TRXvsTHETER

August 13, 2025

TRON_Forecast: Advanced Cryptocurrency Price Prediction

[22]:
```
# REQUIREMENTS
pip install requests pandas numpy scikit-learn pmdarima tensorflow
```

1 Get current date and time

[12]:
```
from datetime import datetime
now = datetime.now()

print("Today is:", now.strftime("%A, %d %B %Y"))
print("Current time:", now.strftime("%H:%M:%S"))
```

```
Today is: Wednesday, 13 August 2025
Current time: 09:33:40
```

1.1 Today is: Wednesday, 13 August 2025

[11]:
```
import ccxt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller, acf, pacf
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense
import time
import warnings
warnings.filterwarnings('ignore')
```

```
# Exchange configuration with fallbacks for TRX/USDT
EXCHANGES = [
    {'name': 'binance', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'kucoin', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'bybit', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'huobi', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'okx', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/USDT'},
]

# 1. Robust Data Acquisition for TRX/USDT
def fetch_historical_data(timeframe='1d', limit=1000):
    """Fetch TRX/USDT data with exchange fallback"""
    for exchange_config in EXCHANGES:
        try:
            exchange_class = getattr(ccxt, exchange_config['name'])
            exchange = exchange_class(exchange_config['params'])
            time.sleep(exchange.rateLimit / 1000)

            symbol = exchange_config['symbol']
            ohlcv = exchange.fetch_ohlcv(symbol, timeframe, limit=limit)

            df = pd.DataFrame(ohlcv, columns=['timestamp', 'open', 'high',␣
 ↪'low', 'close', 'volume'])
            df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
            df.set_index('timestamp', inplace=True)
            print(f"Successfully fetched TRX/USDT data from␣
 ↪{exchange_config['name']}")
            return df

        except (ccxt.NetworkError, ccxt.ExchangeError) as e:
            print(f"Error from {exchange_config['name']}: {str(e)[:100]}...␣
 ↪trying next exchange")
            continue
        except Exception as e:
            print(f"Unexpected error with {exchange_config['name']}: {str(e)[:
 ↪100]}... trying next exchange")
            continue

    raise RuntimeError("All exchanges failed. Please check network connection␣
 ↪or API availability")

# 2. ARIMA Forecasting
```

```
def arima_forecast(data, forecast_steps=30):
    if len(data) < 30:
        return np.array([data.iloc[-1]] * forecast_steps)

    d = 0
    if len(data) > 50:
        d = find_optimal_d(data)

    if d > 0:
        diff_data = data.diff(d).dropna()
    else:
        diff_data = data

    p, q = (1, 1)
    if len(diff_data) > 50:
        try:
            p, q = find_optimal_pq(diff_data)
        except:
            pass

    try:
        model = ARIMA(data, order=(p, d, q))
        model_fit = model.fit()
        return model_fit.forecast(steps=forecast_steps).values
    except:
        return np.array([data.rolling(window=5).mean().iloc[-1]] *␣
 ↪forecast_steps)

def find_optimal_d(data, max_d=2):
    d = 0
    for i in range(max_d + 1):
        try:
            result = adfuller(data if i == 0 else data.diff().dropna())
            if result[1] <= 0.05:
                return d
            d = i
        except:
            break
    return min(d, max_d)

def find_optimal_pq(data, max_p=5, max_q=5):
    try:
        acf_vals = acf(data, nlags=max_p + max_q)
        pacf_vals = pacf(data, nlags=max_p + max_q)

        p = next((i for i in range(1, len(pacf_vals)) if abs(pacf_vals[i]) > 1.
 ↪96/np.sqrt(len(data))), 1)
```

```
        q = next((i for i in range(1, len(acf_vals)) if abs(acf_vals[i]) > 1.96/
 ↪np.sqrt(len(data))), 1)

        return min(p, max_p), min(q, max_q)
    except:
        return (1, 1)

# 3. LSTM Model
def create_dataset(data, window_size=8, target_size=1):
    X, y = [], []
    n = len(data)
    if n < window_size + target_size:
        return np.array([]), np.array([])

    for i in range(n - window_size - target_size):
        X.append(data[i:(i+window_size)])
        y.append(data[(i+window_size):(i+window_size+target_size)])
    return np.array(X), np.array(y)

def lstm_forecast(data, window_size=8, forecast_steps=30):
    if len(data) < window_size + 10:
        return np.array([data.iloc[-1]] * forecast_steps)

    scaler = MinMaxScaler(feature_range=(0,1))
    scaled_data = scaler.fit_transform(data.values.reshape(-1,1))

    X, y = create_dataset(scaled_data, window_size, forecast_steps)
    if len(X) == 0:
        return np.array([data.iloc[-1]] * forecast_steps)

    X = np.reshape(X, (X.shape[0], X.shape[1], 1))

    model = Sequential([
        LSTM(32, input_shape=(window_size, 1)),
        Dense(forecast_steps)
    ])
    model.compile(optimizer='adam', loss='mse')
    model.fit(X, y, epochs=20, batch_size=16, verbose=0)

    last_window = scaled_data[-window_size:].reshape(1, window_size, 1)
    forecast = model.predict(last_window)
    return scaler.inverse_transform(forecast)[0]

# 4. Visualization Functions for TRX
def plot_forecast_comparison(history, arima_forecast, lstm_forecast, title,␣
 ↪forecast_days=30):
    plt.figure(figsize=(14, 8))
```

```
    plt.plot(history.index, history.values, 'b-', label='Historical Prices',␣
↪linewidth=2)

    last_date = history.index[-1]
    forecast_dates = pd.date_range(
        start=last_date + pd.Timedelta(days=1),
        periods=forecast_days,
        freq='D'
    )

    plt.plot(forecast_dates, arima_forecast, 'r--', label='ARIMA Forecast',␣
↪linewidth=2)
    plt.plot(forecast_dates, lstm_forecast, 'g-.', label='LSTM Forecast',␣
↪linewidth=2)

    plt.title(f'TRX/USDT Price Forecast: {title}', fontsize=16)
    plt.xlabel('Date', fontsize=14)
    plt.ylabel('Price (USDT)', fontsize=14)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend(fontsize=12)
    plt.xticks(rotation=45)
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator())
    plt.tight_layout()
    plt.savefig(f'TRX_forecast_comparison_{title.replace(" ", "_")}.png',␣
↪dpi=300)
    plt.show()

def plot_window_comparison(results, interval):
    if interval not in results:
        return

    window_data = results[interval]
    window_names = list(window_data.keys())

    arima_1d = [window_data[w]['ARIMA_1D'] for w in window_names]
    lstm_1d = [window_data[w]['LSTM_1D'] for w in window_names]
    arima_1m = [window_data[w]['ARIMA_1M'] for w in window_names]
    lstm_1m = [window_data[w]['LSTM_1M'] for w in window_names]

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 12))

    # 1-Day Forecast Comparison
    x = np.arange(len(window_names))
    width = 0.35
    ax1.bar(x - width/2, arima_1d, width, label='ARIMA', color='skyblue')
    ax1.bar(x + width/2, lstm_1d, width, label='LSTM', color='salmon')
```

```
    ax1.set_title(f'TRX 1-Day Forecast Comparison ({interval} Interval)',␣
↪fontsize=14)
    ax1.set_ylabel('Price (USDT)', fontsize=12)
    ax1.set_xticks(x)
    ax1.set_xticklabels(window_names)
    ax1.legend()
    ax1.grid(axis='y', linestyle='--', alpha=0.7)

    # 1-Month Forecast Comparison
    ax2.bar(x - width/2, arima_1m, width, label='ARIMA', color='skyblue')
    ax2.bar(x + width/2, lstm_1m, width, label='LSTM', color='salmon')
    ax2.set_title(f'TRX 1-Month Forecast Comparison ({interval} Interval)',␣
↪fontsize=14)
    ax2.set_ylabel('Price (USDT)', fontsize=12)
    ax2.set_xticks(x)
    ax2.set_xticklabels(window_names)
    ax2.legend()
    ax2.grid(axis='y', linestyle='--', alpha=0.7)

    plt.tight_layout()
    plt.savefig(f'TRX_window_comparison_{interval}.png', dpi=300)
    plt.show()

def plot_model_performance(history, arima_fit, lstm_fit, title):
    plt.figure(figsize=(14, 8))
    plt.plot(history.index, history.values, 'b-', label='Historical Prices',␣
↪linewidth=2)
    plt.plot(history.index[-len(arima_fit):], arima_fit, 'r--', label='ARIMA␣
↪Fit', linewidth=1.5)
    plt.plot(history.index[-len(lstm_fit):], lstm_fit, 'g-.', label='LSTM Fit',␣
↪linewidth=1.5)

    plt.title(f'TRX Model Performance: {title}', fontsize=16)
    plt.xlabel('Date', fontsize=14)
    plt.ylabel('Price (USDT)', fontsize=14)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend(fontsize=12)
    plt.xticks(rotation=45)
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator())
    plt.tight_layout()
    plt.savefig(f'TRX_model_performance_{title.replace(" ", "_")}.png', dpi=300)
    plt.show()

# 5. Configuration for TRX analysis
time_intervals = {
    '1D': '1d',
```

```
    '1W': '1w',
    '1M': '1M'
}

window_sizes = {
    '1h': 3600,
    '3h': 10800,
    '6h': 21600,
    '12h': 43200,
    '1d': 86400,
    '3d': 259200,
}

# 6. Main Processing for TRX
def process_and_visualize_trx():
    results = {}
    raw_data = fetch_historical_data(timeframe='1d', limit=500)

    for interval_name, interval_code in time_intervals.items():
        try:
            print(f"\nProcessing {interval_name} interval for TRX...")
            resampled = raw_data.resample(interval_code).agg({
                'open': 'first',
                'high': 'max',
                'low': 'min',
                'close': 'last',
                'volume': 'sum'
            }).ffill().dropna()

            close_series = resampled['close']

            if len(close_series) < 50:
                print(f"  Insufficient data ({len(close_series)} points). Skipping.")
                continue

            interval_results = {}
            best_arima_1m = None
            best_lstm_1m = None

            for window_name, window_seconds in window_sizes.items():
                try:
                    print(f"  Calculating {window_name} windows...")
                    interval_seconds = pd.Timedelta(interval_code).total_seconds()
                    window_bars = max(8, int(window_seconds / interval_seconds))
```

```
                    if len(close_series) < window_bars + 30:
                        print(f"    Not enough data for window. Have␣
↪{len(close_series)}, need {window_bars+30}")
                        continue

                    # Generate forecasts
                    arima_1m = arima_forecast(close_series, forecast_steps=30)
                    lstm_1m = lstm_forecast(close_series,␣
↪window_size=window_bars, forecast_steps=30)

                    # Store results
                    interval_results[window_name] = {
                        'ARIMA_1D': arima_1m[0],
                        'ARIMA_1M': arima_1m[-1],
                        'LSTM_1D': lstm_1m[0],
                        'LSTM_1M': lstm_1m[-1],
                        'ARIMA_Full': arima_1m,
                        'LSTM_Full': lstm_1m
                    }

                    # Track best models
                    last_price = close_series.iloc[-1]
                    if best_arima_1m is None or abs(arima_1m[-1] - last_price)␣
↪< abs(best_arima_1m[-1] - last_price):
                        best_arima_1m = arima_1m
                    if best_lstm_1m is None or abs(lstm_1m[-1] - last_price) <␣
↪abs(best_lstm_1m[-1] - last_price):
                        best_lstm_1m = lstm_1m

                except Exception as e:
                    print(f"    Error processing window {window_name}: {str(e)[:
↪100]}")
                    continue

            results[interval_name] = interval_results

            # Generate visualizations
            if best_arima_1m is not None and best_lstm_1m is not None:
                plot_forecast_comparison(
                    history=close_series[-100:],
                    arima_forecast=best_arima_1m,
                    lstm_forecast=best_lstm_1m,
                    title=f"{interval_name} Interval",
                    forecast_days=30
                )

                plot_window_comparison(results, interval_name)
```

```
                if interval_results:
                    best_window = next(iter(interval_results))
                    # Generate model fit data
                    arima_fit = arima_forecast(close_series[-100:-30],␣
 ↪forecast_steps=70)
                    lstm_fit = lstm_forecast(close_series[-100:-30],␣
 ↪window_size=window_bars, forecast_steps=70)

                    plot_model_performance(
                        history=close_series[-100:],
                        arima_fit=arima_fit,
                        lstm_fit=lstm_fit,
                        title=f"{interval_name} Interval ({best_window} window)"
                    )

            except Exception as e:
                print(f"Failed processing interval {interval_name}: {str(e)[:100]}")
                continue

    return results

# 7. Execute TRX Analysis
if __name__ == "__main__":
    print("Starting TRX/USDT forecasting and visualization...")
    results = process_and_visualize_trx()

    # Display results
    print("\n\n=== TRX/USDT Forecast Results ===")
    for interval, window_data in results.items():
        print(f"\n[{interval} Interval]")
        for window, values in window_data.items():
            print(f"  {window} window:")
            print(f"    ARIMA 1D: {values['ARIMA_1D']:.6f}")
            print(f"    LSTM 1D:  {values['LSTM_1D']:.6f}")
            print(f"    ARIMA 1M: {values['ARIMA_1M']:.6f}")
            print(f"    LSTM 1M:  {values['LSTM_1M']:.6f}")
    print("\nTRX forecasting complete! Visualizations saved as PNG files.")
```

```
Starting TRX/USDT forecasting and visualization…
Error from binance: binance GET https://api.binance.com/api/v3/exchangeInfo 451
{
  "code": 0,
  "msg": "Service unavai… trying next exchange
Successfully fetched TRX/USDT data from kucoin

Processing 1D interval for TRX…
```

