
**NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS**

**SCHOOL OF SCIENCES**
**DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS**

**PROGRAM OF POSTGRADUATE STUDIES**
**Data Science and Information Technologies**
**SPECIALIZATION**
**Machine Learning and Artificial Intelligence**


**MSc THESIS**

# Real-Time Cryptocurrency Price Forecasting with Blockchain Transaction Graphs and Neural Networks


**Georgios P. Gkrekas**

**Supervisors: Aikaterini Doka**, Professor NKUA



**ATHENS**

**MARCH 2024**


**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**
**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ**
**Επιστήμη Δεδομένων και Τεχνολογίες Πληροφορίας**
**ΕΙΔΙΚΕΥΣΗ**
**Μεγάλα Δεδομένα και Τεχνητή Νοημοσύνη**


**ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

# Πρόβλεψη Τιμών Κρυπτονομισμάτων σε Πραγματικό Χρόνο με Γράφους Συναλλαγών Blockchain και Νευρωνικά Δίκτυα


**Γεώργιος Π. Γκρέκας**

**Επιβλέποντες: Αικατερίνη Δόκα**, Καθηγητής ΕΚΠΑ



**ΑΘΗΝΑ**

**ΜΑΡΤΙΟΣ 2024**

**MSc THESIS**

# Real-Time Cryptocurrency Price Forecasting with Blockchain Transaction Graphs and Neural Networks

**Georgios P. Gkrekas**
**S.N.:** DS2200005

**SUPERVISORS: Aikaterini Doka**, Professor NKUA

**EXAMINATION COMMITTEE: Aikaterini Doka**, Professor NKUA
**Mema Rousopoulou**, Professor NKUA
**Ntoulas Alexandros**, Professor NKUA

**Examination Date: 4 March, 2025**

**ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

# Πρόβλεψη Τιμών Κρυπτονομισμάτων σε Πραγματικό Χρόνο με Γράφους Συναλλαγών Blockchain και Νευρωνικά Δίκτυα

**Γεώργιος Π. Γκρέκας**
**Α.Μ.:** DS2200005

**ΕΠΙΒΛΕΠΟΝΤΕΣ: Αικατερίνη Δόκα**, Καθηγητής ΕΚΠΑ

**ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: Αικατερίνη Δόκα**, Καθηγητής ΕΚΠΑ
**Μέμα Ρουσουπούλου**, Καθηγητής ΕΚΠΑ
**Ντούλας Αλέξανδρος**, Καθηγητής ΕΚΠΑ

**Ημερομηνία Εξέτασης: 4 Μαρτίου 2025**

# ABSTRACT

There is a growing body of research focused on cryptocurrency price prediction, particularly for Bitcoin, the most widely traded digital asset. Previous studies have predominantly utilized socio-economic factors and technical market indicators in combination with traditional machine learning models. The primary contribution of this thesis is the development of a real-time algorithm for Bitcoin price forecasting using Neural Networks and blockchain transaction graphs. Unlike previous approaches, this work leverages the dynamic structure of live blockchain data, introducing a novel graph-based methodology for financial forecasting.

Real-time data was collected using free and publicly available APIs to capture live transaction streams and historical price data. The real-time nature of the algorithm required adaptations in both data processing and model architecture. To address these demands, the study exploits a combination of traditional graph embedding techniques and custom-designed methods tailored to the characteristics of live blockchain transaction data. These embeddings, when integrated with NNs, enable the identification of transactional patterns that are crucial in forecasting Bitcoin's price.

Beyond theoretical evaluation, this research incorporates a real-world scenario to assess the model's practical effectiveness. The algorithm's performance is evaluated not only on conventional metrics but also in terms of revenue generation, simulating how well the model would perform in an actual trading environment. This approach highlights the practical applicability of combining graph-based machine learning with real-time blockchain analytics for cryptocurrency price prediction.

**SUBJECT AREA**: Machine Learning and Neural Networks (NNs) for Blockchain Data Analysis and Real-Time Financial Forecasting

**KEYWORDS**: Price Prediction, Graph Neural Networks (NNs), Bitcoin, Blockchain, Transaction Graphs, Machine Learning, Graph Regression, Forecasting, Graph Embedding Techniques, Cryptocurrencies, Time Series Analysis, Transaction Graph Construction Algorithm, Sliding Window Approach, Real-Time Data Processing, Unsupervised Learning, Blockchain Analytics, Financial Forecasting, Decentralized Networks, Deep Learning, Network Topology, Feature Engineering, Dynamic Graphs, Temporal Graph Analysis, Blockchain Data Streams, Predictive Modeling, Live Data Integration.

# ΠΕΡΙΛΗΨΗ

Υπάρχει ένα αυξανόμενο ερευνητικό ενδιαφέρον για την πρόβλεψη τιμών κρυπτονομισμάτων, με επίκεντρο το Bitcoin, το πιο ευρέως διαπραγματεύσιμο ψηφιακό νόμισμα. Προηγούμενες μελέτες βασίζονται κυρίως σε κοινωνικοοικονομικούς παράγοντες και τεχνικούς δείκτες αγοράς, σε συνδυασμό με παραδοσιακά μοντέλα μηχανικής μάθησης. Η κύρια συμβολή αυτής της διπλωματικής εργασίας είναι η ανάπτυξη ενός αλγορίθμου πρόβλεψης τιμών Bitcoin σε πραγματικό χρόνο, χρησιμοποιώντας Νευρωνικά Δίκτυα και γράφους συναλλαγών του blockchain. Σε αντίθεση με προηγούμενες προσεγγίσεις, η παρούσα εργασία αξιοποιεί τη δυναμική δομή των ζωντανών δεδομένων blockchain, εισάγοντας μια νέα μεθοδολογία βασισμένη σε γράφους για χρηματοοικονομική πρόβλεψη.

