Start coding or generate with AI.

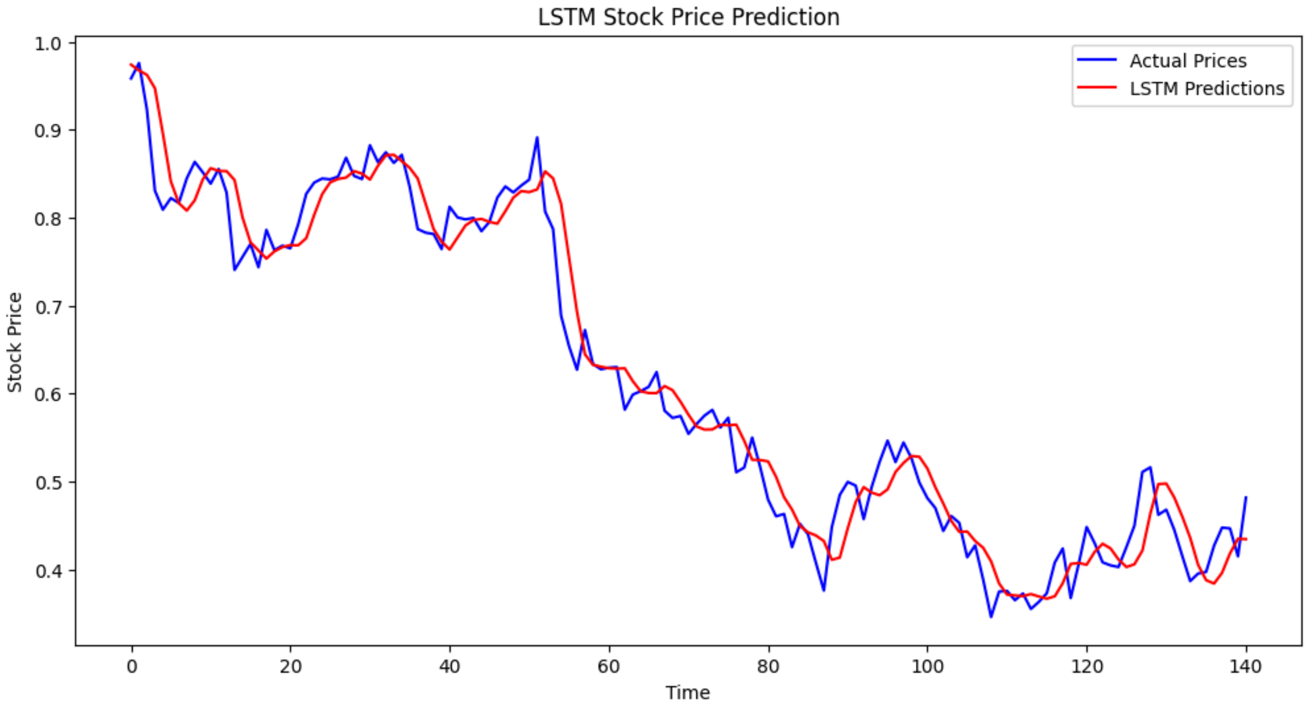
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import LSTM, GRU, Dense, Dropout
import yfinance as yf
import pickle

# Function to load stock data
def load_stock_data(stock_symbol, start_date, end_date):
    data = yf.download(stock_symbol, start=start_date, end=end_date)[['Close', 'High', 'Low', 'Volume']]
    return data

# Function to prepare input data
def prepare_input_data(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, 0])  # Predicting 'Close' price
    return np.array(X), np.array(y)

# Stock details
stock_symbol = "RELIANCE.NS"
start_date = "2022-01-01"
end_date = "2025-02-05"

# Load and preprocess data
data = load_stock_data(stock_symbol, start_date, end_date)
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)

# Save scaler
with open("scaler.pkl", "wb") as f:
    pickle.dump(scaler, f)

# Prepare data
time_steps = 60
X, y = prepare_input_data(data_scaled, time_steps)

# Split data
train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Optimized LSTM model
lstm_model = Sequential([
    LSTM(128, return_sequences=True, input_shape=(time_steps, 4)),
    Dropout(0.2),
    LSTM(64, return_sequences=False),
    Dropout(0.2),
    Dense(32),
    Dense(1)
])
lstm_model.compile(optimizer="adam", loss="mean_squared_error")
lstm_model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=1)

