# 📦 Mã nguồn: coin

## config.py

# Cấu hình dữ liệu  
CRYPTO = "BTC/USDT"   
START\_DATE = "2020-01-01"   
PRE\_DAY = 30  
TEST\_SIZE = 60   
  
# Cấu hình training  
EPOCHS = 10  
BATCH\_SIZE = 32  
UNITS = 60  
  
# Đường dẫn lưu model  
MODELS\_DIR = "models"  
MODEL\_BASENAME = "crypto"

## crypto.py

import investpy  
import pandas as pd  
import datetime as dt  
import numpy as np  
from sklearn.preprocessing import MinMaxScaler  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense, Dropout, LSTM  
import matplotlib.pyplot as plt  
  
# Import config  
from config import CRYPTO, START\_DATE, PRE\_DAY, TEST\_SIZE, EPOCHS, BATCH\_SIZE, UNITS, MODEL\_BASENAME  
  
# ==== Load dữ liệu ====  
END = dt.datetime.now().strftime("%d/%m/%Y")  
df = investpy.get\_crypto\_historical\_data(  
 crypto=CRYPTO,  
 from\_date=START\_DATE,  
 to\_date=END  
)  
  
df = pd.DataFrame(df)  
  
# ==== Tạo feature ====  
df['H-L'] = df['High'] - df['Low']  
df['O-C'] = df['Open'] - df['Close']  
for ma in [7, 14, 21]:  
 df[f'SMA\_{ma}'] = df['Close'].rolling(window=ma).mean()  
df['SD\_7'] = df['Close'].rolling(window=7).std()  
df['SD\_21'] = df['Close'].rolling(window=21).std()  
df.dropna(inplace=True)  
  
# Lưu CSV để tham khảo  
df.to\_csv(f"{CRYPTO}.csv")  
print("✅ Done Loading Data")  
  
# ==== Chuẩn hóa dữ liệu ====  
cols\_x = ['H-L', 'O-C', 'SMA\_7', 'SMA\_14', 'SMA\_21', 'SD\_7', 'SD\_21']  
cols\_y = ['Close']  
scala\_x = MinMaxScaler(feature\_range=(0, 1))  
scala\_y = MinMaxScaler(feature\_range=(0, 1))  
scaled\_x = scala\_x.fit\_transform(df[cols\_x].values)  
scaled\_y = scala\_y.fit\_transform(df[cols\_y].values)  
  
# ==== Tạo dữ liệu train/test ====  
x\_total, y\_total = [], []  
for i in range(PRE\_DAY, len(df)):  
 x\_total.append(scaled\_x[i-PRE\_DAY:i])  
 y\_total.append(scaled\_y[i])  
  
x\_train = np.array(x\_total[:-TEST\_SIZE])  
x\_test = np.array(x\_total[-TEST\_SIZE:])  
y\_train = np.array(y\_total[:-TEST\_SIZE])  
y\_test = np.array(y\_total[-TEST\_SIZE:])  
print(x\_train.shape, y\_train.shape, x\_test.shape, y\_test.shape)  
  
# ==== Build model ====  
model = Sequential([  
 LSTM(UNITS, return\_sequences=True, input\_shape=(x\_train.shape[1], x\_train.shape[2])),  
 Dropout(0.2),  
 LSTM(UNITS, return\_sequences=True),  
 Dropout(0.2),  
 LSTM(UNITS),  
 Dropout(0.2),  
 Dense(1)  
])  
model.compile(optimizer='adam', loss='mean\_squared\_error')  
model.fit(x\_train, y\_train, epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1)  
  
model.save(f"{MODEL\_BASENAME}.h5")  
print("✅ Done Training Model")  
  
