

LSTM NEW

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input
from tensorflow.keras.callbacks import Callback

# Load and parse datetime correctly
df = pd.read_csv('EURUSD_ASK_12HOURS.csv')
df.columns = ['Local time', 'Open', 'High', 'Low', 'Close', 'Volume']
df['Local time'] = pd.to_datetime(df['Local time'].str.split('.').str[0], format='%Y-%m-%d %H:%M:%S')
df = df[df['Volume'] > 0].copy()
df = df[['Open', 'High', 'Low', 'Close']].astype(float)

# Scale data
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df)

# Create sequences (exclude Close from input features)
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length, :-1]) # Use only Open, High, Low
        y.append(data[i+seq_length, -1]) # Predict Close
    return np.array(X), np.array(y)

SEQ_LENGTH = 3
X, y = create_sequences(scaled, SEQ_LENGTH)

# Split train/test
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

# Build LSTM model
model = Sequential()
model.add(Input(shape=(X_train.shape[1], X_train.shape[2]))) # Explicit Input Layer
model.add(LSTM(64, return_sequences=True))
model.add(LSTM(64))
model.add(Dense(1))

# Custom RMSE metric
def rmse(y_true, y_pred):
    return tf.sqrt(tf.reduce_mean(tf.square(y_pred - y_true)))

# Compile without MAPE (handled manually)
model.compile(optimizer='adam', loss='mse', metrics=['mae', rmse])

```

```

# Callback to track metrics on original scale
class MetricsCallback(Callback):
    def __init__(self, X_train, y_train, X_val, y_val, scaler, df):
        super().__init__()
        self.X_train = X_train
        self.y_train = y_train
        self.X_val = X_val
        self.y_val = y_val
        self.scaler = scaler
        self.df = df
        self.metrics = {
            'train_mae': [], 'val_mae': [],
            'train_mape': [], 'val_mape': [],
            'train_rmse': [], 'val_rmse': [],
            'train_r2': [], 'val_r2': []
        }

    def on_epoch_end(self, epoch, logs=None):
        # Predict on train and validation data
        train_pred = self.model.predict(self.X_train, verbose=0).flatten()
        val_pred = self.model.predict(self.X_val, verbose=0).flatten()

        # Inverse transform predictions and actuals
        def inverse_transform(preds, X_original):
            dummy = np.zeros((len(preds), self.df.shape[1]))
            dummy[:, :-1] = X_original.reshape(-1, self.df.shape[1]-1)
            dummy[:, -1] = preds.ravel()
            dummy = self.scaler.inverse_transform(dummy)
            return dummy[:, -1]

        train_pred_inv = inverse_transform(train_pred, self.X_train[:, -1, :])
        val_pred_inv = inverse_transform(val_pred, self.X_val[:, -1, :])
        train_actuals_inv = inverse_transform(self.y_train, self.X_train[:, -1, :])
        val_actuals_inv = inverse_transform(self.y_val, self.X_val[:, -1, :])

        # Compute metrics
        def calculate_metrics(y_true, y_pred):
            mae = mean_absolute_error(y_true, y_pred)
            rmse = np.sqrt(mean_squared_error(y_true, y_pred))
            mape = mean_absolute_percentage_error(y_true, y_pred) * 100
            r2 = r2_score(y_true, y_pred)
            return mae, mape, rmse, r2

        train_mae, train_mape, train_rmse, train_r2 = calculate_metrics(train_pred_inv, train_actuals_inv)
        val_mae, val_mape, val_rmse, val_r2 = calculate_metrics(val_pred_inv, val_actuals_inv)

        # Store metrics
        self.metrics['train_mae'].append(train_mae)
        self.metrics['val_mae'].append(val_mae)
        self.metrics['train_mape'].append(train_mape)
        self.metrics['val_mape'].append(val_mape)
        self.metrics['train_rmse'].append(train_rmse)
        self.metrics['val_rmse'].append(val_rmse)
        self.metrics['train_r2'].append(train_r2)

```

```

        self.metrics['val_r2'].append(val_r2)

        # Print every 5 epochs
        if (epoch + 1) % 5 == 0:
            print(f"Epoch {epoch+1:2d} | "
                  f"Train MAPE: {train_mape:.2f}%, Val MAPE: {val_mape:.2f}%
                  f"Train R2: {train_r2:.4f}, Val R2: {val_r2:.4f}")

    # Initialize callback
    metrics_callback = MetricsCallback(X_train, y_train, X_test, y_test, scaler,
                                         )

    # Train model
    history_lstm = model.fit(
        X_train, y_train,
        epochs=50,
        batch_size=32,
        validation_data=(X_test, y_test),
        callbacks=[metrics_callback]
    )

    # Predictions
    train_pred_lstm = model.predict(X_train).flatten()
    test_pred_lstm = model.predict(X_test).flatten()

    # Inverse transform predictions
    def inverse_transform(preds, X_original):
        dummy = np.zeros((len(preds), df.shape[1]))
        dummy[:, :-1] = X_original.reshape(-1, df.shape[1]-1)
        dummy[:, -1] = preds.ravel()
        dummy = scaler.inverse_transform(dummy)
        return dummy[:, -1]

    train_pred_inv_lstm = inverse_transform(train_pred_lstm, X_train[:, -1, :])
    test_pred_inv_lstm = inverse_transform(test_pred_lstm, X_test[:, -1, :])

    # Actual values (invert y_train/y_test using corresponding X)
    train_actuals = inverse_transform(y_train, X_train[:, -1, :])
    test_actuals = inverse_transform(y_test, X_test[:, -1, :])

    # Compute final metrics
    def evaluate(y_true, y_pred):
        mae = mean_absolute_error(y_true, y_pred)
        rmse_val = np.sqrt(mean_squared_error(y_true, y_pred))
        mape_val = mean_absolute_percentage_error(y_true, y_pred) * 100
        r2_val = r2_score(y_true, y_pred)
        return [mae, rmse_val, mape_val, r2_val]

    train_metrics = evaluate(train_actuals, train_pred_inv_lstm)
    test_metrics = evaluate(test_actuals, test_pred_inv_lstm)

    # Plotting
    plt.figure(figsize=(12, 6))
    plt.plot(history_lstm.history['mae'], label='Train MAE (Scaled)')
    plt.plot(history_lstm.history['val_mae'], label='Val MAE (Scaled)')
    plt.plot(metrics_callback.metrics['train_mae'], label='Train MAE (Original S

```

```
plt.plot(metrics_callback.metrics['val_mae'], label='Val MAE (Original Scale')
plt.legend()
plt.title('MAE per Epoch')
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(metrics_callback.metrics['train_mape'], label='Train MAPE (Original Scale')
plt.plot(metrics_callback.metrics['val_mape'], label='Val MAPE (Original Scale')
plt.legend()
plt.title('MAPE per Epoch (Original Scale)')
plt.xlabel('Epoch')
plt.ylabel('MAPE (%)')
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(metrics_callback.metrics['train_r2'], label='Train R2')
plt.plot(metrics_callback.metrics['val_r2'], label='Val R2)
plt.axhline(y=0, color='r', linestyle='--', label='Baseline (R2 = 0)')
plt.legend()
plt.title('R2 Score Progression')
plt.xlabel('Epoch')
plt.ylabel('R2')
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(test_actuals, label='Actual')
plt.plot(test_pred_inv_lstm, label='Predicted')
plt.legend()
plt.title('Test Set Predictions')
plt.show()

# Print final metrics
print(f"Train Metrics - MAE: {train_metrics[0]:.4f}, RMSE: {train_metrics[1]:.4f}")
print(f"Test Metrics - MAE: {test_metrics[0]:.4f}, RMSE: {test_metrics[1]:.4f}")

