# 📦 Mã nguồn: coin

## config.py

# Cấu hình dữ liệu  
CRYPTO = "BTC/USDT"   
START\_DATE = "2024-01-01"  
END\_DATE = "2025-9-13"   
PRE\_DAY = 30  
TEST\_RATIO = 0.2  
MIN\_TEST\_DAYS = 60   
PRINT\_LAST\_N = 5  
PREDICT\_DAYS = 5  
  
# 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  
import pandas as pd  
from sklearn.preprocessing import MinMaxScaler  
  
from config import (  
 CRYPTO, START\_DATE, END\_DATE, PRE\_DAY, TEST\_RATIO,  
 EPOCHS, BATCH\_SIZE, MODELS\_DIR, PREDICT\_DAYS  
)  
  
from utils import data\_utils, scaler\_utils, model\_utils, train\_utils, plot\_utils, export\_code  
  
  
def main():  
 # --- Load & feature  
 print(f"📥 Loading {CRYPTO} from {START\_DATE} to {END\_DATE}...")  
 df = data\_utils.load\_crypto\_data(CRYPTO, START\_DATE, END\_DATE)  
 df = data\_utils.add\_features(df)  
  
 if df.empty:  
 print("❌ No data.")  
 return  
  
 feature\_cols = ['H-L', 'O-C', 'SMA\_7', 'SMA\_14', 'SMA\_21', 'SD\_7', 'SD\_21']  
 target\_col = 'Close'  
  
 # --- Chia train/test bằng train\_utils  
 x\_train, y\_train, x\_test, y\_test, test\_size = train\_utils.split\_train\_test(  
 df, df[feature\_cols].values, df[[target\_col]].values, PRE\_DAY, test\_ratio=TEST\_RATIO  
 )  
  
 # --- Load scaler nếu có, nếu không thì tạo mới  
 train\_df = df.iloc[:-test\_size]  
 scaler\_x, scaler\_y = scaler\_utils.load\_scalers(prefix="crypto", in\_dir=MODELS\_DIR)  
 if scaler\_x is None or scaler\_y is None:  
 scaler\_x = MinMaxScaler().fit(train\_df[feature\_cols])  
 scaler\_y = MinMaxScaler().fit(train\_df[[target\_col]])  
 scaler\_utils.save\_scalers(scaler\_x, scaler\_y, prefix="crypto", out\_dir=MODELS\_DIR)  
  
 scaled\_x = scaler\_x.transform(df[feature\_cols])  
 scaled\_y = scaler\_y.transform(df[[target\_col]])  
  
 # --- In thông tin dataset  
 print(f"📊 Dataset size: {len(df)} rows")  
 print(f"📊 Train: {x\_train.shape}, Test: {x\_test.shape}")  
  
 # --- Model versioning  
 latest\_version = model\_utils.get\_latest\_version()  
 version = latest\_version + 1  
  
 if latest\_version > 0:  
 prev\_path = os.path.join(MODELS\_DIR, f"v{latest\_version}", "model.h5")  
 print(f"🔄 Finetuning from v{latest\_version} ({prev\_path})")  
 model = model\_utils.load\_existing\_model(prev\_path)  
 model.compile(optimizer="adam", loss="mean\_squared\_error")  
 else:  
 print("🚀 Training new model...")  
 # dùng x\_train shape thay vì x  
 model = model\_utils.build\_new\_model((x\_train.shape[1], x\_train.shape[2]))  
  
 # --- Train  
 callbacks = train\_utils.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  
 model\_path = model\_utils.save\_model\_with\_meta(  
 model, scaler\_x, scaler\_y, version,  
 history=history,  
 config={"EPOCHS": EPOCHS, "BATCH\_SIZE": BATCH\_SIZE, "PRE\_DAY": PRE\_DAY}  
 )  
  
 # --- Dự đoán test set  
 y\_pred\_test = model.predict(x\_test)  
 y\_pred\_test = scaler\_y.inverse\_transform(y\_pred\_test)  
 y\_true\_test = scaler\_y.inverse\_transform(y\_test)  
 dates\_test = df.index[-test\_size:]  
  
