import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

import tensorflow as tf

from tensorflow.keras.layers import Input, Conv1D, ReLU, Concatenate, Dense, Multiply, Lambda, Reshape, GRU, RepeatVector, TimeDistributed

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

import tensorflow.keras.backend as K

from scipy.optimize import differential\_evolution

# Load GOOGL stock data

df = pd.read\_csv("/content/AAPL.csv", parse\_dates=['Date'], index\_col='Date')

def create\_sequences(data, time\_steps=60):

X, y = [], []

for i in range(len(data) - time\_steps):

X.append(data[i:i + time\_steps])

y.append(data[i + time\_steps, 1])

return np.array(X), np.array(y)

# Prepare dataset

scaler = MinMaxScaler()

numeric\_data = df[['Open', 'High', 'Low', 'Close', 'Volume']].values

scaled\_data = scaler.fit\_transform(numeric\_data)

time\_steps = 60

X, y = create\_sequences(scaled\_data, time\_steps)

split = int(0.8 \* X.shape[0])

X\_train, X\_test = X[:split], X[split:]

y\_train, y\_test = y[:split], y[split:]

# Model Components

def multi\_scale\_conv\_block(input\_tensor):

conv1 = Conv1D(filters=32, kernel\_size=3, padding='same')(input\_tensor)

conv1 = ReLU()(conv1)

conv2 = Conv1D(filters=32, kernel\_size=5, padding='same')(input\_tensor)

conv2 = ReLU()(conv2)

conv3 = Conv1D(filters=32, kernel\_size=7, padding='same')(input\_tensor)

conv3 = ReLU()(conv3)

return Concatenate()([conv1, conv2, conv3])

def temporal\_attention\_block(input\_tensor):

e\_t = Dense(128, activation='tanh')(input\_tensor)

alpha\_t = Dense(1, activation='softmax')(e\_t)

alpha\_t = Lambda(lambda x: K.squeeze(x, axis=-1))(alpha\_t)

alpha\_t = Reshape((time\_steps, 1))(alpha\_t)

context\_vector = Multiply()([input\_tensor, alpha\_t])

return Lambda(lambda x: K.sum(x, axis=1))(context\_vector)

# Build Model

input\_tensor = Input(shape=(time\_steps, X.shape[2]))

multi\_scale\_features = multi\_scale\_conv\_block(input\_tensor)

context\_vector = temporal\_attention\_block(multi\_scale\_features)

encoder\_input = Reshape((1, context\_vector.shape[-1]))(context\_vector)

encoder\_output = GRU(128, return\_sequences=False)(encoder\_input)

decoder\_output = RepeatVector(1)(encoder\_output)

decoder\_output = GRU(128, return\_sequences=True)(decoder\_output)

output = TimeDistributed(Dense(1))(decoder\_output)

output = Lambda(lambda x: x[:, -1, :])(output)

model = Model(inputs=input\_tensor, outputs=output)

# Gray Wolf Optimizer (GWO)

def gwo\_optimizer(params):

learning\_rate = params[0]

batch\_size = int(params[1])

model.compile(optimizer=Adam(learning\_rate=learning\_rate), loss='mean\_squared\_error')

model.fit(X\_train, y\_train, epochs=10, batch\_size=batch\_size, verbose=0)

predictions = model.predict(X\_test)

return mean\_squared\_error(y\_test, predictions[:, 0])

bounds = [(1e-5, 1e-3), (32, 128)]

result = differential\_evolution(gwo\_optimizer, bounds, strategy='best1bin', maxiter=10, popsize=5)

optimal\_lr, optimal\_bs = result.x

# Train with optimal parameters

optimal\_bs = int(optimal\_bs)

model.compile(optimizer=Adam(learning\_rate=optimal\_lr), loss='mean\_squared\_error')

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=optimal\_bs, validation\_data=(X\_test, y\_test))

# Evaluate Model

predictions = model.predict(X\_test).flatten()

rmse = np.sqrt(mean\_squared\_error(y\_test, predictions))

mae = mean\_absolute\_error(y\_test, predictions)

mape = np.mean(np.abs((y\_test - predictions) / y\_test)) \* 100

r2 = r2\_score(y\_test, predictions)

mda = np.mean((np.sign(y\_test[1:] - y\_test[:-1]) == np.sign(predictions[1:] - predictions[:-1])).astype(int))

smape = 100 \* np.mean(2 \* np.abs(y\_test - predictions) / (np.abs(y\_test) + np.abs(predictions)))

# Print Metrics

print(f"RMSE: {rmse:.4f}")

print(f"MAE: {mae:.4f}")

print(f"MAPE: {mape:.2f}%")

print(f"R-Squared: {r2:.4f}")

print(f"MDA: {mda:.2f}")

print(f"SMAPE: {smape:.2f}%")

# Plot Predictions

plt.figure(figsize=(12, 6))

plt.plot(y\_test, label='Actual')

plt.plot(predictions, label='Predicted')

plt.legend()

plt.title('Stock Price Prediction')

plt.xlabel('Time')

plt.ylabel('Normalized Price')

plt.show()