Bitcoin_NBEATS_Prediction

February 13, 2026

Bitcoin Price Prediction Using N-BEATS

Tsing Ouyang

Computational Finance and Risk Management (CFRM)

Department of Applied Mathematics

University of Washington

1 Paper Summary

1.1 Introduction

In the article "Bitcoin Price Prediction Using N-BEATS ML Technique," published in *EAI Endorsed Transactions on Scalable Information Systems*, authors G. Asmat and K. M. Maiyama demonstrate that the Neural Basis Expansion Analysis Time Series (N-BEATS) deep learning architecture effectively overcomes the challenges of cryptocurrency volatility to provide highly accurate price forecasts.

The paper utilizes 729-day hourly Bitcoin data to ensure the model captures both minute-by-minute fluctuations and long-term trends. Min-Max scaling and data reshaping are applied to normalize the raw data, making it easier for model process and avoiding disproportionate influence. The study enables the decomposition of time series into interpretable trend and seasonality components without requiring domain-specific feature engineering. Through a comparative analysis, the paper proves that N-BEATS delivers significantly lower error rates than Linear Regression and advanced long-short-term-memory (LSTM) networks. To sum up, N-BEATS is a superior, robust tool for financial forecasting that offers both precision and interpretability for high-volatile cryptocurrency markets.

1.2 Body

The researchers utilized a high-frequency data strategy to capture the inherent variability of the cryptocurrency market. By gathering 729 days of hourly Bitcoin price data from Yahoo Finance, the study ensured the model was trained on both rapid intraday fluctuations and broader long-term trends. The dataset incorporated six key features—opening, high, low, closing, and adjusted closing prices and trading volume, to provide a comprehensive view of market dynamics. This hourly interval was specifically selected because it allows for more accurate forecasting and real-time result incorporation compared to other time intervals. The granular multi-featured hourly data provides the necessary resolution to model the complex, stochastic and high-volatile nature of cryptocurrency.

The study utilized rigorous data preprocessing techniques to normalize raw financial information and prepare it for the N-BEATS architecture. The primary method involved applying Min-Max scaling to normalize all features into a range between 0 and 1, which prevents any single variable—such as trading volume—from exerting disproportionate influence over the model's learning process. Additionally, the researchers performed data reshaping to transform the sequences into a 3D structure, enabling the model to effectively process historical time steps and their associated features. This phase also included the creation of three-hour time steps, where the model analyzes data from the previous three hours to forecast the high and low prices of the subsequent hour. These systematic preprocessing steps ensure that the input data is structurally optimized and statistically balanced for stable neural network training.

The N-BEATS architecture enables the decomposition of time series into interpretable trend and seasonality components without requiring domain-specific feature engineering. The model achieves this by organizing fully connected layers into distinct blocks that perform "backcasting" to analyze historical data and "forecasting" to predict future values. This unique structure uses residual connections to bridge short-term shocks and long-term trends, allowing the network to identify complex temporal patterns autonomously. As a result, the model can break down a forecast into its underlying components, providing clear insights into market movements that are typically obscured in "black box" neural networks. This demonstrates that a modular, block-based deep learning design can provide both high predictive power and the interpretability necessary for financial decision-making.

The N-BEATS architecture demonstrates superior predictive accuracy against traditional econometric and alternative deep learning models for cryptocurrency forecasting. Researchers conducted a comparative test by training N-BEATS, Linear Regression, and Long Short-Term Memory (LSTM) networks on the same 729-day hourly dataset. The models were evaluated using Mean Absolute Error (MAE) and the R-squared (R^2) coefficient of determination, defined as the percentage of total variation described by the model. While Linear Regression and LSTM yielded R^2 scores of 0.9645 and 0.9263 respectively, N-BEATS achieved a nearly perfect R^2 of 0.9998 and a significantly lower MAE of 0.00240. This performance proves that the block-based decomposition of N-BEATS is more effective at capturing the nonlinear, stochastic dependencies of Bitcoin than standard regression or recurrent neural networks.

1.3 Conclusion

The research effectively demonstrates that the N-BEATS architecture bridges the gap between raw, volatile market data and actionable financial intelligence by leveraging a modular architecture that outperforms traditional benchmarks like LSTM and Linear Regression. By combining granular hourly data with automated feature decomposition, the study proves that deep learning can achieve a near-perfect correlation of $R^2 = 0.9998$ in even the most stochastic and nonlinear environments of cryptocurrency market.

1.4 Hypothesis

The study analyzes the predictive capabilities of the Neural Basis Expansion Analysis Time Series (N-BEATS) deep learning architecture compared to traditional and alternative machine learning models. The hypothesis is that N-BEATS can more effectively predict the cryptocurrency market with high uncertainty, nonlinearity and stochasticity over Linear Regression and LSTM networks. The model prediction accuracy has been analyzed. The model utilizes a three-hour look-back

window of historical market data, including the opening, high, low, closing, adjusted closing prices, and trading volume as inputs. The output is the predictive hourly high and low prices of Bitcoin for the subsequent hour. The outcome is that either the N-BEATS is superior in more precise and interpretable forecasts than the benchmark models or it will fail to overcome the Bitcoin data. The hypothesis is tested by dividing the data set into training set (80%) and testing set (20%), then measuring the Mean Absolute Error (MAE) and R-squared (R^2) scores over other models.

2 Literature Review

The research article "Bitcoin Price Prediction Using N-BEATS ML Technique" by Asmat and Maiyama (2025) is situated at the contemporary intersection of decentralized finance and advanced deep learning. It builds upon a research lineage that began with foundational work on Bitcoin and evolved through the application of traditional econometric models, such as ARIMA, which often struggled to capture the non-linear, high-frequency volatility of crypto-assets. As financial forecasting transitioned toward deep learning such as Long Short-Term Memory (LSTM) networks; however, the N-BEATS architecture is a superior alternative due to its unique ability to decompose time series into interpretable trend and seasonality components without manual feature engineering.

2.1 Key References

- Nakamoto, 2008: introduces the idea of a decentralised digital currency named Bitcoin. It is the foundation of cryptocurrency.
- Shumway and Stoffer, 2017: the application classical time series model (i.e. seasonal ARIMA) in finance to model complex nonlinear data like Bitcoin.
- Gamboa, 2017: the Recurrent Neural Networks (RNNs) as well as the LSTM networks are suitable for the analysis of time-dependent patterns. Also, it is effective for the financial market for capturing the temporal dependencies.
- Oreshkin et al., 2020: the original N-BEATS architecture to model stochastic dependencies and decompose time series into interpretable trend and seasonality components without manual feature engineering
- Maswood, 2020: comparing with traditional econometric approaches, machine learning is effectively precise for the uncertain and nonlinear dataset such as cryptocurrency trading

2.2 Dissenting Opinions

The paper only considers the accuracy in terms off MAE and R-squared, but computational overhead or complexity has not been discussed. Also, the precision and accuracy are determined in terms of magnitude only (MAR and R^2), direction of the market movement is not discussed.

