DATA PREPROCESSING

```
import librosa
import scipy.signal
import webrtcvad
TARGET_SR = 16000
def wiener_filter(signal):
    return scipy.signal.wiener(signal)
def normalize_audio(signal):
    return signal / np.max(np.abs(signal)) if
np.max(np.abs(signal)) != 0 else signal
def resample_audio(signal, orig_sr, target_sr=TARGET_SR):
    return librosa.resample(signal, orig_sr=orig_sr,
target_sr=target_sr)
def vad_webrtc(signal, sr, frame_ms=30, mode=3):
    vad = webrtcvad.Vad(mode)
    frame_len = int(sr * frame_ms / 1000)
    voiced = np.array([], dtype=np.float32)
    for i in range(0, len(signal), frame_len):
        frame = signal[i:i+frame_len]
        if len(frame) < frame_len:
            break
        pcm = (frame * 32768).astype(np.int16).tobytes()
        if vad.is_speech(pcm, sr):
            voiced = np.concatenate((voiced, frame))
    return voiced if len(voiced) > 0 else signal
```

DATA AUGMENTATION

```
from audiomentations import AddGaussianNoise,
TimeStretch, PitchShift, Shift, Gain
```

```
augmentations = [
    AddGaussianNoise(min_amplitude=0.001,
max_amplitude=0.015, p=1.0),
    TimeStretch(min_rate=0.8, max_rate=1.2, p=1.0),
    PitchShift(min_semitones=-4, max_semitones=4, p=1.0),
    Shift(min_shift=-0.5, max_shift=0.5, p=1.0),
    Gain(min_gain_db=-6, max_gain_db=6, p=1.0)
]
def apply_augmentation(audio, sr, transform):
    return transform(samples=audio, sample_rate=sr)
```

FEATURE LOADING, ENCODING AND SCALING

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder,
StandardScaler
from sklearn.model_selection import train_test_split
df = pd.read_csv("100wav_wavlm_large_features.csv")
X = df.iloc[:, 2:].values
y = df["emotion"].values
scaler = StandardScaler()
X = scaler.fit_transform(X)
le = LabelEncoder()
y = le.fit_transform(y)
y_cat = tf.keras.utils.to_categorical(y)
X_train, X_test, y_train, y_test = train_test_split(X, y_cat,
test_size=0.2, random_state=42)
X_train = X_train.reshape((X_train.shape[0], 1,
X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

MODEL BUILDING: BILSTM + SELF-ATTENTION

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout,
Bidirectional, LSTM, MultiHeadAttention, LayerNormalization,
GlobalAveragePooling1D
from tensorflow.keras.optimizers import Adam
def build_model(seq_len, feat_dim, num_cls):
    inputs = Input(shape=(seq_len, feat_dim))
    x = Bidirectional(LSTM(64,
return_sequences=True))(inputs)
    x_proj = Dense(128)(x)
    attn = MultiHeadAttention(num_heads=4,
key_dim=128)(x_proj, x_proj)
    x = LayerNormalization()(x_proj + attn)
    x = GlobalAveragePooling1D()(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.3)(x)
    outputs = Dense(num_cls, activation='softmax')(x)
    model = Model(inputs, outputs)
    model.compile(optimizer=Adam(1e-
3),loss='categorical_crossentropy', metrics=['accuracy'])
    return model
model = build_model(1, X_train.shape[2], y_cat.shape[1])
model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_data=(X_test, y_test)
```

EVALUATION METRICS

```
from sklearn.metrics import classification_report,
confusion_matrix, balanced_accuracy_score, r2_score
```

```
y_pred = np.argmax(model.predict(X_test), axis=1)
y_true = np.argmax(y_test, axis=1)
print("Accuracy:", model.evaluate(X_test, y_test)[1])
print("Balanced Accuracy:", balanced_accuracy_score(y_true,
y_pred))
print("R² Score:", r2_score(y_true, y_pred))

print("Classification Report:\n", classification_report(y_true,
y_pred))
```