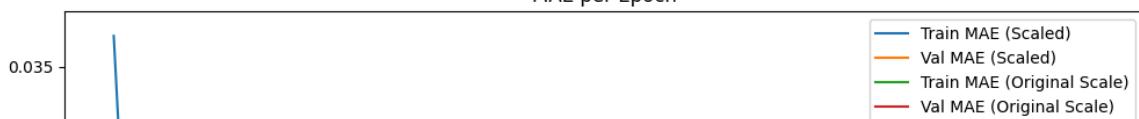
# Print metrics every 5 epochs
for epoch in range(4, 50, 5): # Starts from epoch 5 (index 4)
    print(f"Epoch {epoch+1:2d} | "
          f"Train MAE: {metrics_callback.metrics['train_mae'][epoch]:.4f}, "
          f"Val MAE: {metrics_callback.metrics['val_mae'][epoch]:.4f} | "
          f"Train MAPE: {metrics_callback.metrics['train_mape'][epoch]:.2f}% "
          f"Val MAPE: {metrics_callback.metrics['val_mape'][epoch]:.2f}% | "
          f"Train RMSE: {metrics_callback.metrics['train_rmse'][epoch]:.4f}, "
          f"Val RMSE: {metrics_callback.metrics['val_rmse'][epoch]:.4f} | "
          f"Train R2: {metrics_callback.metrics['train_r2'][epoch]:.4f}, "
          f"Val R2: {metrics_callback.metrics['val_r2'][epoch]:.4f}")
```

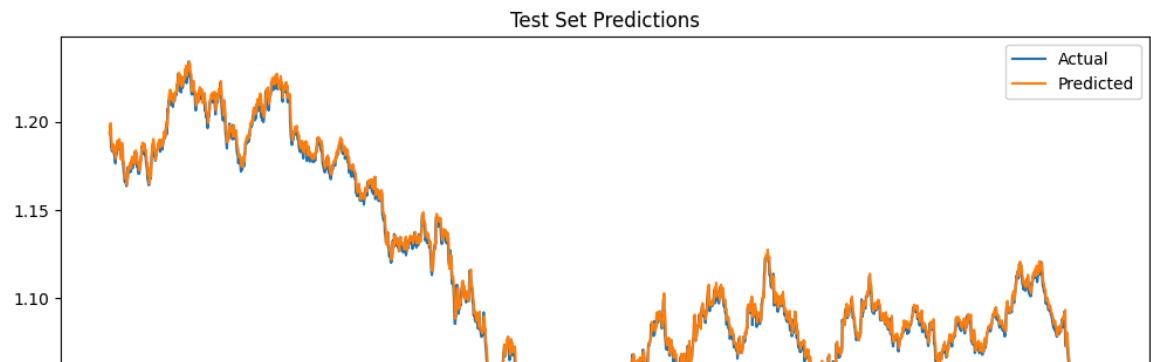
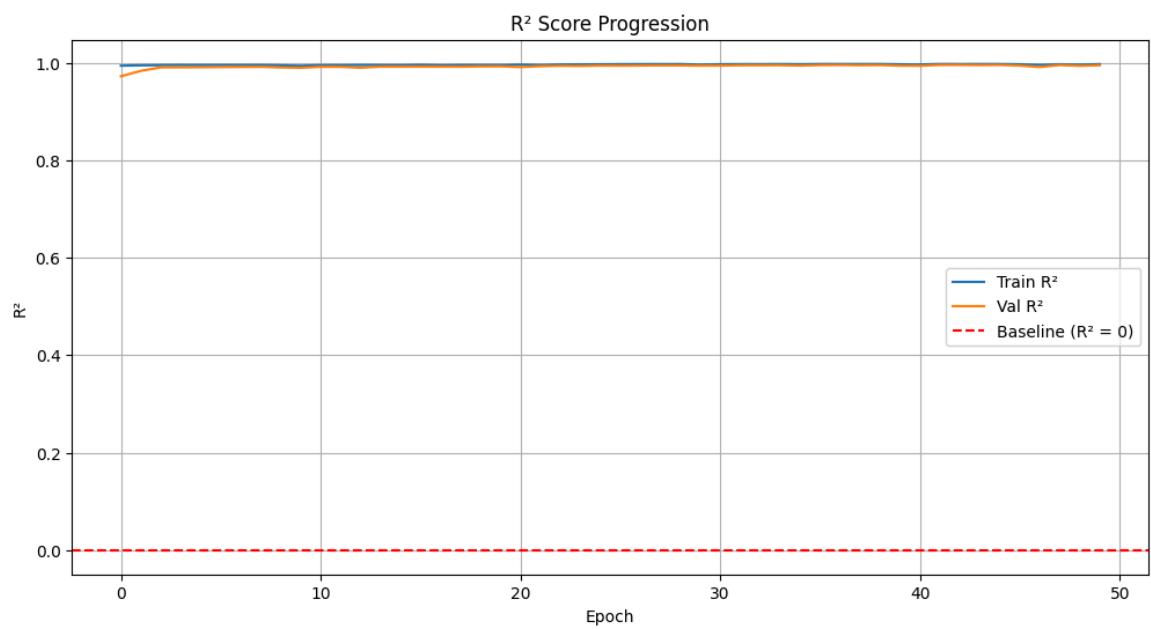
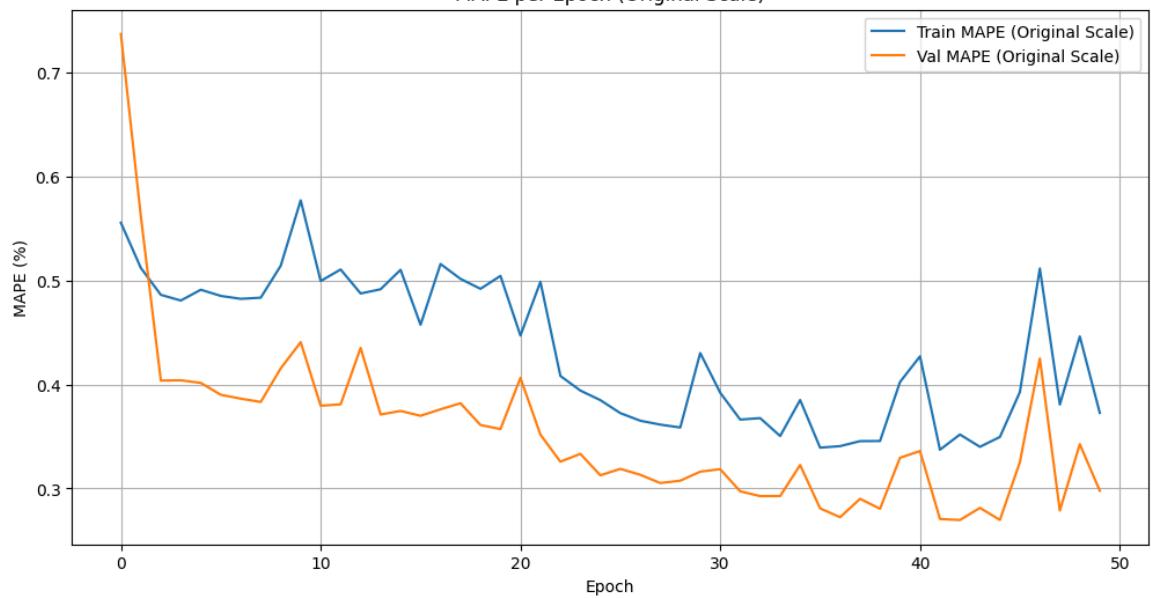
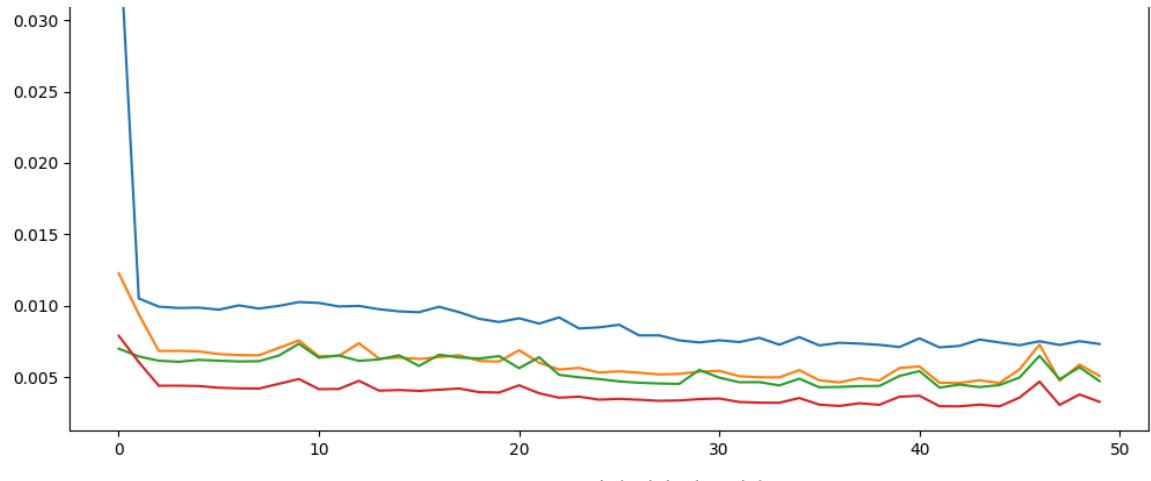


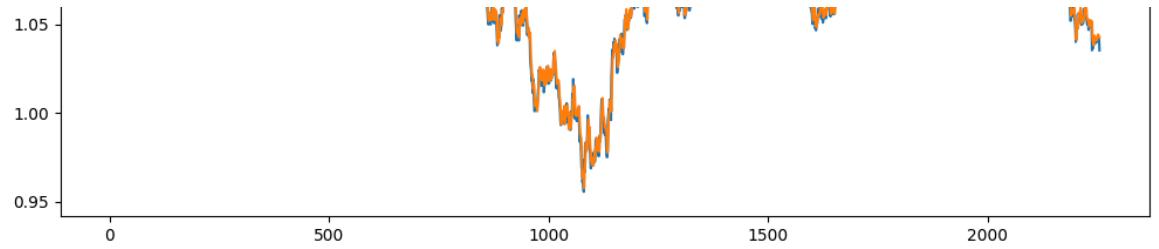
```
<>:14: SyntaxWarning: invalid escape sequence '\.'  
<>:14: SyntaxWarning: invalid escape sequence '\.'  
/tmp/ipython-input-1184094498.py:14: SyntaxWarning: invalid escape sequence  
    df['Local time'] = pd.to_datetime(df['Local time'].str.split('.').str[0]).  
Epoch 1/50  
282/282 7s 16ms/step - loss: 0.0303 - mae: 0.0964 - rmse:  
Epoch 2/50  
282/282 2s 8ms/step - loss: 2.0498e-04 - mae: 0.0108 -  
Epoch 3/50  
282/282 2s 9ms/step - loss: 1.7075e-04 - mae: 0.0098 -  
Epoch 4/50  
282/282 2s 8ms/step - loss: 1.8591e-04 - mae: 0.0100 -  
Epoch 5/50  
279/282 0s 6ms/step - loss: 1.7849e-04 - mae: 0.0099 -  
282/282 4s 13ms/step - loss: 1.7850e-04 - mae: 0.0099 -  
Epoch 6/50  
282/282 4s 9ms/step - loss: 1.7444e-04 - mae: 0.0097 -  
Epoch 7/50  
282/282 2s 9ms/step - loss: 1.8113e-04 - mae: 0.0099 -  
Epoch 8/50  
282/282 3s 9ms/step - loss: 1.8359e-04 - mae: 0.0100 -  
Epoch 9/50  
282/282 4s 14ms/step - loss: 1.7974e-04 - mae: 0.0099 -  
Epoch 10/50  
277/282 0s 5ms/step - loss: 1.8380e-04 - mae: 0.0103 -  
282/282 4s 9ms/step - loss: 1.8387e-04 - mae: 0.0103 -  
Epoch 11/50  
282/282 3s 9ms/step - loss: 1.9419e-04 - mae: 0.0104 -  
Epoch 12/50  
282/282 3s 9ms/step - loss: 1.7548e-04 - mae: 0.0098 -  
Epoch 13/50  
282/282 4s 13ms/step - loss: 2.0309e-04 - mae: 0.0106 -  
Epoch 14/50  
282/282 2s 9ms/step - loss: 1.6943e-04 - mae: 0.0097 -  
Epoch 15/50  
275/282 0s 5ms/step - loss: 1.6509e-04 - mae: 0.0096 -  
282/282 3s 9ms/step - loss: 1.6516e-04 - mae: 0.0096 -  
Epoch 16/50  
282/282 2s 9ms/step - loss: 1.6991e-04 - mae: 0.0097 -  
Epoch 17/50  
282/282 3s 11ms/step - loss: 1.6678e-04 - mae: 0.0096 -  
Epoch 18/50  
282/282 3s 11ms/step - loss: 1.6952e-04 - mae: 0.0096 -  
Epoch 19/50  
282/282 2s 9ms/step - loss: 1.4689e-04 - mae: 0.0091 -  
Epoch 20/50  
273/282 0s 5ms/step - loss: 1.4531e-04 - mae: 0.0088 -  
282/282 2s 9ms/step - loss: 1.4527e-04 - mae: 0.0088 -  
Epoch 21/50  
282/282 2s 9ms/step - loss: 1.6228e-04 - mae: 0.0095 -  
Epoch 22/50  
282/282 3s 12ms/step - loss: 1.3554e-04 - mae: 0.0087 -  
Epoch 23/50  
282/282 4s 9ms/step - loss: 1.4503e-04 - mae: 0.0091 -  
Epoch 24/50  
282/282 2s 9ms/step - loss: 1.3348e-04 - mae: 0.0086 -  
Epoch 25/50  
282/282 0s 5ms/step - loss: 1.2437e-04 - mae: 0.0083 -  
282/282 3s 9ms/step - loss: 1.2440e-04 - mae: 0.0083 -  
Epoch 26/50
```

282/282 3s 12ms/step - loss: 1.4773e-04 - mae: 0.0091
 Epoch 27/50
282/282 2s 9ms/step - loss: 1.1522e-04 - mae: 0.0080 -
 Epoch 28/50
282/282 3s 9ms/step - loss: 1.1145e-04 - mae: 0.0078 -
 Epoch 29/50
282/282 2s 9ms/step - loss: 1.0188e-04 - mae: 0.0074 -
 Epoch 30/50
274/282 0s 5ms/step - loss: 1.0099e-04 - mae: 0.0075 -
282/282 3s 12ms/step - loss: 1.0096e-04 - mae: 0.0075
 Epoch 31/50
282/282 3s 10ms/step - loss: 1.0311e-04 - mae: 0.0075
 Epoch 32/50
282/282 2s 9ms/step - loss: 9.8021e-05 - mae: 0.0073 -
 Epoch 33/50
282/282 2s 9ms/step - loss: 1.0765e-04 - mae: 0.0077 -
 Epoch 34/50
282/282 3s 9ms/step - loss: 9.9419e-05 - mae: 0.0074 -
 Epoch 35/50
278/282 0s 7ms/step - loss: 1.0777e-04 - mae: 0.0077 -
282/282 3s 12ms/step - loss: 1.0782e-04 - mae: 0.0077
 Epoch 36/50
282/282 2s 9ms/step - loss: 9.5774e-05 - mae: 0.0072 -
 Epoch 37/50
282/282 2s 9ms/step - loss: 9.6993e-05 - mae: 0.0072 -
 Epoch 38/50
282/282 2s 8ms/step - loss: 9.6246e-05 - mae: 0.0072 -
 Epoch 39/50
282/282 3s 9ms/step - loss: 9.6059e-05 - mae: 0.0072 -
 Epoch 40/50
282/282 0s 8ms/step - loss: 9.2909e-05 - mae: 0.0071 -
282/282 3s 12ms/step - loss: 9.2914e-05 - mae: 0.0071
 Epoch 41/50
282/282 2s 9ms/step - loss: 1.0984e-04 - mae: 0.0078 -
 Epoch 42/50
282/282 2s 9ms/step - loss: 9.7600e-05 - mae: 0.0071 -
 Epoch 43/50
282/282 3s 9ms/step - loss: 1.0230e-04 - mae: 0.0074 -
 Epoch 44/50
282/282 3s 12ms/step - loss: 1.0626e-04 - mae: 0.0076
 Epoch 45/50
278/282 0s 5ms/step - loss: 1.0596e-04 - mae: 0.0076 -
282/282 2s 9ms/step - loss: 1.0588e-04 - mae: 0.0076 -
 Epoch 46/50
282/282 2s 9ms/step - loss: 9.6236e-05 - mae: 0.0072 -
 Epoch 47/50
282/282 2s 9ms/step - loss: 1.0284e-04 - mae: 0.0074 -
 Epoch 48/50
282/282 3s 9ms/step - loss: 9.7227e-05 - mae: 0.0072 -
 Epoch 49/50
282/282 4s 12ms/step - loss: 1.1311e-04 - mae: 0.0079
 Epoch 50/50
279/282 0s 5ms/step - loss: 1.0827e-04 - mae: 0.0077 -
282/282 2s 9ms/step - loss: 1.0815e-04 - mae: 0.0077 -
282/282 1s 2ms/step
71/71 0s 2ms/step

MAE per Epoch







Train Metrics - MAE: 0.0047, RMSE: 0.0064, MAPE: 0.37%, R2: 0.9973

Test Metrics - MAE: 0.0033, RMSE: 0.0043, MAPE: 0.30%, R2: 0.9951

Epoch 5 | Train MAE: 0.0062, Val MAE: 0.0044 | Train MAPE: 0.49%, Val MAPE

Epoch 10 | Train MAE: 0.0073, Val MAE: 0.0049 | Train MAPE: 0.58%, Val MAPE

Epoch 15 | Train MAE: 0.0065, Val MAE: 0.0041 | Train MAPE: 0.51%, Val MAPE

Epoch 20 | Train MAE: 0.0065, Val MAE: 0.0039 | Train MAPE: 0.50%, Val MAPE

Epoch 25 | Train MAE: 0.0049, Val MAE: 0.0034 | Train MAPE: 0.38%, Val MAPE

Epoch 30 | Train MAE: 0.0055, Val MAE: 0.0035 | Train MAPE: 0.43%, Val MAPE

Epoch 35 | Train MAE: 0.0049, Val MAE: 0.0035 | Train MAPE: 0.39%, Val MAPE

Epoch 40 | Train MAE: 0.0051, Val MAE: 0.0036 | Train MAPE: 0.40%, Val MAPE

Epoch 45 | Train MAE: 0.0044, Val MAE: 0.0030 | Train MAPE: 0.35%, Val MAPE

Epoch 50 | Train MAE: 0.0047, Val MAE: 0.0033 | Train MAPE: 0.37%, Val MAPE

ANN NEW

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_ab
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.callbacks import Callback

# Load and parse datetime correctly
df = pd.read_csv('EUROUSD_ASK_12HOURS.csv')
df.columns = ['Local time', 'Open', 'High', 'Low', 'Close', 'Volume']
df['Local time'] = pd.to_datetime(df['Local time'].str.split('.').str[0], f
df = df[df['Volume'] > 0].copy()
df = df[['Open', 'High', 'Low', 'Close']].astype(float)

# Scale data
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df)

# Create sequences (exclude Close from input features)
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length, :-1]) # Use only Open, High, Low
        y.append(data[i+seq_length, -1]) # Predict Close
    return np.array(X), np.array(y)

SEQ_LENGTH = 3
X, y = create_sequences(scaled, SEQ_LENGTH)

# Split train/test
split = int(0.8 * len(X))
X_train = X[:split]
X_test = X[split:]
y_train = y[:split]
y_test = y[split:]

# Reshape for ANN input
X_train_reshaped = X_train.reshape(X_train.shape[0], -1) # (samples, timestamp)
X_test_reshaped = X_test.reshape(X_test.shape[0], -1)

# Build ANN model
model = Sequential()
model.add(Input(shape=(X_train_reshaped.shape[1],))) # Flattened sequence
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))

# Custom RMSE metric