# Save LSTM model
lstm_model.save("lstm_model.h5")

# Optimized GRU model
gru_model = Sequential([
    GRU(128, return_sequences=True, input_shape=(time_steps, 4)),
    Dropout(0.2),
    GRU(64, return_sequences=False),
    Dropout(0.2),
    Dense(32),
    Dense(1)
])
gru_model.compile(optimizer="adam", loss="mean_squared_error")
gru_model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=1)

# Save GRU model
gru_model.save("gru_model.h5")

# Make predictions
y_pred_lstm = lstm_model.predict(X_test)
```

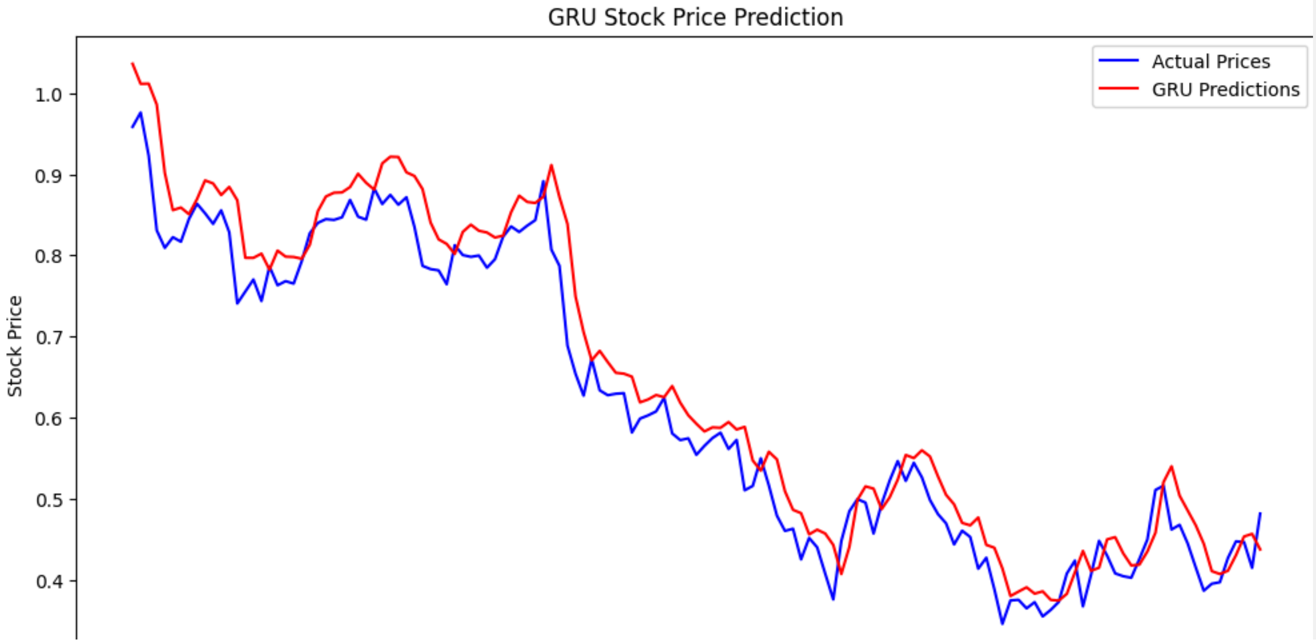
```
y_pred_lstm = lstm_model.predict(X_test)
y_pred_gru = gru_model.predict(X_test)

# Stacking predictions for ensemble learning
stacked_predictions = np.hstack([y_pred_lstm, y_pred_gru])

# Train meta-model
meta_model = LinearRegression()
meta_model.fit(stacked_predictions, y_test)

# Make final predictions using the ensemble model
ensemble_predictions = meta_model.predict(stacked_predictions)

# Evaluate models
mse_lstm = mean_squared_error(y_test, y_pred_lstm)
mse_gru = mean_squared_error(y_test, y_pred_gru)
mse_ensemble = mean_squared_error(y_test, ensemble_predictions)

r2_lstm = r2_score(y_test, y_pred_lstm) * 100
r2_gru = r2_score(y_test, y_pred_gru) * 100
r2_ensemble = r2_score(y_test, ensemble_predictions) * 100

print(f"LSTM Model Accuracy: {r2_lstm:.2f}% (MSE: {mse_lstm:.4f})")
print(f"GRU Model Accuracy: {r2_gru:.2f}% (MSE: {mse_gru:.4f})")
print(f"Ensemble Model Accuracy: {r2_ensemble:.2f}% (MSE: {mse_ensemble:.4f})")