Τα δεδομένα συλλέχθηκαν σε πραγματικό χρόνο χρησιμοποιώντας δωρεάν και δημόσια διαθέσιμα APIs για τη λήψη ζωντανών συναλλαγών και ιστορικών δεδομένων τιμών. Η φύση του αλγορίθμου σε πραγματικό χρόνο απαίτησε προσαρμογές τόσο στη διαδικασία επεξεργασίας δεδομένων όσο και στην αρχιτεκτονική του μοντέλου. Για την αντιμετώπιση αυτών των απαιτήσεων, η μελέτη εκμεταλλεύεται έναν συνδυασμό από παραδοσιακές τεχνικές μηχανικής μάθησης γράφων και μεθόδους ειδικά σχεδιασμένες για τα χαρακτηριστικά των live συναλλαγών του blockchain. Αυτές οι τεχνικές, όταν χρησιμοποιούνται σε NNs, επιτρέπουν την αναγνώριση προτύπων συναλλαγών που είναι καθοριστικά για την πρόβλεψη της τιμής του Bitcoin.

Πέρα από τη θεωρητική αξιολόγηση, η έρευνα αυτή εξερευνά κι ένα πραγματικό σενάριο για την αξιολόγηση της πρακτικής αποτελεσματικότητας του μοντέλου. Η απόδοση του αλγορίθμου αξιολογείται όχι μόνο με συμβατικούς δείκτες, αλλά και με βάση τα έσοδα, προσομοιώνοντας την απόδοση του μοντέλου σε ένα πραγματικό περιβάλλον συναλλαγών. Αυτή η προσέγγιση αναδεικνύει τη χρηστική εφαρμογή του συνδυασμού μηχανικής μάθησης βασισμένης σε γράφους με δεδομένα blockchain σε πραγματικό χρόνο για την πρόβλεψη τιμών κρυπτονομισμάτων.

**ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ**: Μηχανική Μάθηση και Νευρωνικά Δίκτυα Γράφων (GNNs) για Ανάλυση Δεδομένων Blockchain και Χρηματοοικονομική Πρόβλεψη σε Πραγματικό Χρόνο

**ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ**: Πρόβλεψη Τιμών, Νευρωνικά Δίκτυα Γράφων (GNNs), Bitcoin, Blockchain, Γράφοι Συναλλαγών, Μηχανική Μάθηση, Παλινδρόμηση σε Γράφους, Πρόβλεψη, Τεχνικές Ενσωμάτωσης Γράφων, Κρυπτονομίσματα, Ανάλυση Χρονοσειρών, Αλγόριθμος Κατασκευής Γράφων Συναλλαγών, Προσέγγιση Κινούμενου Παραθύρου (Sliding Window), Επεξεργασία Δεδομένων σε Πραγματικό Χρόνο, Μη

Εποπτευόμενη Μάθηση, Ανάλυση Δεδομένων Blockchain, Χρηματοοικονομική Πρόβλεψη, Αποκεντρωμένα Δίκτυα, Βαθιά Μάθηση (Deep Learning), Τοπολογία Δικτύου, Μηχανική Χαρακτηριστικών (Feature Engineering), Δυναμικοί Γράφοι, Ανάλυση Χρονικών Γράφων, Ροές Δεδομένων Blockchain, Προγνωστική Μοντελοποίηση, Ενσωμάτωση Ζωντανών Δεδομένων.

# ACKNOWLEDGMENTS

First, I extend my deepest appreciation to my advisor, Aikaterini Doka, for her support, insightful guidance, and constructive feedback throughout this research.

I am also grateful to Charis Kleitsikas for the advices and feedback that enabled me to conduct this research effectively.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# PREFACE

The rapid growth of blockchain technology and cryptocurrency markets has led to an increasing need for accurate and efficient forecasting models. As digital assets continue to reshape the financial landscape, understanding and predicting their price movements remains a challenge due to their high volatility and complex transactional structures.

The motivation behind this research stems from the limitations of traditional financial models, which often rely solely on historical price data while overlooking the network dynamics of blockchain transactions. By employing graph-based approaches, this work aims to capture the structural and temporal patterns in cryptocurrency markets, offering new perspectives for predictive analytics and algorithmic trading.

This thesis presents a comprehensive pipeline for data collection, feature engineering, and model evaluation, ensuring a robust and scalable approach to cryptocurrency forecasting. Additionally, it goes beyond theoretical model assessments by incorporating a trading simulation framework, which evaluates the real-world implications of the predictions.

I hope this work contributes to the growing field of blockchain analytics, algorithmic trading, and financial forecasting, providing valuable insights for both researchers and practitioners in the domain.

# 1. DATA ENGINEERING

Data engineering is the discipline focused on designing, building, and maintaining the infrastructure and systems that facilitate the efficient collection, storage, and processing of data. This process includes data ingestion (gathering data from APIs, databases, or other sources), data transformation (cleaning and organizing raw data), and data integration (combining data from multiple sources into a unified format) [1]. In this project, I implemented a real-time data engineering pipeline to collect, process, and prepare blockchain transaction data for Bitcoin price forecasting. I used the Blockchain.info WebSocket API [2] to capture live streams of Bitcoin transactions and the CoinGecko API [3] for historical price data. The raw transaction data was ingested and processed into transaction graphs. I organized the data into 30-minute windows allowing me to capture snapshots of the transaction network over time. This process involved extensive data cleaning to remove irrelevant transactions, structuring the data to reflect the network's connectivity, and integrating additional market data to enhance the predictive features of the model.

## 1.1 APIs

An API, or Application Programming Interface, is a set of protocols and tools that allow different software applications to communicate with each other [4]. APIs define the methods and data structures that developers can use to interact with the functionalities of an existing software application, enabling the integration of services and data across different platforms. APIs are essential in modern software development as they provide a standardized way to access the features and data of another application, without needing to understand the internal workings of that application.