```

```

def rmse(y_true, y_pred):
    return tf.sqrt(tf.reduce_mean(tf.square(y_pred - y_true)))

# Compile model
model.compile(optimizer='adam', loss='mse', metrics=['mae', rmse])

# Callback to track metrics on original scale
class MetricsCallback(Callback):
    def __init__(self, X_train, y_train, X_val, y_val, scaler, df):
        super().__init__()
        self.X_train = X_train
        self.y_train = y_train
        self.X_val = X_val
        self.y_val = y_val
        self.scaler = scaler
        self.df = df
        self.metrics = {
            'train_mae': [], 'val_mae': [],
            'train_mape': [], 'val_mape': [],
            'train_rmse': [], 'val_rmse': [],
            'train_r2': [], 'val_r2': []
        }

    def on_epoch_end(self, epoch, logs=None):
        # Reshape for ANN prediction
        X_train_reshaped = self.X_train.reshape(len(self.X_train), -1)
        X_val_reshaped = self.X_val.reshape(len(self.X_val), -1)

        train_pred = self.model.predict(X_train_reshaped, verbose=0).flatten()
        val_pred = self.model.predict(X_val_reshaped, verbose=0).flatten()

        # Use last time step features for inverse transform
        X_train_last = self.X_train[:, -1, :]
        X_val_last = self.X_val[:, -1, :]

        def inverse_transform(preds, X_original_last):
            dummy = np.zeros((len(preds), self.df.shape[1]))
            dummy[:, :-1] = X_original_last
            dummy[:, -1] = preds.ravel()
            dummy = self.scaler.inverse_transform(dummy)
            return dummy[:, -1]

        train_pred_inv = inverse_transform(train_pred, X_train_last)
        val_pred_inv = inverse_transform(val_pred, X_val_last)
        train_actuals_inv = inverse_transform(self.y_train, X_train_last)
        val_actuals_inv = inverse_transform(self.y_val, X_val_last)

        def calculate_metrics(y_true, y_pred):
            mae = mean_absolute_error(y_true, y_pred)
            rmse = np.sqrt(mean_squared_error(y_true, y_pred))
            mape = mean_absolute_percentage_error(y_true, y_pred) * 100
            r2 = r2_score(y_true, y_pred)
            return mae, mape, rmse, r2

        train_mae, train_mape, train_rmse, train_r2 = calculate_metrics(trai

```

```

val_mae, val_mape, val_rmse, val_r2 = calculate_metrics(val_actuals_)

    self.metrics['train_mae'].append(train_mae)
    self.metrics['val_mae'].append(val_mae)
    self.metrics['train_mape'].append(train_mape)
    self.metrics['val_mape'].append(val_mape)
    self.metrics['train_rmse'].append(train_rmse)
    self.metrics['val_rmse'].append(val_rmse)
    self.metrics['train_r2'].append(train_r2)
    self.metrics['val_r2'].append(val_r2)

if (epoch + 1) % 5 == 0:
    print(f"Epoch {epoch+1:2d} | "
          f"Train MAPE: {train_mape:.2f}%, Val MAPE: {val_mape:.2f}%
          f"Train R2: {train_r2:.4f}, Val R2: {val_r2:.4f}")

# Initialize callback
metrics_callback = MetricsCallback(X_train, y_train, X_test, y_test, scaler,

# Train model
history_ann = model.fit(
    X_train_reshaped, y_train,
    epochs=50,
    batch_size=32,
    validation_data=(X_test_reshaped, y_test),
    callbacks=[metrics_callback]
)

# Predictions
train_pred_ann = model.predict(X_train_reshaped).flatten()
test_pred_ann = model.predict(X_test_reshaped).flatten()

# Inverse transform predictions
def inverse_transform(preds, X_original_last):
    dummy = np.zeros((len(preds), df.shape[1]))
    dummy[:, :-1] = X_original_last
    dummy[:, -1] = preds.ravel()
    dummy = scaler.inverse_transform(dummy)
    return dummy[:, -1]

train_pred_inv_ann = inverse_transform(train_pred_ann, X_train[:, -1, :])
test_pred_inv_ann = inverse_transform(test_pred_ann, X_test[:, -1, :])

# Actual values (invert y_train/y_test using corresponding X)
train_actuals = inverse_transform(y_train, X_train[:, -1, :])
test_actuals = inverse_transform(y_test, X_test[:, -1, :])

# Compute final metrics
def evaluate(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse_val = np.sqrt(mean_squared_error(y_true, y_pred))
    mape_val = mean_absolute_percentage_error(y_true, y_pred) * 100
    r2_val = r2_score(y_true, y_pred)
    return [mae, rmse_val, mape_val, r2_val]