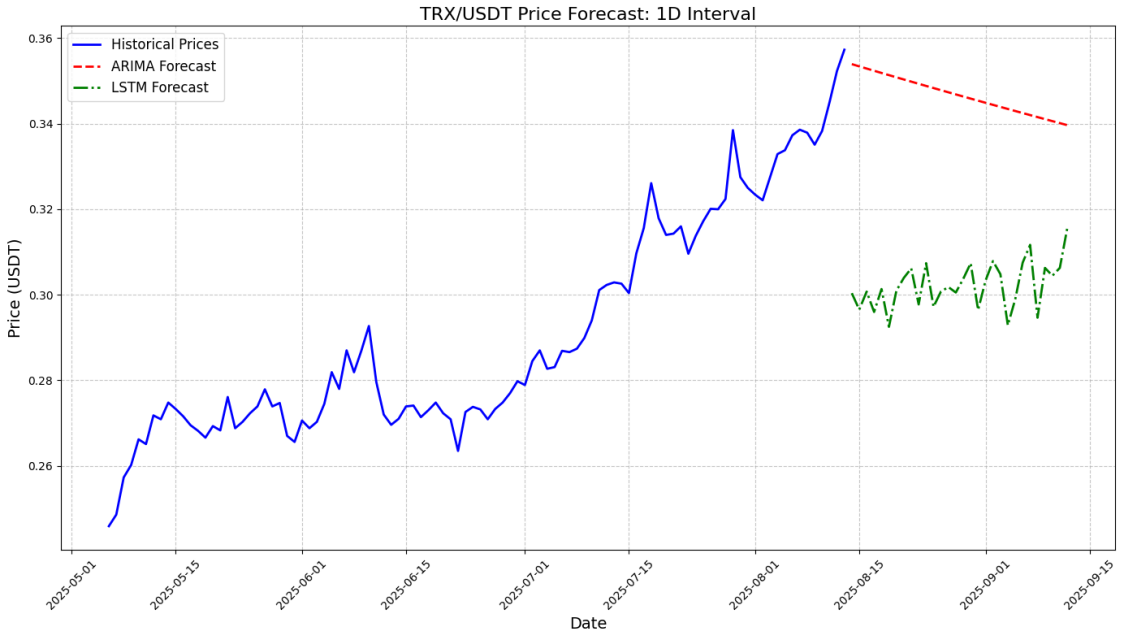
```
  Calculating 1h windows…
1/1             0s 177ms/step
  Calculating 3h windows…
1/1             0s 172ms/step
  Calculating 6h windows…
1/1             0s 174ms/step
  Calculating 12h windows…
1/1             0s 291ms/step
  Calculating 1d windows…
```

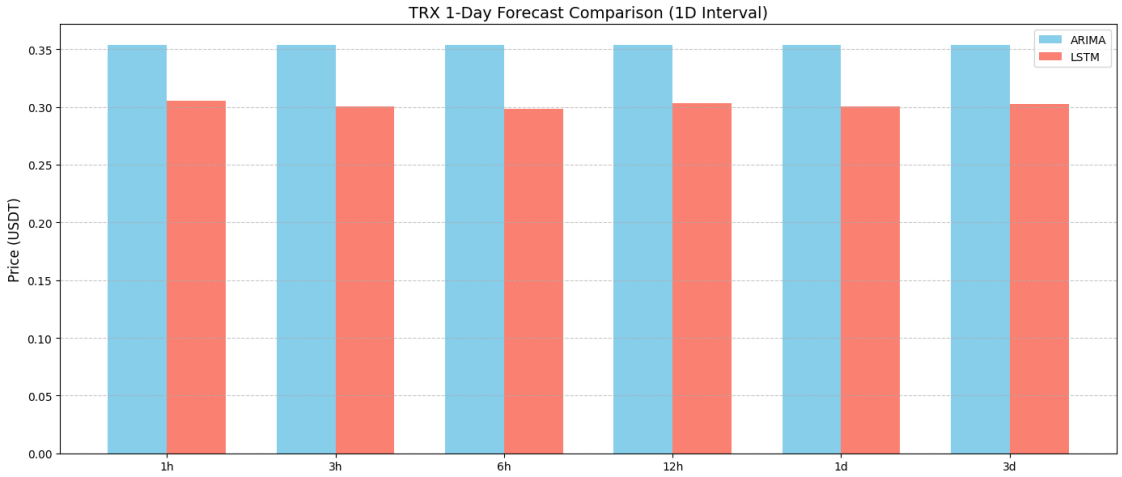
```
WARNING:tensorflow:5 out of the last 5 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7b381821d120> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

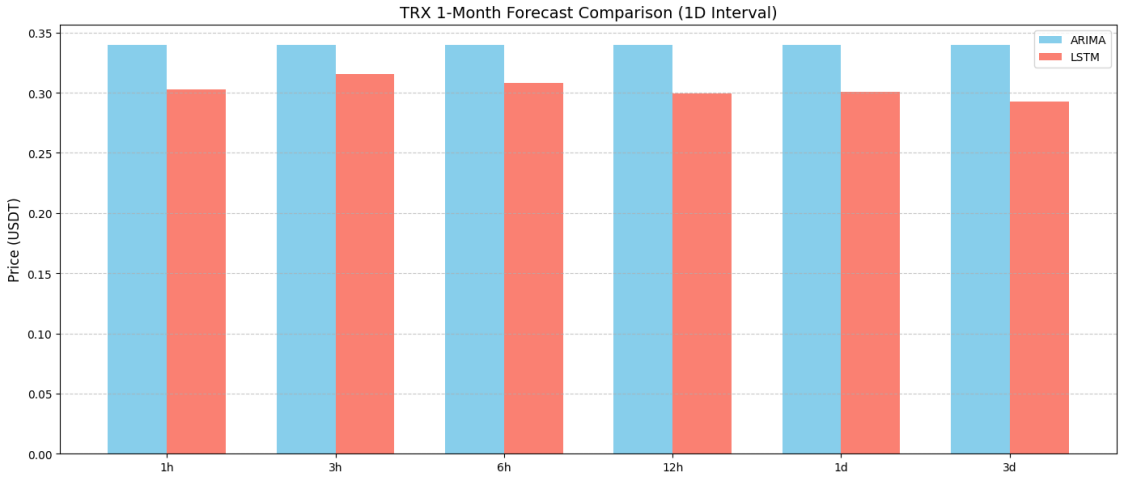
```
1/1             0s 193ms/step
  Calculating 3d windows…
```

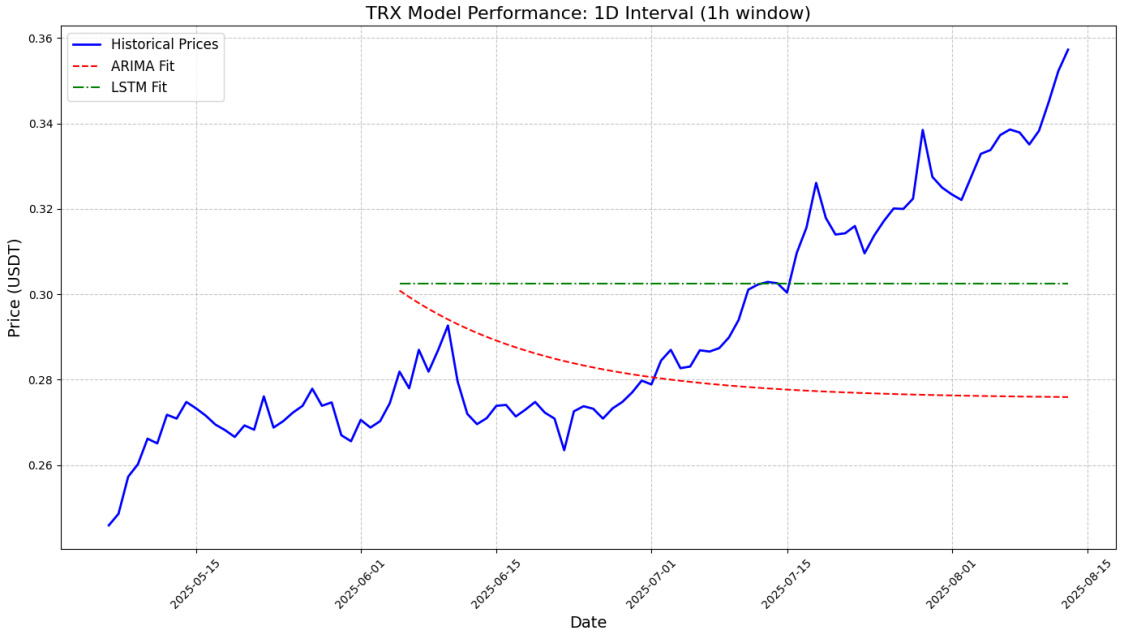
```
WARNING:tensorflow:6 out of the last 6 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7b3820058360> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
1/1             0s 181ms/step
```

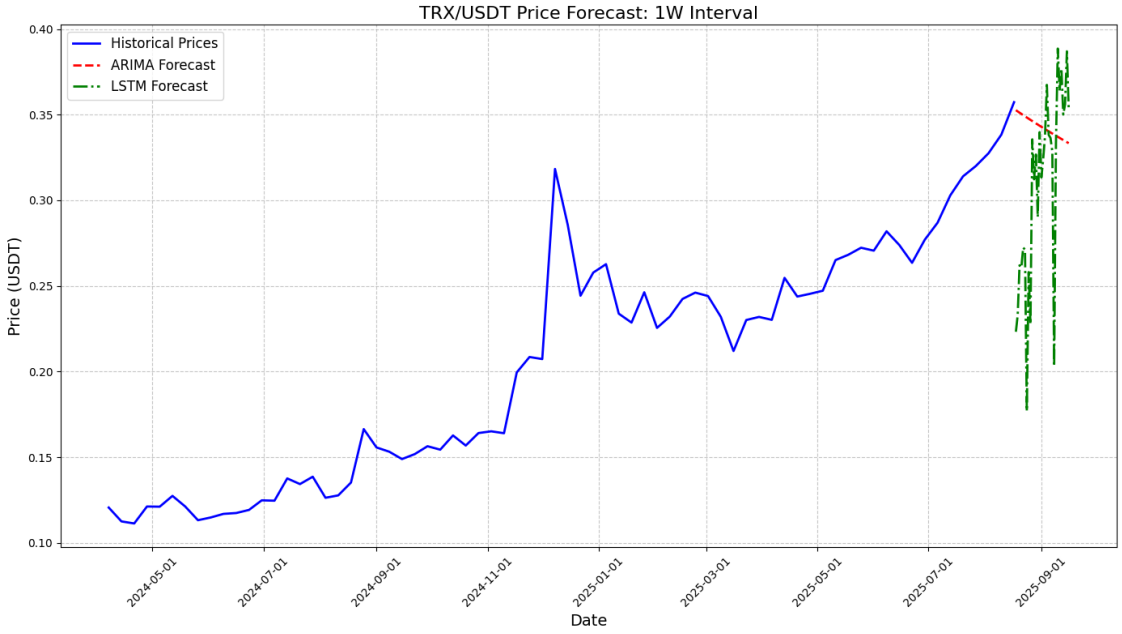

TRX/USDT Price Forecast: 1D Interval
Historical Prices
ARIMA Forecast
LSTM Forecast
Price (USDT)
Date
0.36
0.34
0.32
0.30
0.28
0.26
2025-05-01
2025-05-15
2025-06-01
2025-06-15
2025-07-01
2025-07-15
2025-08-01
2025-08-15
2025-09-01
2025-09-15

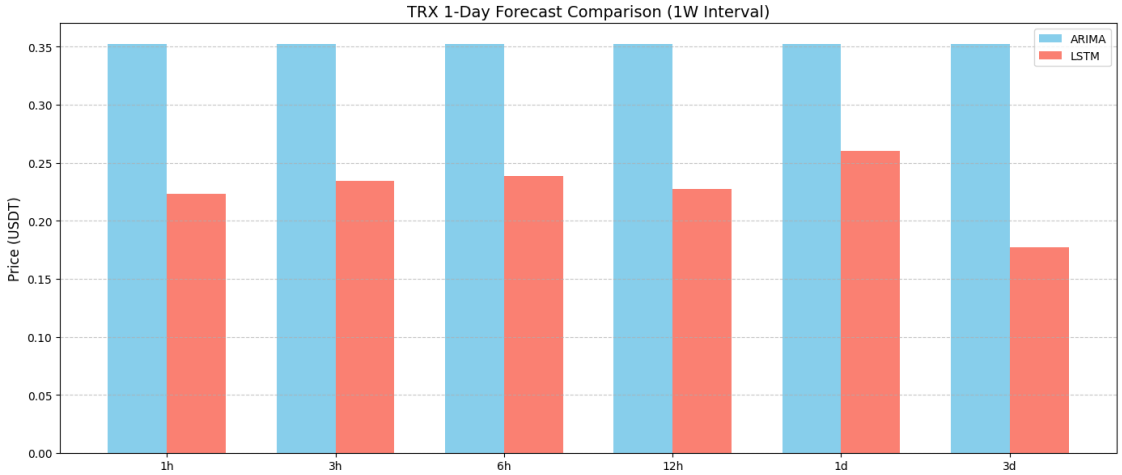

TRX 1-Day Forecast Comparison (1D Interval)
ARIMA
LSTM
Price (USDT)
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1h
3h
6h
12h
1d
3d

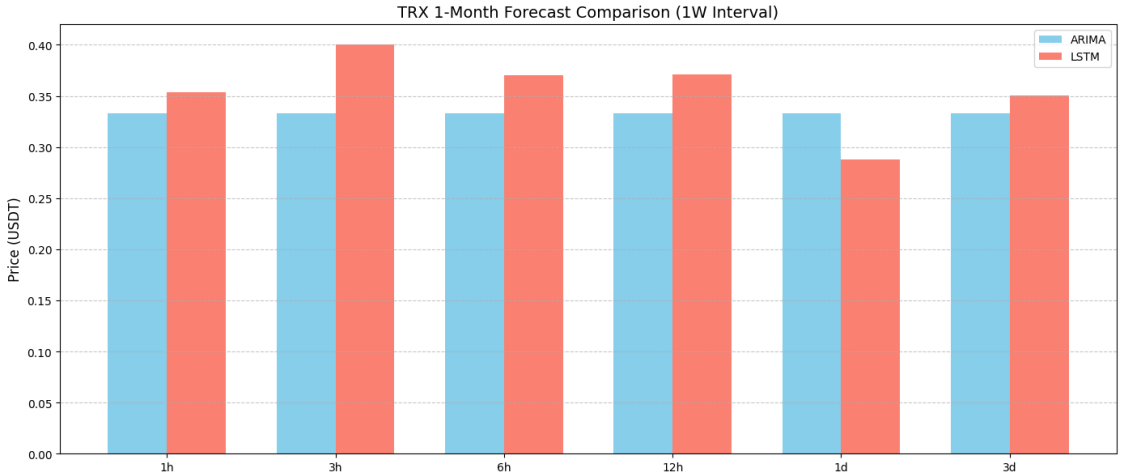

TRX 1-Month Forecast Comparison (1D Interval)
ARIMA
LSTM
Price (USDT)
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1h
3h
6h
12h
1d
3d

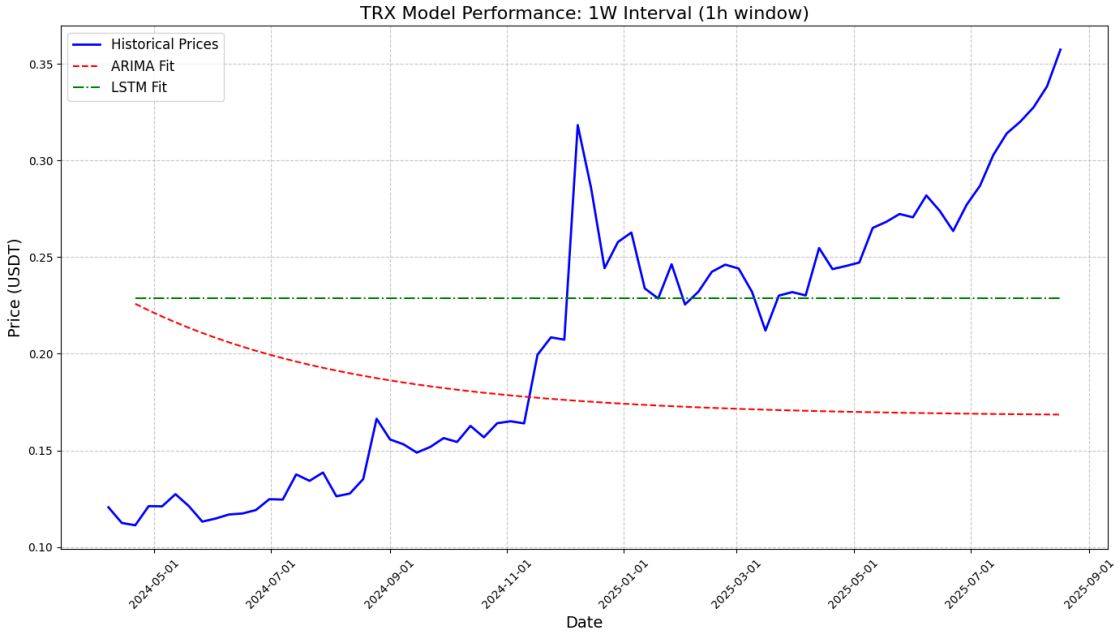



