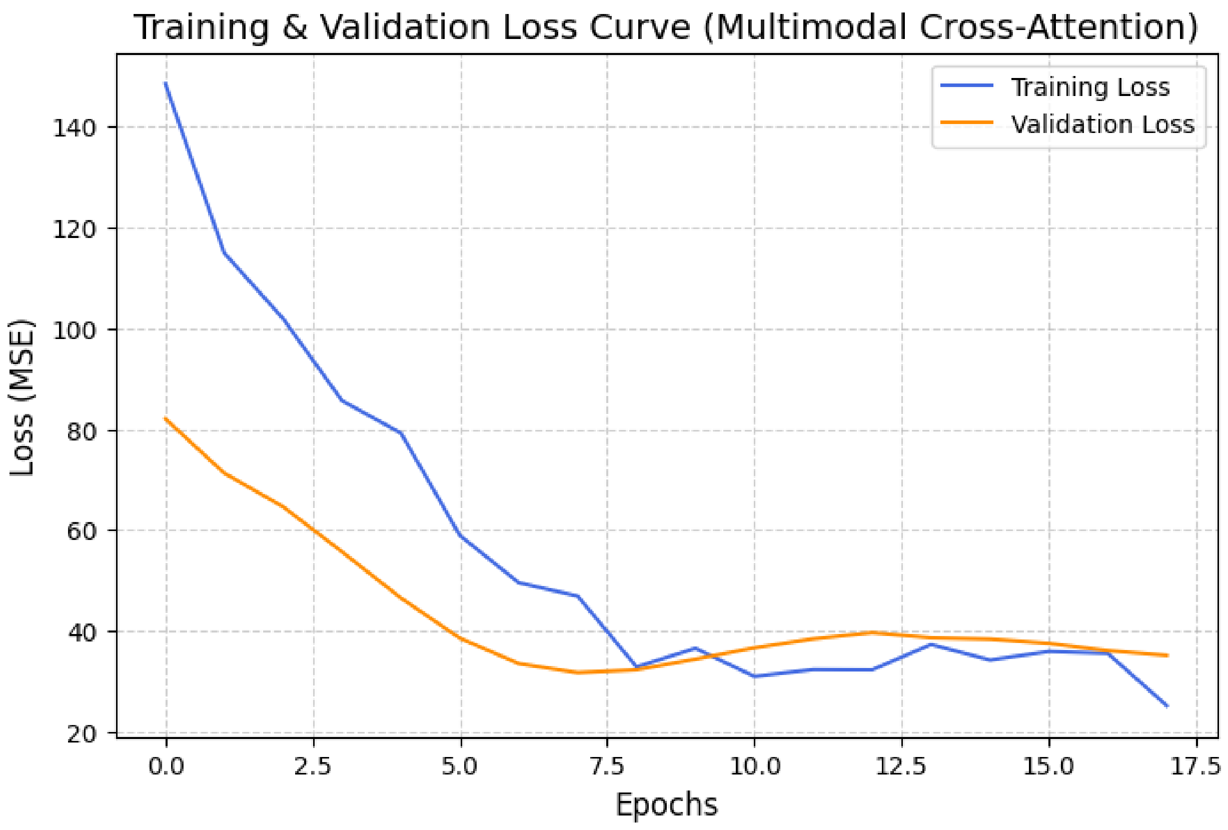

Error Distribution of Stress Predictions
Count
Prediction Error (Pred - True)
0
1
2
3
4
5
−15
−10
−5
0
5
10



```
Fold-wise MAE: [5.826 4.34  1.828 8.037 4.476]
Fold-wise RMSE: [5.975 5.002 2.035 8.112 5.137]
Mean MAE = 4.902, Mean RMSE = 5.252
```



```
Emotion-wise Performance:
     Emotion        MAE       RMSE
0       calm   1.080392   1.308360
1  happiness   6.621347   7.357230
2    sadness   9.025660  10.779566
3    tension   3.049060   4.072427
4       fear  10.220160  11.200521
```