```

```

train_metrics_ann = evaluate(train_actuals, train_pred_inv_ann)
test_metrics_ann = evaluate(test_actuals, test_pred_inv_ann)

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(history_ann.history['mae'], label='Train MAE (Scaled)')
plt.plot(history_ann.history['val_mae'], label='Val MAE (Scaled)')
plt.plot(metrics_callback.metrics['train_mae'], label='Train MAE (Original Scale')
plt.plot(metrics_callback.metrics['val_mae'], label='Val MAE (Original Scale')
plt.legend()
plt.title('MAE per Epoch')
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(metrics_callback.metrics['train_mape'], label='Train MAPE (Original Scale')
plt.plot(metrics_callback.metrics['val_mape'], label='Val MAPE (Original Scale')
plt.legend()
plt.title('MAPE per Epoch (Original Scale)')
plt.xlabel('Epoch')
plt.ylabel('MAPE (%)')
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(metrics_callback.metrics['train_r2'], label='Train R2')
plt.plot(metrics_callback.metrics['val_r2'], label='Val R2)
plt.axhline(y=0, color='r', linestyle='--', label='Baseline (R2 = 0)')
plt.legend()
plt.title('R2 Score Progression')
plt.xlabel('Epoch')
plt.ylabel('R2')
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(test_actuals, label='Actual')
plt.plot(test_pred_inv_ann, label='Predicted')
plt.legend()
plt.title('Test Set Predictions')
plt.show()

# Print final metrics
print(f"Train Metrics - MAE: {train_metrics_ann[0]:.4f}, RMSE: {train_metrics_a
print(f"Test Metrics - MAE: {test_metrics_ann[0]:.4f}, RMSE: {test_metrics_a

# Print metrics every 5 epochs
for epoch in range(4, 50, 5): # Starts from epoch 5 (index 4)
    print(f"Epoch {epoch+1:2d} | "
          f"Train MAE: {metrics_callback.metrics['train_mae'][epoch]:.4f}, "
          f"Val MAE: {metrics_callback.metrics['val_mae'][epoch]:.4f} | "
          f"Train MAPE: {metrics_callback.metrics['train_mape'][epoch]:.2f}% "
          f"Val MAPE: {metrics_callback.metrics['val_mape'][epoch]:.2f}% | "
          f"Train RMSE: {metrics_callback.metrics['train_rmse'][epoch]:.4f}, "
          f"Val RMSE: {metrics_callback.metrics['val_rmse'][epoch]:.4f} | "

```

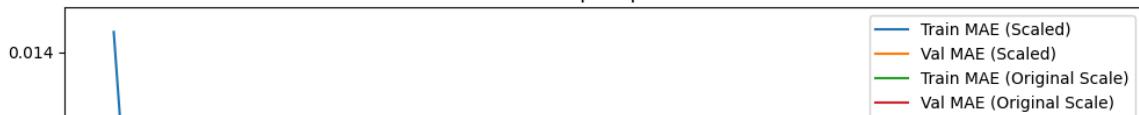
```
f"Train R2: {metrics_callback.metrics['train_r2'][epoch]:.4f}, "
f"Val R2: {metrics_callback.metrics['val_r2'][epoch]:.4f}")
```

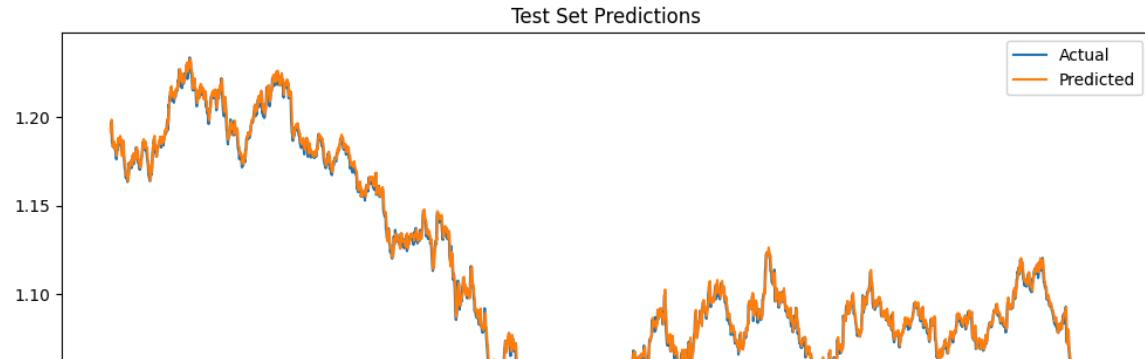
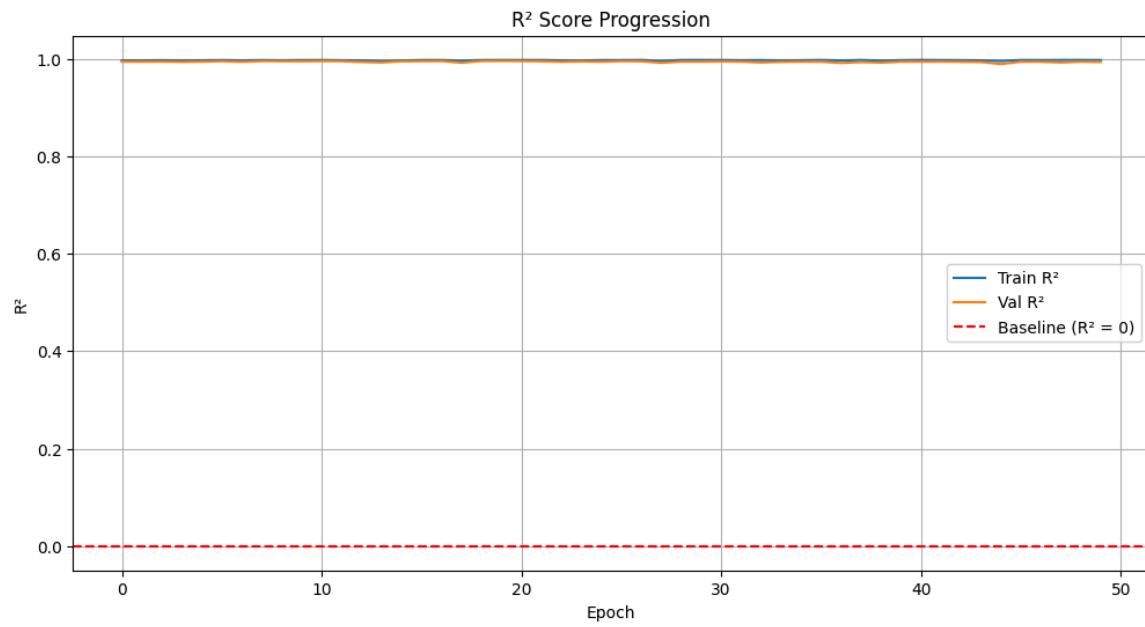
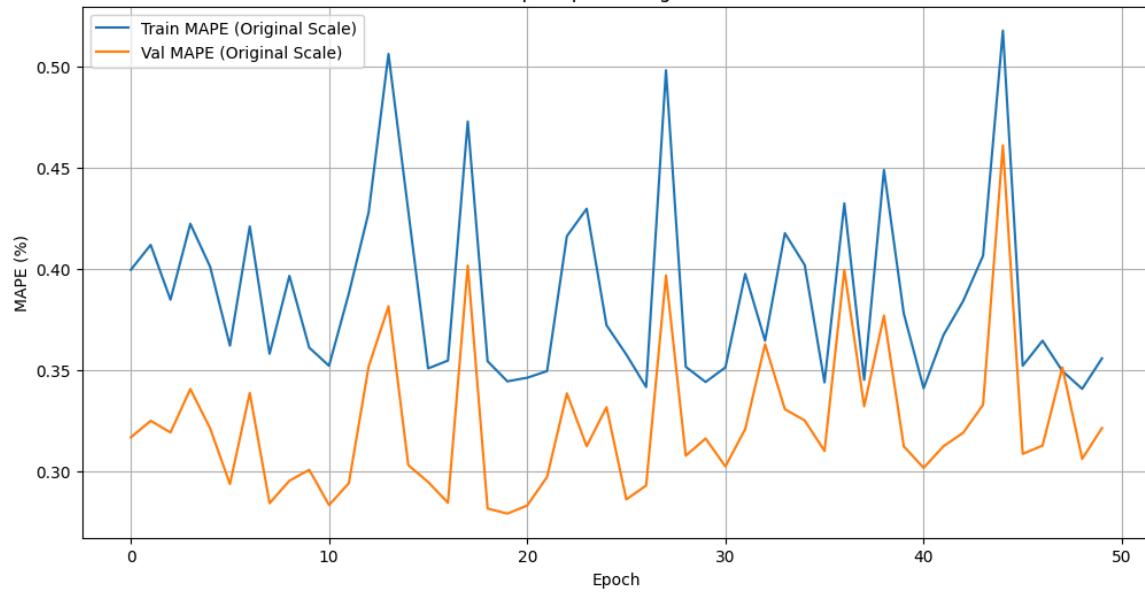
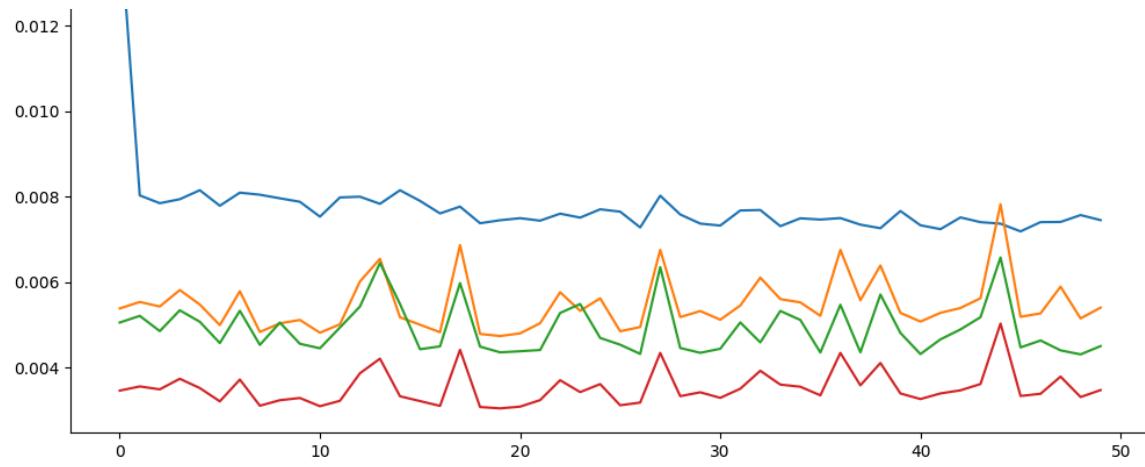