### 1.1.1 CoinGecko API

The CoinGecko API is a widely used service that provides real-time and historical data on cryptocurrency markets. This API is particularly valuable for retrieving information about various cryptocurrencies, including their current prices, market capitalization, trading volumes, and other key metrics. The specific endpoint used (`/coins/markets`) allows access to detailed market data for specified cryptocurrencies, in this case, Bitcoin. The CoinGecko API updates this data every 10 minutes, ensuring that the information retrieved is up-to-date and reflect the latest market conditions.

Example CoinGecko API Data

Below is the example data entry collected using the CoinGecko API on July 2, 2024:


```
{
    'id': 'bitcoin',
    'symbol': 'btc',
    'name': 'Bitcoin',
    'image': 'https://coin-images.coingecko.com/coins/images/1/large/bitcoin.
    'current_price': 61871,
    'market_cap': 1220053807334,
    'market_cap_rank': 1,
    'fully_diluted_valuation': 1299292918350,
    'total_volume': 23474981674,
    'high_24h': 63791,
    'low_24h': 61848,
    'price_change_24h': -1004.2204519376755,
    'price_change_percentage_24h': -1.59716,
    'market_cap_change_24h': -19975175307.439453,
    'market_cap_change_percentage_24h': -1.61086,
    'circulating_supply': 19719287.0,
    'total_supply': 21000000.0,
    'max_supply': 21000000.0,
    'ath': 73738,
    'ath_change_percentage': -15.98852,
    'ath_date': '2024-03-14T07:10:36.635Z',
    'atl': 67.81,
    'atl_change_percentage': 91257.0926,
    'atl_date': '2013-07-06T00:00:00.000Z',
    'roi': None,
    'last_updated': '2024-07-02T15:30:06.459Z'
}
```


Explanation of Each Feature

- **ID (`id`)**: A unique identifier for the cryptocurrency, which is `bitcoin` in this case.
- **Symbol (`symbol`)**: The shorthand or ticker symbol for the cryptocurrency, here it is `btc` for Bitcoin.
- **Name (`name`)**: The full name of the cryptocurrency, which is `Bitcoin`.
- **Image (`image`)**: A URL link to an image representing the cryptocurrency, often used in apps or websites to visually identify the coin. You can view the im-

age using this URL: https://coin-images.coingecko.com/coins/images/1/large/bitcoin.png?1696501400

- **Current Price (`current_price`)**: The latest market price of Bitcoin, given in USD. As of the time of data collection, Bitcoin was priced at $61,871.
- **Market Cap (`market_cap`)**: The total market value of Bitcoin in circulation, calculated by multiplying the current price by the circulating supply. Here, it is about $1.22 trillion.
- **Market Cap Rank (`market_cap_rank`)**: The ranking of Bitcoin based on its market capitalization compared to other cryptocurrencies. Bitcoin is ranked number 1.
- **Fully Diluted Valuation (`fully_diluted_valuation`)**: The market capitalization if all potential coins (the total supply) were in circulation. For Bitcoin, this figure is approximately $1.3 trillion.
- **Total Volume (24h) (`total_volume`)**: The total amount of Bitcoin traded in the last 24 hours, reflecting liquidity and market activity, which was about $23.47 billion.
- **24h High (`high_24h`)**: The highest price that Bitcoin reached in the past 24 hours, which was $63,791.
- **24h Low (`low_24h`)**: The lowest price that Bitcoin reached in the past 24 hours, which was $61,848.
- **Price Change (24h) (`price_change_24h`)**: The absolute change in Bitcoin's price over the last 24 hours, showing a decrease of $1,004.22.
- **Price Change Percentage (24h) (`price_change_percentage_24h`)**: The percentage change in Bitcoin's price over the last 24 hours, which is a decrease of 1.60%.
- **Market Cap Change (24h) (`market_cap_change_24h`)**: The absolute change in Bitcoin's market capitalization over the last 24 hours, showing a decrease of approximately $19.98 billion.
- **Market Cap Change Percentage (24h) (`market_cap_change_percentage_24h`)**: The percentage change in Bitcoin's market capitalization over the last 24 hours, which is a decrease of 1.61%.
- **Circulating Supply (`circulating_supply`)**: The number of Bitcoin currently in circulation, which is 19,719,287 BTC.

- **Total Supply (`total_supply`)**: The total number of Bitcoin that exists so far, which is capped at 21 million BTC.
- **Max Supply (`max_supply`)**: The maximum supply of Bitcoin that will ever exist, set at 21 million BTC.
- **All-Time High (ATH) (`ath`)**: The highest price Bitcoin has ever reached, which was $73,738.
- **ATH Change Percentage (`ath_change_percentage`)**: The percentage change from the all-time high to the current price, showing a decrease of about 15.99%.
- **ATH Date (`ath_date`)**: The date when Bitcoin reached its all-time high, which was March 14, 2024.
- **All-Time Low (ATL) (`atl`)**: The lowest price Bitcoin has ever reached, which was $67.81.
- **ATL Change Percentage (`atl_change_percentage`)**: The percentage increase from the all-time low to the current price, showing a significant rise of 91,257.09%.
- **ATL Date (`atl_date`)**: The date when Bitcoin was at its all-time low, which was July 6, 2013.
- **Return on Investment (ROI) (`roi`)**: This is typically used for other cryptocurrencies but is not applicable for Bitcoin in this dataset (`None`).
- **Last Updated (`last_updated`)**: The timestamp for when this data was last updated, indicating the data's recency. Here, the last update was on July 2, 2024, at 15:30:06 UTC.