 # --- Dự đoán mở rộng PREDICT\_DAYS  
 scaled\_x = scaler\_x.transform(df[feature\_cols]) # đảm bảo dùng scaler  
 last\_window = scaled\_x[-PRE\_DAY:]  
 preds\_future = []  
 current = last\_window.copy()  
  
 for \_ in range(PREDICT\_DAYS):  
 pred = model.predict(current.reshape(1, PRE\_DAY, len(feature\_cols)))  
 preds\_future.append(pred[0, 0])  
 pred\_scaled = scaler\_y.transform(pred)[0]   
 new\_row = current[-1].copy()  
 new\_row[-1] = pred  
 current = np.vstack([current[1:], new\_row])  
  
 preds\_future = scaler\_y.inverse\_transform(np.array(preds\_future).reshape(-1, 1))  
 future\_dates = pd.date\_range(df.index[-1] + pd.Timedelta(days=1), periods=PREDICT\_DAYS, freq="D")  
  
 print("🔮 Future predictions:")  
 for d, p in zip(future\_dates, preds\_future):  
 print(f"{d.date()} | {p[0]:.2f} USD")  
  
 # --- Vẽ biểu đồ bằng plot\_utils  
 plot\_utils.plot\_forecast(  
 df, y\_true\_test, y\_pred\_test, dates\_test,  
 future\_dates, preds\_future, 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  
from config import START\_DATE, END\_DATE  
  
def load\_crypto\_data(symbol="BTC/USDT", start\_date=START\_DATE, end\_date= END\_DATE, timeframe="1d"):  
 exchange = ccxt.binance()  
 since = exchange.parse8601(start\_date + "T00:00:00Z")  
 all\_ohlcv = []  
 while True:  
 ohlcv = exchange.fetch\_ohlcv(symbol, timeframe=timeframe, since=since, limit=2000)  
 if not ohlcv:  
 break  
 all\_ohlcv += ohlcv  
 # next since = timestamp cuối + 1ms  
 since = ohlcv[-1][0] + 1  
 if end\_date and pd.to\_datetime(since, unit="ms") > pd.to\_datetime(end\_date):  
 break  
 df = pd.DataFrame(all\_ohlcv, columns=["Timestamp","Open","High","Low","Close","Volume"])  
 df["Date"] = pd.to\_datetime(df["Timestamp"], unit="ms")  
 df.set\_index("Date", inplace=True)  
 if end\_date:  
 df = df.loc[:end\_date]  
 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 = "1.1"  
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, json, time, 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\_version():  
 if not os.path.exists(MODELS\_DIR):  
 return 0  
 versions = [int(d[1:]) for d in os.listdir(MODELS\_DIR) if d.startswith("v") and d[1:].isdigit()]  
 return max(versions) if versions else 0  
  
def load\_existing\_model(path):  
 return load\_model(path)  
  
def save\_model\_with\_meta(model, scaler\_x, scaler\_y, version, history=None, config=None):  
 version\_dir = os.path.join(MODELS\_DIR, f"v{version}")  
 os.makedirs(version\_dir, exist\_ok=True)  
  
 # --- Save model  
 model\_path = os.path.join(version\_dir, "model.h5")  
 model.save(model\_path)  
  
 # --- Save scalers  
 joblib.dump(scaler\_x, os.path.join(version\_dir, "scaler\_x.pkl"))  
 joblib.dump(scaler\_y, os.path.join(version\_dir, "scaler\_y.pkl"))  
  
 # --- Save training history  
 if history is not None:  
 with open(os.path.join(version\_dir, "history.json"), "w") as f:  
 json.dump(history.history, f, indent=2)  
  
 # --- Save metadata  
 meta = {  
 "version": version,  
 "model\_path": model\_path,  
 "scaler\_x": os.path.join(version\_dir, "scaler\_x.pkl"),  
 "scaler\_y": os.path.join(version\_dir, "scaler\_y.pkl"),  
 "timestamp": time.strftime("%Y-%m-%d %H:%M:%S"),  
 "config": config or {}  
 }  
 with open(os.path.join(version\_dir, "meta.json"), "w") as f:  
 json.dump(meta, f, indent=2)  
  