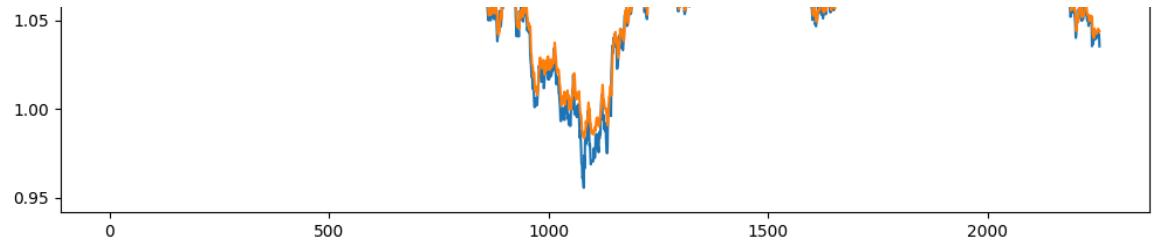
```
Epoch 1/50
<>:14: SyntaxWarning: invalid escape sequence '\.'
<>:14: SyntaxWarning: invalid escape sequence '\.'
/tmp/ipython-input-3647764198.py:14: SyntaxWarning: invalid escape sequence
    df['Local time'] = pd.to_datetime(df['Local time'].str.split('.').str[0])
282/282 2s 5ms/step - loss: 0.0032 - mae: 0.0309 - rmsd: 0.0309
Epoch 2/50
282/282 1s 4ms/step - loss: 1.2288e-04 - mae: 0.0081 - rmsd: 0.0081
Epoch 3/50
282/282 1s 4ms/step - loss: 1.1923e-04 - mae: 0.0079 - rmsd: 0.0079
Epoch 4/50
282/282 3s 5ms/step - loss: 1.1294e-04 - mae: 0.0078 - rmsd: 0.0078
Epoch 5/50
274/282 0s 2ms/step - loss: 1.1372e-04 - mae: 0.0078 - rmsd: 0.0078
282/282 1s 4ms/step - loss: 1.1394e-04 - mae: 0.0078 - rmsd: 0.0078
Epoch 6/50
282/282 1s 4ms/step - loss: 1.1121e-04 - mae: 0.0078 - rmsd: 0.0078
Epoch 7/50
282/282 1s 4ms/step - loss: 1.2370e-04 - mae: 0.0082 - rmsd: 0.0082
Epoch 8/50
282/282 1s 4ms/step - loss: 1.1414e-04 - mae: 0.0080 - rmsd: 0.0080
Epoch 9/50
282/282 1s 4ms/step - loss: 1.0606e-04 - mae: 0.0076 - rmsd: 0.0076
Epoch 10/50
269/282 0s 2ms/step - loss: 1.1389e-04 - mae: 0.0079 - rmsd: 0.0079
282/282 1s 4ms/step - loss: 1.1386e-04 - mae: 0.0079 - rmsd: 0.0079
Epoch 11/50
282/282 1s 4ms/step - loss: 1.0811e-04 - mae: 0.0076 - rmsd: 0.0076
Epoch 12/50
282/282 2s 7ms/step - loss: 1.0826e-04 - mae: 0.0077 - rmsd: 0.0077
Epoch 13/50
282/282 1s 5ms/step - loss: 1.1148e-04 - mae: 0.0078 - rmsd: 0.0078
Epoch 14/50
282/282 1s 4ms/step - loss: 1.1227e-04 - mae: 0.0079 - rmsd: 0.0079
Epoch 15/50
266/282 0s 2ms/step - loss: 1.1583e-04 - mae: 0.0080 - rmsd: 0.0080
282/282 1s 4ms/step - loss: 1.1598e-04 - mae: 0.0080 - rmsd: 0.0080
Epoch 16/50
282/282 1s 4ms/step - loss: 1.1335e-04 - mae: 0.0080 - rmsd: 0.0080
Epoch 17/50
282/282 1s 4ms/step - loss: 1.0843e-04 - mae: 0.0076 - rmsd: 0.0076
Epoch 18/50
282/282 1s 4ms/step - loss: 1.0685e-04 - mae: 0.0076 - rmsd: 0.0076
Epoch 19/50
282/282 1s 4ms/step - loss: 1.0450e-04 - mae: 0.0075 - rmsd: 0.0075
Epoch 20/50
269/282 0s 2ms/step - loss: 1.0503e-04 - mae: 0.0076 - rmsd: 0.0076
282/282 1s 4ms/step - loss: 1.0495e-04 - mae: 0.0076 - rmsd: 0.0076
Epoch 21/50
282/282 2s 6ms/step - loss: 1.0665e-04 - mae: 0.0077 - rmsd: 0.0077
Epoch 22/50
282/282 2s 6ms/step - loss: 9.9383e-05 - mae: 0.0073 - rmsd: 0.0073
Epoch 23/50
282/282 1s 4ms/step - loss: 1.0491e-04 - mae: 0.0076 - rmsd: 0.0076
Epoch 24/50
282/282 1s 4ms/step - loss: 1.0580e-04 - mae: 0.0076 - rmsd: 0.0076
Epoch 25/50
281/282 0s 2ms/step - loss: 1.0588e-04 - mae: 0.0076 - rmsd: 0.0076
282/282 1s 4ms/step - loss: 1.0589e-04 - mae: 0.0076 - rmsd: 0.0076
Epoch 26/50
```

282/282 1s 4ms/step - loss: 1.1275e-04 - mae: 0.0079 -
 Epoch 27/50
282/282 1s 4ms/step - loss: 9.8479e-05 - mae: 0.0073 -
 Epoch 28/50
282/282 1s 4ms/step - loss: 1.1408e-04 - mae: 0.0080 -
 Epoch 29/50
282/282 1s 4ms/step - loss: 1.0919e-04 - mae: 0.0078 -
 Epoch 30/50
282/282 0s 2ms/step - loss: 1.0223e-04 - mae: 0.0074 -
282/282 2s 5ms/step - loss: 1.0222e-04 - mae: 0.0074 -
 Epoch 31/50
282/282 2s 5ms/step - loss: 9.7726e-05 - mae: 0.0072 -
 Epoch 32/50
282/282 1s 4ms/step - loss: 1.0425e-04 - mae: 0.0075 -
 Epoch 33/50
282/282 1s 4ms/step - loss: 1.1693e-04 - mae: 0.0080 -
 Epoch 34/50
282/282 1s 4ms/step - loss: 9.1132e-05 - mae: 0.0070 -
 Epoch 35/50
264/282 0s 2ms/step - loss: 1.0456e-04 - mae: 0.0076 -
282/282 1s 4ms/step - loss: 1.0448e-04 - mae: 0.0076 -
 Epoch 36/50
282/282 1s 4ms/step - loss: 1.0758e-04 - mae: 0.0077 -
 Epoch 37/50
282/282 1s 4ms/step - loss: 9.6773e-05 - mae: 0.0073 -
 Epoch 38/50
282/282 1s 4ms/step - loss: 9.6888e-05 - mae: 0.0072 -
 Epoch 39/50
282/282 2s 7ms/step - loss: 1.0573e-04 - mae: 0.0075 -
 Epoch 40/50
271/282 0s 3ms/step - loss: 1.0290e-04 - mae: 0.0076 -
282/282 2s 5ms/step - loss: 1.0304e-04 - mae: 0.0076 -
 Epoch 41/50
282/282 1s 4ms/step - loss: 1.0056e-04 - mae: 0.0073 -
 Epoch 42/50
282/282 1s 4ms/step - loss: 1.0032e-04 - mae: 0.0073 -
 Epoch 43/50
282/282 1s 4ms/step - loss: 1.0230e-04 - mae: 0.0075 -
 Epoch 44/50
282/282 1s 4ms/step - loss: 9.9341e-05 - mae: 0.0073 -
 Epoch 45/50
276/282 0s 2ms/step - loss: 9.7594e-05 - mae: 0.0073 -
282/282 1s 4ms/step - loss: 9.7648e-05 - mae: 0.0073 -
 Epoch 46/50
282/282 1s 4ms/step - loss: 9.9372e-05 - mae: 0.0073 -
 Epoch 47/50
282/282 1s 4ms/step - loss: 1.0624e-04 - mae: 0.0075 -
 Epoch 48/50
282/282 2s 6ms/step - loss: 1.0045e-04 - mae: 0.0074 -
 Epoch 49/50
282/282 2s 6ms/step - loss: 1.0271e-04 - mae: 0.0076 -
 Epoch 50/50
275/282 0s 2ms/step - loss: 9.9250e-05 - mae: 0.0073 -
282/282 1s 4ms/step - loss: 9.9340e-05 - mae: 0.0073 -
282/282 0s 1ms/step
71/71 0s 1ms/step

MAE per Epoch







Train Metrics - MAE: 0.0045, RMSE: 0.0062, MAPE: 0.36%, R2: 0.9974

Test Metrics - MAE: 0.0035, RMSE: 0.0049, MAPE: 0.32%, R2: 0.9936

Epoch 5 | Train MAE: 0.0051, Val MAE: 0.0035 | Train MAPE: 0.40%, Val MAPE

Epoch 10 | Train MAE: 0.0046, Val MAE: 0.0033 | Train MAPE: 0.36%, Val MAPE

Epoch 15 | Train MAE: 0.0055, Val MAE: 0.0033 | Train MAPE: 0.43%, Val MAPE

Epoch 20 | Train MAE: 0.0044, Val MAE: 0.0031 | Train MAPE: 0.34%, Val MAPE

Epoch 25 | Train MAE: 0.0047, Val MAE: 0.0036 | Train MAPE: 0.37%, Val MAPE

Epoch 30 | Train MAE: 0.0044, Val MAE: 0.0034 | Train MAPE: 0.34%, Val MAPE

Epoch 35 | Train MAE: 0.0051, Val MAE: 0.0036 | Train MAPE: 0.40%, Val MAPE

Epoch 40 | Train MAE: 0.0048, Val MAE: 0.0034 | Train MAPE: 0.38%, Val MAPE

Epoch 45 | Train MAE: 0.0066, Val MAE: 0.0050 | Train MAPE: 0.52%, Val MAPE

Epoch 50 | Train MAE: 0.0045, Val MAE: 0.0035 | Train MAPE: 0.36%, Val MAPE

CNN NEW

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_ab
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, In
from tensorflow.keras.callbacks import Callback

# Load and parse datetime correctly
df = pd.read_csv('EUROUSD_ASK_12HOURS.csv')
df.columns = ['Local time', 'Open', 'High', 'Low', 'Close', 'Volume']
df['Local time'] = pd.to_datetime(df['Local time'].str.split('.').str[0], f
df = df[df['Volume'] > 0].copy()
df = df[['Open', 'High', 'Low', 'Close']].astype(float)

# Scale data
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df)

# Create sequences (exclude Close from input features)
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length, :-1]) # Use only Open, High, Low
        y.append(data[i+seq_length, -1]) # Predict Close
    return np.array(X), np.array(y)

SEQ_LENGTH = 3
X, y = create_sequences(scaled, SEQ_LENGTH)

# Split train/test
split = int(0.8 * len(X))
X_train = X[:split]
X_test = X[split:]
y_train = y[:split]
y_test = y[split:]

# Build CNN model
model = Sequential()
model.add(Input(shape=(X_train.shape[1], X_train.shape[2]))) # (timesteps,
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', padding='same'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(1))

# Custom RMSE metric
def rmse(y_true, y_pred):
    return tf.sqrt(tf.reduce_mean(tf.square(y_pred - y_true)))

```

```

# Compile model
model.compile(optimizer='adam', loss='mse', metrics=['mae', rmse])

# Callback to track metrics on original scale
class MetricsCallback(Callback):
    def __init__(self, X_train, y_train, X_val, y_val, scaler, df):
        super().__init__()
        self.X_train = X_train
        self.y_train = y_train
        self.X_val = X_val
        self.y_val = y_val
        self.scaler = scaler
        self.df = df
        self.metrics = {
            'train_mae': [], 'val_mae': [],
            'train_mape': [], 'val_mape': [],
            'train_rmse': [], 'val_rmse': [],
            'train_r2': [], 'val_r2': []
        }

    def on_epoch_end(self, epoch, logs=None):
        # Predict on train and validation data
        train_pred = self.model.predict(self.X_train, verbose=0).flatten()
        val_pred = self.model.predict(self.X_val, verbose=0).flatten()