```
Processing 1W interval for TRX…
  Calculating 1h windows…
1/1             0s 195ms/step
  Calculating 3h windows…
1/1             0s 175ms/step
  Calculating 6h windows…
1/1             0s 169ms/step
  Calculating 12h windows…
1/1             0s 169ms/step
  Calculating 1d windows…
1/1             0s 168ms/step
  Calculating 3d windows…
1/1             0s 169ms/step
```


TRX/USDT Price Forecast: 1W Interval
Historical Prices
ARIMA Forecast
LSTM Forecast
Price (USDT)
Date
0.40
0.35
0.30
0.25
0.20
0.15
0.10
2024-05-01
2024-07-01
2024-09-01
2024-11-01
2025-01-01
2025-03-01
2025-05-01
2025-07-01
2025-09-01


TRX 1-Day Forecast Comparison (1W Interval)
ARIMA
LSTM
Price (USDT)
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1h
3h
6h
12h
1d
3d


TRX 1-Month Forecast Comparison (1W Interval)
ARIMA
LSTM
Price (USDT)
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1h
3h
6h
12h
1d
3d




```
Processing 1M interval for TRX…
  Insufficient data (17 points). Skipping.


=== TRX/USDT Forecast Results ===

[1D Interval]
  1h window:
    ARIMA 1D: 0.353929
    LSTM 1D:  0.304990
    ARIMA 1M: 0.339669
    LSTM 1M:  0.302929
  3h window:
    ARIMA 1D: 0.353929
    LSTM 1D:  0.300383
    ARIMA 1M: 0.339669
    LSTM 1M:  0.315582
  6h window:
    ARIMA 1D: 0.353929
    LSTM 1D:  0.298249
    ARIMA 1M: 0.339669
    LSTM 1M:  0.307861
  12h window:
```

```
    ARIMA 1D: 0.353929
    LSTM 1D:  0.302986
    ARIMA 1M: 0.339669
    LSTM 1M:  0.299722
  1d window:
    ARIMA 1D: 0.353929
    LSTM 1D:  0.300622
    ARIMA 1M: 0.339669
    LSTM 1M:  0.300693
  3d window:
    ARIMA 1D: 0.353929
    LSTM 1D:  0.302258
    ARIMA 1M: 0.339669
    LSTM 1M:  0.292675

[1W Interval]
  1h window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.223239
    ARIMA 1M: 0.333356
    LSTM 1M:  0.353374
  3h window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.234057
    ARIMA 1M: 0.333356
    LSTM 1M:  0.400258
  6h window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.238869
    ARIMA 1M: 0.333356
    LSTM 1M:  0.370169
  12h window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.227247
    ARIMA 1M: 0.333356
    LSTM 1M:  0.371152
  1d window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.260096
    ARIMA 1M: 0.333356
    LSTM 1M:  0.287577
  3d window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.176993
    ARIMA 1M: 0.333356
    LSTM 1M:  0.350722

TRX forecasting complete! Visualizations saved as PNG files.
```

[13]:
```
import ccxt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller, acf, pacf
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense
import time
import warnings
warnings.filterwarnings('ignore')

# Exchange configuration with fallbacks for TRX/USDT
EXCHANGES = [
    {'name': 'binance', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'kucoin', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'bybit', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'huobi', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'okx', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/USDT'},
]

# 1. Robust Data Acquisition for TRX/USDT
def fetch_historical_data(timeframe='1d', limit=1000):
    """Fetch TRX/USDT data with exchange fallback"""
    for exchange_config in EXCHANGES:
        try:
            exchange_class = getattr(ccxt, exchange_config['name'])
            exchange = exchange_class(exchange_config['params'])
            time.sleep(exchange.rateLimit / 1000)

            symbol = exchange_config['symbol']
            ohlcv = exchange.fetch_ohlcv(symbol, timeframe, limit=limit)

            df = pd.DataFrame(ohlcv, columns=['timestamp', 'open', 'high',␣
 ↪'low', 'close', 'volume'])
            df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
            df.set_index('timestamp', inplace=True)
            print(f"Successfully fetched TRX/USDT data from␣
 ↪{exchange_config['name']}")
            return df
```

```
        except (ccxt.NetworkError, ccxt.ExchangeError) as e:
            print(f"Error from {exchange_config['name']}: {str(e)[:100]}...␣
 ↪trying next exchange")
            continue
        except Exception as e:
            print(f"Unexpected error with {exchange_config['name']}: {str(e)[:
 ↪100]}... trying next exchange")
            continue

    raise RuntimeError("All exchanges failed. Please check network connection␣
 ↪or API availability")

# 2. ARIMA Forecasting
def arima_forecast(data, forecast_steps=30):
    if len(data) < 30:
        return np.array([data.iloc[-1]] * forecast_steps)

    d = 0
    if len(data) > 50:
        d = find_optimal_d(data)

    if d > 0:
        diff_data = data.diff(d).dropna()
    else:
        diff_data = data

    p, q = (1, 1)
    if len(diff_data) > 50:
        try:
            p, q = find_optimal_pq(diff_data)
        except:
            pass

    try:
        model = ARIMA(data, order=(p, d, q))
        model_fit = model.fit()
        return model_fit.forecast(steps=forecast_steps).values
    except:
        return np.array([data.rolling(window=5).mean().iloc[-1]] *␣
 ↪forecast_steps)

def find_optimal_d(data, max_d=2):
    d = 0
    for i in range(max_d + 1):
        try:
            result = adfuller(data if i == 0 else data.diff().dropna())
            if result[1] <= 0.05:
```

```
                return d
            d = i
        except:
            break
    return min(d, max_d)

def find_optimal_pq(data, max_p=5, max_q=5):
    try:
        acf_vals = acf(data, nlags=max_p + max_q)
        pacf_vals = pacf(data, nlags=max_p + max_q)

        p = next((i for i in range(1, len(pacf_vals)) if abs(pacf_vals[i]) > 1.
 ↪96/np.sqrt(len(data))), 1)
        q = next((i for i in range(1, len(acf_vals)) if abs(acf_vals[i]) > 1.96/
 ↪np.sqrt(len(data))), 1)

        return min(p, max_p), min(q, max_q)
    except:
        return (1, 1)

# 3. LSTM Model
def create_dataset(data, window_size=8, target_size=1):
    X, y = [], []
    n = len(data)
    if n < window_size + target_size:
        return np.array([]), np.array([])

    for i in range(n - window_size - target_size):
        X.append(data[i:(i+window_size)])
        y.append(data[(i+window_size):(i+window_size+target_size)])
    return np.array(X), np.array(y)

def lstm_forecast(data, window_size=8, forecast_steps=30):
    if len(data) < window_size + 10:
        return np.array([data.iloc[-1]] * forecast_steps)

    scaler = MinMaxScaler(feature_range=(0,1))
    scaled_data = scaler.fit_transform(data.values.reshape(-1,1))

    X, y = create_dataset(scaled_data, window_size, forecast_steps)
    if len(X) == 0:
        return np.array([data.iloc[-1]] * forecast_steps)

    X = np.reshape(X, (X.shape[0], X.shape[1], 1))

    model = Sequential([
        LSTM(32, input_shape=(window_size, 1)),
```

```
        Dense(forecast_steps)
    ])
    model.compile(optimizer='adam', loss='mse')
    model.fit(X, y, epochs=20, batch_size=16, verbose=0)

    last_window = scaled_data[-window_size:].reshape(1, window_size, 1)
    forecast = model.predict(last_window)
    return scaler.inverse_transform(forecast)[0]

# 4. Visualization Functions for TRX
def plot_forecast_comparison(history, arima_forecast, lstm_forecast, title,␣
 ↪forecast_days=30):
    plt.figure(figsize=(14, 8))
    plt.plot(history.index, history.values, 'b-', label='Historical Prices',␣
 ↪linewidth=2)

    last_date = history.index[-1]
    forecast_dates = pd.date_range(
        start=last_date + pd.Timedelta(days=1),
        periods=forecast_days,
        freq='D'
    )

    plt.plot(forecast_dates, arima_forecast, 'r--', label='ARIMA Forecast',␣
 ↪linewidth=2)
    plt.plot(forecast_dates, lstm_forecast, 'g-.', label='LSTM Forecast',␣
 ↪linewidth=2)

    plt.title(f'TRX/USDT Price Forecast: {title}', fontsize=16)
    plt.xlabel('Date', fontsize=14)
    plt.ylabel('Price (USDT)', fontsize=14)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend(fontsize=12)
    plt.xticks(rotation=45)
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator())
    plt.tight_layout()
    plt.savefig(f'TRX_forecast_comparison_{title.replace(" ", "_")}.png',␣
 ↪dpi=300)
    plt.show()

def plot_window_comparison(results, interval):
    if interval not in results:
        return

    window_data = results[interval]
    window_names = list(window_data.keys())
```

```
    arima_1d = [window_data[w]['ARIMA_1D'] for w in window_names]
    lstm_1d = [window_data[w]['LSTM_1D'] for w in window_names]
    arima_1m = [window_data[w]['ARIMA_1M'] for w in window_names]
    lstm_1m = [window_data[w]['LSTM_1M'] for w in window_names]