# ==== Test dự đoán ====  
predict\_prices = model.predict(x\_test)  
predict\_prices = scala\_y.inverse\_transform(predict\_prices)  
  
real\_price = df[-TEST\_SIZE:]['Close'].values.reshape(-1, 1)  
  
plt.plot(real\_price, color="red", label=f"Real {CRYPTO} Prices")  
plt.plot(predict\_prices, color="blue", label=f"Predicted {CRYPTO} Prices")  
plt.title(f"{CRYPTO} Prices Prediction")  
plt.xlabel("Time")  
plt.ylabel("Price")  
plt.legend()  
plt.show()  
  
# ==== Dự đoán ngày tiếp theo ====  
x\_predict = df[-PRE\_DAY:][cols\_x].values  
x\_predict = scala\_x.transform(x\_predict)  
x\_predict = np.array(x\_predict).reshape(1, PRE\_DAY, len(cols\_x))  
  
prediction = model.predict(x\_predict)  
prediction = scala\_y.inverse\_transform(prediction)  
print(f"📈 Next day predicted price: {prediction}")

## main.py

import os  
import numpy as np  
from sklearn.preprocessing import MinMaxScaler  
from config import CRYPTO, START\_DATE, PRE\_DAY, EPOCHS, BATCH\_SIZE, MODEL\_BASENAME, MODELS\_DIR, TEST\_SIZE  
from utils.data\_utils import load\_crypto\_data, add\_features, create\_sequences  
from utils.scaler\_utils import load\_scalers, save\_scalers  
from utils.model\_utils import build\_new\_model, get\_latest\_model, load\_existing\_model, save\_model\_with\_meta  
from utils.plot\_utils import plot\_predictions  
from utils.train\_utils import get\_callbacks, save\_history  
import tensorflow as tf  
tf.config.run\_functions\_eagerly(True)  
print("🔎 Eager execution:", tf.executing\_eagerly())  
  
def main():  
 print(f"📥 Đang tải dữ liệu {CRYPTO}...")  
 df = load\_crypto\_data(CRYPTO, start\_date=START\_DATE)  
 df = add\_features(df)  
  
 if df.empty:  
 print("❌ Không có dữ liệu.")  
 return  
  
 feature\_cols = ['H-L', 'O-C', 'SMA\_7', 'SMA\_14', 'SMA\_21', 'SD\_7', 'SD\_21']  
 target\_col = 'Close'  
  
 scaler\_x, scaler\_y = load\_scalers(in\_dir=MODELS\_DIR)  
 if scaler\_x is None or scaler\_y is None:  
 train\_df = df.iloc[:-TEST\_SIZE]  
 test\_df = df.iloc[-TEST\_SIZE:]  
  
 scaler\_x = MinMaxScaler().fit(train\_df[feature\_cols])  
 scaler\_y = MinMaxScaler().fit(train\_df[[target\_col]])  
  
 scaled\_x = scaler\_x.transform(df[feature\_cols])  
 scaled\_y = scaler\_y.transform(df[[target\_col]])  
  
 else:  
 scaled\_x = scaler\_x.transform(df[feature\_cols].values)  
 scaled\_y = scaler\_y.transform(df[[target\_col]].values)  
  
 x, y = create\_sequences(scaled\_x, scaled\_y, PRE\_DAY)  
 train\_size = len(x) - TEST\_SIZE  
 x\_train, y\_train = x[:train\_size], y[:train\_size]  
 x\_test, y\_test = x[train\_size:], y[train\_size:]  
  
 if len(x) == 0:  
 print("❌ Không đủ dữ liệu.")  
 return  
  
 latest\_model, latest\_num = get\_latest\_model(MODELS\_DIR)  
 version = latest\_num + 1  
  
 if latest\_model:  
 print(f"🔄 Finetune {latest\_model} ...")  
 model = load\_existing\_model(latest\_model)  
 model.compile(optimizer='adam', loss='mean\_squared\_error')  
 else:  
 print("🚀 Train mới từ đầu...")  
 model = build\_new\_model((x.shape[1], x.shape[2]))  
  
 callbacks = get\_callbacks(version)  
 history = model.fit(  
 x\_train, y\_train,  
 validation\_split=0.1,  
 epochs=EPOCHS,  
 batch\_size=BATCH\_SIZE,  
 verbose=1,  
 callbacks=callbacks  
 )  
 save\_history(history, version)  
  