In the context of forecasting Bitcoin's future price, certain features hold varying degrees of predictive value. Total Volume (24h) stands out with high forecasting value as it directly reflects market activity and liquidity, which are critical indicators of potential price movements. Several features possess moderate forecasting value, including Current Price, Market Cap, 24h High, 24h Low, Price Change (24h), Price Change Percentage (24h), Market Cap Change (24h), Market Cap Change Percentage (24h), All-Time High (ATH), and ATH Change Percentage. These metrics provide insights into recent market trends, price levels, and overall market sentiment, making them useful for short-term predictions. On the other hand, features like ID, Symbol, Name, Image, Circulating Supply, Total Supply, Max Supply, ATH Date, ATL, ATL Change Percentage, ATL Date, ROI, Last Updated, Market Cap Rank, and Fully Diluted Valuation offer low or no forecasting value, as they are more descriptive or historical in nature and do not directly influence future price changes.

### 1.1.2 Blockchain.info WebSocket API

The Blockchain.info WebSocket API is a specialized API designed to provide real-time data from the Bitcoin network. By connecting to this API, users can receive continuous updates on Bitcoin transactions and blocks as they occur, making it a valuable tool for developers and analysts who require live data for monitoring, analysis, or trading purposes. The WebSocket protocol, used by this API, is particularly well-suited for applications that need to maintain a persistent connection to a server, allowing data to be pushed to the client as soon as it becomes available, rather than requiring the client to repeatedly request updates.

This API offers various operations, including subscriptions to unconfirmed transactions, new blocks, and specific Bitcoin addresses. By subscribing to unconfirmed transactions, for instance, users can monitor transactions as they are broadcast to the network but before they are included in a block. The data provided by the API includes detailed information about each transaction, such as the input and output addresses, transaction value, and time of the transaction. This real-time access to Bitcoin data is essential for applications that require up-to-the-second information, such as trading platforms, monitoring tools, or research applications.

Python offers several libraries, such as requests for HTTP-based APIs and websocket-client for WebSocket APIs, which make it easier to connect, communicate, and process data from these interfaces. These libraries provide built-in functions to handle the complexities of network communication, data parsing, and error handling, allowing developers to focus on the core logic of their applications rather than the underlying protocol details.

The transaction data obtained from the Blockchain.info WebSocket API follows a structured format that provides detailed information about each Bitcoin transaction. The JSON object typically includes several key fields.

Bitcoin Transaction Example

Below is an example of a Bitcoin transaction as provided by the Blockchain.info WebSocket API:


```
{
    ”op”: ”utx”,
    ”x”: {
        ”lock_time”: 0,
        ”ver”: 2,
```

```
"size": 207,
"inputs": [
    {
        "sequence": 4294967293,
        "prev_out": {
            "spent": True,
            "tx_index": 0,
            "type": 0,
            "addr": "bc1q53l73j028sh09axkwvgum22eedcvna630hp8x7",
            "value": 8113,
            "n": 1,
            "script": "0014a47fe8c9ea3c2ef2f4d67311cda959cb70c9f751"
        },
        "script": ""
    }
],
"time": 1719936955,
"tx_index": 0,
"vin_sz": 1,
"hash": "7080c3cbb83e62bb448468ca8a31f9421fb882f016c0a65e1",
"vout_sz": 2,
"relayed_by": "0.0.0.0",
"out": [
    {
        "spent": False,
        "tx_index": 0,
        "type": 0,
        "addr": None,
        "value": 0,
        "n": 0,
        "script": "6a5d0414011400"
    },
    {
        "spent": True,
        "tx_index": 0,
        "type": 0,
        "addr": "bc1q53l73j028sh09axkwvgum22eedcvna630hp8x7",
        "value": 7281,
        "n": 1,
        "script": "0014a47fe8c9ea3c2ef2f4d67311cda959cb70c9f751"
    }
]
```

```
    }
}
```


Explanation of the Transaction Example

- **op:** `"utx"` indicates that this is an unconfirmed transaction.
- **x:** This key holds all the transaction-specific data.
  - **lock_time:** `0`, the transaction can be processed immediately.
  - **ver:** `2`, indicating the version of the transaction format.
  - **size:** `207`, the size of the transaction in bytes.
  - **inputs:** Contains a list of inputs used in this transaction. Each input includes:
    - **sequence:** `4294967293`, the sequence number of the input.
    - **prev_out:** Details of the previous output being spent:
      - **spent:** `True`, indicating the output has been used.
      - **tx_index:** `0`, the transaction index of the previous output.
      - **type:** `0`, the type of transaction.
      - **addr:** `"bc1q53l73j028sh09axkwvgum22eedcvna630hp8x7"`, the Bitcoin address from which the input is derived.
      - **value:** `8113`, the amount of Bitcoin (in satoshis) associated with the input.
      - **n:** `1`, the index of the output in the previous transaction.
      - **script:** `"0014a47fe8c9ea3c2ef2f4d67311cda959cb70c9f751"`, the script associated with the input.
  - **time:** `1719936955`, the UNIX timestamp of when the transaction was broadcast, it represents the number of seconds that have elapsed since the Unix epoch, which is defined as 00:00:00 UTC on January 1, 1970. This system of timekeeping is widely used in computing and programming because it is simple, consistent, and time zone-independent.
  - **tx_index:** `0`, the transaction index (usually provided by the blockchain system).
  - **vin_sz:** `1`, the number of inputs in the transaction.
  - **hash:** `"7080c3cbb83e62bb448468ca8a31f9421fb882f016c0a65e11539135285c`
    the unique hash identifier for this transaction.
  - **vout_sz:** `2`, the number of outputs in the transaction.