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 12))

    # 1-Day Forecast Comparison
    x = np.arange(len(window_names))
    width = 0.35
    ax1.bar(x - width/2, arima_1d, width, label='ARIMA', color='skyblue')
    ax1.bar(x + width/2, lstm_1d, width, label='LSTM', color='salmon')
    ax1.set_title(f'TRX 1-Day Forecast Comparison ({interval} Interval)',␣
↪fontsize=14)
    ax1.set_ylabel('Price (USDT)', fontsize=12)
    ax1.set_xticks(x)
    ax1.set_xticklabels(window_names)
    ax1.legend()
    ax1.grid(axis='y', linestyle='--', alpha=0.7)

    # 1-Month Forecast Comparison
    ax2.bar(x - width/2, arima_1m, width, label='ARIMA', color='skyblue')
    ax2.bar(x + width/2, lstm_1m, width, label='LSTM', color='salmon')
    ax2.set_title(f'TRX 1-Month Forecast Comparison ({interval} Interval)',␣
↪fontsize=14)
    ax2.set_ylabel('Price (USDT)', fontsize=12)
    ax2.set_xticks(x)
    ax2.set_xticklabels(window_names)
    ax2.legend()
    ax2.grid(axis='y', linestyle='--', alpha=0.7)

    plt.tight_layout()
    plt.savefig(f'TRX_window_comparison_{interval}.png', dpi=300)
    plt.show()

def plot_model_performance(history, arima_fit, lstm_fit, title):
    plt.figure(figsize=(14, 8))
    plt.plot(history.index, history.values, 'b-', label='Historical Prices',␣
↪linewidth=2)
    plt.plot(history.index[-len(arima_fit):], arima_fit, 'r--', label='ARIMA␣
↪Fit', linewidth=1.5)
    plt.plot(history.index[-len(lstm_fit):], lstm_fit, 'g-.', label='LSTM Fit',␣
↪linewidth=1.5)

    plt.title(f'TRX Model Performance: {title}', fontsize=16)
    plt.xlabel('Date', fontsize=14)
```

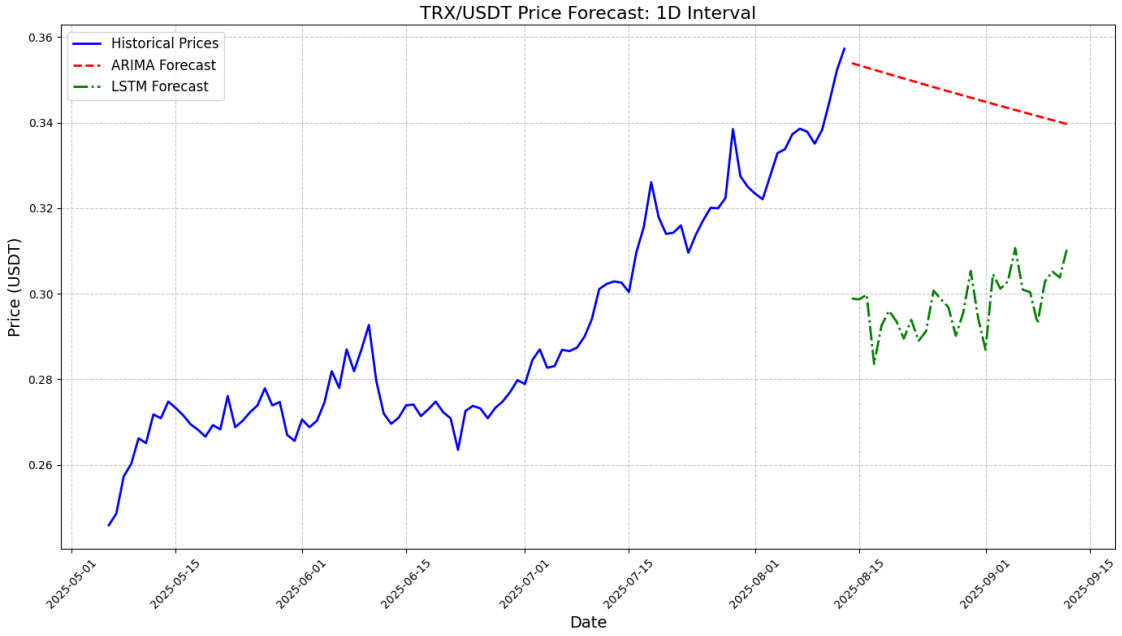
```
    plt.ylabel('Price (USDT)', fontsize=14)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend(fontsize=12)
    plt.xticks(rotation=45)
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
    plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator())
    plt.tight_layout()
    plt.savefig(f'TRX_model_performance_{title.replace(" ", "_")}.png', dpi=300)
    plt.show()

# 5. Configuration for TRX analysis
time_intervals = {
    '1D': '1d',
    '1W': '1w',
    '1M': '1M'
}

window_sizes = {
    '1h': 3600,
    '3h': 10800,
    '6h': 21600,
    '12h': 43200,
    '1d': 86400,
    '3d': 259200,
}

# 6. Main Processing for TRX
def process_and_visualize_trx():
    results = {}
    raw_data = fetch_historical_data(timeframe='1d', limit=500)

    for interval_name, interval_code in time_intervals.items():
        try:
            print(f"\nProcessing {interval_name} interval for TRX...")
            resampled = raw_data.resample(interval_code).agg({
                'open': 'first',
                'high': 'max',
                'low': 'min',
                'close': 'last',
                'volume': 'sum'
            }).ffill().dropna()

            close_series = resampled['close']

            if len(close_series) < 50:
                print(f"  Insufficient data ({len(close_series)} points).␣
 ↪Skipping.")
```

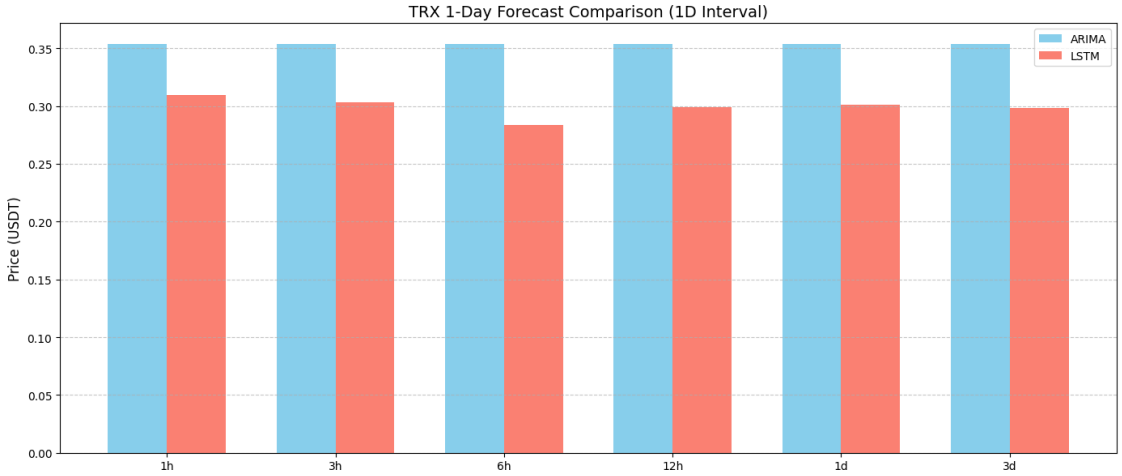
```
                continue

            interval_results = {}
            best_arima_1m = None
            best_lstm_1m = None

            for window_name, window_seconds in window_sizes.items():
                try:
                    print(f"  Calculating {window_name} windows...")
                    interval_seconds = pd.Timedelta(interval_code).
↪total_seconds()
                    window_bars = max(8, int(window_seconds / interval_seconds))

                    if len(close_series) < window_bars + 30:
                        print(f"    Not enough data for window. Have␣
↪{len(close_series)}, need {window_bars+30}")
                        continue

                    # Generate forecasts
                    arima_1m = arima_forecast(close_series, forecast_steps=30)
                    lstm_1m = lstm_forecast(close_series,␣
↪window_size=window_bars, forecast_steps=30)

                    # Store results
                    interval_results[window_name] = {
                        'ARIMA_1D': arima_1m[0],
                        'ARIMA_1M': arima_1m[-1],
                        'LSTM_1D': lstm_1m[0],
                        'LSTM_1M': lstm_1m[-1],
                        'ARIMA_Full': arima_1m,
                        'LSTM_Full': lstm_1m
                    }

                    # Track best models
                    last_price = close_series.iloc[-1]
                    if best_arima_1m is None or abs(arima_1m[-1] - last_price)␣
↪< abs(best_arima_1m[-1] - last_price):
                        best_arima_1m = arima_1m
                    if best_lstm_1m is None or abs(lstm_1m[-1] - last_price) <␣
↪abs(best_lstm_1m[-1] - last_price):
                        best_lstm_1m = lstm_1m

                except Exception as e:
                    print(f"    Error processing window {window_name}: {str(e)[:
↪100]}")
                    continue
```

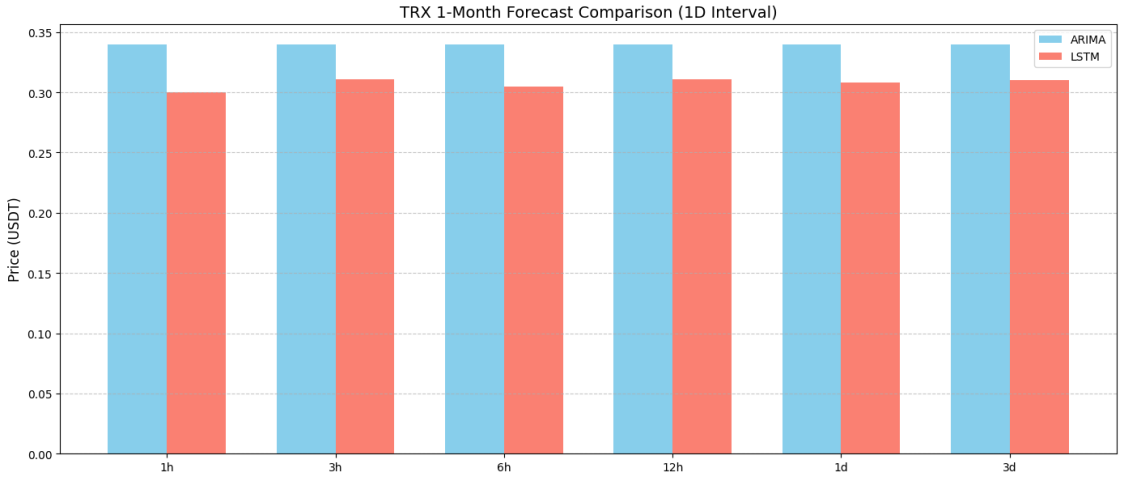
```
            results[interval_name] = interval_results

            # Generate visualizations
            if best_arima_1m is not None and best_lstm_1m is not None:
                plot_forecast_comparison(
                    history=close_series[-100:],
                    arima_forecast=best_arima_1m,
                    lstm_forecast=best_lstm_1m,
                    title=f"{interval_name} Interval",
                    forecast_days=30
                )

                plot_window_comparison(results, interval_name)

                if interval_results:
                    best_window = next(iter(interval_results))
                    # Generate model fit data
                    arima_fit = arima_forecast(close_series[-100:-30],␣
↪forecast_steps=70)
                    lstm_fit = lstm_forecast(close_series[-100:-30],␣
↪window_size=window_bars, forecast_steps=70)

                    plot_model_performance(
                        history=close_series[-100:],
                        arima_fit=arima_fit,
                        lstm_fit=lstm_fit,
                        title=f"{interval_name} Interval ({best_window} window)"
                    )

        except Exception as e:
            print(f"Failed processing interval {interval_name}: {str(e)[:100]}")
            continue

    return results

# 7. Execute TRX Analysis
if __name__ == "__main__":
    print("Starting TRX/USDT forecasting and visualization...")
    results = process_and_visualize_trx()