 model\_path = save\_model\_with\_meta(  
 model, scaler\_x, scaler\_y, version,  
 history=history,  
 config={  
 "EPOCHS": EPOCHS,  
 "BATCH\_SIZE": BATCH\_SIZE,  
 "PRE\_DAY": PRE\_DAY  
 }  
 )  
 print(f"📊 Dataset: {len(df)} rows")  
 print(f"📊 Train: {x\_train.shape}, Test: {x\_test.shape}")  
 print(f"✅ Đã lưu model phiên bản {version}: {model\_path}")  
  
 # Inverse transform về giá thật  
 y\_true = scaler\_y.inverse\_transform(y\_test)  
 y\_pred = scaler\_y.inverse\_transform(y\_pred)  
   
 # Vẽ biểu đồ  
 plot\_predictions(y\_true, y\_pred, version, out\_dir=MODELS\_DIR)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## train\_crypto.py

import os  
import re  
import numpy as np  
import pandas as pd  
import datetime as dt  
import investpy  
from sklearn.preprocessing import MinMaxScaler  
from tensorflow.keras.models import Sequential, load\_model  
from tensorflow.keras.layers import Dense, Dropout, LSTM  
  
# Import cấu hình chung  
from config import CRYPTO, START\_DATE, PRE\_DAY, TEST\_SIZE, EPOCHS, BATCH\_SIZE, UNITS, MODEL\_BASENAME  
  
  
# ==== Hàm tiện ích ====  
  
def add\_features(df):  
 df['H-L'] = df['High'] - df['Low']  
 df['O-C'] = df['Open'] - df['Close']  
 for ma in [7, 14, 21]:  
 df[f'SMA\_{ma}'] = df['Close'].rolling(window=ma).mean()  
 df['SD\_7'] = df['Close'].rolling(window=7).std()  
 df['SD\_21'] = df['Close'].rolling(window=21).std()  
 df.dropna(inplace=True)  
 return df  
  
def scale\_data(df, feature\_cols, target\_col):  
 scaler\_x = MinMaxScaler()  
 scaler\_y = MinMaxScaler()  
 scaled\_x = scaler\_x.fit\_transform(df[feature\_cols].values)  
 scaled\_y = scaler\_y.fit\_transform(df[[target\_col]].values)  
 return scaled\_x, scaled\_y  
  
def create\_sequences(scaled\_x, scaled\_y, pre\_day):  
 x, y = [], []  
 for i in range(pre\_day, len(scaled\_x)):  
 x.append(scaled\_x[i-pre\_day:i])  
 y.append(scaled\_y[i])  
 return np.array(x), np.array(y)  
  
def get\_latest\_model():  
 models = [f for f in os.listdir(".") if f.startswith("model\_") and f.endswith(".h5")]  
 if not models:  
 return None, 0  
 numbers = [int(re.findall(r"model\_(\d+)\.h5", f)[0]) for f in models]  
 latest\_num = max(numbers)  
 return f"model\_{latest\_num}.h5", latest\_num  
  
def build\_new\_model(input\_shape):  
 model = Sequential([  
 LSTM(UNITS, return\_sequences=True, input\_shape=input\_shape),  
 Dropout(0.2),  
 LSTM(UNITS, return\_sequences=True),  
 Dropout(0.2),  
 LSTM(UNITS),  
 Dropout(0.2),  
 Dense(1)  
 ])  
 model.compile(optimizer='adam', loss='mean\_squared\_error')  
 return model  
  