 # --- Update registry.json  
 registry\_path = os.path.join(MODELS\_DIR, "registry.json")  
 registry = []  
 if os.path.exists(registry\_path):  
 with open(registry\_path, "r") as f:  
 registry = json.load(f)  
 registry.append(meta)  
 with open(registry\_path, "w") as f:  
 json.dump(registry, f, indent=2)  
  
 # --- Update latest.json  
 latest\_path = os.path.join(MODELS\_DIR, "latest.json")  
 with open(latest\_path, "w") as f:  
 json.dump(meta, f, indent=2)  
  
 return model\_path

## utils\plot\_utils.py

import os  
import matplotlib.pyplot as plt  
import matplotlib.ticker as mtick  
  
def plot\_forecast(df, y\_true\_test, y\_pred\_test, dates\_test,  
 future\_dates, preds\_future, version, out\_dir="models"):  
  
 plots\_dir = os.path.join(out\_dir, "plots")  
 os.makedirs(plots\_dir, exist\_ok=True)  
  
 plt.figure(figsize=(12, 6))  
  
 # Thực tế toàn bộ  
 plt.plot(df.index, df["Close"], color="blue", label="Thực tế")  
  
 # Dự đoán test  
 plt.plot(dates\_test, y\_pred\_test, color="orange", linestyle="--", label="Dự đoán (test)")  
  
 # Dự đoán tương lai  
 plt.plot(future\_dates, preds\_future, color="red", linestyle="--", marker="o", label="Dự đoán (tương lai)")  
  
 # Vạch đánh dấu END\_DATE  
 plt.axvline(df.index[-1], color="gray", linestyle=":", label="END\_DATE")  
  
 # Format trục Y thành USD  
 ax = plt.gca()  
 ax.yaxis.set\_major\_formatter(mtick.StrMethodFormatter('${x:,.0f}'))  
  
 plt.title(f"Dự báo giá {df.columns.name or 'crypto'} - Model v{version}")  
 plt.xlabel("Ngày")  
 plt.ylabel("Giá (USD)")  
 plt.legend()  
 plt.tight\_layout()  
  
 path = os.path.join(plots\_dir, f"forecast\_v{version}.png")  
 plt.savefig(path)  
 plt.close()  
 print(f"📊 Đã lưu biểu đồ dự báo: {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, random, numpy as np, tensorflow as tf  
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint  
from config import MODELS\_DIR  
  
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, checkpoint lưu vào thư mục version."""  
 version\_dir = os.path.join(MODELS\_DIR, f"v{version}")  
 os.makedirs(version\_dir, exist\_ok=True)  
  
 callbacks = [  
 EarlyStopping(monitor="val\_loss", patience=8, restore\_best\_weights=True),  
 ModelCheckpoint(  
 os.path.join(version\_dir, "best\_model\_tmp.h5"),  
 monitor="val\_loss",  
 save\_best\_only=True  
 )  
 ]  
 return callbacks  
  
def split\_train\_test(df, scaled\_x, scaled\_y, pre\_day, test\_ratio=0.2, min\_test\_days=60):  
 n = len(df)  
 test\_size = max(min\_test\_days, int(n \* test\_ratio))  
 if test\_size >= n - pre\_day:  
 test\_size = max(1, n - pre\_day - 1)  
  
 x, y = [], []  
 for i in range(pre\_day, len(scaled\_x)):  
 x.append(scaled\_x[i-pre\_day:i])  
 y.append(scaled\_y[i])  
 x, y = np.array(x), np.array(y)  
  
 x\_train, y\_train = x[:-test\_size], y[:-test\_size]  
 x\_test, y\_test = x[-test\_size:], y[-test\_size:]  
  
 return x\_train, y\_train, x\_test, y\_test, test\_size