    # Display results
    print("\n\n=== TRX/USDT Forecast Results ===")
    for interval, window_data in results.items():
        print(f"\n[{interval} Interval]")
        for window, values in window_data.items():
            print(f"  {window} window:")
```

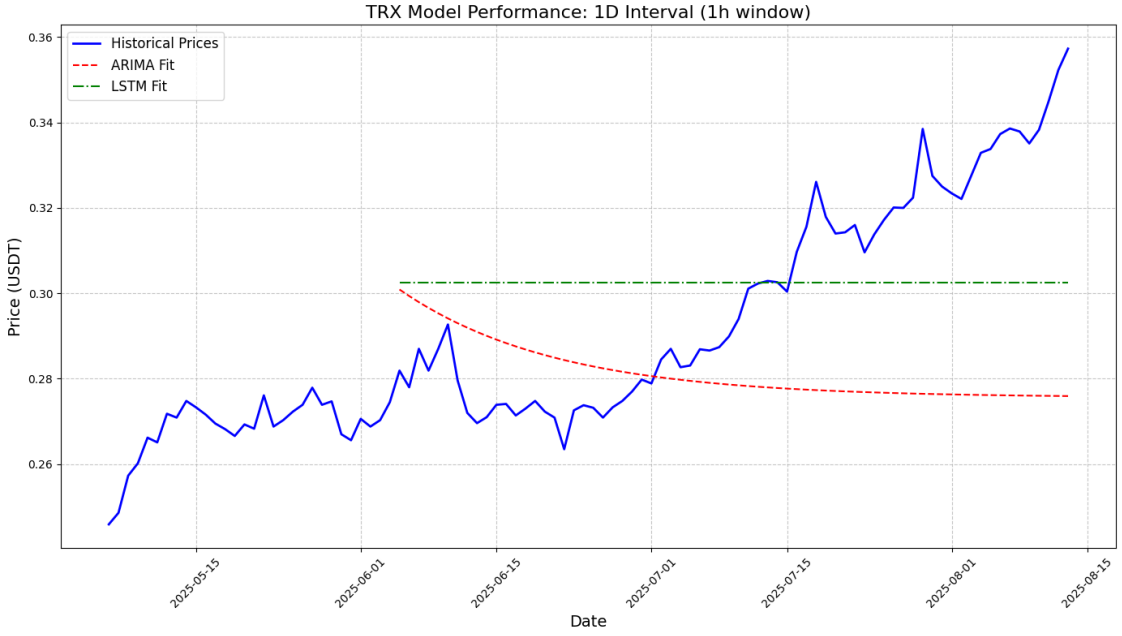
```
            print(f"    ARIMA 1D: {values['ARIMA_1D']:.6f}")
            print(f"    LSTM 1D:  {values['LSTM_1D']:.6f}")
            print(f"    ARIMA 1M: {values['ARIMA_1M']:.6f}")
            print(f"    LSTM 1M:  {values['LSTM_1M']:.6f}")
    print("\nTRX forecasting complete! Visualizations saved as PNG files.")
```

```
Starting TRX/USDT forecasting and visualization…
Error from binance: binance GET https://api.binance.com/api/v3/exchangeInfo 451
{
  "code": 0,
  "msg": "Service unavai… trying next exchange
Successfully fetched TRX/USDT data from kucoin

Processing 1D interval for TRX…
  Calculating 1h windows…
1/1             0s 196ms/step
  Calculating 3h windows…
1/1             0s 192ms/step
  Calculating 6h windows…
1/1             0s 206ms/step
  Calculating 12h windows…
1/1             0s 295ms/step
  Calculating 1d windows…
1/1             0s 204ms/step
  Calculating 3d windows…
1/1             0s 199ms/step
```

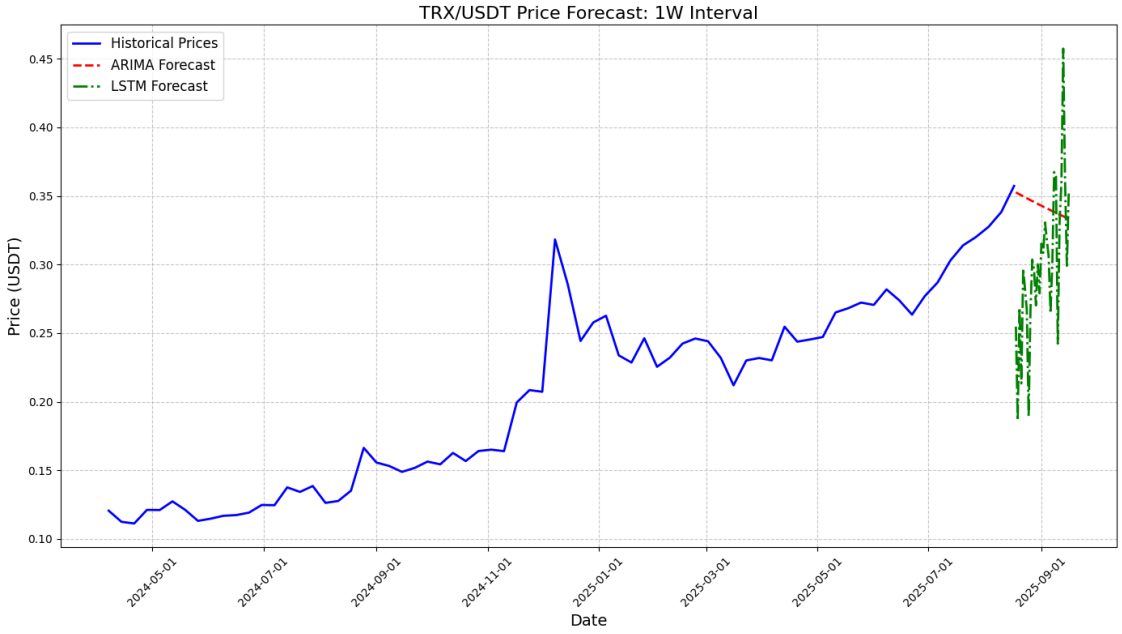


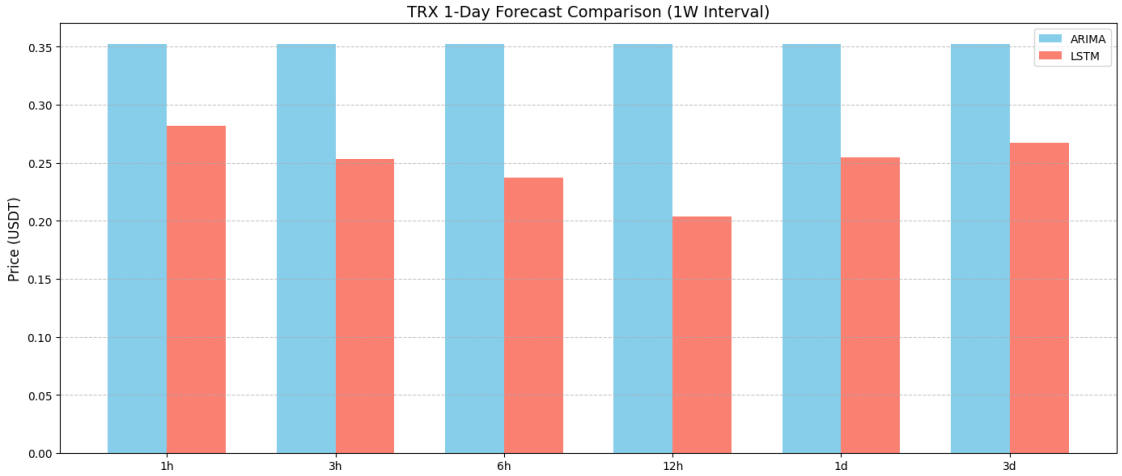

TRX 1-Day Forecast Comparison (1D Interval)
ARIMA
LSTM
Price (USDT)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
1h
3h
6h
12h
1d
3d

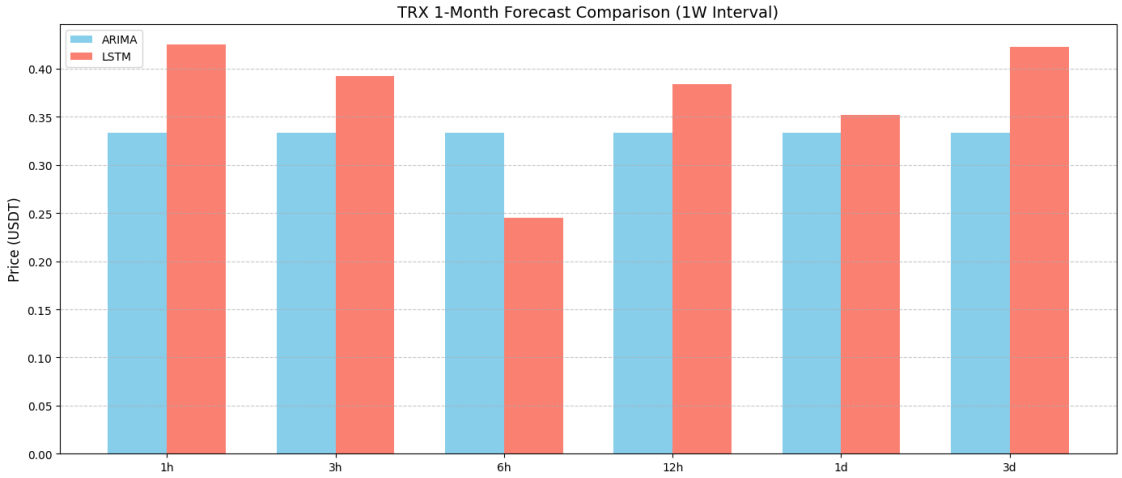

TRX 1-Month Forecast Comparison (1D Interval)
ARIMA
LSTM
Price (USDT)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
1h
3h
6h
12h
1d
3d

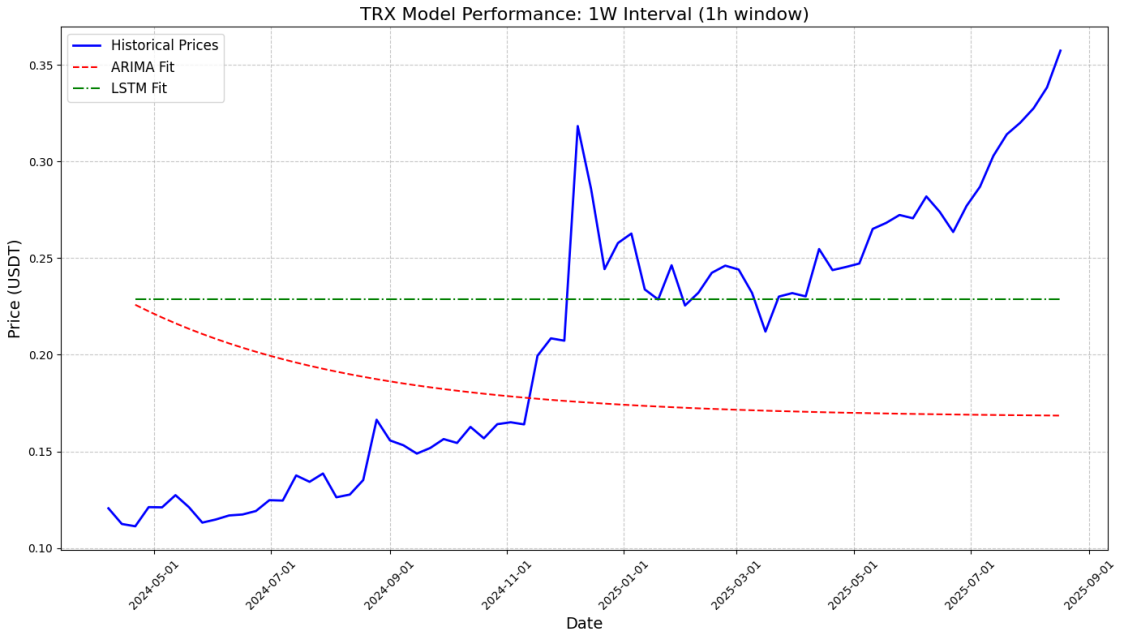

TRX Model Performance: 1D Interval (1h window)
Historical Prices
ARIMA Fit
LSTM Fit
Price (USDT)
0.26
0.28
0.30
0.32
0.34
0.36
2025-05-15
2025-06-01
2025-06-15
2025-07-01
2025-07-15
2025-08-01
2025-08-15
Date

```
Processing 1W interval for TRX…
  Calculating 1h windows…
1/1             0s 342ms/step
  Calculating 3h windows…
1/1             0s 320ms/step
  Calculating 6h windows…
1/1             0s 219ms/step
  Calculating 12h windows…
1/1             0s 212ms/step
  Calculating 1d windows…
1/1             0s 201ms/step
  Calculating 3d windows…
1/1             0s 193ms/step
```




TRX 1-Day Forecast Comparison (1W Interval)
ARIMA
LSTM
Price (USDT)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
1h
3h
6h
12h
1d
3d


TRX 1-Month Forecast Comparison (1W Interval)
ARIMA
LSTM
Price (USDT)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
1h
3h
6h
12h
1d
3d


TRX Model Performance: 1W Interval (1h window)
Historical Prices
ARIMA Fit
LSTM Fit
Price (USDT)
0.10
0.15
0.20
0.25
0.30
0.35
2024-05-01
2024-07-01
2024-09-01
2024-11-01
2025-01-01
2025-03-01
2025-05-01
2025-07-01
2025-09-01
Date

```
Processing 1M interval for TRX…
  Insufficient data (17 points). Skipping.