2.3 Implementation Resources (Python packages)

- `Pandas`: data structural manipulation
- `NumPy`: reshaping the data
- `yfinance`: data acquisition for historial Bitcoin data
- `Scikit-Learn`: data processing, model selection by quantifying performance metrics, Train-Test sets model development
- `TensorFlow`, `Keras`: deep learning framework to construct N-BEATS architecture with layers

2.4 Similar Work

- Wang et al., 2025: evaluating deep-learning (TimeGPT) models' prediction accuracy, speeding and statistical significance over colatile crypto market.

3 Set up the environment

Import necessary libraries and define configuration parameters for running the Bitcoin price prediction code in tensorflow.

[18]:
```
import pandas as pd
import numpy as np
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

print("TensorFlow Version:", tf.__version__)

# ------------------------------------------------------------------------
# Configuration Parameters (from Paper)
# ------------------------------------------------------------------------
TICKER = 'BTC-USD'
DATA_PERIOD = "730d" # Fetch slightly more to ensure enough hourly data after␣
 ↪potential gaps
DATA_INTERVAL = "1h"
SEQUENCE_LENGTH = 3 # Input: Use previous 3 hours
FORECAST_HORIZON = 1 # Output: Predict next 1 hour
FEATURES = ['Open', 'High', 'Low', 'Close', 'Volume'] # Adjusted for yfinance␣
 ↪auto_adjust=True
TARGETS = ['High', 'Low']
TRAIN_SPLIT_RATIO = 0.8
```

```
TensorFlow Version: 2.20.0
```

[20]:
```
# N-BEATS Hyperparameters
N_BLOCKS = 3
N_NEURONS = 128 # Units per block
N_STACKS = 1 # Paper doesn't explicitly mention stacks, assuming 1 stack of␣
 ↪N_BLOCKS
N_LAYERS_PER_BLOCK = 4 # Common N-BEATS setting, not specified in paper
THETA_SIZE = N_NEURONS # Size of the expansion coefficients, can be tuned
BATCH_SIZE = 64
EPOCHS = 50
```

```
LEARNING_RATE = 0.001
```

4 Data Acquisition

This step focuses on acquiring and preparing the Bitcoin price data for modeling. It involves:

1. Downloading historical price data using yfinance for the past 730 days at an hourly interval.
2. Selecting relevant features: Open, High, Low, Close, and Volume.
3. Handling missing values (NaNs) by removing rows with NaNs in essential price columns and filling NaN volumes with 0.
4. Scaling the data to a range of 0-1 using MinMaxScaler.
5. Splitting the data into training and testing sets (80% train, 20% test).

[21]:
```
print("\n--- Data Acquisition ---")
# Use auto_adjust=True (default), which provides adjusted prices
# and removes 'Adj Close' column.
data_raw = yf.download(TICKER, period=DATA_PERIOD, interval=DATA_INTERVAL,␣
 ↪auto_adjust=True)

# --- DIAGNOSTIC PRINT 1 ---
print("\n--- After yf.download() ---")
print(f"Raw data shape: {data_raw.shape}")
print(f"Raw data columns: {data_raw.columns}")
print("Raw Data Head:\n", data_raw.head())
# --- END DIAGNOSTIC PRINT 1 ---

# Check if data is empty
if data_raw.empty:
    raise ValueError(f"Failed to download data for {TICKER}. DataFrame is empty.
 ↪")

# Select features (Volume might be NaN sometimes, handle later)
# Check if FEATURES exist before selecting
missing_cols = [col for col in FEATURES if col not in data_raw.columns]
if missing_cols:
     raise KeyError(f"Columns {missing_cols} not found in downloaded data.␣
 ↪Available columns: {data_raw.columns}")

data = data_raw[FEATURES].copy() # Use .copy() to avoid SettingWithCopyWarning␣
 ↪later

# --- DIAGNOSTIC PRINT 2 ---
print("\n--- After Selecting FEATURES ---")
print(f"Selected data shape: {data.shape}")
print(f"Selected data columns: {data.columns}")
print("Selected Data Head:\n", data.head())
# --- END DIAGNOSTIC PRINT 2 ---
```

```python
# --- Replace dropna with boolean indexing ---
essential_price_cols = ['Open', 'High', 'Low', 'Close']
# Check if essential columns exist before filtering
missing_essential = [col for col in essential_price_cols if col not in data.
 ↪columns]
if missing_essential:
    raise KeyError(f"Essential columns {missing_essential} are missing before␣
 ↪filtering NaNs. Data columns: {data.columns}")

print(f"\nShape before removing NaNs in essential columns: {data.shape}")
# Keep rows where ALL essential price columns are NOT NaN
data = data[data[essential_price_cols].notna().all(axis=1)]
print(f"Shape after removing NaNs in essential columns: {data.shape}")
# --- End Replace dropna ---


# Fill NaN volume with 0, common practice
if 'Volume' in data.columns:
    data['Volume'] = data['Volume'].fillna(0)
else:
    print("Warning: 'Volume' column not found after NaN filtering.")


print(f"\nFinal data shape for preprocessing: {data.shape}")
if data.empty:
    raise ValueError("DataFrame became empty after removing NaNs. Check␣
 ↪downloaded data quality.")

# --- End Revised Step 1 ---
```

```
--- Data Acquisition ---
[**********************100%**********************]  1 of 1 completed

--- After yf.download() ---
Raw data shape: (17363, 5)
Raw data columns: MultiIndex([( 'Close', 'BTC-USD'),
            (  'High', 'BTC-USD'),
            (   'Low', 'BTC-USD'),
            (  'Open', 'BTC-USD'),
            ('Volume', 'BTC-USD')],
           names=['Price', 'Ticker'])
Raw Data Head:
 Price                        Close          High           Low  \
Ticker                      BTC-USD       BTC-USD       BTC-USD
```

```
Datetime
2024-02-15 00:00:00+00:00  52360.242188  52467.960938  51829.496094
2024-02-15 01:00:00+00:00  51943.589844  52371.015625  51801.421875
2024-02-15 02:00:00+00:00  52122.542969  52233.503906  51959.035156
2024-02-15 03:00:00+00:00  52378.894531  52394.914062  52104.105469
2024-02-15 04:00:00+00:00  52203.054688  52385.132812  52193.335938

Price                             Open      Volume
Ticker                         BTC-USD     BTC-USD
Datetime
2024-02-15 00:00:00+00:00  51836.785156           0
2024-02-15 01:00:00+00:00  52371.015625  1201115136
2024-02-15 02:00:00+00:00  51959.035156   699662336
2024-02-15 03:00:00+00:00  52104.105469   529379328
2024-02-15 04:00:00+00:00  52384.472656   160198656

--- After Selecting FEATURES ---
Selected data shape: (17363, 5)
Selected data columns: MultiIndex([(  'Open', 'BTC-USD'),
            (  'High', 'BTC-USD'),
            (   'Low', 'BTC-USD'),
            ( 'Close', 'BTC-USD'),
            ('Volume', 'BTC-USD')],
           names=['Price', 'Ticker'])
Selected Data Head:
 Price                             Open          High           Low  \
Ticker                         BTC-USD       BTC-USD       BTC-USD
Datetime
2024-02-15 00:00:00+00:00  51836.785156  52467.960938  51829.496094
2024-02-15 01:00:00+00:00  52371.015625  52371.015625  51801.421875
2024-02-15 02:00:00+00:00  51959.035156  52233.503906  51959.035156
2024-02-15 03:00:00+00:00  52104.105469  52394.914062  52104.105469
2024-02-15 04:00:00+00:00  52384.472656  52385.132812  52193.335938

Price                            Close      Volume
Ticker                         BTC-USD     BTC-USD
Datetime
2024-02-15 00:00:00+00:00  52360.242188           0
2024-02-15 01:00:00+00:00  51943.589844  1201115136
2024-02-15 02:00:00+00:00  52122.542969   699662336
2024-02-15 03:00:00+00:00  52378.894531   529379328
2024-02-15 04:00:00+00:00  52203.054688   160198656

Shape before removing NaNs in essential columns: (17363, 5)
Shape after removing NaNs in essential columns: (17363, 5)

Final data shape for preprocessing: (17363, 5)
```