  
# ==== Main ====  
  
def main():  
 try:  
 # 1. Load dữ liệu crypto  
 END = dt.datetime.now().strftime("%d/%m/%Y")  
 df = investpy.get\_crypto\_historical\_data(crypto=CRYPTO, from\_date=START\_DATE, to\_date=END)  
 df = pd.DataFrame(df)  
 df = add\_features(df)  
  
 if df.empty:  
 print("❌ Lỗi: Không có dữ liệu crypto để train.")  
 return  
  
 # 2. Chuẩn hóa  
 feature\_cols = ['H-L', 'O-C', 'SMA\_7', 'SMA\_14', 'SMA\_21', 'SD\_7', 'SD\_21']  
 target\_col = 'Close'  
 scaled\_x, scaled\_y = scale\_data(df, feature\_cols, target\_col)  
 x, y = create\_sequences(scaled\_x, scaled\_y, PRE\_DAY)  
  
 if len(x) == 0:  
 print("❌ Lỗi: Không đủ dữ liệu để tạo chuỗi train/test.")  
 return  
  
 # 3. Tự động chọn train mới hoặc finetune  
 latest\_model, latest\_num = get\_latest\_model()  
  
 if latest\_model:  
 print(f"🔄 Phát hiện model cũ: {latest\_model} → Train tiếp...")  
 model = load\_model(latest\_model)  
 model.fit(x, y, epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1)  
 new\_model\_path = f"model\_{latest\_num+1}.h5"  
 model.save(new\_model\_path)  
 print(f"✅ Đã lưu model mới: {new\_model\_path}")  
 else:  
 print("🚀 Không có model cũ → Train mới từ đầu...")  
 model = build\_new\_model((x.shape[1], x.shape[2]))  
 model.fit(x, y, epochs=EPOCHS, batch\_size=BATCH\_SIZE, verbose=1)  
 model.save(f"{MODEL\_BASENAME}.h5")  
 print(f"✅ Đã lưu model: {MODEL\_BASENAME}.h5")  
  
 except Exception as e:  
 print(f"❌ Lỗi không mong muốn: {type(e).\_\_name\_\_} → {str(e)}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## models\nope\_have.py

## utils\data\_utils.py

import ccxt  
import pandas as pd  
import numpy as np  
  
def load\_crypto\_data(symbol="BTC/USDT", start\_date="2017-01-01", timeframe="1d", limit=2000):  
 exchange = ccxt.binance()  
 since = exchange.parse8601(start\_date + "T00:00:00Z")  
 ohlcv = exchange.fetch\_ohlcv(symbol, timeframe=timeframe, since=since, limit=limit)  
 df = pd.DataFrame(ohlcv, columns=["Timestamp", "Open", "High", "Low", "Close", "Volume"])  
 df["Date"] = pd.to\_datetime(df["Timestamp"], unit="ms")  
 df.set\_index("Date", inplace=True)  
 return df  
  
def add\_features(df):  
 df['H-L'] = df['High'] - df['Low']  
 df['O-C'] = df['Open'] - df['Close']  
 for ma in [7, 14, 21]:  
 df[f'SMA\_{ma}'] = df['Close'].rolling(window=ma).mean()  
 df['SD\_7'] = df['Close'].rolling(window=7).std()  
 df['SD\_21'] = df['Close'].rolling(window=21).std()  
 df.dropna(inplace=True)  
 return df  
  
def create\_sequences(scaled\_x, scaled\_y, pre\_day):  
 x, y = [], []  
 for i in range(pre\_day, len(scaled\_x)):  
 x.append(scaled\_x[i-pre\_day:i])  
 y.append(scaled\_y[i])  
 return np.array(x), np.array(y)

## utils\export\_code.py

import os  
from docx import Document  
  
VERSION = "1.0"  
SERVICE\_NAME = "coin"  
OUTPUT\_NAME = f"{SERVICE\_NAME}\_export\_{VERSION}.docx"  
  
def export\_code():  
 current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
 print("[DEBUG] Đường dẫn file export\_code.py:", current\_dir)  
  
 # 📁 Đường dẫn đến thư mục service (cùng cấp)  
 service\_path = os.path.abspath(os.path.join(current\_dir, ".."))  
 print("[DEBUG] Đường dẫn đến service:", service\_path)  
  
 if not os.path.exists(service\_path):  
 print("[❌] Không tìm thấy thư mục:", service\_path)  
 return  
  