=== TRX/USDT Forecast Results ===

[1D Interval]
  1h window:
    ARIMA 1D: 0.353929
    LSTM 1D:  0.309232
    ARIMA 1M: 0.339669
    LSTM 1M:  0.300251
  3h window:
    ARIMA 1D: 0.353929
    LSTM 1D:  0.303373
    ARIMA 1M: 0.339669
    LSTM 1M:  0.310721
  6h window:
    ARIMA 1D: 0.353929
    LSTM 1D:  0.283693
    ARIMA 1M: 0.339669
    LSTM 1M:  0.304839
  12h window:
    ARIMA 1D: 0.353929
    LSTM 1D:  0.298886
    ARIMA 1M: 0.339669
    LSTM 1M:  0.310844
  1d window:
    ARIMA 1D: 0.353929
    LSTM 1D:  0.301331
    ARIMA 1M: 0.339669
    LSTM 1M:  0.308169
  3d window:
    ARIMA 1D: 0.353929
    LSTM 1D:  0.298410
    ARIMA 1M: 0.339669
    LSTM 1M:  0.309971

[1W Interval]
  1h window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.282066
    ARIMA 1M: 0.333356
    LSTM 1M:  0.425150
```

```
  3h window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.253562
    ARIMA 1M: 0.333356
    LSTM 1M:  0.392040
  6h window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.237235
    ARIMA 1M: 0.333356
    LSTM 1M:  0.245004
  12h window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.203936
    ARIMA 1M: 0.333356
    LSTM 1M:  0.384251
  1d window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.254750
    ARIMA 1M: 0.333356
    LSTM 1M:  0.351793
  3d window:
    ARIMA 1D: 0.352612
    LSTM 1D:  0.267495
    ARIMA 1M: 0.333356
    LSTM 1M:  0.422566

TRX forecasting complete! Visualizations saved as PNG files.
```

[20]:
```python
import ccxt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller, acf, pacf
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense
import time
import warnings
import matplotlib as mpl

warnings.filterwarnings('ignore')


# STYLE SETTINGS #
```

```
plt.style.use('default')

# Custom styling parameters
mpl.rcParams['font.family'] = 'DejaVu Sans'
mpl.rcParams['axes.linewidth'] = 1.2
mpl.rcParams['axes.edgecolor'] = '0.15'
mpl.rcParams['axes.labelcolor'] = '#404040'
mpl.rcParams['axes.titlesize'] = 16
mpl.rcParams['axes.titleweight'] = 'bold'
mpl.rcParams['axes.titlepad'] = 12
mpl.rcParams['axes.labelsize'] = 13
mpl.rcParams['axes.labelweight'] = 'medium'
mpl.rcParams['xtick.labelsize'] = 11
mpl.rcParams['ytick.labelsize'] = 11
mpl.rcParams['legend.fontsize'] = 11
mpl.rcParams['grid.color'] = '0.92'
mpl.rcParams['grid.linestyle'] = '-'
mpl.rcParams['grid.linewidth'] = 0.8
mpl.rcParams['figure.figsize'] = (14, 8)
mpl.rcParams['figure.dpi'] = 120
mpl.rcParams['savefig.dpi'] = 300
mpl.rcParams['savefig.transparent'] = True
mpl.rcParams['lines.linewidth'] = 2.2

# Color scheme
COLOR_HIST = '#1f77b4'
COLOR_ARIMA = '#d62728'
COLOR_LSTM = '#ff7f0e'
COLOR_BG = '#f8f9fa'
COLOR_GRID = '#e9ecef'
COLOR_TEXT = '#343a40'

# Exchange configuration for TRX/USDT
EXCHANGES = [
    {'name': 'binance', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'kucoin', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'bybit', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'huobi', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/
 ↪USDT'},
    {'name': 'okx', 'params': {'enableRateLimit': True}, 'symbol': 'TRX/USDT'},
]

# ====================== #
# DATA & MODEL FUNCTIONS #
```

```
# ====================== #
def fetch_historical_data(timeframe='1d', limit=1000):
    """Fetch TRX/USDT data with exchange fallback"""
    for exchange_config in EXCHANGES:
        try:
            exchange_class = getattr(ccxt, exchange_config['name'])
            exchange = exchange_class(exchange_config['params'])
            time.sleep(exchange.rateLimit / 1000)

            symbol = exchange_config['symbol']
            ohlcv = exchange.fetch_ohlcv(symbol, timeframe, limit=limit)

            df = pd.DataFrame(ohlcv, columns=['timestamp', 'open', 'high',␣
 ↪'low', 'close', 'volume'])
            df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
            df.set_index('timestamp', inplace=True)
            print(f"  Fetched TRX/USDT data from {exchange_config['name']}")
            return df

        except (ccxt.NetworkError, ccxt.ExchangeError) as e:
            print(f"  Error from {exchange_config['name']}: {str(e)[:100]}...␣
 ↪trying next")
            continue
        except Exception as e:
            print(f"  Unexpected error with {exchange_config['name']}: {str(e)[:
 ↪100]}... trying next")
            continue

    raise RuntimeError("  All exchanges failed. Please check network␣
 ↪connection")

def arima_forecast(data, forecast_steps=30):
    """ARIMA forecasting with error handling"""
    if len(data) < 30:
        return np.array([data.iloc[-1]] * forecast_steps)

    d = 0
    if len(data) > 50:
        d = find_optimal_d(data)

    if d > 0:
        diff_data = data.diff(d).dropna()
    else:
        diff_data = data

    p, q = (1, 1)
    if len(diff_data) > 50:
```

```
        try:
            p, q = find_optimal_pq(diff_data)
        except:
            pass

    try:
        model = ARIMA(data, order=(p, d, q))
        model_fit = model.fit()
        return model_fit.forecast(steps=forecast_steps).values
    except:
        return np.array([data.rolling(window=5).mean().iloc[-1]] *␣
↪forecast_steps)

def find_optimal_d(data, max_d=2):
    """Determine optimal differencing order"""
    d = 0
    for i in range(max_d + 1):
        try:
            result = adfuller(data if i == 0 else data.diff().dropna())
            if result[1] <= 0.05:
                return d
            d = i
        except:
            break
    return min(d, max_d)

def find_optimal_pq(data, max_p=5, max_q=5):
    """Find optimal ARMA parameters"""
    try:
        acf_vals = acf(data, nlags=max_p + max_q)
        pacf_vals = pacf(data, nlags=max_p + max_q)

        # Find significant lags with threshold
        p = next((i for i in range(1, len(pacf_vals)) if abs(pacf_vals[i]) > 1.
↪96/np.sqrt(len(data))), 1)
        q = next((i for i in range(1, len(acf_vals)) if abs(acf_vals[i]) > 1.96/
↪np.sqrt(len(data))), 1)

        return min(p, max_p), min(q, max_q)
    except:
        return (1, 1)

def create_dataset(data, window_size=8, target_size=1):
    """Create training dataset with safe bounds checking"""
    X, y = [], []
    n = len(data)
    if n < window_size + target_size:
```

```
        return np.array([]), np.array([])

    for i in range(n - window_size - target_size):
        X.append(data[i:(i+window_size)])
        y.append(data[(i+window_size):(i+window_size+target_size)])
    return np.array(X), np.array(y)

def lstm_forecast(data, window_size=8, forecast_steps=30):
    """LSTM forecasting with error handling"""
    if len(data) < window_size + 10:
        return np.array([data.iloc[-1]] * forecast_steps)

    scaler = MinMaxScaler(feature_range=(0,1))
    scaled_data = scaler.fit_transform(data.values.reshape(-1,1))

    X, y = create_dataset(scaled_data, window_size, forecast_steps)
    if len(X) == 0:
        return np.array([data.iloc[-1]] * forecast_steps)

    X = np.reshape(X, (X.shape[0], X.shape[1], 1))

    model = Sequential([
        LSTM(32, input_shape=(window_size, 1)),
        Dense(forecast_steps)
    ])
    model.compile(optimizer='adam', loss='mse')
    model.fit(X, y, epochs=20, batch_size=16, verbose=0)

    last_window = scaled_data[-window_size:].reshape(1, window_size, 1)
    forecast = model.predict(last_window)
    return scaler.inverse_transform(forecast)[0]

# ======================== #
# PROFESSIONAL VISUALIZATIONS #
# ======================== #
def plot_forecast_comparison(history, arima_forecast, lstm_forecast, title,␣
 ↪forecast_days=30):
    """Create artistic forecast visualization"""
    fig, ax = plt.subplots(figsize=(16, 9))
    fig.patch.set_facecolor(COLOR_BG)
    ax.set_facecolor(COLOR_BG)

    # Plot historical data
    ax.plot(history.index, history.values, color=COLOR_HIST,
            label='Historical Prices', linewidth=3.0, alpha=0.9)

    # Generate future dates
```

```
    last_date = history.index[-1]
    forecast_dates = pd.date_range(
        start=last_date + pd.Timedelta(days=1),
        periods=forecast_days,
        freq='D'
    )

    # Plot forecasts with artistic styles
    ax.plot(forecast_dates, arima_forecast, color=COLOR_ARIMA,
            linestyle='--', label='ARIMA Forecast', linewidth=2.8, alpha=0.95)
    ax.plot(forecast_dates, lstm_forecast, color=COLOR_LSTM,
            linestyle='-.', label='LSTM Forecast', linewidth=2.8, alpha=0.95)

    # Forecast confidence bands (artistic effect)
    ax.fill_between(forecast_dates,
                    np.minimum(arima_forecast, lstm_forecast),
                    np.maximum(arima_forecast, lstm_forecast),
                    color=COLOR_ARIMA, alpha=0.08)

    # Vertical separator between history and forecast
    ax.axvline(x=last_date, color='#6c757d', linestyle=':', linewidth=1.8,␣
↪alpha=0.7)

    # Formatting
    ax.set_title(f'TRX/USDT Price Forecast: {title}\n',
                 fontsize=20, fontweight='bold', color=COLOR_TEXT, pad=20)
    ax.set_xlabel('Date', fontsize=14, labelpad=15, color=COLOR_TEXT)
    ax.set_ylabel('Price (USDT)', fontsize=14, labelpad=15, color=COLOR_TEXT)

    # Grid and spines
    ax.grid(True, color=COLOR_GRID, linestyle='-', alpha=0.8)
    for spine in ax.spines.values():
        spine.set_visible(False)

    # Legend with artistic effect
    legend = ax.legend(loc='upper left', frameon=True, framealpha=0.95,
                       facecolor='white', edgecolor=COLOR_GRID, fontsize=12)
    legend.get_frame().set_linewidth(1.2)

    # Date formatting
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %d, %Y'))
    ax.xaxis.set_major_locator(mdates.AutoDateLocator())
    fig.autofmt_xdate(rotation=45, ha='center')

    # Price formatting
    ax.yaxis.set_major_formatter(ticker.StrMethodFormatter('${x:,.4f}'))
```

```
    # Annotations
    plt.annotate(f'Last Price: ${history.iloc[-1]:.4f}',
                 xy=(last_date, history.iloc[-1]),
                 xytext=(10, 10), textcoords='offset points',
                 arrowprops=dict(arrowstyle='->', color=COLOR_HIST),
                 fontsize=11, backgroundcolor='white')

    plt.tight_layout()
    plt.subplots_adjust(top=0.92, bottom=0.15)
    #plt.savefig(f'TRX_forecast_{title.replace(" ", "_")}.pdf',␣
 ↪bbox_inches='tight', pad_inches=0.5)
    #plt.close()
    plt.show()

def plot_window_comparison(results, interval):
    """Artistic bar chart comparison of window sizes"""
    if interval not in results:
        return

    window_data = results[interval]
    window_names = list(window_data.keys())

    # Prepare data
    arima_1d = [window_data[w]['ARIMA_1D'] for w in window_names]
    lstm_1d = [window_data[w]['LSTM_1D'] for w in window_names]
    arima_1m = [window_data[w]['ARIMA_1M'] for w in window_names]
    lstm_1m = [window_data[w]['LSTM_1M'] for w in window_names]

    # Create figure
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(16, 14))
    fig.patch.set_facecolor(COLOR_BG)
    ax1.set_facecolor(COLOR_BG)
    ax2.set_facecolor(COLOR_BG)
    fig.suptitle(f'TRX Forecast Performance by Window Size ({interval}␣
 ↪Interval)',
                 fontsize=20, fontweight='bold', color=COLOR_TEXT, y=0.96)

    # Plot 1D forecasts
    x = np.arange(len(window_names))
    width = 0.38
    ax1.bar(x - width/2, arima_1d, width, label='ARIMA', color=COLOR_ARIMA,
           edgecolor='white', linewidth=1.2, alpha=0.92)
    ax1.bar(x + width/2, lstm_1d, width, label='LSTM', color=COLOR_LSTM,
           edgecolor='white', linewidth=1.2, alpha=0.92)

    # Add data labels
    for i, v in enumerate(arima_1d):
```

```
        ax1.text(i - width/2, v + max(arima_1d+lstm_1d)*0.01, f'${v:.4f}',
                 ha='center', fontsize=9, color=COLOR_ARIMA, fontweight='bold')
    for i, v in enumerate(lstm_1d):
        ax1.text(i + width/2, v + max(arima_1d+lstm_1d)*0.01, f'${v:.4f}',
                 ha='center', fontsize=9, color=COLOR_LSTM, fontweight='bold')

    ax1.set_title('1-Day Forecast', fontsize=16, pad=15, color=COLOR_TEXT)
    ax1.set_ylabel('Price (USDT)', fontsize=13, labelpad=10, color=COLOR_TEXT)
    ax1.set_xticks(x)
    ax1.set_xticklabels([f"{w}\nWindow" for w in window_names], fontsize=11)
    ax1.legend(frameon=True, framealpha=0.95, facecolor='white',␣
↪edgecolor=COLOR_GRID)
    ax1.grid(axis='y', color=COLOR_GRID, linestyle='-', alpha=0.8)
    ax1.yaxis.set_major_formatter(ticker.StrMethodFormatter('${x:,.4f}'))