4.1 Extensive Data Acquisition

The replicating paper uses the most current bitcoin data: **from February 11th, 2024 to February 9th, 2026** in UTC

[16]:
```
data
```

[16]:
```
Price                           Open          High           Low  \
Ticker                       BTC-USD       BTC-USD       BTC-USD
Datetime
2024-02-15 00:00:00+00:00  51836.785156  52467.960938  51829.496094
2024-02-15 01:00:00+00:00  52371.015625  52371.015625  51801.421875
2024-02-15 02:00:00+00:00  51959.035156  52233.503906  51959.035156
2024-02-15 03:00:00+00:00  52104.105469  52394.914062  52104.105469
2024-02-15 04:00:00+00:00  52384.472656  52385.132812  52193.335938
…                                   …             …             …
2026-02-13 18:00:00+00:00  68998.921875  69346.125000  68880.742188
2026-02-13 19:00:00+00:00  69100.257812  69190.609375  68925.750000
2026-02-13 20:00:00+00:00  68967.867188  68967.867188  68649.242188
2026-02-13 21:00:00+00:00  68738.234375  69012.054688  68738.234375
2026-02-13 22:00:00+00:00  68850.875000  68943.343750  68708.734375

Price                            Close      Volume
Ticker                        BTC-USD     BTC-USD
Datetime
2024-02-15 00:00:00+00:00  52360.242188           0
2024-02-15 01:00:00+00:00  51943.589844  1201115136
2024-02-15 02:00:00+00:00  52122.542969   699662336
2024-02-15 03:00:00+00:00  52378.894531   529379328
2024-02-15 04:00:00+00:00  52203.054688   160198656
…                                   …           …
2026-02-13 18:00:00+00:00  69091.992188           0
2026-02-13 19:00:00+00:00  68956.765625  2093064192
2026-02-13 20:00:00+00:00  68698.007812  1003900928
2026-02-13 21:00:00+00:00  68800.843750  1164029952
2026-02-13 22:00:00+00:00  68909.062500  1496993792

[17363 rows x 5 columns]
```

5 Data Preprocessing

This step prepares the data for the N-BEATS model by creating input sequences and corresponding target values. This involves:

1. Defining a function to create sequences of historical data points for model input.

2. Applying this function to the scaled training and testing data to generate input sequences and target values for High and Low prices.

[5]:
```
print("\n--- Data Preprocessing ---")

# Scaling - Fit only on training data
feature_scaler = MinMaxScaler()
target_scaler = MinMaxScaler()

# Split data chronologically BEFORE scaling
split_index = int(TRAIN_SPLIT_RATIO * len(data))
train_data = data.iloc[:split_index]
test_data = data.iloc[split_index:]

print(f"Training data shape: {train_data.shape}")
print(f"Testing data shape: {test_data.shape}")

# Fit scalers ONLY on training data
scaled_train_features = feature_scaler.fit_transform(train_data[FEATURES])
# Fit target scaler on High/Low columns of training data
target_scaler.fit(train_data[TARGETS])

# Transform train and test data using fitted scalers
scaled_test_features = feature_scaler.transform(test_data[FEATURES])

# Combine scaled features into DataFrames for easier indexing
scaled_train_df = pd.DataFrame(scaled_train_features, columns=FEATURES,␣
 ↪index=train_data.index)
scaled_test_df = pd.DataFrame(scaled_test_features, columns=FEATURES,␣
 ↪index=test_data.index)

# Also scale the target columns separately for sequence creation
scaled_train_targets_df = pd.DataFrame(target_scaler.
 ↪transform(train_data[TARGETS]), columns=TARGETS, index=train_data.index)
scaled_test_targets_df = pd.DataFrame(target_scaler.
 ↪transform(test_data[TARGETS]), columns=TARGETS, index=test_data.index)


# Sequence Creation Function [cite: 132]
def create_sequences(input_features_df, input_targets_df, sequence_length,␣
 ↪forecast_horizon):
    X, y = [], []
    indices = []
    # Ensure target indices align with the end of the input sequence
    for i in range(len(input_features_df) - sequence_length - forecast_horizon␣
 ↪+ 1):
```

```
        feature_sequence = input_features_df.iloc[i:(i + sequence_length)].
 ↪values
        target_values = input_targets_df.iloc[i + sequence_length : i +␣
 ↪sequence_length + forecast_horizon].values

        # Ensure the target shape is (forecast_horizon, num_targets) -> (1, 2)
        if target_values.shape == (forecast_horizon, len(TARGETS)):
            X.append(feature_sequence)
            y.append(target_values.flatten()) # Flatten to (2,) for Dense layer␣
 ↪output
            indices.append(input_targets_df.index[i + sequence_length]) # Store␣
 ↪index of the target time step
        # else: # Debugging shape issues
            # print(f"Skipping index {i} due to shape mismatch: Target shape␣
 ↪{target_values.shape}")

    return np.array(X), np.array(y), indices

# Create sequences for training and testing sets
X_train, y_train, train_indices = create_sequences(scaled_train_df,␣
 ↪scaled_train_targets_df, SEQUENCE_LENGTH, FORECAST_HORIZON)
X_test, y_test, test_indices = create_sequences(scaled_test_df,␣
 ↪scaled_test_targets_df, SEQUENCE_LENGTH, FORECAST_HORIZON)

print(f"X_train shape: {X_train.shape}") # Should be (num_samples, 3,␣
 ↪num_features)
print(f"y_train shape: {y_train.shape}") # Should be (num_samples, 2)
print(f"X_test shape: {X_test.shape}")
print(f"y_test shape: {y_test.shape}")

# Verify shapes
num_features = len(FEATURES)
num_targets = len(TARGETS)
if X_train.shape[1:] != (SEQUENCE_LENGTH, num_features):
    raise ValueError(f"X_train shape mismatch: expected (None,␣
 ↪{SEQUENCE_LENGTH}, {num_features}), got {X_train.shape}")
if y_train.shape[1:] != (num_targets,):
    raise ValueError(f"y_train shape mismatch: expected (None, {num_targets}),␣
 ↪got {y_train.shape}")
```

```
--- Data Preprocessing ---
Training data shape: (13890, 5)
Testing data shape: (3473, 5)
X_train shape: (13887, 3, 5)
y_train shape: (13887, 2)
X_test shape: (3470, 3, 5)
```

```
y_test shape: (3470, 2)
```