# Function to plot predictions
def plot_predictions(y_test, y_pred, model_name):
    plt.figure(figsize=(12, 6))
    plt.plot(y_test, label="Actual Prices", color="blue")
    plt.plot(y_pred, label=f"{model_name} Predictions", color="red")
    plt.title(f"{model_name} Stock Price Prediction")
    plt.xlabel("Time")
    plt.ylabel("Stock Price")
    plt.legend()
    plt.show()

# Plot predictions
plot_predictions(y_test, y_pred_lstm, "LSTM")
plot_predictions(y_test, y_pred_gru, "GRU")
plot_predictions(y_test, ensemble_predictions, "Ensemble Model")

# ** Predict Tomorrow's Stock Price **
def predict_tomorrow():
    last_60_days = data_scaled[-time_steps:]  # Get last 60 days data
    last_60_days = np.array([last_60_days])  # Reshape for model input

    # Get predictions
    pred_lstm = lstm_model.predict(last_60_days)[0][0]
    pred_gru = gru_model.predict(last_60_days)[0][0]
    ensemble_input = np.array([[pred_lstm, pred_gru]])
    pred_ensemble = meta_model.predict(ensemble_input)[0]

    # Convert predictions back to actual price scale
    pred_lstm_actual = scaler.inverse_transform([[pred_lstm, 0, 0, 0]])[0][0]
    pred_gru_actual = scaler.inverse_transform([[pred_gru, 0, 0, 0]])[0][0]
    pred_ensemble_actual = scaler.inverse_transform([[pred_ensemble, 0, 0, 0]])[0][0]

    print("\nTomorrow's Predicted Closing Prices:")
    print(f"LSTM Model Prediction: {pred_lstm_actual:.2f} INR")
    print(f"GRU Model Prediction: {pred_gru_actual:.2f} INR")
    print(f"Ensemble Model Prediction: {pred_ensemble_actual:.2f} INR")