- **relayed_by:** `"0.0.0.0"`, the IP address of the node that relayed this transaction (anonymized in this case).
- **out:** Contains a list of outputs generated by this transaction:
  - **Output 1:**
    - **spent:** `False` - This field indicates whether the output has already been spent. If `True`, it means the output has been used as an input in another transaction.
    - **tx_index:** `0` - The transaction index within the blockchain. This is a unique identifier that helps locate the specific transaction in which this output was created.
    - **type:** `0` - This field typically indicates the type of output. For standard transactions, it is usually `0`.
    - **addr:** `None` - The Bitcoin address that receives the output. This is the recipient's address to which the specified amount of Bitcoin is being sent.
    - **value:** `0` - The amount of Bitcoin in satoshis (1 Bitcoin = 100,000,000 satoshis) that is being transferred to the `addr`. This represents the actual value of the output.
    - **n:** `0` - The index of the output in the transaction. This is used to identify this particular output among possibly multiple outputs in the same transaction.
    - **script:** `"6a5d0414011400"` - This field contains the locking script (also known as the scriptPubKey), which specifies the conditions that must be met to spend the output. For example, it usually includes the public key hash corresponding to the recipient's Bitcoin address.
  - **Output 2:**
    - **spent:** `True` - This field indicates whether the output has already been spent. If `True`, it means the output has been used as an input in another transaction.
    - **tx_index:** `0` - The transaction index within the blockchain. This is a unique identifier that helps locate the specific transaction in which this output was created.
    - **type:** `0` - This field typically indicates the type of output. For standard transactions, it is usually `0`.
    - **addr:** `"bc1q53l73j028sh09axkwvgum22eedcvna630hp8x7"` - The Bitcoin address that receives the output. This is the recipient's address to which the specified amount of Bitcoin is being sent.

- **value:** `7281` - The amount of Bitcoin in satoshis (1 Bitcoin = 100,000,000 satoshis) that is being transferred to the `addr`. This represents the actual value of the output.
- **n:** `1` - The index of the output in the transaction. This is used to identify this particular output among possibly multiple outputs in the same transaction.
- **script:** `"0014a47fe8c9ea3c2ef2f4d67311cda959cb70c9f751"` - This field contains the locking script (also known as the scriptPubKey), which specifies the conditions that must be met to spend the output. For example, it usually includes the public key hash corresponding to the recipient's Bitcoin address.

## 1.2 Graph Creation

### 1.2.1 Early Graph Creation

Nodes in the Graph The graph's nodes represent Bitcoin addresses involved in the transactions. Each unique address, whether it's sending or receiving Bitcoin, becomes a node in the graph. The code collects these addresses from the inputs and outputs of the transactions.

Edges in the Graph The edges in the graph represent the flow of Bitcoin from one address to another. Each transaction is analyzed to determine the sending (input) and receiving (output) addresses, and an edge is created from the input address node to the output address node. These edges are directed, meaning they indicate the direction of Bitcoin flow (from sender to recipient).

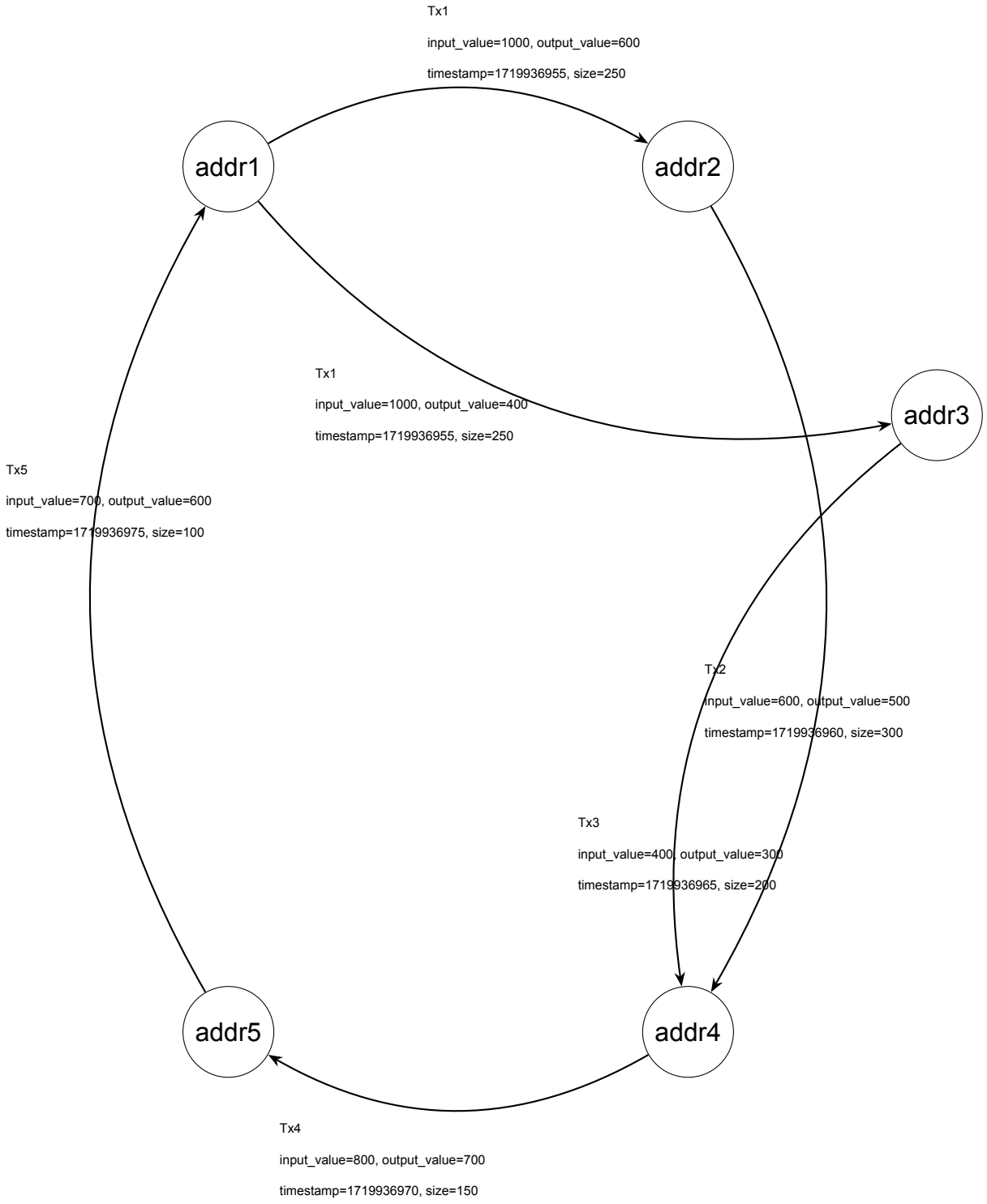
Edge Features Along with creating edges, the code also collects features for each edge. These features include attributes of the transaction such as:

- **size:** The size of the transaction in bytes.
- **input_value:** The amount of Bitcoin (in satoshis) being spent by the input address.
- **output_value:** The amount of Bitcoin being received by the output address.
- **timestamp:** The time when the transaction was broadcast.

These features are important because they can be used to analyze the characteristics of the transactions represented by the edges in the graph.

For forecasting the future price of Bitcoin, the most valuable edge features are likely to be input value, output value, and timestamp, with size also having some relevance. These features can provide insights into market sentiment, large trades, and the timing of transactions, which are crucial for predicting price movements. Additionally, vin sz and vout sz will be represented as directed edges from the node, further enhancing the model's ability to capture the flow and distribution of transaction inputs and outputs across the network.

Example




This graph represents the flow of Bitcoin between different addresses in a simplified network of transactions. Each node (circle) in the graph corresponds to a unique

Bitcoin address (addr1, addr2, addr3, addr4, and addr5), and each directed edge (arrow) represents a transaction in which Bitcoin is sent from one address to another.

Key Components

- **Nodes:** The nodes addr1, addr2, addr3, addr4, and addr5 represent different Bitcoin addresses involved in the transactions.
- **Edges:** The arrows indicate the direction of the Bitcoin flow. For example, an arrow from addr1 to addr2 indicates that addr1 sent Bitcoin to addr2 in a transaction labeled as Tx1.
- **Transaction Labels:** Each edge is labeled with the transaction identifier (Tx1, Tx2, etc.) followed by specific features of that transaction, such as size, input_value, output_value and timestamp. These features provide detailed information about the transactions, including when they were broadcast, the amounts involved, and the conditions under which the Bitcoin can be spent.

Summary of the Flow

- **Transaction 1 (Tx1):** addr1 sends Bitcoin to both addr2 and addr3.
- **Transaction 2 (Tx2):** addr2 sends Bitcoin to addr4.
- **Transaction 3 (Tx3):** addr3 also sends Bitcoin to addr4.
- **Transaction 4 (Tx4):** addr4 sends Bitcoin to addr5.
- **Transaction 5 (Tx5):** addr5 sends Bitcoin back to addr1, completing the cycle.