6 Key Techniques (N-BEATS)

The Neural Basis Expansion Analysis for Time Series (N-BEATS) is a deep learning architecture specifically engineered for time series forecasting without the need for domain-specific feature engineering. It is characterized by a hierarchical structure of blocks and stacks that utilize fully connected layers and residual connections to decompose data into interpretable signals like trend and seasonality.

6.1 Mathematical Formulation

6.1.1 Block Operation

For each block b with input x_b, the model computes a set of expansion coefficients (θ) using fully connected layers:

$$\theta_b^f = FC_{b,f}(x_b) \quad \text{and} \quad \theta_b^b = FC_{b,b}(x_b)$$

Where FC represents the fully connected layers.

6.1.2 Basis Expansion

The forecast ($\hat{y}_b$) and backcast ($\hat{x}_b$) are then generated by projecting these coefficients onto basis functions (g):

$$\hat{y}_b = \sum_{i=1}^{\dim(\theta_b^f)} \theta_{b,i}^f g_i^f \quad \text{and} \quad \hat{x}_b = \sum_{i=1}^{\dim(\theta_b^b)} \theta_{b,i}^b g_i^b$$

The basis functions g can be generic (like a polynomial for trend) or learned through the network.

6.1.3 Residual Hierarchy (Backcasting)

The input for the next block (x_{b+1}) is the residual of the previous backcast:

$$x_{b+1} = x_b - \hat{x}_b$$

This enables the "decomposition" mentioned in the paper, as each block subtracts what it has already "explained" from the input.

6.1.4 Global Forecast

The final prediction ($\hat{y}$) is the sum of all individual block forecasts across the entire stack:

$$\hat{y} = \sum_b \hat{y}_b$$

This summation allows the model to combine short-term oscillations and long-term trends into a single, highly accurate Bitcoin price prediction.

7 N-BEATS Model Definition (TensorFlow/Keras)

This step defines the N-BEATS model architecture using TensorFlow/Keras. It involves:

1. Creating a custom NBeatsBlock layer, the fundamental building block of the N-BEATS architecture.
2. Building the complete N-BEATS model by stacking NBeatsBlock layers and incorporating residual connections.
3. Compiling the model with the Mean Absolute Error (MAE) loss function and the Adam optimizer.