    # Plot 1M forecasts
    ax2.bar(x - width/2, arima_1m, width, label='ARIMA', color=COLOR_ARIMA,
           edgecolor='white', linewidth=1.2, alpha=0.92)
    ax2.bar(x + width/2, lstm_1m, width, label='LSTM', color=COLOR_LSTM,
           edgecolor='white', linewidth=1.2, alpha=0.92)

    # Add data labels
    for i, v in enumerate(arima_1m):
        ax2.text(i - width/2, v + max(arima_1m+lstm_1m)*0.01, f'${v:.4f}',
                 ha='center', fontsize=9, color=COLOR_ARIMA, fontweight='bold')
    for i, v in enumerate(lstm_1m):
        ax2.text(i + width/2, v + max(arima_1m+lstm_1m)*0.01, f'${v:.4f}',
                 ha='center', fontsize=9, color=COLOR_LSTM, fontweight='bold')

    ax2.set_title('1-Month Forecast', fontsize=16, pad=15, color=COLOR_TEXT)
    ax2.set_ylabel('Price (USDT)', fontsize=13, labelpad=10, color=COLOR_TEXT)
    ax2.set_xticks(x)
    ax2.set_xticklabels([f"{w}\nWindow" for w in window_names], fontsize=11)
    ax2.legend(frameon=True, framealpha=0.95, facecolor='white',␣
↪edgecolor=COLOR_GRID)
    ax2.grid(axis='y', color=COLOR_GRID, linestyle='-', alpha=0.8)
    ax2.yaxis.set_major_formatter(ticker.StrMethodFormatter('${x:,.4f}'))

    # Remove spines
    for ax in [ax1, ax2]:
        for spine in ax.spines.values():
            spine.set_visible(False)

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    #plt.savefig(f'TRX_window_comparison_{interval}.pdf', bbox_inches='tight',␣
↪pad_inches=0.5)
    #plt.close()
```

```
    plt.show()

def plot_model_performance(history, arima_fit, lstm_fit, title):
    """Artistic visualization of model performance"""
    fig, ax = plt.subplots(figsize=(16, 9))
    fig.patch.set_facecolor(COLOR_BG)
    ax.set_facecolor(COLOR_BG)

    # Plot historical data
    ax.plot(history.index, history.values, color=COLOR_HIST,
            label='Historical Prices', linewidth=3.2, alpha=0.95)

    # Plot model fits
    fit_dates = history.index[-len(arima_fit):]
    ax.plot(fit_dates, arima_fit, color=COLOR_ARIMA,
            linestyle='--', label='ARIMA Fit', linewidth=2.6, alpha=0.9)
    ax.plot(fit_dates, lstm_fit, color=COLOR_LSTM,
            linestyle='-.', label='LSTM Fit', linewidth=2.6, alpha=0.9)

    # Calculate error metrics
    arima_diff = np.abs(arima_fit - history[-len(arima_fit):].values)
    lstm_diff = np.abs(lstm_fit - history[-len(lstm_fit):].values)
    arima_mae = np.mean(arima_diff)
    lstm_mae = np.mean(lstm_diff)

    # Annotation box
    textstr = '\n'.join((
        f'ARIMA Fit:',
        f'• MAE = ${arima_mae:.6f}',
        f'',
        f'LSTM Fit:',
        f'• MAE = ${lstm_mae:.6f}'))

    props = dict(boxstyle='round', facecolor='white', edgecolor=COLOR_GRID,
                 linewidth=1.5, alpha=0.95)
    ax.text(0.05, 0.95, textstr, transform=ax.transAxes, fontsize=12,
            verticalalignment='top', bbox=props, fontfamily='monospace')

    # Formatting
    ax.set_title(f'TRX Model Performance: {title}\n',
                 fontsize=20, fontweight='bold', color=COLOR_TEXT, pad=20)
    ax.set_xlabel('Date', fontsize=14, labelpad=15, color=COLOR_TEXT)
    ax.set_ylabel('Price (USDT)', fontsize=14, labelpad=15, color=COLOR_TEXT)

    # Grid and spines
    ax.grid(True, color=COLOR_GRID, linestyle='-', alpha=0.8)
    for spine in ax.spines.values():
```

```
        spine.set_visible(False)

    # Legend
    legend = ax.legend(loc='upper left', frameon=True, framealpha=0.95,
                       facecolor='white', edgecolor=COLOR_GRID, fontsize=12)
    legend.get_frame().set_linewidth(1.2)

    # Date formatting
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %d, %Y'))
    ax.xaxis.set_major_locator(mdates.AutoDateLocator())
    fig.autofmt_xdate(rotation=45, ha='center')

    # Price formatting
    ax.yaxis.set_major_formatter(ticker.StrMethodFormatter('${x:,.4f}'))

    plt.tight_layout()
    plt.subplots_adjust(top=0.92, bottom=0.15)
    #plt.savefig(f'TRX_model_performance_{title.replace(" ", "_")}.pdf',␣
 ↪bbox_inches='tight', pad_inches=0.5)
    #plt.close()
    plt.show()

# =============== #
# MAIN PROCESSING #
# =============== #
def process_and_visualize_trx():
    print("\n" + "="*60)
    print(" TRX/USDT FORECASTING SYSTEM ".center(60, ' '))
    print("="*60 + "\n")

    # Fetch data
    print("Fetching TRX/USDT historical data...")
    raw_data = fetch_historical_data(timeframe='1d', limit=500)

    # Configuration
    time_intervals = {
        '1D': '1d',
        '1W': '1w',
        '1M': '1M'
    }

    window_sizes = {
        '1H': 3600,
        '4H': 14400,
        '12H': 43200,
        '1D': 86400,
        '3D': 259200,
```

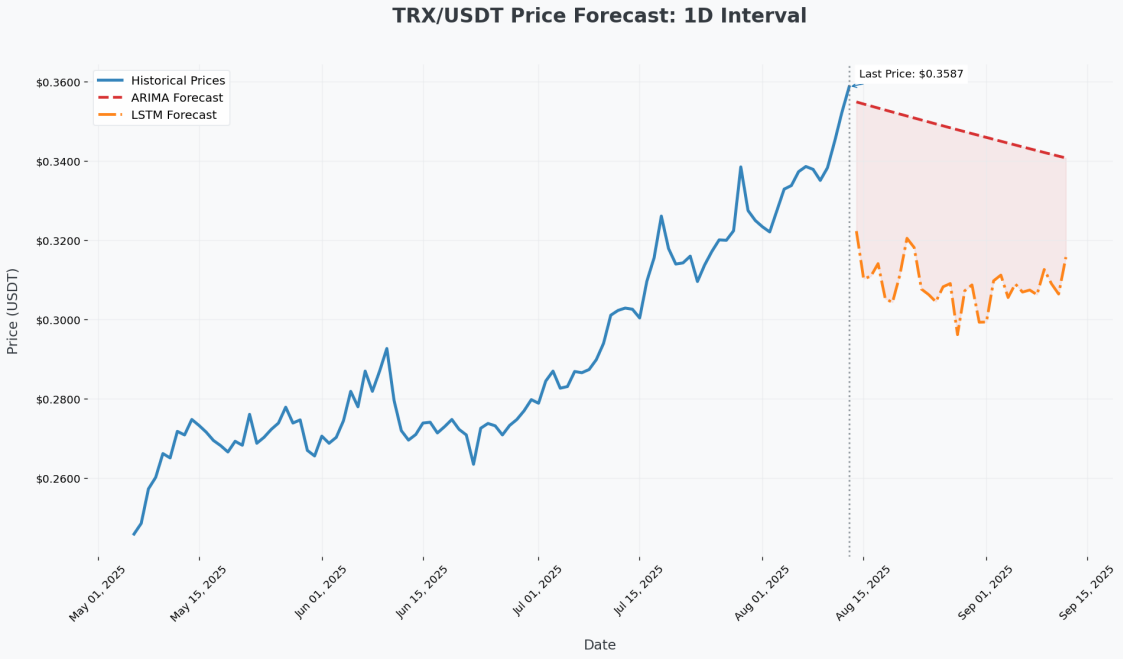
```
    }

    results = {}

    for interval_name, interval_code in time_intervals.items():
        try:
            print(f"\n{'='*40}")
            print(f" Processing {interval_name} interval ".center(40, ' '))
            print("="*40)

            # Resample data
            resampled = raw_data.resample(interval_code).agg({
                'open': 'first',
                'high': 'max',
                'low': 'min',
                'close': 'last',
                'volume': 'sum'
            }).ffill().dropna()

            close_series = resampled['close']

            if len(close_series) < 50:
                print(f"    Insufficient data ({len(close_series)} points).␣
↪Skipping.")
                continue

            interval_results = {}
            best_arima_1m = None
            best_lstm_1m = None

            for window_name, window_seconds in window_sizes.items():
                try:
                    print(f"  • Calculating {window_name} window...")
                    interval_seconds = pd.Timedelta(interval_code).
↪total_seconds()
                    window_bars = max(8, int(window_seconds / interval_seconds))

                    if len(close_series) < window_bars + 30:
                        print(f"      Not enough data. Have␣
↪{len(close_series)}, need {window_bars+30}")
                        continue

                    # Generate forecasts
                    arima_1m = arima_forecast(close_series, forecast_steps=30)
                    lstm_1m = lstm_forecast(close_series,␣
↪window_size=window_bars, forecast_steps=30)
```

```
                    # Store results
                    interval_results[window_name] = {
                        'ARIMA_1D': arima_1m[0],
                        'ARIMA_1M': arima_1m[-1],
                        'LSTM_1D': lstm_1m[0],
                        'LSTM_1M': lstm_1m[-1],
                        'ARIMA_Full': arima_1m,
                        'LSTM_Full': lstm_1m
                    }

                    # Track best models
                    last_price = close_series.iloc[-1]
                    if best_arima_1m is None or abs(arima_1m[-1] - last_price)␣
↪< abs(best_arima_1m[-1] - last_price):
                        best_arima_1m = arima_1m
                    if best_lstm_1m is None or abs(lstm_1m[-1] - last_price) <␣
↪abs(best_lstm_1m[-1] - last_price):
                        best_lstm_1m = lstm_1m

                except Exception as e:
                    print(f"      Error processing window: {str(e)[:100]}")
                    continue

            results[interval_name] = interval_results

            # Generate visualizations
            if best_arima_1m is not None and best_lstm_1m is not None:
                print("    Creating forecast comparison visualization...")
                plot_forecast_comparison(
                    history=close_series[-100:],
                    arima_forecast=best_arima_1m,
                    lstm_forecast=best_lstm_1m,
                    title=f"{interval_name} Interval",
                    forecast_days=30
                )

                print("    Creating window comparison visualization...")
                plot_window_comparison(results, interval_name)

                if interval_results:
                    best_window = next(iter(interval_results))
                    print(f"    Creating model performance visualization␣
↪({best_window} window)...")
                    # Generate model fit data
                    arima_fit = arima_forecast(close_series[-100:-30],␣
↪forecast_steps=70)
```

```
                    lstm_fit = lstm_forecast(close_series[-100:-30],␣
 ↪window_size=window_bars, forecast_steps=70)

                    plot_model_performance(
                        history=close_series[-100:],
                        arima_fit=arima_fit,
                        lstm_fit=lstm_fit,
                        title=f"{interval_name} Interval"
                    )

            except Exception as e:
                print(f"  Failed processing interval: {str(e)[:100]}")
                continue

    return results

# ============== #
# EXECUTION MAIN #
# ============== #
if __name__ == "__main__":
    # Start processing
    results = process_and_visualize_trx()

    # Display results
    print("\n\n" + "="*60)
    print(" FORECAST RESULTS ".center(60, ' '))
    print("="*60)

    for interval, window_data in results.items():
        print(f"\n {interval} Interval  ")
        for window, values in window_data.items():
            print(f"\n    {window} Window:")
            print(f"    ARIMA 1D: ${values['ARIMA_1D']:.6f}")
            print(f"    LSTM 1D:  ${values['LSTM_1D']:.6f}")
            print(f"    ARIMA 1M: ${values['ARIMA_1M']:.6f}")
            print(f"    LSTM 1M:  ${values['LSTM_1M']:.6f}")

    print("\n" + "="*60)
    print(" VISUALIZATIONS SAVED TO CURRENT DIRECTORY ".center(60))
    print("="*60)
    print("\nForecasting complete! Professional charts saved as PNG files.\n")
```

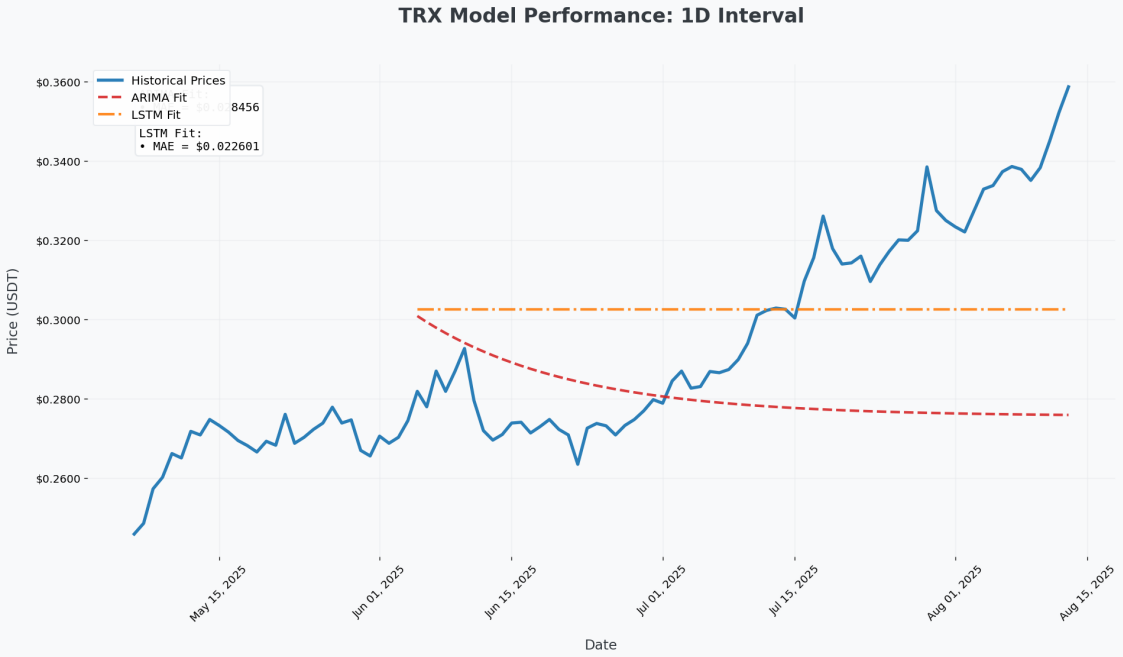
```
============================================================
         TRX/USDT FORECASTING SYSTEM
============================================================