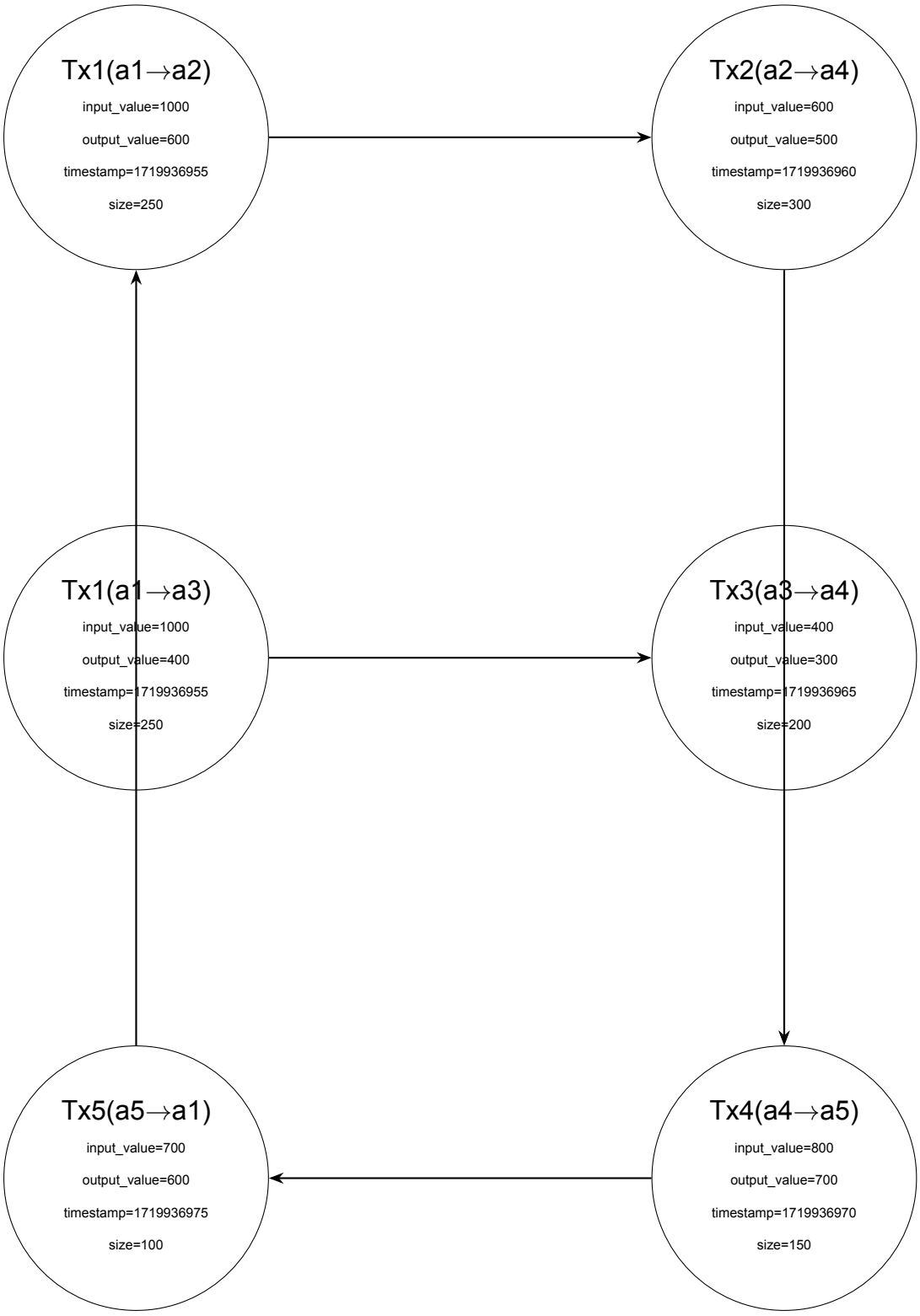
Purpose of the Graph

This graph visually illustrates the connections between different Bitcoin addresses as they participate in a series of transactions. It shows not only the directional flow of Bitcoin but also provides insight into the nature and details of each transaction through the associated edge features. This kind of visualization can be useful for analyzing transaction patterns, tracking the flow of funds, or detecting any unusual activities in a network of Bitcoin transactions.

### 1.2.2 Line Graph Transformation

The transformation of a directed graph into a line graph is a process where each node in the line graph represents a directed edge from the original graph [5] . This transformation is particularly useful when edge features need to be incorporated into models, like modern Graph Neural Networks (GNNs), that are primarily designed to work with node features [6]. In directed graphs, the directionality of edges is crucial, as it indicates the flow from one node to another. Therefore, when creating the line graph, it is essential to preserve this directionality: two nodes in the line graph are connected if their corresponding edges in the original graph are consecutive and share the same directional flow. This approach shifts the focus from the entities (nodes) in the original graph to the interactions (edges) between them, allowing the GNN to effectively utilize the rich information embedded in these interactions. This is especially beneficial in contexts such as financial networks or communication systems, where the direction and sequence of transactions or communications are vital for understanding the network's behavior and making accurate predictions.

Directed Line Graph with Node Features


Tx1(a1→a2)
input_value=1000
output_value=600
timestamp=1719936955
size=250
Tx2(a2→a4)
input_value=600
output_value=500
timestamp=1719936960
size=300
Tx1(a1→a3)
input_value=1000
output_value=400
timestamp=1719936955
size=250
Tx3(a3→a4)
input_value=400
output_value=300
timestamp=1719936965
size=200
Tx5(a5→a1)
input_value=700
output_value=600
timestamp=1719936975
size=100
Tx4(a4→a5)
input_value=800
output_value=700
timestamp=1719936970
size=150

## 1.3 Data Collection

The data collection process was structured into 12-hour live streaming sessions, with each dataset representing a complete 12-hour window of blockchain transactions. This approach ensured that a significant volume of real-time transactional data was collected, capturing the natural fluctuations and activity patterns of the Bitcoin network over extended periods. By focusing on 12-hour streams, I was able to observe both short-term and medium-term transactional behaviors, which are essential for accurate price forecasting.

Initially, transaction graphs were constructed by representing Bitcoin addresses as nodes and the flow of transactions as directed edges. While this approach effectively captured the basic connectivity of the network, it did not fully exploit the strengths of GNNs, which are better suited to handling feature-enriched nodes of graphs. To enhance the model's performance, the graph structure was transformed into a more complex representation using the line graph transformation.

To transform this raw data into a usable format for machine learning, the transaction data was organized into 30-minute graph windows. A new graph was generated every 10 minutes, creating overlapping snapshots of the transaction network. This overlapping window strategy allowed for the detection of emerging patterns and short-term shifts in transaction behavior, while still maintaining a broader view of the network's activity over time. These graphs formed the foundation for the subsequent graph-based machine learning models, enabling effective and timely Bitcoin price predictions.