[6]:
```
print("\n--- N-BEATS Model Definition ---")

# N-BEATS Block Layer Implementation (Same as before)
class NBeatsBlock(layers.Layer):
    def __init__(self, input_size, theta_size, horizon, n_neurons, n_layers,␣
 ↪**kwargs):
        super().__init__(**kwargs)
        self.horizon = horizon
        self.theta_size = theta_size
        # Fully connected layers (stack)
        self.hidden = [layers.Dense(n_neurons, activation='relu') for _ in␣
 ↪range(n_layers)]
        # Output layers for basis coefficients (thetas)
        self.theta_f = layers.Dense(theta_size, activation='linear',␣
 ↪name='theta_f') # Forecast
        self.theta_b = layers.Dense(theta_size, activation='linear',␣
 ↪name='theta_b') # Backcast

    def call(self, inputs):
        x = inputs
        for layer in self.hidden:
            x = layer(x)
        theta_f = self.theta_f(x)
        theta_b = self.theta_b(x)
        # Simplified basis expansion - direct use of thetas
        # Final projection handled in the main model build
        return theta_b, theta_f

# Build the full N-BEATS model with residual connections (Revised forecast_sum␣
 ↪handling)
def create_nbeats_model(seq_len, horizon, num_features, num_targets, n_blocks,␣
 ↪n_neurons, n_layers_per_block, theta_size):
```

```
    input_layer = layers.Input(shape=(seq_len, num_features),␣
↪name='model_input')
    # Flatten the input sequence for Dense layers
    x_flat = layers.Flatten()(input_layer)

    residuals = x_flat
    forecast_sum = None # Initialize as None

    for i in range(n_blocks):
        block = NBeatsBlock(input_size=seq_len * num_features, # Block operates␣
↪on flattened input
                            theta_size=theta_size,
                            horizon=horizon,
                            n_neurons=n_neurons,
                            n_layers=n_layers_per_block,
                            name=f'nbeats_block_{i}')
        # Get backcast and forecast thetas from the block
        theta_b, theta_f = block(residuals)

        # Simplified backcast/forecast generation
        backcast_layer = layers.Dense(seq_len * num_features,␣
↪name=f'backcast_{i}')
        backcast = backcast_layer(theta_b)

        forecast_layer = layers.Dense(horizon * num_targets,␣
↪name=f'forecast_{i}')
        forecast = forecast_layer(theta_f) # Shape (batch_size, horizon *␣
↪num_targets)

        # Double Residual Connection: Subtract backcast from input to block
        residuals = layers.subtract([residuals, backcast], name=f'subtract_{i}')

        # Add the block's forecast to the total forecast
        if forecast_sum is None:
            # Initialize with the first block's forecast
            forecast_sum = forecast
        else:
            # Add subsequent forecasts
            forecast_sum = layers.add([forecast_sum, forecast], name=f'add_{i}')

    # Ensure final output shape is (batch_size, num_targets) for horizon=1
    if horizon == 1:
      # If horizon is 1, forecast_sum already has shape (batch_size,␣
↪num_targets)
      final_forecast = forecast_sum
    else:
```

```
    # If horizon > 1, reshape might be needed
    final_forecast = layers.Reshape((horizon, num_targets),␣
↪name='final_forecast')(forecast_sum)
    # If you need only the first step of a multi-step forecast:
    # final_forecast = layers.Lambda(lambda x: x[:, 0, :],␣
↪name='first_step_forecast')(final_forecast)


  model = tf.keras.Model(inputs=input_layer, outputs=final_forecast,␣
↪name='NBEATS_model')
  return model

# Instantiate the model
model = create_nbeats_model(
    seq_len=SEQUENCE_LENGTH,
    horizon=FORECAST_HORIZON,
    num_features=num_features,
    num_targets=num_targets,
    n_blocks=N_BLOCKS,
    n_neurons=N_NEURONS,
    n_layers_per_block=N_LAYERS_PER_BLOCK,
    theta_size=THETA_SIZE
)

# Compile the model
model.compile(loss='mae', # Mean Absolute Error
              optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE),
              metrics=['mae'])

model.summary()
```

```
--- N-BEATS Model Definition ---
WARNING:tensorflow:From
C:\Users\Tsing\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\backend\tensorflow\core.py:232: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

Model: "NBEATS_model"


 Layer (type)                Output Shape              Param #  ␣
↪Connected to

 model_input (InputLayer)    (None, 3, 5)              0        -␣
↪
```

```
flatten (Flatten)               (None, 15)                        0  ␣
↪model_input[0][0]

nbeats_block_0 (NBeatsBlock)    [(None, 128), (None,              84,608  ␣
↪flatten[0][0]
                                128)]                                 ␣
↪

backcast_0 (Dense)              (None, 15)                        1,935  ␣
↪nbeats_block_0[0][0]

subtract_0 (Subtract)           (None, 15)                        0  ␣
↪flatten[0][0],
                                                                    ␣
↪backcast_0[0][0]

nbeats_block_1 (NBeatsBlock)    [(None, 128), (None,              84,608  ␣
↪subtract_0[0][0]
                                128)]                                 ␣
↪

backcast_1 (Dense)              (None, 15)                        1,935  ␣
↪nbeats_block_1[0][0]

subtract_1 (Subtract)           (None, 15)                        0  ␣
↪subtract_0[0][0],
                                                                    ␣
↪backcast_1[0][0]

forecast_0 (Dense)              (None, 2)                           258  ␣
↪nbeats_block_0[0][1]

forecast_1 (Dense)              (None, 2)                           258  ␣
↪nbeats_block_1[0][1]

nbeats_block_2 (NBeatsBlock)    [(None, 128), (None,              84,608  ␣
↪subtract_1[0][0]
                                128)]                                 ␣
↪

add_1 (Add)                     (None, 2)                         0  ␣
↪forecast_0[0][0],
                                                                    ␣
↪forecast_1[0][0]
```

```
forecast_2 (Dense)              (None, 2)                      258  ␣
↪nbeats_block_2[0][1]

add_2 (Add)                     (None, 2)                        0  ␣
↪add_1[0][0],

                                                                   ␣
↪forecast_2[0][0]


Total params: 258,468 (1009.64 KB)

Trainable params: 258,468 (1009.64 KB)

Non-trainable params: 0 (0.00 B)
```

8 Model Training

This step trains the defined N-BEATS model using the preprocessed training data. It involves:

1. Utilizing early stopping to prevent overfitting and ensure optimal training duration.
2. Implementing learning rate reduction to fine-tune the learning process and improve convergence.
3. Monitoring the training and validation loss to assess model performance.

9 Overfitting Detection

The error curve (MAE) of the model over the iterations (epoch) is overlapping well at the righ-hand side, this indicates that model fits both the training data and the validation (test) data, and it generalizes well. Therefore, there is very low probability that the model is overfitting.

[7]:
```
print("\n--- Model Training and Overfitting Detection ---")

# Define callbacks
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                  patience=10, # Stop if no␣
 ↪improvement for 10 epochs
                                                  restore_best_weights=True)

reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
                                                 factor=0.2, # Reduce LR by␣
 ↪factor of 5
                                                 patience=5,
                                                 min_lr=LEARNING_RATE / 100)
```

```python
# Train the model
history = model.fit(
    X_train, y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_split=0.1, # Use last 10% of training data for validation during␣
 ↪training
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)

# Plot training history
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss History')
plt.xlabel('Epoch')
plt.ylabel('MAE Loss')
plt.legend()
plt.show()
```

