1. Source code

```
# stock_price_prediction/app.py

import gradio as gr
import pandas as pd
import numpy as np
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import yfinance as yf
import os

# Load or train model
MODEL_PATH = "stock_model.h5"
scaler = MinMaxScaler()

def get_stock_data(ticker="AAPL"):
    df = yf.download(ticker, period="2y")
```

```
df = df[['Close']].dropna() return
df

  def create_dataset(data, time_step=60):
      X, y = [], []
      for i in range(time_step, len(data)):
          X.append(data[i-time_step:i, 0])
          y.append(data[i, 0])
      return np.array(X), np.array(y)

  def train_model(df):
      data = scaler.fit_transform(df.values)
      X, y = create_dataset(data)
      X = X.reshape(X.shape[0], X.shape[1], 1)

      model = Sequential()
      model.add(LSTM(units=50, return_sequences=True, input_shape=(X.shape[1],
  1)))
      model.add(LSTM(units=50))
      model.add(Dense(1))
      model.compile(optimizer='adam', loss='mean_squared_error')

      model.fit(X, y, epochs=5, batch_size=32, verbose=1)
      model.save(MODEL_PATH)
      return model

  def load_or_train_model(df):
      if os.path.exists(MODEL_PATH):
          return load_model(MODEL_PATH)
      else:
          return train_model(df)

  def prepare_input_sequence(df, window=60):
      data = scaler.transform(df.values)
      return np.reshape(data[-window:], (1, window, 1))

  def predict_prices(days, ticker):
      df = get_stock_data(ticker)
      model = load_or_train_model(df)
      input_seq = prepare_input_sequence(df)

      predictions = []
      for _ in range(days):
          pred = model.predict(input_seq, verbose=0)[0, 0]
```

```
        predictions.append(pred)
        input_seq = np.append(input_seq[0][1:],
        [[pred]], axis=0) input_seq =
        input_seq.reshape(1, input_seq.shape[0], 1)

    return scaler.inverse_transform(np.array(predictions).reshape(-1,

1)).flatten() def predict_and_plot(days, ticker):
    preds = predict_prices(days, ticker)
    days_range = list(range(1, days + 1))

    plt.figure(figsize=(8, 4))
    plt.plot(days_range, preds, marker='o', linestyle='-',
    color='blue') plt.title(f"Predicted Stock Prices for
    {ticker.upper()}") plt.xlabel("Days Ahead")
    plt.ylabel("P
    rice (USD)")
    plt.grid(True
    )

    return plt.gcf(), {f"Day {i+1}": f"${p:.2f}" for i, p in

enumerate(preds)} # Gradio Interface
demo = gr.Interface(
    fn=predict
    _and_plot,
    inputs=[
        gr.Slider(1, 10, step=1, label="Days Ahead"),
        gr.Textbox(value="AAPL", label="Stock
        Ticker")
    ],
    outputs=["plot",
    "label"], title="Stock
    Price Prediction App",
    description="Predict the next few days of stock closing prices using
an LSTM model trained on historical data."
)

demo.launch()
```