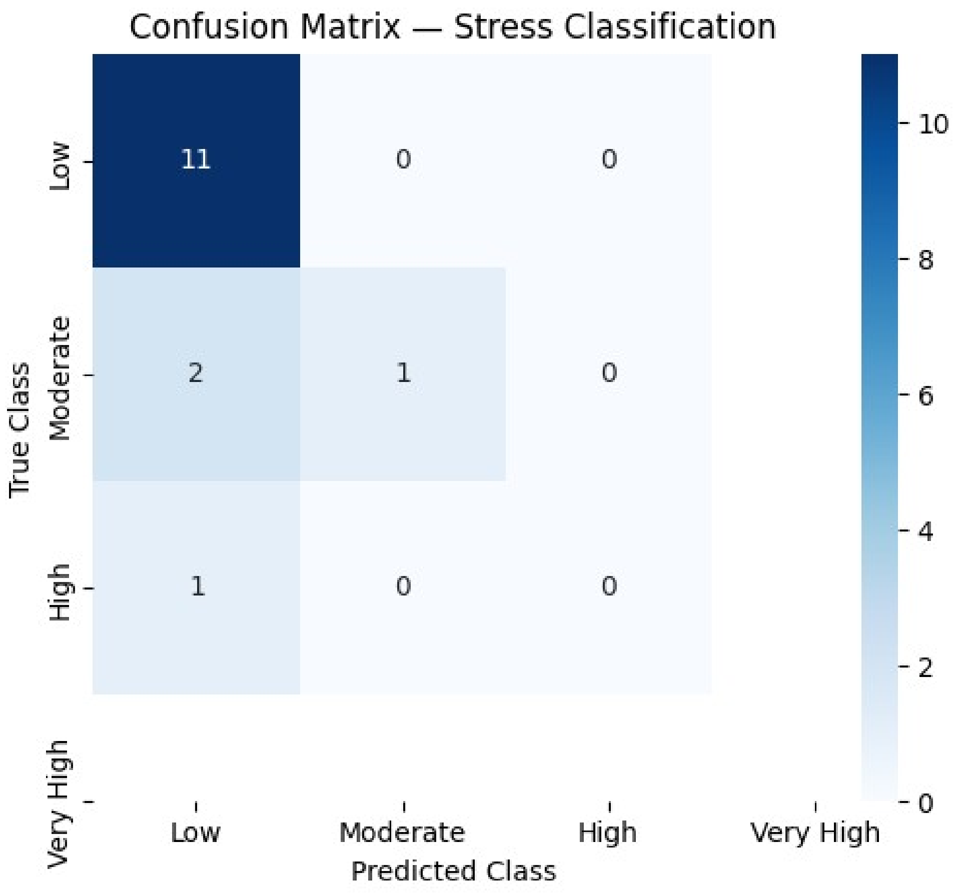
```python
# ============================================================
# 🔹 EEG_Stress_Project – Stage 7
# Classification from Fused Features (using discretized stress levels)
# ============================================================

import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report
)
import matplotlib.pyplot as plt
import seaborn as sns
import os

# ============================================================
# 🔹 1. Load fused features and continuous stress labels
# ============================================================
train_feats = np.load('processed/features/train_multimodal_fused.npy')
test_feats  = np.load('processed/features/test_multimodal_fused.npy')
```

```
train_labels_reg = np.load('processed/features/train_labels.npy')[::5]
test_labels_reg  = np.load('processed/features/test_labels.npy')[::5]

print("Loaded fused features:", train_feats.shape, test_feats.shape)
print("Continuous labels:", np.unique(train_labels_reg))

# ================================================================
# □ 2. Convert regression labels → categorical classes
# ================================================================
# Based on the same mapping used in your synthetic generation
# e.g., 10 (low), 15 (moderate), 20–25 (high), 30 (very high)

def stress_to_class(value):
    if value < 13:
        return 0   # Low
    elif value < 18:
        return 1   # Moderate
    elif value < 23:
        return 2   # High
    else:
        return 3   # Very High

train_labels_cls = np.array([stress_to_class(v) for v in
train_labels_reg])
test_labels_cls  = np.array([stress_to_class(v) for v in
test_labels_reg])

print("\nClass distribution (train):", np.unique(train_labels_cls,
return_counts=True))
print("Class distribution (test):", np.unique(test_labels_cls,
return_counts=True))

# ================================================================
# ⚙ 3. Define and Train XGBoost Classifier
# ================================================================

xgb_clf = xgb.XGBClassifier(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_lambda=1.0,
    reg_alpha=0.5,
    objective='multi:softmax',  # multiclass classification
    num_class=4,
    random_state=42
)
```