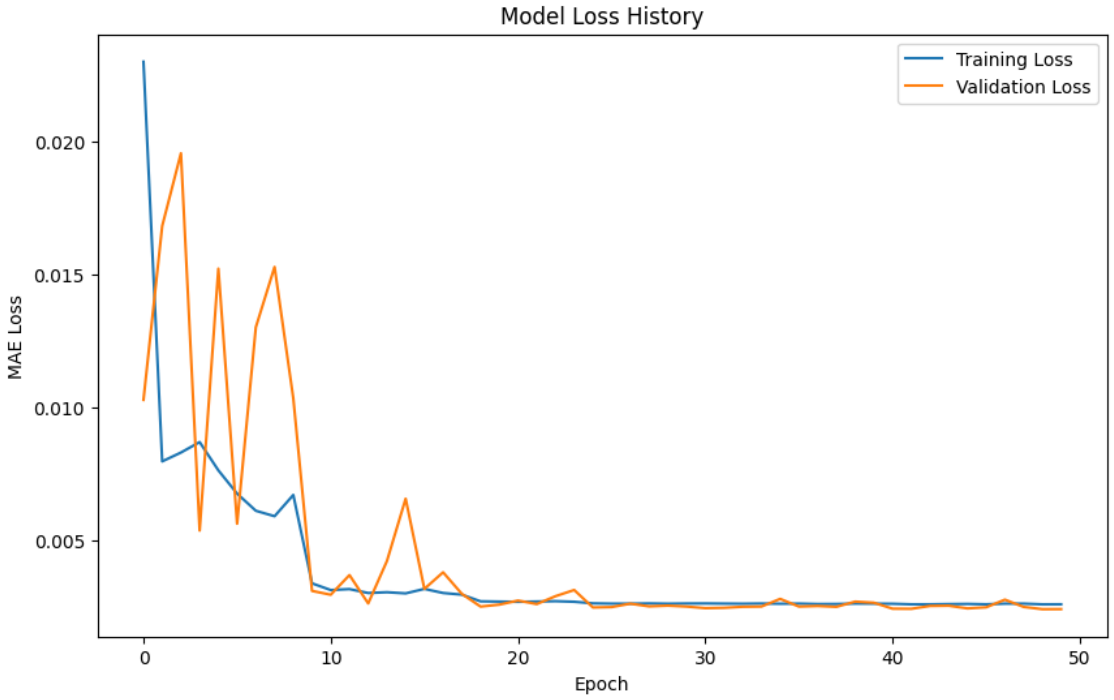
```
--- Model Training and Overfitting Detection ---
Epoch 1/50
C:\Users\Tsing\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\optimizers\base_optimizer.py:870: UserWarning: Gradients do
not exist for variables ['nbeats_block_2/theta_b/kernel',
'nbeats_block_2/theta_b/bias'] when minimizing the loss. If using
`model.compile()`, did you forget to provide a `loss` argument?
  warnings.warn(
196/196            5s 4ms/step -
loss: 0.0230 - mae: 0.0230 - val_loss: 0.0103 - val_mae: 0.0103 - learning_rate:
0.0010
Epoch 2/50
196/196            1s 3ms/step -
loss: 0.0080 - mae: 0.0080 - val_loss: 0.0168 - val_mae: 0.0168 - learning_rate:
0.0010
Epoch 3/50
196/196            1s 3ms/step -
loss: 0.0083 - mae: 0.0083 - val_loss: 0.0196 - val_mae: 0.0196 - learning_rate:
0.0010
Epoch 4/50
196/196            1s 4ms/step -
loss: 0.0087 - mae: 0.0087 - val_loss: 0.0053 - val_mae: 0.0053 - learning_rate:
0.0010
Epoch 5/50
196/196            1s 3ms/step -
```

```
loss: 0.0076 - mae: 0.0076 - val_loss: 0.0152 - val_mae: 0.0152 - learning_rate:
0.0010
Epoch 6/50
196/196             1s 3ms/step -
loss: 0.0067 - mae: 0.0067 - val_loss: 0.0056 - val_mae: 0.0056 - learning_rate:
0.0010
Epoch 7/50
196/196             1s 3ms/step -
loss: 0.0061 - mae: 0.0061 - val_loss: 0.0130 - val_mae: 0.0130 - learning_rate:
0.0010
Epoch 8/50
196/196             1s 3ms/step -
loss: 0.0059 - mae: 0.0059 - val_loss: 0.0153 - val_mae: 0.0153 - learning_rate:
0.0010
Epoch 9/50
196/196             1s 4ms/step -
loss: 0.0067 - mae: 0.0067 - val_loss: 0.0103 - val_mae: 0.0103 - learning_rate:
0.0010
Epoch 10/50
196/196             1s 3ms/step -
loss: 0.0034 - mae: 0.0034 - val_loss: 0.0031 - val_mae: 0.0031 - learning_rate:
2.0000e-04
Epoch 11/50
196/196             1s 3ms/step -
loss: 0.0031 - mae: 0.0031 - val_loss: 0.0029 - val_mae: 0.0029 - learning_rate:
2.0000e-04
Epoch 12/50
196/196             1s 3ms/step -
loss: 0.0031 - mae: 0.0031 - val_loss: 0.0037 - val_mae: 0.0037 - learning_rate:
2.0000e-04
Epoch 13/50
196/196             1s 3ms/step -
loss: 0.0030 - mae: 0.0030 - val_loss: 0.0026 - val_mae: 0.0026 - learning_rate:
2.0000e-04
Epoch 14/50
196/196             1s 3ms/step -
loss: 0.0030 - mae: 0.0030 - val_loss: 0.0042 - val_mae: 0.0042 - learning_rate:
2.0000e-04
Epoch 15/50
196/196             1s 3ms/step -
loss: 0.0030 - mae: 0.0030 - val_loss: 0.0066 - val_mae: 0.0066 - learning_rate:
2.0000e-04
Epoch 16/50
196/196             1s 3ms/step -
loss: 0.0031 - mae: 0.0031 - val_loss: 0.0031 - val_mae: 0.0031 - learning_rate:
2.0000e-04
Epoch 17/50
196/196             1s 3ms/step -
```

```
loss: 0.0030 - mae: 0.0030 - val_loss: 0.0038 - val_mae: 0.0038 - learning_rate:
2.0000e-04
Epoch 18/50
196/196             1s 3ms/step -
loss: 0.0029 - mae: 0.0029 - val_loss: 0.0030 - val_mae: 0.0030 - learning_rate:
2.0000e-04
Epoch 19/50
196/196             1s 3ms/step -
loss: 0.0027 - mae: 0.0027 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
4.0000e-05
Epoch 20/50
196/196             1s 3ms/step -
loss: 0.0027 - mae: 0.0027 - val_loss: 0.0026 - val_mae: 0.0026 - learning_rate:
4.0000e-05
Epoch 21/50
196/196             1s 3ms/step -
loss: 0.0027 - mae: 0.0027 - val_loss: 0.0027 - val_mae: 0.0027 - learning_rate:
4.0000e-05
Epoch 22/50
196/196             1s 3ms/step -
loss: 0.0027 - mae: 0.0027 - val_loss: 0.0026 - val_mae: 0.0026 - learning_rate:
4.0000e-05
Epoch 23/50
196/196             1s 3ms/step -
loss: 0.0027 - mae: 0.0027 - val_loss: 0.0029 - val_mae: 0.0029 - learning_rate:
4.0000e-05
Epoch 24/50
196/196             1s 3ms/step -
loss: 0.0027 - mae: 0.0027 - val_loss: 0.0031 - val_mae: 0.0031 - learning_rate:
4.0000e-05
Epoch 25/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0024 - val_mae: 0.0024 - learning_rate:
1.0000e-05
Epoch 26/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 27/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0026 - val_mae: 0.0026 - learning_rate:
1.0000e-05
Epoch 28/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 29/50
196/196             1s 3ms/step -
```

```
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 30/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 31/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0024 - val_mae: 0.0024 - learning_rate:
1.0000e-05
Epoch 32/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0024 - val_mae: 0.0024 - learning_rate:
1.0000e-05
Epoch 33/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 34/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 35/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0028 - val_mae: 0.0028 - learning_rate:
1.0000e-05
Epoch 36/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 37/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 38/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 39/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0027 - val_mae: 0.0027 - learning_rate:
1.0000e-05
Epoch 40/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0026 - val_mae: 0.0026 - learning_rate:
1.0000e-05
Epoch 41/50
196/196             1s 3ms/step -
```

```
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0024 - val_mae: 0.0024 - learning_rate:
1.0000e-05
Epoch 42/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0024 - val_mae: 0.0024 - learning_rate:
1.0000e-05
Epoch 43/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 44/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 45/50
196/196             1s 2ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0024 - val_mae: 0.0024 - learning_rate:
1.0000e-05
Epoch 46/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 47/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0027 - val_mae: 0.0027 - learning_rate:
1.0000e-05
Epoch 48/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0025 - val_mae: 0.0025 - learning_rate:
1.0000e-05
Epoch 49/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0024 - val_mae: 0.0024 - learning_rate:
1.0000e-05
Epoch 50/50
196/196             1s 3ms/step -
loss: 0.0026 - mae: 0.0026 - val_loss: 0.0024 - val_mae: 0.0024 - learning_rate:
1.0000e-05
```



