import numpy as np  
import pandas as pd  
import torch  
import torch.nn as nn  
import matplotlib.pyplot as plt  
import seaborn as snsfrom sklearn.preprocessing import MinMaxScaler  
from torch.utils.data import DataLoader, TensorDataset  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')  
print(f"Using device: {device}")

data = pd.read\_csv('.csv', encoding='gbk')

num\_samples = len(X\_data)  
train\_size = int(0.8 \* num\_samples)  
val\_size = int(0.1 \* num\_samples)  
test\_size = num\_samples - train\_size - val\_size

class TCN\_BiLSTM\_SEAttention(nn.Module):

def \_\_init\_\_(self, input\_size, num\_channels, hidden\_size, num\_layers, kernel\_size, dropout\_rate=0.5):

super(TCN\_BiLSTM\_SEAttention, self).\_\_init\_\_()

self.tcn = TCN(input\_size, num\_channels, kernel\_size, dropout\_rate)

self.lstm = nn.LSTM(num\_channels[-1], hidden\_size, num\_layers, batch\_first=True, bidirectional=True,

dropout=dropout\_rate)

self.se\_attention = SEAttention(hidden\_size \* 2)

self.fc = nn.Linear(hidden\_size \* 2, 1)

self.dropout = nn.Dropout(dropout\_rate)

def forward(self, x):

tcn\_out = self.tcn(x)

lstm\_out, \_ = self.lstm(tcn\_out)

se\_out, attn\_weights = self.se\_attention(lstm\_out)

x = self.dropout(se\_out)

x = self.fc(x)

return x, attn\_weightsdef train\_model(model, train\_loader, val\_loader, criterion, optimizer, epochs=):  
 train\_losses = []  
 val\_losses = []  
 for epoch in range(epochs):  
 model.train()  
 running\_loss = 0.0  
 for inputs, targets in train\_loader:  
 inputs = inputs.to(device)  
 targets = targets.to(device)  
 optimizer.zero\_grad()  
 outputs, \_ = model(inputs)  
 loss = criterion(outputs, targets)  
 loss.backward()  
 optimizer.step()  
 running\_loss += loss.item()  
 train\_losses.append(running\_loss / len(train\_loader))val\_loss = 0.0  
 model.eval()  
 with torch.no\_grad():  
 for val\_inputs, val\_targets in val\_loader:  
 val\_inputs = val\_inputs.to(device)  
 val\_targets = val\_targets.to(device)  
 val\_outputs, \_ = model(val\_inputs)  
 v\_loss = criterion(val\_outputs, val\_targets)  
 val\_loss += v\_loss.item()  
 val\_losses.append(val\_loss / len(val\_loader))print(f"Epoch {epoch + 1}/{epochs}, Loss: {running\_loss / len(train\_loader):.4f}, Val Loss: {val\_loss / len(val\_loader):.4f}")  
 return train\_losses, val\_losses  
train\_losses, val\_losses = train\_model(model, train\_loader, val\_loader, criterion, optimizer, epochs=)mse = mean\_squared\_error(train\_actuals, train\_predictions)  
rmse = np.sqrt(mse)  
mae = mean\_absolute\_error(train\_actuals, train\_predictions)  
r2 = r2\_score(train\_actuals, train\_predictions)  
print(f"MSE: {mse}")  
print(f"RMSE: {rmse}")  
print(f"MAE: {mae}")  
print(f"R² Score: {r2}")plt.figure(figsize=(10, 6))  
plt.plot(range(len(train\_actuals)), train\_actuals, label='Actual', linestyle='-', marker='o')  
plt.plot(range(len(train\_predictions)), train\_predictions, label='Predicted', linestyle='--', marker='x')  
plt.xlabel('Sample Index')  
plt.ylabel('Stress')  
plt.title('Predicted vs Actual Stress on Training Set')  
plt.legend()  
plt.show()plt.figure(figsize=(10, 6))  
plt.scatter(train\_actuals, train\_predictions, s=25, c='b', label='Predicted vs Actual')  
plt.plot([min(train\_actuals), max(train\_actuals)], [min(train\_actuals), max(train\_actuals)], '--k',label='Reference Line')  
plt.xlabel('Actual Stress')  
plt.ylabel('Predicted Stress')  
plt.title('Scatter Plot of Predicted vs Actual Stress on Training Set')  
plt.legend()  
plt.show()plt.figure(figsize=(10, 6))  
sns.lineplot(data=train\_losses, label='Train Loss')  
sns.lineplot(data=val\_losses, label='Validation Loss')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.title('Training and Validation Loss')  
plt.legend()  
plt.show()