predict_tomorrow()
```

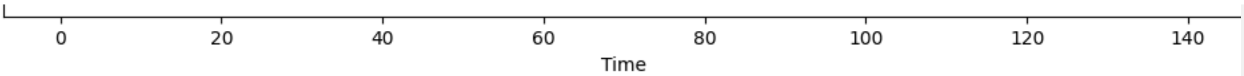
```
[*********************100%***********************]  1 of 1 completed
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` ar
  super().__init__(**kwargs)
Epoch 1/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 8s 95ms/step - loss: 0.0524
Epoch 2/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 94ms/step - loss: 0.0069
Epoch 3/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 94ms/step - loss: 0.0046
Epoch 4/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 100ms/step - loss: 0.0046
Epoch 5/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 164ms/step - loss: 0.0051
Epoch 6/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 93ms/step - loss: 0.0040
Epoch 7/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 96ms/step - loss: 0.0043
Epoch 8/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 95ms/step - loss: 0.0036
Epoch 9/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 138ms/step - loss: 0.0039
Epoch 10/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 149ms/step - loss: 0.0032
Epoch 11/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 94ms/step - loss: 0.0037
Epoch 12/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 94ms/step - loss: 0.0033
Epoch 13/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 93ms/step - loss: 0.0041
Epoch 14/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 95ms/step - loss: 0.0031
Epoch 15/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 171ms/step - loss: 0.0034
Epoch 16/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 95ms/step - loss: 0.0033
Epoch 17/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 94ms/step - loss: 0.0032
Epoch 18/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 94ms/step - loss: 0.0032
Epoch 19/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 105ms/step - loss: 0.0027
Epoch 20/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 177ms/step - loss: 0.0030
Epoch 21/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 94ms/step - loss: 0.0028
Epoch 22/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 95ms/step - loss: 0.0028
Epoch 23/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 95ms/step - loss: 0.0028
Epoch 24/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 146ms/step - loss: 0.0031
Epoch 25/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 95ms/step - loss: 0.0027
Epoch 26/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 94ms/step - loss: 0.0032
Epoch 27/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 160ms/step - loss: 0.0030
Epoch 28/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 161ms/step - loss: 0.0022
Epoch 29/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 129ms/step - loss: 0.0022
Epoch 30/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - loss: 0.0026
Epoch 31/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 196ms/step - loss: 0.0027
Epoch 32/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 94ms/step - loss: 0.0027
Epoch 33/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 146ms/step - loss: 0.0023
Epoch 34/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 206ms/step - loss: 0.0021
Epoch 35/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 171ms/step - loss: 0.0023
Epoch 36/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 102ms/step - loss: 0.0022
Epoch 37/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 132ms/step - loss: 0.0024
Epoch 38/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 154ms/step - loss: 0.0021
Epoch 39/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 94ms/step - loss: 0.0024
Epoch 40/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 96ms/step - loss: 0.0024
Epoch 41/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 96ms/step - loss: 0.0022
Epoch 42/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - loss: 0.0022
Epoch 43/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 164ms/step - loss: 0.0020
Epoch 44/50
```

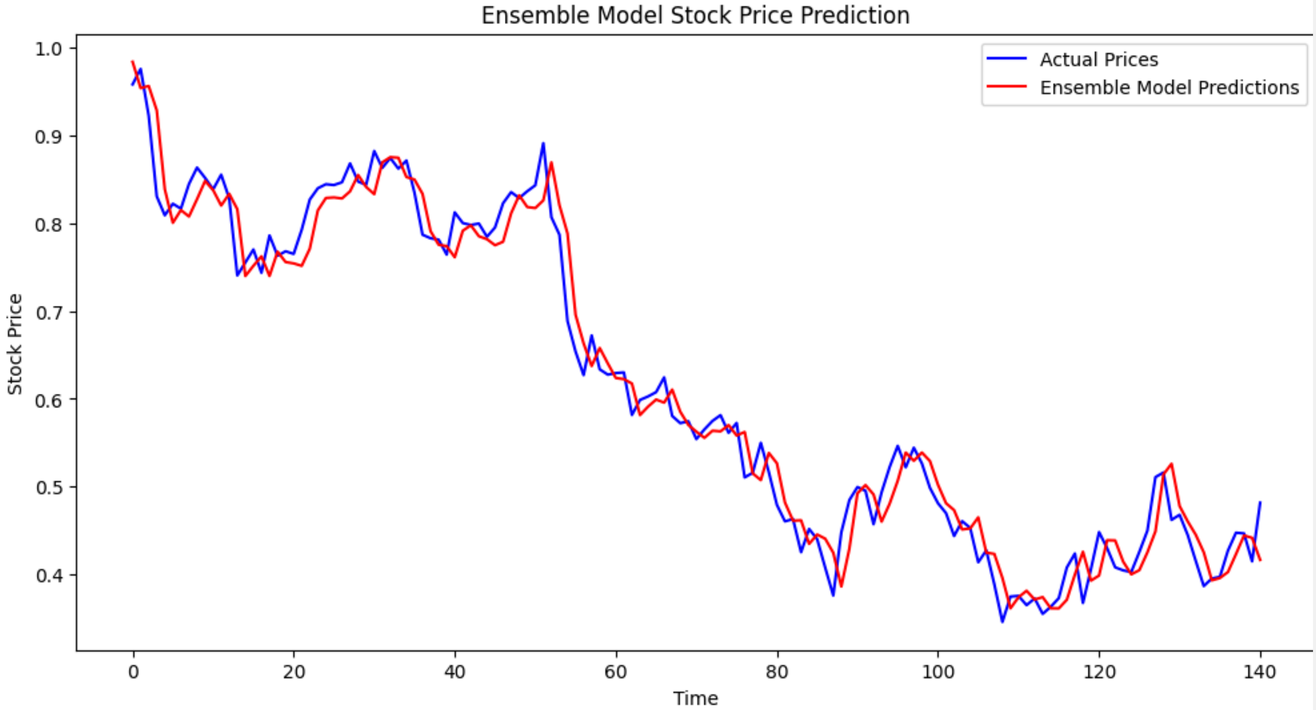
```
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 124ms/step - loss: 0.0021
Epoch 45/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - loss: 0.0024
Epoch 46/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 95ms/step - loss: 0.0020
Epoch 47/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 97ms/step - loss: 0.0019
Epoch 48/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - loss: 0.0023
Epoch 49/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 127ms/step - loss: 0.0019
Epoch 50/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 156ms/step - loss: 0.0018
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format i
Epoch 1/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 7s 94ms/step - loss: 0.0693
Epoch 2/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 163ms/step - loss: 0.0061
Epoch 3/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 93ms/step - loss: 0.0040
Epoch 4/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 94ms/step - loss: 0.0038
Epoch 5/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - loss: 0.0038
Epoch 6/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 93ms/step - loss: 0.0037
Epoch 7/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 149ms/step - loss: 0.0041
Epoch 8/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 95ms/step - loss: 0.0036
Epoch 9/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 108ms/step - loss: 0.0028
Epoch 10/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 96ms/step - loss: 0.0031
Epoch 11/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 147ms/step - loss: 0.0034
Epoch 12/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 117ms/step - loss: 0.0029
Epoch 13/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 144ms/step - loss: 0.0028
Epoch 14/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 95ms/step - loss: 0.0030
Epoch 15/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 164ms/step - loss: 0.0026
Epoch 16/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 94ms/step - loss: 0.0023
Epoch 17/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 95ms/step - loss: 0.0026
Epoch 18/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 95ms/step - loss: 0.0025
Epoch 19/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 110ms/step - loss: 0.0023
Epoch 20/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 163ms/step - loss: 0.0023
Epoch 21/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 94ms/step - loss: 0.0029
Epoch 22/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 94ms/step - loss: 0.0022
Epoch 23/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 94ms/step - loss: 0.0023
Epoch 24/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - loss: 0.0022
Epoch 25/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 146ms/step - loss: 0.0025
Epoch 26/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 94ms/step - loss: 0.0024
Epoch 27/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 94ms/step - loss: 0.0018
Epoch 28/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 94ms/step - loss: 0.0020
Epoch 29/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 106ms/step - loss: 0.0018
Epoch 30/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 150ms/step - loss: 0.0019
Epoch 31/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 93ms/step - loss: 0.0020
Epoch 32/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 97ms/step - loss: 0.0018
Epoch 33/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 97ms/step - loss: 0.0019
Epoch 34/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 94ms/step - loss: 0.0019
Epoch 35/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 125ms/step - loss: 0.0018
Epoch 36/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 140ms/step - loss: 0.0016
Epoch 37/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - loss: 0.0021
Epoch 38/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - loss: 0.0024
Epoch 39/50
```