 # 📝 Tạo file docx  
 doc = Document()  
 doc.add\_heading(f"📦 Mã nguồn: {SERVICE\_NAME}", level=1)  
  
 file\_count = 0  
  
 for root, dirs, files in os.walk(service\_path):  
 print("[DEBUG] Đang đọc thư mục:", root)  
  
 for file in files:  
 if not file.endswith(".py"):  
 continue  
  
 file\_path = os.path.join(root, file)  
 rel\_path = os.path.relpath(file\_path, service\_path)  
  
 print(f" 📄 Đọc file: {rel\_path}")  
  
 try:  
 with open(file\_path, 'r', encoding='utf-8') as f:  
 content = f.read()  
  
 doc.add\_heading(rel\_path, level=2)  
 doc.add\_paragraph(content, style='Normal')  
 file\_count += 1  
  
 except Exception as e:  
 print(f"[⚠️] Không đọc được file: {file\_path} → {type(e).\_\_name\_\_}: {str(e)}")  
  
 # 📤 Lưu file  
 output\_path = os.path.abspath(os.path.join(current\_dir, "../doc", OUTPUT\_NAME))  
 doc.save(output\_path)  
 print(f"✅ Hoàn tất. Đã ghi {file\_count} file vào: {output\_path}")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 export\_code()

## utils\model\_utils.py

import os, re, json, time, shutil  
import joblib  
from tensorflow.keras.models import Sequential, load\_model  
from tensorflow.keras.layers import LSTM, Dropout, Dense  
from config import UNITS, MODELS\_DIR  
  
def build\_new\_model(input\_shape):  
 model = Sequential([  
 LSTM(UNITS, return\_sequences=True, input\_shape=input\_shape),  
 Dropout(0.2),  
 LSTM(UNITS, return\_sequences=True),  
 Dropout(0.2),  
 LSTM(UNITS),  
 Dropout(0.2),  
 Dense(1)  
 ])  
 model.compile(optimizer='adam', loss='mean\_squared\_error')  
 return model  
  
def get\_latest\_model(models\_dir=MODELS\_DIR):  
 os.makedirs(models\_dir, exist\_ok=True)  
 models = [f for f in os.listdir(models\_dir) if f.startswith("model\_") and f.endswith(".h5")]  
 if not models:  
 return None, 0  
 numbers = [int(re.findall(r"model\_(\d+)\.h5", f)[0]) for f in models]  
 latest\_num = max(numbers)  
 return os.path.join(models\_dir, f"model\_{latest\_num}.h5"), latest\_num  
  
def load\_existing\_model(path):  
 return load\_model(path)  
  
def update\_registry(models\_dir, version, info):  
 path = os.path.join(models\_dir, "registry.json")  
 registry = []  
 if os.path.exists(path):  
 with open(path, "r") as f:  
 registry = json.load(f)  
 registry.append(info)  
 with open(path, "w") as f:  
 json.dump(registry, f, indent=2)  
  
def save\_model\_with\_meta(model, scaler\_x, scaler\_y, version, history=None, config=None):  
 os.makedirs(MODELS\_DIR, exist\_ok=True)  
  
 # --- Save model  
 model\_fn = f"model\_{version}.h5"  
 model\_path = os.path.join(MODELS\_DIR, model\_fn)  
 model.save(model\_path)  
  
 # --- Save scalers  
 scaler\_x\_fn = f"scaler\_{version}\_x.pkl"  
 scaler\_y\_fn = f"scaler\_{version}\_y.pkl"  
 joblib.dump(scaler\_x, os.path.join(MODELS\_DIR, scaler\_x\_fn))  
 joblib.dump(scaler\_y, os.path.join(MODELS\_DIR, scaler\_y\_fn))  
  
 # --- Save training history  
 if history is not None:  
 with open(os.path.join(MODELS\_DIR, f"history\_{version}.json"), "w") as f:  
 json.dump(history.history, f)  
  