        # Use last time step features for inverse transform
        X_train_last = self.X_train[:, -1, :]
        X_val_last = self.X_val[:, -1, :]

        def inverse_transform(preds, X_original_last):
            dummy = np.zeros((len(preds), self.df.shape[1]))
            dummy[:, :-1] = X_original_last
            dummy[:, -1] = preds.ravel()
            dummy = self.scaler.inverse_transform(dummy)
            return dummy[:, -1]

        train_pred_inv = inverse_transform(train_pred, X_train_last)
        val_pred_inv = inverse_transform(val_pred, X_val_last)
        train_actuals_inv = inverse_transform(self.y_train, X_train_last)
        val_actuals_inv = inverse_transform(self.y_val, X_val_last)

        def calculate_metrics(y_true, y_pred):
            mae = mean_absolute_error(y_true, y_pred)
            rmse = np.sqrt(mean_squared_error(y_true, y_pred))
            mape = mean_absolute_percentage_error(y_true, y_pred) * 100
            r2 = r2_score(y_true, y_pred)
            return mae, mape, rmse, r2

        train_mae, train_mape, train_rmse, train_r2 = calculate_metrics(train_pred_inv, train_actuals_inv)
        val_mae, val_mape, val_rmse, val_r2 = calculate_metrics(val_pred_inv, val_actuals_inv)

        self.metrics['train_mae'].append(train_mae)
        self.metrics['val_mae'].append(val_mae)
        self.metrics['train_mape'].append(train_mape)

```

```
        self.metrics['val_mape'].append(val_mape)
        self.metrics['train_rmse'].append(train_rmse)
        self.metrics['val_rmse'].append(val_rmse)
        self.metrics['train_r2'].append(train_r2)
        self.metrics['val_r2'].append(val_r2)

    if (epoch + 1) % 5 == 0:
        print(f"Epoch {epoch+1:2d} | "
              f"Train MAPE: {train_mape:.2f}%, Val MAPE: {val_mape:.2f}%
              f"Train R2: {train_r2:.4f}, Val R2: {val_r2:.4f}")

# Initialize callback
metrics_callback = MetricsCallback(X_train, y_train, X_test, y_test, scaler,
```



```
# Train model
history_cnn = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_data=(X_test, y_test),
    callbacks=[metrics_callback]
)

# Predictions
train_pred_cnn = model.predict(X_train).flatten()
test_pred_cnn = model.predict(X_test).flatten()

# Inverse transform predictions
def inverse_transform(preds, X_original_last):
    dummy = np.zeros((len(preds), df.shape[1]))
    dummy[:, :-1] = X_original_last
    dummy[:, -1] = preds.ravel()
    dummy = scaler.inverse_transform(dummy)
    return dummy[:, -1]

train_pred_inv_cnn = inverse_transform(train_pred_cnn, X_train[:, -1, :])
test_pred_inv_cnn = inverse_transform(test_pred_cnn, X_test[:, -1, :])

# Actual values (invert y_train/y_test using corresponding X)
train_actuals = inverse_transform(y_train, X_train[:, -1, :])
test_actuals = inverse_transform(y_test, X_test[:, -1, :])

# Compute final metrics
def evaluate(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse_val = np.sqrt(mean_squared_error(y_true, y_pred))
    mape_val = mean_absolute_percentage_error(y_true, y_pred) * 100
    r2_val = r2_score(y_true, y_pred)
    return [mae, rmse_val, mape_val, r2_val]

train_metrics_cnn = evaluate(train_actuals, train_pred_inv_cnn)
test_metrics_cnn = evaluate(test_actuals, test_pred_inv_cnn)

# Plotting
plt.figure(figsize=(12, 6))
```

```

plt.plot(history_cnn.history['mae'], label='Train MAE (Scaled)')
plt.plot(history_cnn.history['val_mae'], label='Val MAE (Scaled)')
plt.plot(metrics_callback.metrics['train_mae'], label='Train MAE (Original Scale')
plt.plot(metrics_callback.metrics['val_mae'], label='Val MAE (Original Scale')
plt.legend()
plt.title('MAE per Epoch')
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(metrics_callback.metrics['train_mape'], label='Train MAPE (Original Scale')
plt.plot(metrics_callback.metrics['val_mape'], label='Val MAPE (Original Scale')
plt.legend()
plt.title('MAPE per Epoch (Original Scale)')
plt.xlabel('Epoch')
plt.ylabel('MAPE (%)')
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(metrics_callback.metrics['train_r2'], label='Train R2')
plt.plot(metrics_callback.metrics['val_r2'], label='Val R2)
plt.axhline(y=0, color='r', linestyle='--', label='Baseline (R2 = 0)')
plt.legend()
plt.title('R2 Score Progression')
plt.xlabel('Epoch')
plt.ylabel('R2')
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(test_actuals, label='Actual')
plt.plot(test_pred_inv_cnn, label='Predicted')
plt.legend()
plt.title('Test Set Predictions')
plt.show()

# Print final metrics
print(f"Train Metrics - MAE: {train_metrics_cnn[0]:.4f}, RMSE: {train_metrics_cnn[1]:.4f}")
print(f"Test Metrics - MAE: {test_metrics_cnn[0]:.4f}, RMSE: {test_metrics_cnn[1]:.4f}")

# Print metrics every 5 epochs
for epoch in range(4, 50, 5): # Starts from epoch 5 (index 4)
    print(f"Epoch {epoch+1:2d} | "
          f"Train MAE: {metrics_callback.metrics['train_mae'][epoch]:.4f}, "
          f"Val MAE: {metrics_callback.metrics['val_mae'][epoch]:.4f} | "
          f"Train MAPE: {metrics_callback.metrics['train_mape'][epoch]:.2f}%, "
          f"Val MAPE: {metrics_callback.metrics['val_mape'][epoch]:.2f}% | "
          f"Train RMSE: {metrics_callback.metrics['train_rmse'][epoch]:.4f}, "
          f"Val RMSE: {metrics_callback.metrics['val_rmse'][epoch]:.4f} | "
          f"Train R2: {metrics_callback.metrics['train_r2'][epoch]:.4f}, "
          f"Val R2: {metrics_callback.metrics['val_r2'][epoch]:.4f}")