```
Epoch 39/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 96ms/step - loss: 0.0023
Epoch 40/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 96ms/step - loss: 0.0017
Epoch 41/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 132ms/step - loss: 0.0016
Epoch 42/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 139ms/step - loss: 0.0016
Epoch 43/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 96ms/step - loss: 0.0018
Epoch 44/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 95ms/step - loss: 0.0019
Epoch 45/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - loss: 0.0016
Epoch 46/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 159ms/step - loss: 0.0019
Epoch 47/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 96ms/step - loss: 0.0014
Epoch 48/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - loss: 0.0016
Epoch 49/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - loss: 0.0015
Epoch 50/50
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 95ms/step - loss: 0.0017
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format i
WARNING:tensorflow:5 out of the last 11 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
4/5 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/stepWARNING:tensorflow:5 out of the last 11 calls to <function TensorFlowTrainer.make_predict_fu
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 181ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 368ms/step
LSTM Model Accuracy: 96.05% (MSE: 0.0013)
GRU Model Accuracy: 93.50% (MSE: 0.0021)
Ensemble Model Accuracy: 97.03% (MSE: 0.0010)
```










```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 187ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step

Tomorrow's Predicted Closing Prices:
LSTM Model Prediction: 1266.48 INR
GRU Model Prediction: 1281.81 INR
Ensemble Model Prediction: 1272.93 INR
```

```
import numpy as np
import yfinance as yf
import pickle
from tensorflow.keras.models import load_model
from sklearn.preprocessing import MinMaxScaler

# Function to load stock data
def load_stock_data(stock_symbol, start_date, end_date):
    data = yf.download(stock_symbol, start=start_date, end=end_date)[['Close', 'High', 'Low', 'Volume']]
    return data

# Function to prepare input data
def prepare_input_data(data, time_steps=60):
    return np.array([data[-time_steps:]])  # Take last 60 days and reshape

# Function to predict tomorrow's stock price
def predict_tomorrow(stock_symbol):
    # Load models and scaler
    lstm_model = load_model("lstm_model.h5")
    gru_model = load_model("gru_model.h5")
    with open("scaler.pkl", "rb") as f:
        scaler = pickle.load(f)

    # Load stock data for last 60 days
    end_date = "2025-02-05"
    start_date = "2024-12-01"  # Adjust for 60-day window
    data = load_stock_data(stock_symbol, start_date, end_date)

    if data.empty:
        print(f"No data available for {stock_symbol}")
        return

    # Scale the data
    data_scaled = scaler.transform(data)

    # Prepare last 60 days data
    last_60_days = prepare_input_data(data_scaled)

    # Get predictions
    pred_lstm = lstm_model.predict(last_60_days)[0][0]
    pred_gru = gru_model.predict(last_60_days)[0][0]
```