10 Statistical Performance Results

10.1 R-squared (R^2)

It is indicating the percentage of variation in a dependent variable that is explained by an independent variable(s) in a regression model. It's Ranging from 0 to 1 (0% to 100%), it represents how well a model fits the data, with 1 indicating a perfect fit. ## Mean Absolute Error (MAE) It is a foundational regression model metric that measures the average magnitude of errors between predicted and actual values.It's calculated as the average of the absolute differences:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Lower MAE scores (0 is ideal) indicate better performance.

11 Evaluation

This step evaluates the trained N-BEATS model on the test data to assess its predictive accuracy. It involves:

1. Making predictions on the test set using the trained model.
2. Calculating the Mean Absolute Error (MAE) and R-squared (R2) scores to quantify the model's performance.
3. Comparing the results with the reported values from the research paper for benchmarking.

[8]:
```
print("\n--- Evaluation ---")

# Make predictions on the test set
predictions_scaled = model.predict(X_test)

# Inverse transform predictions and actual values
# Predictions shape: (num_samples, num_targets)
# y_test shape: (num_samples, num_targets)
predictions_original = target_scaler.inverse_transform(predictions_scaled)
y_test_original = target_scaler.inverse_transform(y_test)

# Calculate metrics on the original scale
mae = mean_absolute_error(y_test_original, predictions_original)
r2 = r2_score(y_test_original, predictions_original)

print(f"\nTest Set Evaluation (Original Scale):")
print(f"Mean Absolute Error (MAE): {mae:.6f}")
print(f"R-squared (R2 Score):      {r2:.6f}")

# Comparison with paper's results (Note potential ambiguity/typo in paper's␣
 ↪reported values)
# Abstract: MAE 0.9998, R2 0.00240 [cite: 7]
# Results : MAE 0.00240, R2 0.9998 [cite: 156, 158]
# Assuming results section values are more likely intended for scaled data
print("\nComparison with Paper (Results Section - likely scaled values):")
print(f"Paper MAE (Scaled?): 0.00240")
print(f"Paper R2  (Scaled?): 0.9998")
print("\nComparison with Paper (Abstract - likely scaled values):")
print(f"Paper MAE (Scaled?): 0.9998")
print(f"Paper R2  (Scaled?): 0.00240")
print("\nNote: The MAE/R2 calculated here are on the *original price scale*,")
print("while the paper's values might be on the *scaled* data (0-1 range).")
print("Direct comparison might be misleading without knowing the paper's exact␣
 ↪evaluation scale.")
```

```
--- Evaluation ---
109/109             0s 2ms/step

Test Set Evaluation (Original Scale):
Mean Absolute Error (MAE): 218.476475
R-squared (R2 Score):      0.999035

Comparison with Paper (Results Section - likely scaled values):
Paper MAE (Scaled?): 0.00240
Paper R2  (Scaled?): 0.9998

Comparison with Paper (Abstract - likely scaled values):
```

```
Paper MAE (Scaled?): 0.9998
Paper R2  (Scaled?): 0.00240

Note: The MAE/R2 calculated here are on the *original price scale*,
while the paper's values might be on the *scaled* data (0-1 range).
Direct comparison might be misleading without knowing the paper's exact
evaluation scale.
```

12 Visualization

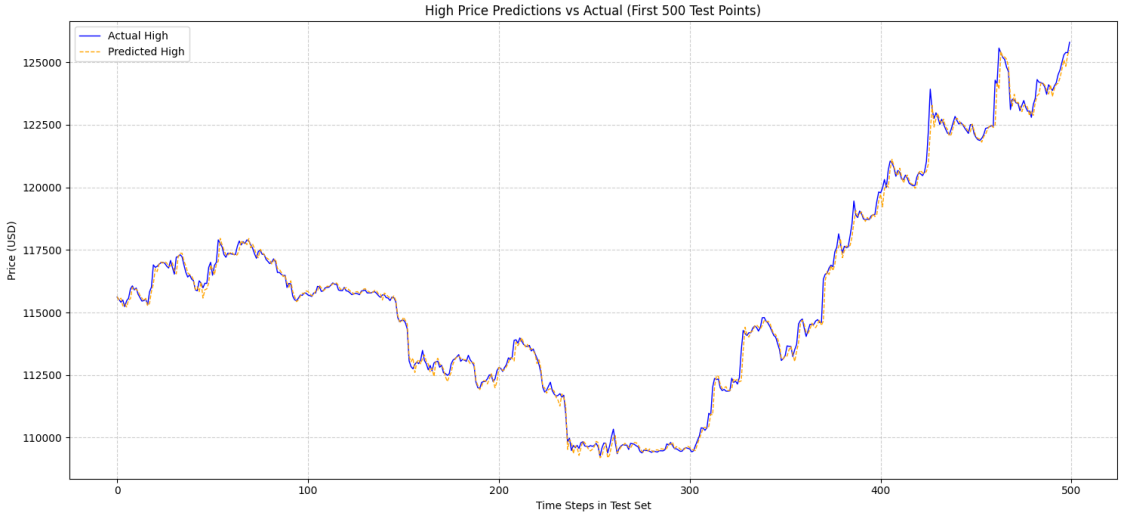
This step visualizes the model's predictions against the actual Bitcoin prices to gain further insights into its performance. It involves:

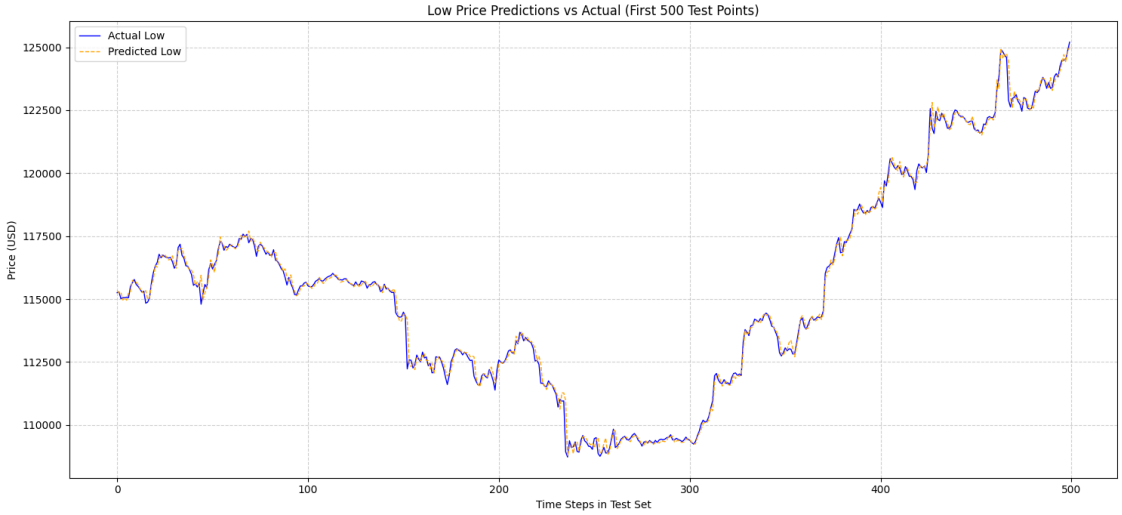
1. Plotting the predicted and actual High and Low prices over time.
2. Analyzing the visual alignment of predictions with actual price movements.

[14]:
```python
print("\n--- Visualization ---")

# Use the test_indices to align predictions with the original dates if needed
# prediction_dates = test_indices[:len(predictions_original)] # Get␣
 ↪corresponding dates

# Plot actual vs predicted High/Low prices for a subset of the test data
plot_points = 500
high_col_index = TARGETS.index('High')
low_col_index = TARGETS.index('Low')

plt.figure(figsize=(15, 7))
plt.plot(y_test_original[:plot_points, high_col_index], label='Actual High',␣
 ↪color='blue', linewidth=1)
plt.plot(predictions_original[:plot_points, high_col_index], label='Predicted␣
 ↪High', color='orange', linestyle='--', linewidth=1)
plt.title(f'High Price Predictions vs Actual (First {plot_points} Test Points)')
plt.xlabel('Time Steps in Test Set')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()

plt.figure(figsize=(15, 7))
plt.plot(y_test_original[:plot_points, low_col_index], label='Actual Low',␣
 ↪color='blue', linewidth=1)
plt.plot(predictions_original[:plot_points, low_col_index], label='Predicted␣
 ↪Low', color='orange', linestyle='--', linewidth=1)
plt.title(f'Low Price Predictions vs Actual (First {plot_points} Test Points)')
plt.xlabel('Time Steps in Test Set')
```

```
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
--- Visualization ---
```





13 Future Research

13.1 Multi-Dimensional Evaluation Framework

The original paper's reliance on Mean Absolute Error (MAE) and R^2 offers a limited view of model performance that may be misleading in financial contexts. In non-stationary time series like Bitcoin, a high R^2 often indicates that the model is simply tracking the current price trend (persistence) rather than identifying future price reversals. To ensure the model is truly capturing predictive signals, future research should adopt a more comprehensive evaluation suite:

- **Advanced Error Metrics**: Future studies should include Root Mean Squared Error (RMSE) to specifically penalize large, high-risk outliers and Mean Absolute Percentage Error (MAPE) to facilitate scale-independent comparisons across different price levels.
- **Statistical Significance**: The Diebold–Mariano test should be employed to statistically validate whether the predictive superiority of N-BEATS over benchmarks like LSTM is significant or merely a result of random luck within the specific 729-day sample.
- **Economic Value**: Beyond statistical fit, the Sharpe Ratio must be used to evaluate the model's risk-adjusted return potential. A model with low MAE is only useful in practice if its predictions can be translated into a profitable trading strategy that justifies the inherent market risk.

13.2 Cross-Asset Generalization

While Bitcoin is the primary representative of crypto market uncertainty, its high liquidity makes its price behavior distinct from the thousands of smaller-cap "altcoins". Generalizing the N-BEATS architecture across a wider array of assets presents several opportunities: - **Altcoin Dynamics**: Applying the model to low-liquidity tokens would test its ability to handle extreme "flash crash" patterns and irregular volatility clusters that are less common in the more "mature" Bitcoin market. - **Transfer Learning**: Research could investigate a transfer learning approach, where a "foundation" N-BEATS model is pre-trained on high-volume assets (Bitcoin or traditional stocks) and then fine-tuned on smaller tokens to overcome the challenges of limited historical data. - **Spillover Analysis**: The model could be extended to study volatility spillovers, analyzing how price fluctuations in Bitcoin serve as a leading indicator for movements in the broader digital asset ecosystem.

14 References

Asmat, G., and K. M. Maiyama. "Bitcoin Price Prediction Using N-Beats ML Technique." EAI Endorsed Transactions on Scalable Information Systems, April 1, 2025. https://doi.org/10.4108/eetsis.9006.

Maswood, Mirza Mohd, and Abdullah G. Alharbi. "Deep Learning-Based Stock Price Prediction Using LSTM and Bi-Directional LSTM Model." 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), October 24, 2020, 87–92. https://doi.org/10.1109/niles50944.2020.9257950.

Nakamoto, Satoshi. Bitcoin: A peer-to-peer electronic cash system, 2008. https://bitcoin.org/bitcoin.pdf.

Oreshkin, Boris N., Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. "N-Beats: Neural

Basis Expansion Analysis for Interpretable Time Series Forecasting." arXiv.org, February 20, 2020. https://arxiv.org/abs/1905.10437.

Shumway, Robert H., and David S. Stoffer. "ARIMA Models. In: Time Series Analysis and Its Applications." SpringerLink, January 1, 1970.

Wang, Minxing, Pavel Braslavski, and Dmitry I. Ignatov. "TimeGPT's Potential in Cryptocurrency Forecasting: Efficiency, Accuracy, and Economic Value." Forecasting 7, no. 3 (September 10, 2025): 48. https://doi.org/10.3390/forecast7030048.