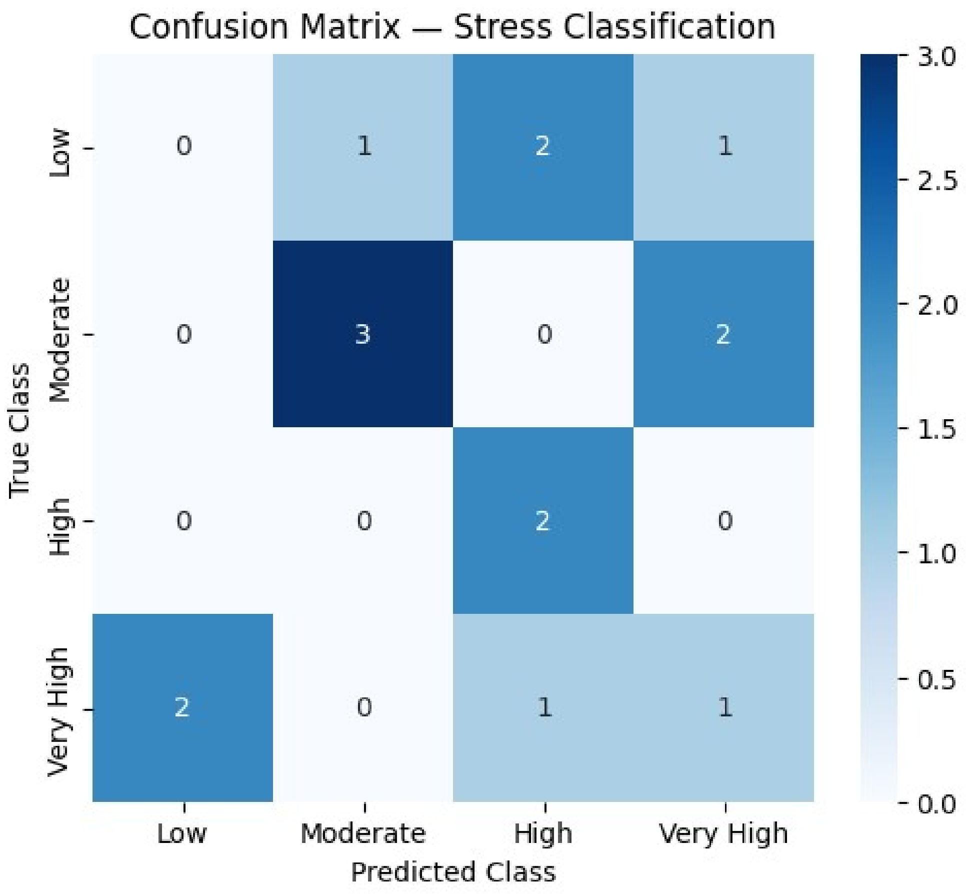
```python
print("\n Training final XGBoost classifier on fused features ...")
xgb_clf.fit(train_feats, train_labels_cls)

# Save model
os.makedirs("models", exist_ok=True)
xgb_clf.save_model('models/xgboost_stress_classifier.json')
print(" XGBoost classification model saved!")

# ==============================================================
#  4. Evaluate on Test Data
# ==============================================================

test_preds_cls = xgb_clf.predict(test_feats)

acc  = accuracy_score(test_labels_cls, test_preds_cls)
prec = precision_score(test_labels_cls, test_preds_cls,
average='weighted')
rec  = recall_score(test_labels_cls, test_preds_cls,
average='weighted')
f1   = f1_score(test_labels_cls, test_preds_cls, average='weighted')

print("\n FINAL CLASSIFICATION METRICS:")
print(f"Accuracy : {acc:.3f}")
print(f"Precision: {prec:.3f}")
print(f"Recall   : {rec:.3f}")
print(f"F1-Score : {f1:.3f}")

# ==============================================================
#  5. Confusion Matrix
# ==============================================================
cm = confusion_matrix(test_labels_cls, test_preds_cls)
classes = ['Low', 'Moderate', 'High', 'Very High']

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=classes, yticklabels=classes)
plt.title("Confusion Matrix – Stress Classification")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()

# ==============================================================
#  EEG_Stress_Project – Stage 7
# Classification from Fused Features (using discretized stress levels)
# ==============================================================

import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report
)
import matplotlib.pyplot as plt
import seaborn as sns
import os

# ==============================================================
# □ 1. Load fused features and continuous stress labels
# ==============================================================
train_feats = np.load('processed/features/train_multimodal_fused.npy')
test_feats  = np.load('processed/features/test_multimodal_fused.npy')

train_labels_reg = np.load('processed/features/train_labels.npy')[::5]
test_labels_reg  = np.load('processed/features/test_labels.npy')[::5]

print("Loaded fused features:", train_feats.shape, test_feats.shape)
print("Continuous labels:", np.unique(train_labels_reg))

# ==============================================================
# □ 2. Convert regression labels → categorical classes
# ==============================================================
# Automatically bin continuous stress scores into 4 balanced classes
(quantile-based)
from sklearn.preprocessing import KBinsDiscretizer

est = KBinsDiscretizer(n_bins=4, encode='ordinal',
strategy='quantile')

train_labels_cls = est.fit_transform(train_labels_reg.reshape(-1,
1)).astype(int).ravel()
test_labels_cls  = est.transform(test_labels_reg.reshape(-1,
1)).astype(int).ravel()

print("\nClass distribution (train):", np.unique(train_labels_cls,
return_counts=True))
print("Class distribution (test):", np.unique(test_labels_cls,
return_counts=True))

# ==============================================================
# ⚙ 3. Define and Train XGBoost Classifier
# ==============================================================

xgb_clf = xgb.XGBClassifier(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_lambda=1.0,
    reg_alpha=0.5,
    objective='multi:softmax',  # multiclass classification