Fetching TRX/USDT historical data…
```

```
  Error from binance: binance GET https://api.binance.com/api/v3/exchangeInfo
451  {
  "code": 0,
  "msg": "Service unavai… trying next
  Fetched TRX/USDT data from kucoin

========================================
     Processing 1D interval
========================================
  • Calculating 1H window…
1/1             0s 265ms/step
  • Calculating 4H window…
1/1             0s 186ms/step
  • Calculating 12H window…
1/1             0s 173ms/step
  • Calculating 1D window…
1/1             0s 171ms/step
  • Calculating 3D window…
1/1             0s 169ms/step
    Creating forecast comparison visualization…
```

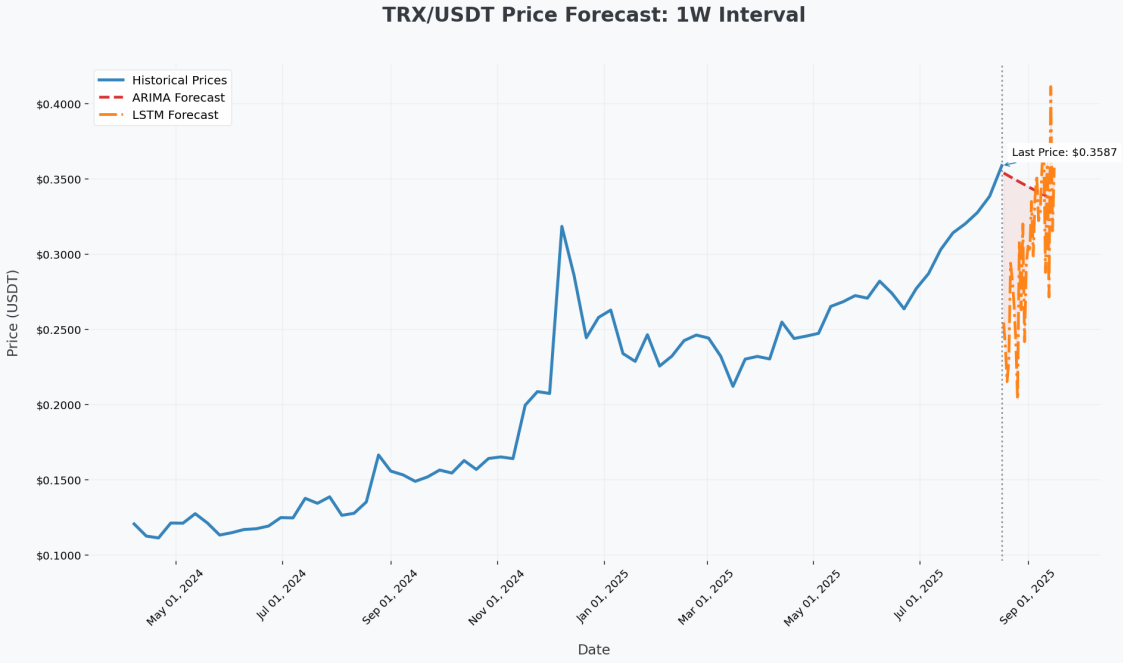



```
    Creating window comparison visualization…
```

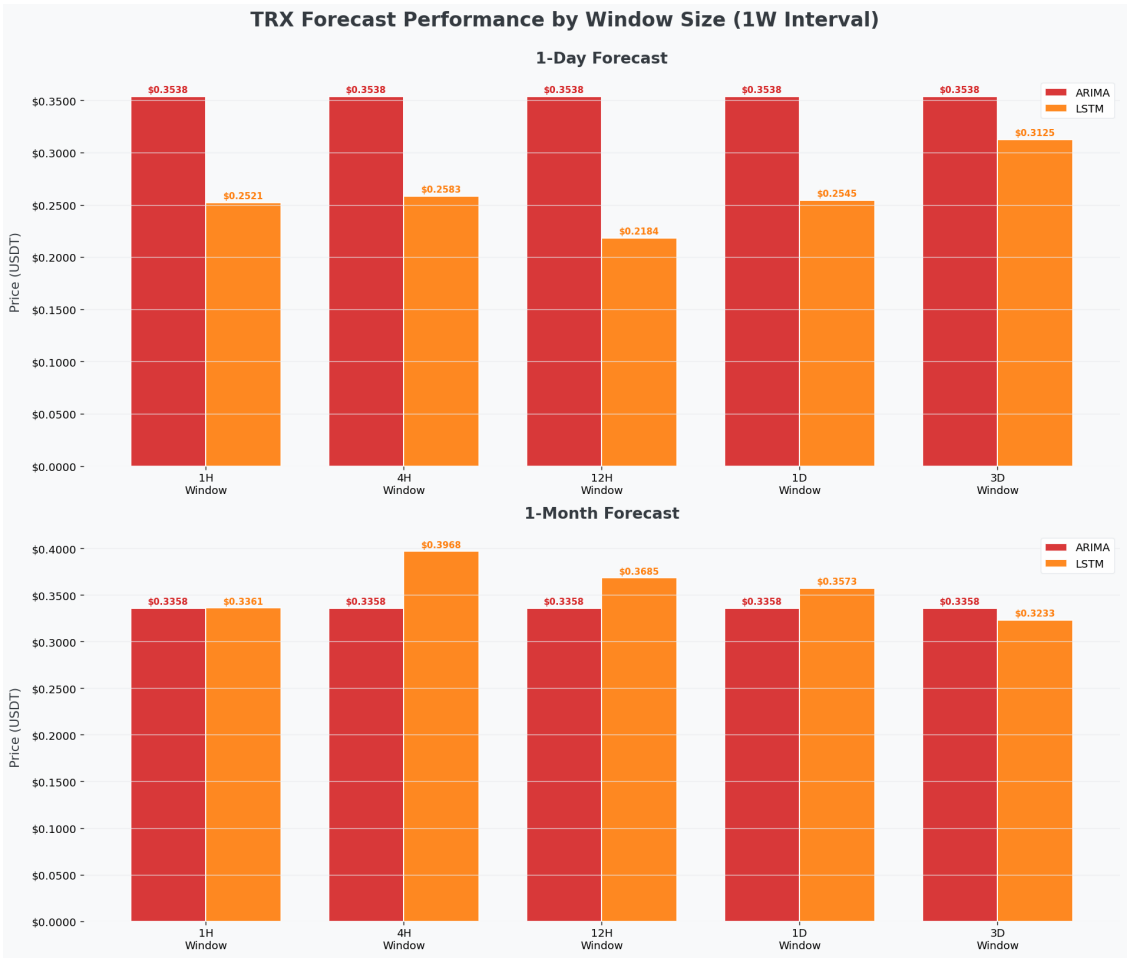
Creating model performance visualization (1H window)…



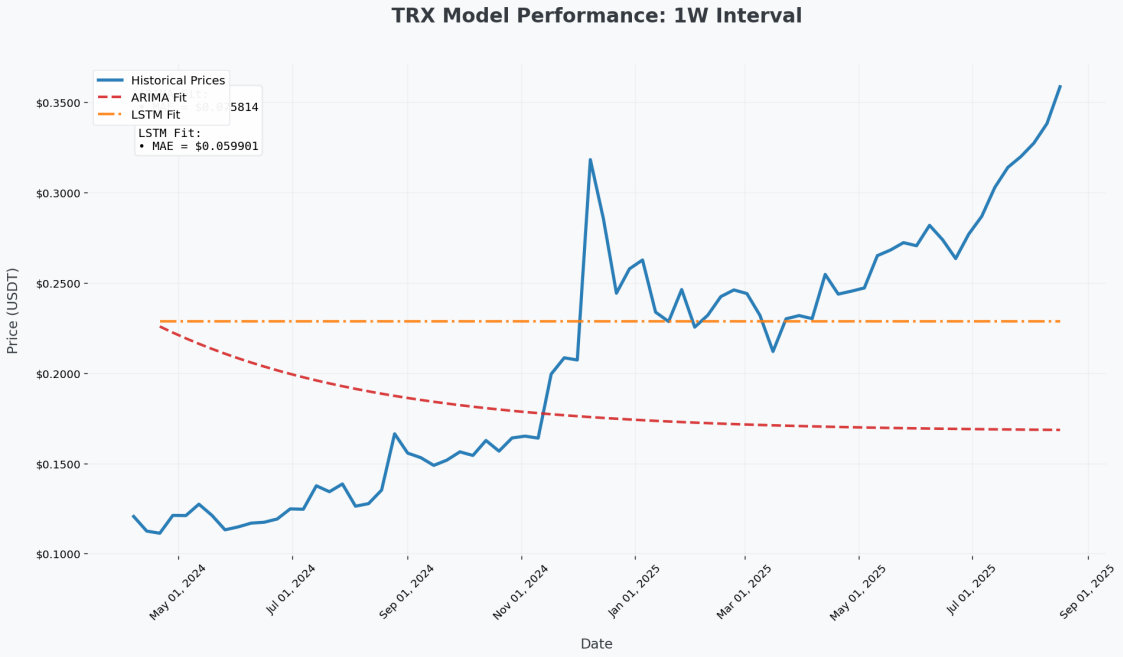

```
========================================
     Processing 1W interval
========================================
  • Calculating 1H window…
1/1             0s 300ms/step
  • Calculating 4H window…
1/1             0s 181ms/step
  • Calculating 12H window…
1/1             0s 175ms/step
  • Calculating 1D window…
1/1             0s 285ms/step
  • Calculating 3D window…
1/1             0s 176ms/step
    Creating forecast comparison visualization…
```



Creating window comparison visualization…



Creating model performance visualization (1H window)…




```
========================================
     Processing 1M interval
========================================
   Insufficient data (17 points). Skipping.


============================================================
            FORECAST RESULTS
============================================================

  1D Interval

    1H Window:
     ARIMA 1D: $0.354889
     LSTM 1D:  $0.319411
     ARIMA 1M: $0.340756
     LSTM 1M:  $0.310317

    4H Window:
     ARIMA 1D: $0.354889
     LSTM 1D:  $0.298199
     ARIMA 1M: $0.340756
     LSTM 1M:  $0.300773

    12H Window:
```

```
   ARIMA 1D: $0.354889
   LSTM 1D:  $0.315553
   ARIMA 1M: $0.340756
   LSTM 1M:  $0.306094

  1D Window:
   ARIMA 1D: $0.354889
   LSTM 1D:  $0.302733
   ARIMA 1M: $0.340756
   LSTM 1M:  $0.300147

  3D Window:
   ARIMA 1D: $0.354889
   LSTM 1D:  $0.322311
   ARIMA 1M: $0.340756
   LSTM 1M:  $0.315746

1W Interval

  1H Window:
   ARIMA 1D: $0.353786
   LSTM 1D:  $0.252147
   ARIMA 1M: $0.335764
   LSTM 1M:  $0.336100

  4H Window:
   ARIMA 1D: $0.353786
   LSTM 1D:  $0.258260
   ARIMA 1M: $0.335764
   LSTM 1M:  $0.396813

  12H Window:
   ARIMA 1D: $0.353786
   LSTM 1D:  $0.218377
   ARIMA 1M: $0.335764
   LSTM 1M:  $0.368477

  1D Window:
   ARIMA 1D: $0.353786
   LSTM 1D:  $0.254474
   ARIMA 1M: $0.335764
   LSTM 1M:  $0.357300

  3D Window:
   ARIMA 1D: $0.353786
   LSTM 1D:  $0.312474
   ARIMA 1M: $0.335764
   LSTM 1M:  $0.323324
```

```
============================================================
         VISUALIZATIONS SAVED TO CURRENT DIRECTORY
============================================================

Forecasting complete! Professional charts saved as PNG files.
```

[6]:

1.2 ———