 # --- Save metadata  
 meta = {  
 "version": version,  
 "model\_file": model\_fn,  
 "scaler\_x": scaler\_x\_fn,  
 "scaler\_y": scaler\_y\_fn,  
 "timestamp": time.strftime("%Y-%m-%d %H:%M:%S"),  
 "config": config or {}  
 }  
 with open(os.path.join(MODELS\_DIR, f"meta\_{version}.json"), "w") as f:  
 json.dump(meta, f, indent=2)  
  
 # --- Update registry  
 update\_registry(MODELS\_DIR, version, meta)  
  
 # --- Copy to latest.h5  
 latest\_path = os.path.join(MODELS\_DIR, "latest.h5")  
 try:  
 if os.path.exists(latest\_path):  
 os.remove(latest\_path)  
 shutil.copy(model\_path, latest\_path)  
 except Exception as e:  
 print(f"⚠️ Không tạo được latest.h5: {e}")  
  
 return model\_path

## utils\plot\_utils.py

import os  
import matplotlib.pyplot as plt  
import numpy as np  
  
def plot\_predictions(y\_true, y\_pred, version, out\_dir="models"):  
 """Vẽ biểu đồ giá thực tế vs dự đoán và lưu thành PNG."""  
 plt.figure(figsize=(12,6))  
 plt.plot(y\_true, color="blue", label="Thực tế")  
 plt.plot(y\_pred, color="red", linestyle="--", label="Dự đoán")  
 plt.title(f"Dự đoán giá crypto - Model v{version}")  
 plt.xlabel("Thời gian")  
 plt.ylabel("Giá (USD)")  
 plt.legend()  
 plt.tight\_layout()  
  
 path = os.path.join(out\_dir, f"plot\_model\_{version}.png")  
 plt.savefig(path)  
 plt.close()  
 print(f"📊 Đã lưu biểu đồ: {path}")

## utils\scaler\_utils.py

import joblib  
from sklearn.preprocessing import MinMaxScaler  
  
def save\_scalers(scaler\_x, scaler\_y, prefix="scaler", out\_dir="."):  
 joblib.dump(scaler\_x, f"{out\_dir}/{prefix}\_x.pkl")  
 joblib.dump(scaler\_y, f"{out\_dir}/{prefix}\_y.pkl")  
 print(f"💾 Đã lưu scaler vào {out\_dir}")  
  
def load\_scalers(prefix="scaler", in\_dir="."):  
 try:  
 scaler\_x = joblib.load(f"{in\_dir}/{prefix}\_x.pkl")  
 scaler\_y = joblib.load(f"{in\_dir}/{prefix}\_y.pkl")  
 print("📂 Đã load scaler từ file.")  
 return scaler\_x, scaler\_y  
 except FileNotFoundError:  
 print("⚠️ Không tìm thấy scaler cũ, sẽ tạo mới.")  
 return None, None

## utils\train\_utils.py

import os, json, random, numpy as np, tensorflow as tf  
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint  
from config import MODELS\_DIR  
  
# ==== Fix seed để tái lập kết quả ====  
SEED = 42  
random.seed(SEED)  
np.random.seed(SEED)  
tf.random.set\_seed(SEED)  
  
def get\_callbacks(version):  
 """Trả về danh sách callbacks cho training."""  
 callbacks = [  
 EarlyStopping(monitor="val\_loss", patience=8, restore\_best\_weights=True),  
 ModelCheckpoint(  
 os.path.join(MODELS\_DIR, f"best\_model\_tmp\_{version}.h5"),  
 monitor="val\_loss",  
 save\_best\_only=True  
 )  
 ]  
 return callbacks  
  
def save\_history(history, version):  
 """Lưu history ra file JSON để phân tích sau."""  
 if history is not None:  
 path = os.path.join(MODELS\_DIR, f"history\_{version}.json")  
 with open(path, "w") as f:  
 json.dump(history.history, f, indent=2)  
 print(f"💾 Đã lưu history: {path}")