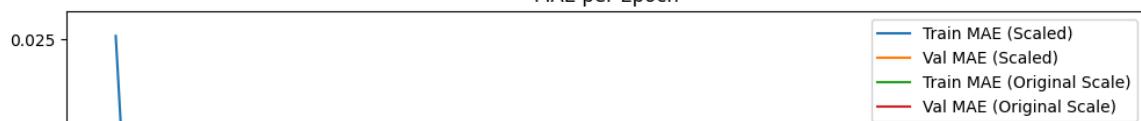
```

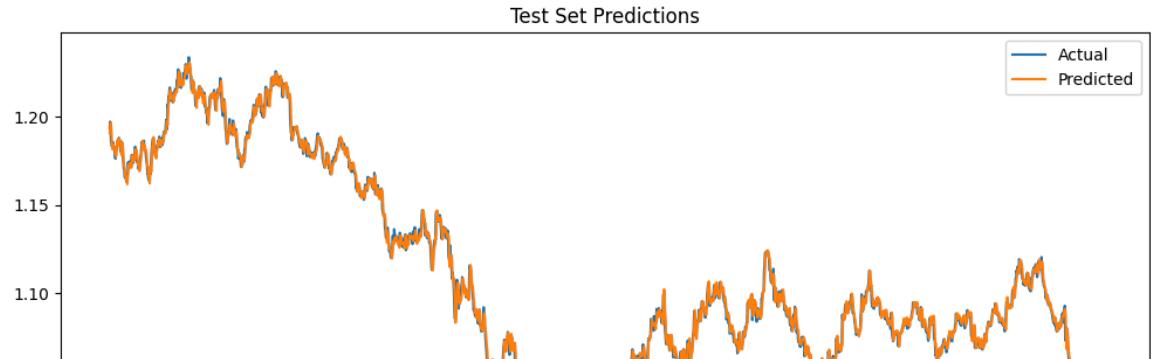
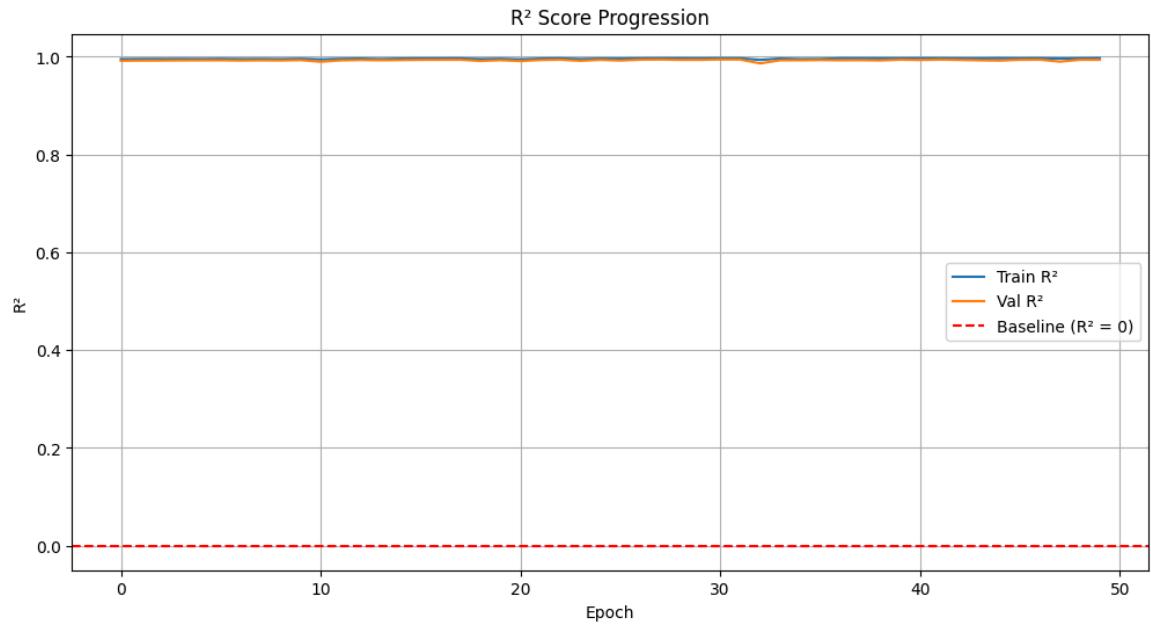
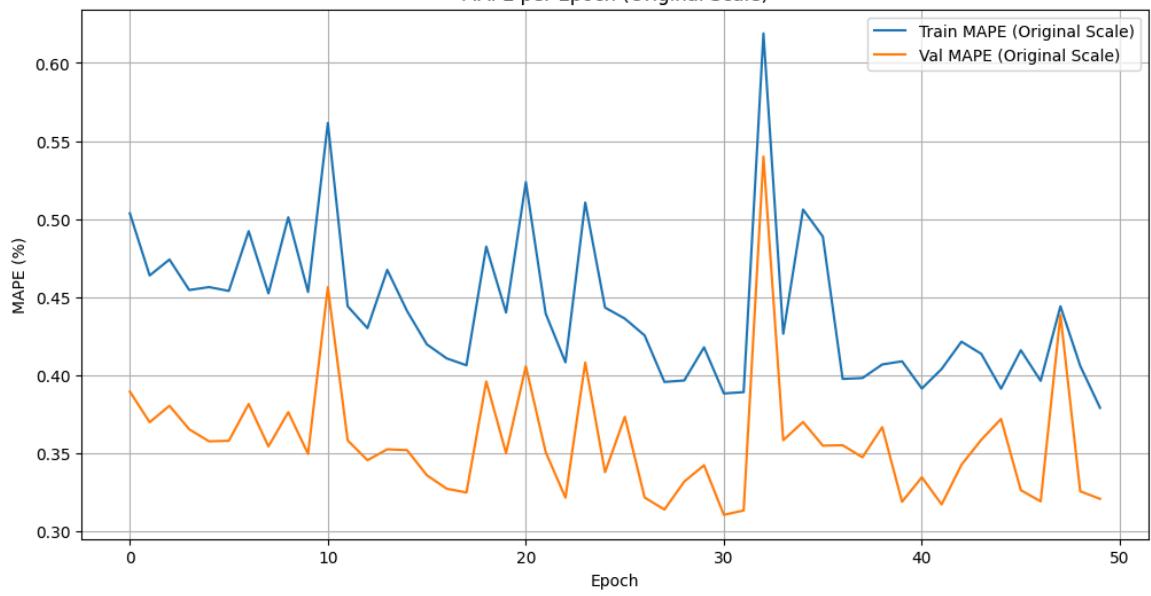
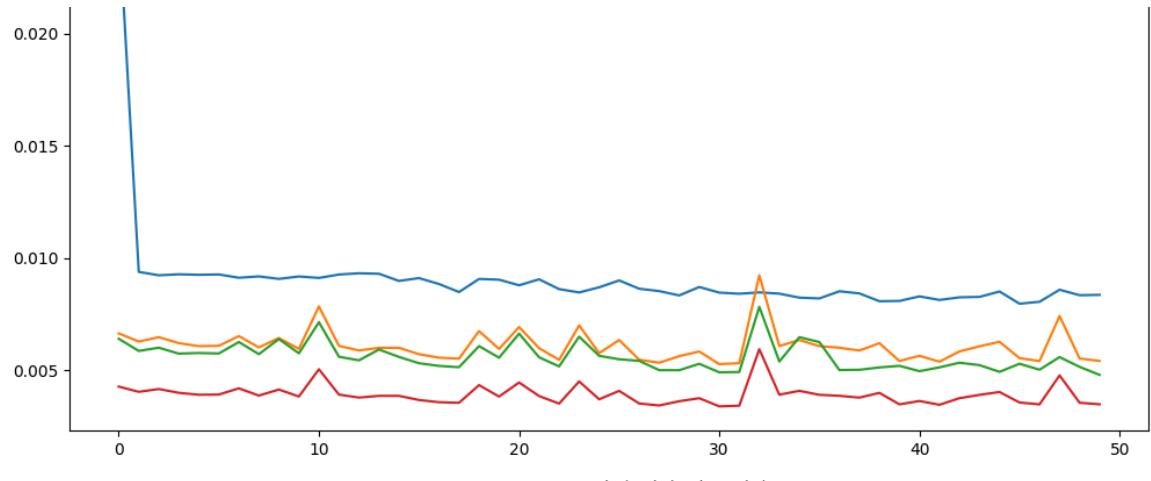


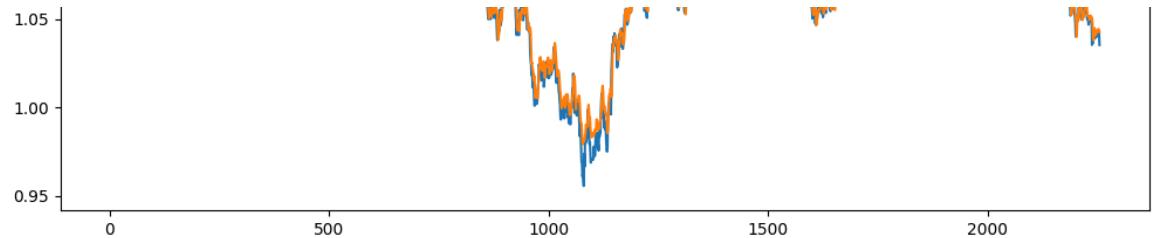
```
Epoch 1/50
<>:14: SyntaxWarning: invalid escape sequence '\.'
<>:14: SyntaxWarning: invalid escape sequence '\.'
/tmp/ipython-input-2454966769.py:14: SyntaxWarning: invalid escape sequence
    df['Local time'] = pd.to_datetime(df['Local time'].str.split('.').str[0])
282/282 3s 5ms/step - loss: 0.0153 - mae: 0.0634 - rmsle: 0.153
Epoch 2/50
282/282 1s 4ms/step - loss: 1.6448e-04 - mae: 0.0093 - rmsle: 0.0093
Epoch 3/50
282/282 1s 4ms/step - loss: 1.5803e-04 - mae: 0.0092 - rmsle: 0.0092
Epoch 4/50
282/282 2s 5ms/step - loss: 1.6479e-04 - mae: 0.0094 - rmsle: 0.0094
Epoch 5/50
266/282 0s 3ms/step - loss: 1.5499e-04 - mae: 0.0092 - rmsle: 0.0092
282/282 2s 7ms/step - loss: 1.5528e-04 - mae: 0.0092 - rmsle: 0.0092
Epoch 6/50
282/282 1s 5ms/step - loss: 1.5930e-04 - mae: 0.0093 - rmsle: 0.0093
Epoch 7/50
282/282 1s 4ms/step - loss: 1.5461e-04 - mae: 0.0091 - rmsle: 0.0091
Epoch 8/50
282/282 2s 5ms/step - loss: 1.5349e-04 - mae: 0.0092 - rmsle: 0.0092
Epoch 9/50
282/282 1s 4ms/step - loss: 1.5048e-04 - mae: 0.0090 - rmsle: 0.0090
Epoch 10/50
281/282 0s 2ms/step - loss: 1.5759e-04 - mae: 0.0092 - rmsle: 0.0092
282/282 1s 4ms/step - loss: 1.5757e-04 - mae: 0.0092 - rmsle: 0.0092
Epoch 11/50
282/282 1s 4ms/step - loss: 1.4905e-04 - mae: 0.0091 - rmsle: 0.0091
Epoch 12/50
282/282 1s 4ms/step - loss: 1.6414e-04 - mae: 0.0095 - rmsle: 0.0095
Epoch 13/50
282/282 2s 6ms/step - loss: 1.8653e-04 - mae: 0.0101 - rmsle: 0.0101
Epoch 14/50
282/282 2s 6ms/step - loss: 1.5221e-04 - mae: 0.0093 - rmsle: 0.0093
Epoch 15/50
254/282 0s 2ms/step - loss: 1.4168e-04 - mae: 0.0089 - rmsle: 0.0089
282/282 1s 4ms/step - loss: 1.4233e-04 - mae: 0.0089 - rmsle: 0.0089
Epoch 16/50
282/282 1s 4ms/step - loss: 1.5030e-04 - mae: 0.0091 - rmsle: 0.0091
Epoch 17/50
282/282 1s 4ms/step - loss: 1.4190e-04 - mae: 0.0088 - rmsle: 0.0088
Epoch 18/50
282/282 1s 4ms/step - loss: 1.3246e-04 - mae: 0.0085 - rmsle: 0.0085
Epoch 19/50
282/282 1s 4ms/step - loss: 1.4139e-04 - mae: 0.0088 - rmsle: 0.0088
Epoch 20/50
255/282 0s 2ms/step - loss: 1.4916e-04 - mae: 0.0092 - rmsle: 0.0092
282/282 1s 4ms/step - loss: 1.4906e-04 - mae: 0.0092 - rmsle: 0.0092
Epoch 21/50
282/282 1s 4ms/step - loss: 1.5068e-04 - mae: 0.0090 - rmsle: 0.0090
Epoch 22/50
282/282 2s 6ms/step - loss: 1.5845e-04 - mae: 0.0094 - rmsle: 0.0094
Epoch 23/50
282/282 2s 6ms/step - loss: 1.3247e-04 - mae: 0.0085 - rmsle: 0.0085
Epoch 24/50
282/282 1s 5ms/step - loss: 1.3426e-04 - mae: 0.0084 - rmsle: 0.0084
Epoch 25/50
257/282 0s 2ms/step - loss: 1.3908e-04 - mae: 0.0088 - rmsle: 0.0088
282/282 1s 4ms/step - loss: 1.3895e-04 - mae: 0.0088 - rmsle: 0.0088
Epoch 26/50
```

282/282 1s 4ms/step - loss: 1.4174e-04 - mae: 0.0089 -
 Epoch 27/50
282/282 1s 4ms/step - loss: 1.2909e-04 - mae: 0.0085 -
 Epoch 28/50
282/282 1s 5ms/step - loss: 1.2723e-04 - mae: 0.0083 -
 Epoch 29/50
282/282 1s 4ms/step - loss: 1.2607e-04 - mae: 0.0084 -
 Epoch 30/50
260/282 0s 2ms/step - loss: 1.3198e-04 - mae: 0.0085 -
282/282 1s 4ms/step - loss: 1.3255e-04 - mae: 0.0085 -
 Epoch 31/50
282/282 2s 7ms/step - loss: 1.3087e-04 - mae: 0.0085 -
 Epoch 32/50
282/282 2s 6ms/step - loss: 1.3538e-04 - mae: 0.0086 -
 Epoch 33/50
282/282 2s 5ms/step - loss: 1.3096e-04 - mae: 0.0084 -
 Epoch 34/50
282/282 1s 5ms/step - loss: 1.3631e-04 - mae: 0.0087 -
 Epoch 35/50
257/282 0s 2ms/step - loss: 1.2795e-04 - mae: 0.0084 -
282/282 1s 4ms/step - loss: 1.2761e-04 - mae: 0.0084 -
 Epoch 36/50
282/282 1s 5ms/step - loss: 1.2653e-04 - mae: 0.0084 -
 Epoch 37/50
282/282 1s 5ms/step - loss: 1.2944e-04 - mae: 0.0086 -
 Epoch 38/50
282/282 1s 4ms/step - loss: 1.3369e-04 - mae: 0.0086 -
 Epoch 39/50
282/282 2s 6ms/step - loss: 1.2160e-04 - mae: 0.0081 -
 Epoch 40/50
279/282 0s 3ms/step - loss: 1.2158e-04 - mae: 0.0081 -
282/282 2s 6ms/step - loss: 1.2156e-04 - mae: 0.0081 -
 Epoch 41/50
282/282 1s 5ms/step - loss: 1.2365e-04 - mae: 0.0082 -
 Epoch 42/50
282/282 1s 5ms/step - loss: 1.1569e-04 - mae: 0.0079 -
 Epoch 43/50
282/282 1s 5ms/step - loss: 1.2788e-04 - mae: 0.0084 -
 Epoch 44/50
282/282 1s 4ms/step - loss: 1.2029e-04 - mae: 0.0082 -
 Epoch 45/50
277/282 0s 2ms/step - loss: 1.3254e-04 - mae: 0.0086 -
282/282 1s 5ms/step - loss: 1.3248e-04 - mae: 0.0086 -
 Epoch 46/50
282/282 1s 5ms/step - loss: 1.1200e-04 - mae: 0.0079 -
 Epoch 47/50
282/282 1s 4ms/step - loss: 1.2461e-04 - mae: 0.0083 -
 Epoch 48/50
282/282 2s 6ms/step - loss: 1.2814e-04 - mae: 0.0085 -
 Epoch 49/50
282/282 2s 7ms/step - loss: 1.2397e-04 - mae: 0.0083 -
 Epoch 50/50
263/282 0s 2ms/step - loss: 1.2908e-04 - mae: 0.0085 -
282/282 2s 4ms/step - loss: 1.2880e-04 - mae: 0.0085 -
282/282 0s 1ms/step
71/71 0s 1ms/step

MAE per Epoch







Train Metrics - MAE: 0.0048, RMSE: 0.0066, MAPE: 0.38%, R2: 0.9971

Test Metrics - MAE: 0.0035, RMSE: 0.0048, MAPE: 0.32%, R2: 0.9939

Epoch 5 | Train MAE: 0.0058, Val MAE: 0.0039 | Train MAPE: 0.46%, Val MAPE

Epoch 10 | Train MAE: 0.0058, Val MAE: 0.0038 | Train MAPE: 0.45%, Val MAPE

Epoch 15 | Train MAE: 0.0056, Val MAE: 0.0039 | Train MAPE: 0.44%, Val MAPE

Epoch 20 | Train MAE: 0.0056, Val MAE: 0.0038 | Train MAPE: 0.44%, Val MAPE

Epoch 25 | Train MAE: 0.0056, Val MAE: 0.0037 | Train MAPE: 0.44%, Val MAPE

Epoch 30 | Train MAE: 0.0053, Val MAE: 0.0038 | Train MAPE: 0.42%, Val MAPE

Epoch 35 | Train MAE: 0.0065, Val MAE: 0.0041 | Train MAPE: 0.51%, Val MAPE

Epoch 40 | Train MAE: 0.0052, Val MAE: 0.0035 | Train MAPE: 0.41%, Val MAPE

Epoch 45 | Train MAE: 0.0049, Val MAE: 0.0040 | Train MAPE: 0.39%, Val MAPE

Epoch 50 | Train MAE: 0.0048, Val MAE: 0.0035 | Train MAPE: 0.38%, Val MAPE

Start coding or generate with AI.