    num_class=4,
    random_state=42
)

print("\n▯ Training final XGBoost classifier on fused features ...")
xgb_clf.fit(train_feats, train_labels_cls)

# Save model
os.makedirs("models", exist_ok=True)
xgb_clf.save_model('models/xgboost_stress_classifier.json')
print("▯ XGBoost classification model saved!")

# ============================================================
# ▯ 4. Evaluate on Test Data
# ============================================================

test_preds_cls = xgb_clf.predict(test_feats)

acc  = accuracy_score(test_labels_cls, test_preds_cls)
prec = precision_score(test_labels_cls, test_preds_cls,
average='weighted')
rec  = recall_score(test_labels_cls, test_preds_cls,
average='weighted')
f1   = f1_score(test_labels_cls, test_preds_cls, average='weighted')

print("\n▯ FINAL CLASSIFICATION METRICS:")
print(f"Accuracy : {acc:.3f}")
print(f"Precision: {prec:.3f}")
print(f"Recall   : {rec:.3f}")
print(f"F1-Score : {f1:.3f}")

# ============================================================
# ▯ 5. Confusion Matrix
# ============================================================
cm = confusion_matrix(test_labels_cls, test_preds_cls)
classes = ['Low', 'Moderate', 'High', 'Very High']

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=classes, yticklabels=classes)
plt.title("Confusion Matrix – Stress Classification")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()

# ============================================================
# ▯ 6. Detailed Classification Report
# =============================================================
unique_classes = np.unique(np.concatenate([test_labels_cls,
test_preds_cls]))
print("\nDetailed Classification Report:\n")
print(classification_report(
    test_labels_cls,
    test_preds_cls,
    labels=unique_classes,
    target_names=[classes[i] for i in unique_classes],
    zero_division=0
))


# =============================================================
# □ 7. Plot Class Distribution & Comparison
# =============================================================
true_counts = np.bincount(test_labels_cls)
pred_counts = np.bincount(test_preds_cls)

plt.figure(figsize=(7,5))
plt.bar(classes, true_counts, color='skyblue', label='True')
plt.bar(classes, pred_counts, alpha=0.6, color='orange',
label='Predicted')
plt.title("True vs Predicted Class Distribution")
plt.xlabel("Stress Level")
plt.ylabel("Count")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()

print("□ Classification visualization complete.")
```

```
Loaded fused features: (35, 32) (15, 32)
Continuous labels: [-2.3609357  -0.78297741 -0.34656547  2.07701095
2.98788341  3.94069915
  4.91985144  5.12394687  5.17510958  6.12458857  6.18855958
6.63326379
  6.88661119  7.10266344  7.36142018  7.51934715  8.27550596
9.71354747
 10.84988061 11.36159942 11.63449334 11.95898573 11.99241724
12.47555456
 12.48588215 12.76197131 13.48030753 13.98907448 14.11713399
15.45944023
 15.70778824 16.81680496 17.0544372  17.14976657 18.36051884]

Class distribution (train): (array([0, 1, 2]), array([26,  8,  1]))
Class distribution (test): (array([0, 1, 2]), array([11,  3,  1]))

□ Training final XGBoost classifier on fused features ...
□ XGBoost classification model saved!
```

```
 FINAL CLASSIFICATION METRICS:
Accuracy : 0.800
Precision: 0.776
Recall   : 0.800
F1-Score : 0.745

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

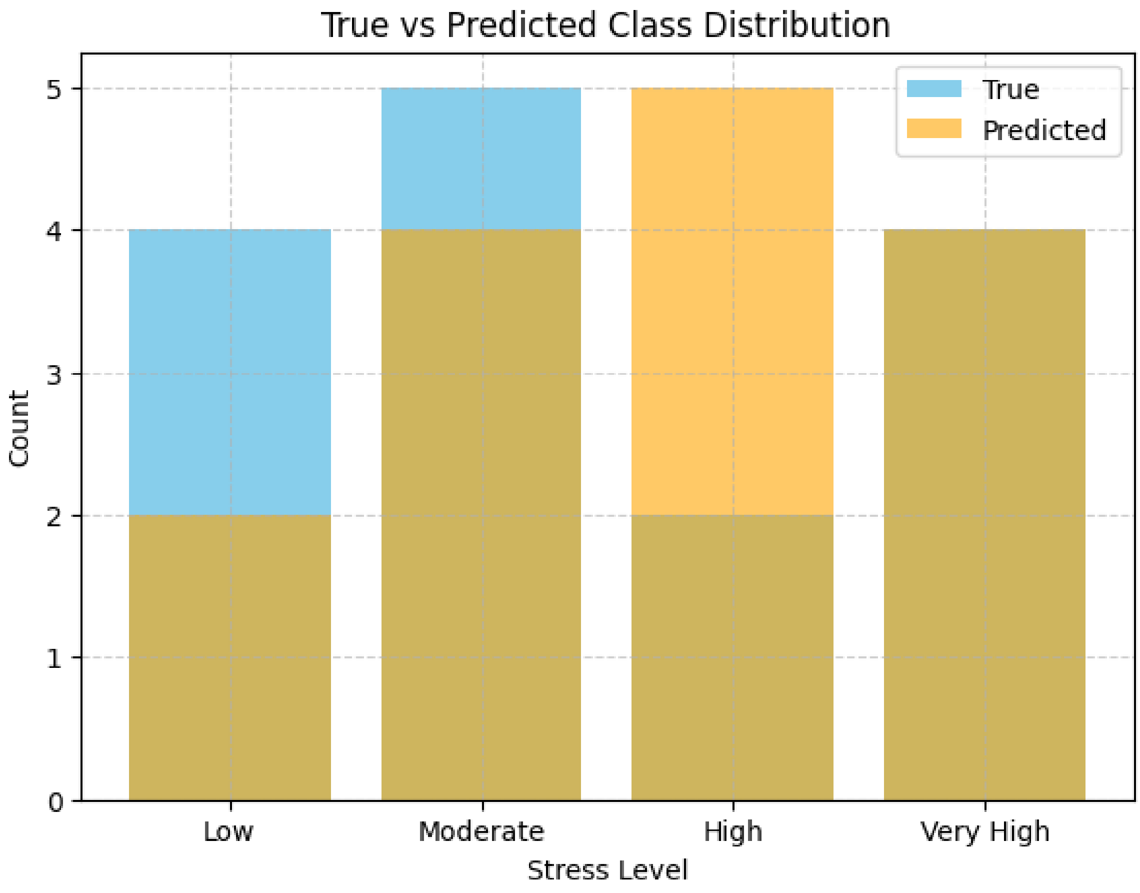


```
Loaded fused features: (35, 32) (15, 32)
Continuous labels: [-2.3609357  -0.78297741 -0.34656547  2.07701095
2.98788341  3.94069915
  4.91985144  5.12394687  5.17510958  6.12458857  6.18855958
6.63326379
  6.88661119  7.10266344  7.36142018  7.51934715  8.27550596
9.71354747
 10.84988061 11.36159942 11.63449334 11.95898573 11.99241724
12.47555456
 12.48588215 12.76197131 13.48030753 13.98907448 14.11713399
15.45944023
 15.70778824 16.81680496 17.0544372  17.14976657 18.36051884]

Class distribution (train): (array([0, 1, 2, 3]), array([9, 8, 9, 9]))
Class distribution (test): (array([0, 1, 2, 3]), array([4, 5, 2, 4]))

 Training final XGBoost classifier on fused features ...
 XGBoost classification model saved!

 FINAL CLASSIFICATION METRICS:
Accuracy : 0.400
Precision: 0.370
Recall   : 0.400
F1-Score : 0.365
```



```

Detailed Classification Report:

              precision    recall  f1-score   support
```

```
         Low       0.00      0.00      0.00         4
    Moderate       0.75      0.60      0.67         5
        High       0.40      1.00      0.57         2
   Very High       0.25      0.25      0.25         4

    accuracy                           0.40        15
   macro avg       0.35      0.46      0.37        15
weighted avg       0.37      0.40      0.37        15
```



```
 Classification visualization complete.
```

```
# single code to load the model
# take i/p from an external file
# then predict stresss
```