```
plt.figure(figsize=(12, 6))
# Plot LSTM
plt.plot(history_lstm.history['val_mae'], label='LSTM Val MAE', linestyle='--')
# Plot ANN
plt.plot(history_ann.history['val_mae'], label='ANN Val MAE', linestyle='-.')
# Plot CNN
plt.plot(history_cnn.history['val_mae'], label='CNN Val MAE', linestyle=':')
plt.legend()
plt.title('Comparison MAE Across Epochs (All Models)')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.grid(True)
plt.show()

# Plotting with enhanced clarity and custom zoom
plt.figure(figsize=(16, 8))

# Full test set comparison (unchanged)
plt.subplot(1, 2, 1)
plt.plot(test_actuals, label='Actual', color='yellow', linewidth=2, zorder=5)
plt.plot(test_pred_inv_lstm, label='LSTM Predicted', color='tab:blue', lines
plt.plot(test_pred_inv_ann, label='ANN Predicted', color='tab:orange', lines
plt.plot(test_pred_inv_cnn, label='CNN Predicted', color='tab:green', linest

plt.title('Model Predictions vs Actual (Full Test Set)', fontsize=14)
plt.xlabel('Time Steps', fontsize=12)
plt.ylabel('Close Price (EUR/USD)', fontsize=12)
plt.legend(loc='upper left', fontsize=10, frameon=True, fancybox=True, shad
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()

# Zoomed-in subplot (focus on steps 800-1200)
zoom_start = 800
zoom_end = 1200

# Check if indices are valid
if zoom_end < len(test_actuals):
    plt.subplot(1, 2, 2)
    plt.plot(
        np.arange(zoom_start, zoom_end),
        test_actuals[zoom_start:zoom_end],
        label='Actual', color='yellow', linewidth=2, zorder=5
    )
    plt.plot(
        np.arange(zoom_start, zoom_end),
        test_pred_inv_lstm[zoom_start:zoom_end],
        label='LSTM', color='tab:blue', linestyle='-', linewidth=2
    )
    plt.plot(
        np.arange(zoom_start, zoom_end),
        test_pred_inv_ann[zoom_start:zoom_end],
```

```
label='ANN', color='tab:orange', linestyle='--', linewidth=2
)
plt.plot(
    np.arange(zoom_start, zoom_end),
    test_pred_inv_cnn[zoom_start:zoom_end],
    label='CNN', color='tab:green', linestyle='--', linewidth=2
)

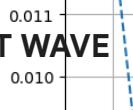
plt.title(f'Zoomed-In Comparison (Steps {zoom_start}-{zoom_end})', font=
plt.xlabel('Time Steps', fontsize=12)
plt.ylabel('Close Price (EUR/USD)', fontsize=12)
plt.legend(loc='upper left', fontsize=10, frameon=True, fancybox=True, s
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
else:
    print(f"Error: Zoom range exceeds test data length ({len(test_actuals)})"

plt.show()
```


Comparison MAE Across Epochs (All Models)

Start coding or generate with AI.

ELLIOT WAVE



```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_err

# Load and preprocess data
# Remove header=None and names parameters to use the first row as header
# Remove explicit dtype specification if columns are consistently numeric af
df = pd.read_csv("EUROUSD_ASK_12HOURS.csv")

# Ensure column names are as expected (adjust if the actual header names are
# Assuming the header row contains 'Local time', 'Open', 'High', 'Low', 'Close',
df.columns = ['datetime', 'Open', 'High', 'Low', 'Close', 'Volume']

# Explicitly convert numeric columns to float after loading
# This handles cases where some rows might have non-numeric data that wasn't
numeric_cols = ['Open', 'High', 'Low', 'Close', 'Volume']
for col in numeric_cols:
    df[col] = pd.to_numeric(df[col], errors='coerce') # Use errors='coerce'

# Drop rows where volume is NaN after coercion
df = df.dropna(subset=['Volume'])
df = df[df['Volume'] > 0].copy()

# Parse datetime correctly
# Adjust the regex if necessary based on the actual datetime string format i
df['datetime'] = pd.to_datetime(df['datetime'].astype(str).str.replace(r'\.\.',

# Drop rows where datetime parsing failed
df = df.dropna(subset=['datetime'])

df = df.sort_values('datetime').reset_index(drop=True)

# Generate predictions using Fibonacci retracement (61.8% level)
predicted_closes = []
actual_closes = []

# Ensure we have enough data for the initial window
if len(df) >= 5:
    for i in range(5, len(df)):
        window = df.iloc[i-5:i]
        high = window['High'].max()
        low = window['Low'].min()
        diff = high - low
        pred_close = high - 0.618 * diff # 61.8% retracement

```

```
predicted_closes.append(pred_close)
actual_closes.append(df.iloc[i]['Close'])

# Convert to numpy arrays
actual = np.array(actual_closes)
predicted = np.array(predicted_closes)
# Ensure dates align with actual/predicted by starting from the 5th index
dates = df['datetime'].iloc[5:].reset_index(drop=True)

# Split into train and test (80/20)
split_idx = int(len(actual) * 0.8)
train_actual, test_actual = actual[:split_idx], actual[split_idx:]
train_pred_ew, test_pred_ew = predicted[:split_idx], predicted[split_idx:]
train_dates, test_dates = dates[:split_idx], dates[split_idx:]

# Evaluate model
def evaluate_model(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mape = mean_absolute_percentage_error(y_true, y_pred) * 100
    r2 = r2_score(y_true, y_pred)
    return {'MAE': mae, 'RMSE': rmse, 'MAPE': mape, 'R2': r2}

train_metrics_ew = evaluate_model(train_actual, train_pred_ew)
test_metrics_ew = evaluate_model(test_actual, test_pred_ew)

# Print metrics
print("Training Metrics:")
# Corrected: Use train_metrics_ew (which is a dictionary) instead of train
for k, v in train_metrics_ew.items():
    print(f"{k}: {v:.4f}")

print("\nTesting Metrics:")
# Corrected: Use test_metrics_ew (which is a dictionary) instead of test
for k, v in test_metrics_ew.items():
    print(f"{k}: {v:.4f}")

# Plotting
plt.figure(figsize=(14, 6))

# Training plot
plt.subplot(1, 2, 1)
plt.plot(train_dates, train_actual, label='Actual (Train)', color='blue')
plt.plot(train_dates, train_pred_ew, label='Predicted (Train)', color='red')
plt.title('Training: Actual vs Predicted')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.xticks(rotation=45)
plt.legend()

# Testing plot
plt.subplot(1, 2, 2)
plt.plot(test_dates, test_actual, label='Actual (Test)', color='green')
```

```

plt.plot(test_dates, test_pred_ew, label='Predicted (Test)', color='orange')
plt.title('Testing: Actual vs Predicted')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.xticks(rotation=45)
plt.legend()

plt.tight_layout()
plt.show()
else:
    print("Not enough data points in the dataframe to perform analysis with")

```

Training Metrics:

MAE: 0.0065

RMSE: 0.0087

MAPE: 0.5100

R2: 0.9950

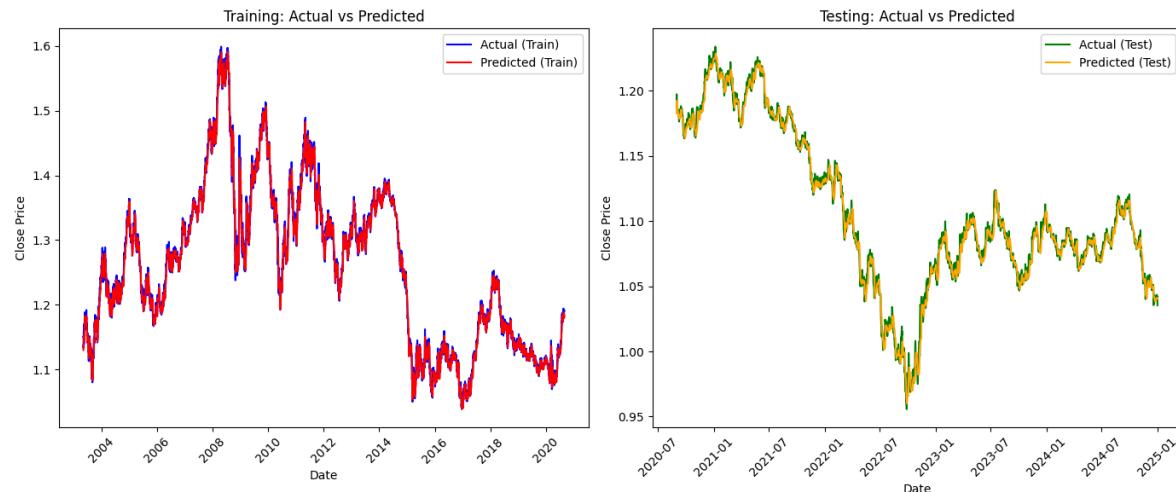
Testing Metrics:

MAE: 0.0043

RMSE: 0.0057

MAPE: 0.3890

R2: 0.9914



Start coding or generate with AI.

Start coding or generate with AI.

```

# 1. Direct Comparison Visualization
plt.figure(figsize=(14, 7))
plt.plot(test_dates, test_actual, label='Actual', color='red', linewidth=2)

```