### 1.3.1 Dataset Analytics

| Dataset Index | Start Time | End Time |
|---|---|---|
| 0 | 2024-08-21 15:45:43 | 2024-08-22 03:38:01 |
| 1 | 2024-08-25 06:15:24 | 2024-08-25 18:17:51 |
| 2 | 2024-08-25 23:58:49 | 2024-08-26 12:02:01 |
| 3 | 2024-08-26 19:13:24 | 2024-08-27 07:16:01 |
| 4 | 2024-08-29 11:34:18 | 2024-08-29 23:27:50 |
| 5 | 2024-08-31 20:23:55 | 2024-09-01 08:27:51 |
| 6 | 2024-09-01 15:50:06 | 2024-09-02 03:42:40 |
| 7 | 2024-09-02 15:24:36 | 2024-09-03 03:27:02 |
| 8 | 2024-09-03 03:27:02 | 2024-09-03 15:21:24 |
| 9 | 2024-09-03 19:42:59 | 2024-09-04 07:35:42 |
| 10 | 2024-09-04 07:45:25 | 2024-09-04 19:39:01 |
| 11 | 2024-09-04 21:09:54 | 2024-09-05 09:03:37 |
| 12 | 2024-09-07 15:17:46 | 2024-09-08 03:11:48 |
| 13 | 2024-09-18 15:17:11 | 2024-09-19 03:20:52 |
| 14 | 2024-09-24 22:07:44 | 2024-09-25 10:11:48 |
| 15 | 2024-11-08 11:55:02 | 2024-11-08 23:58:16 |
| 16 | 2024-11-09 02:09:59 | 2024-11-09 14:11:57 |
| 17 | 2024-11-10 04:15:18 | 2024-11-10 16:18:01 |
| 18 | 2024-11-11 11:52:00 | 2024-11-11 23:44:49 |
| 19 | 2024-11-12 02:35:45 | 2024-11-12 14:29:47 |
| 20 | 2024-11-12 14:49:45 | 2024-11-13 02:52:09 |
| 21 | 2024-11-13 11:34:37 | 2024-11-13 23:34:33 |
| 22 | 2024-11-14 10:48:52 | 2024-11-14 22:50:53 |
| 23 | 2024-11-15 01:41:59 | 2024-11-15 13:54:31 |
| 24 | 2024-11-15 16:35:05 | 2024-11-16 04:38:00 |
| 25 | 2024-11-17 20:46:24 | 2024-11-18 08:48:15 |
| 26 | 2024-11-18 14:20:00 | 2024-11-19 02:21:58 |
| 27 | 2024-11-27 12:00:41 | 2024-11-28 00:03:52 |
| 28 | 2024-12-01 22:27:39 | 2024-12-02 10:20:31 |
| 29 | 2024-12-09 22:30:02 | 2024-12-10 10:22:48 |
| 30 | 2024-12-11 16:48:50 | 2024-12-12 04:51:04 |
| 31 | 2024-12-12 19:24:56 | 2024-12-13 07:27:25 |
| 32 | 2024-12-15 22:14:42 | 2024-12-16 10:06:57 |
| 33 | 2024-12-16 10:56:37 | 2024-12-16 22:59:54 |
| 34 | 2024-12-16 23:19:34 | 2024-12-17 11:22:27 |
| 35 | 2024-12-17 14:32:35 | 2024-12-18 02:25:57 |
| 36 | 2024-12-18 20:39:46 | 2024-12-19 08:32:27 |
| 37 | 2024-12-19 09:42:26 | 2024-12-19 21:34:30 |
| 38 | 2024-12-20 09:57:20 | 2024-12-20 21:50:05 |
| 39 | 2024-12-22 18:59:16 | 2024-12-23 07:02:07 |
| 40 | 2025-01-01 18:49:29 | 2025-01-02 06:32:00 |
| 41 | 2025-01-02 12:43:18 | 2025-01-03 00:45:58 |
| 42 | 2025-01-03 02:45:39 | 2025-01-03 14:39:02 |
| 43 | 2025-01-05 18:40:17 | 2025-01-06 06:52:47 |
| 44 | 2025-01-19 19:54:39 | 2025-01-20 07:57:24 |
| 45 | 2025-01-27 20:18:28 | 2025-01-28 08:24:02 |
| 46 | 2025-02-02 21:17:37 | 2025-02-03 09:23:08 |

**Table 1.1: Overview of Collected Datasets**

# 2. GRAPH REPRESENTATION AND FEATURE ENGINEERING TECHNIQUES

In this chapter, we explore the methods used to transform raw blockchain transaction graphs into meaningful numerical representations suitable for machine learning models. These embeddings convert complex graph data into lower-dimensional vector spaces, preserving information about key properties like node connectivity and neighborhood information. Following the embedding process, we apply Principal Component Analysis (PCA) to reduce the dimensionality of the feature space, enhancing computational efficiency while retaining the most informative components of the data. Additionally, we introduce differencing techniques to address temporal dependencies and non-stationarity in the data, which are critical for improving the accuracy of time-series predictions. Together, these methods form the foundation for preparing the graph-based features that will be utilized in the subsequent machine learning models for Bitcoin price forecasting.

## 2.1 Supervised and Unsupervised Graph Embedding Methods

In contemporary graph-based machine learning, there is a clear conceptual and methodological divide between graph neural network (GNN) models trained with a target variable and those methods that generate embeddings without direct supervision. Supervised GNN architectures—such as Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), or GraphSAGE—learn node or graph representations by optimizing against a known target, allowing them to tailor their embeddings to specific predictive tasks. In contrast, unsupervised graph embedding techniques like Node2Vec, DeepWalk, and spectral methods rely solely on the inherent structure and unlabeled attributes of the graph to produce meaningful representations. These unsupervised approaches do not incorporate labeled outcomes, but instead capture patterns through random walks, eigenvalue decompositions, or simple aggregations of node features. By differentiating between supervised GNN-based methods and unsupervised embedding techniques, we can better understand their respective strengths, intended uses, and potential integration points in real-world analytical pipelines.

### 2.1.1 Rationale for Using Feature Engineering for Graph Embeddings

In the development of a live algorithm for analyzing blockchain transactions, I utilized **blockchain** and **CoinGecko APIs** to query and collect real-time transactional data. The entire data collection, preprocessing, and engineering pipeline was im-

plemented on my personal laptop, giving me full control over the dataset but also imposing certain limitations regarding data volume and structure. Given the characteristics of the data and the specific challenges encountered in constructing the transaction graph, with restricted data volume and structure, it was more feasible to design handcrafted features that capture the essential structural and temporal characteristics of the blockchain transaction graph.

By using feature engineering, I was able to extract interpretable metrics that reflect both the connectivity and dynamics of the network. This approach allowed us to create efficient, rapidly computable graph embeddings without the need for complex neural network models, which are often data-hungry and computationally intensive.

1. **Limited and Self-Engineered Data Availability**
   Since all data were **collected and engineered independently**, the dataset was inherently limited in size and scope compared to large, pre-existing labeled datasets. The process of collecting and labeling blockchain transaction graphs for supervised learning tasks, such as forecasting, is highly demanding and difficult to perform in a live environment. Real-time data collection adds additional complexity due to the continuous nature of blockchain transactions and the absence of immediate, reliable labels. Given these constraints, I anticipated that my data would not be of the highest quality. Therefore, I adapted my approach to utilize unsupervised techniques, which do not require labeled data and allowed me to extract meaningful node and graph-level representations despite these limitations.

2. **Sparse and Structurally Simple Transaction Graphs**
   The transaction graphs constructed from the collected data, although consisting of thousands of nodes (transactions), exhibited **sparse connectivity** and **minimal structural complexity**. The graphs were characterized by **short path lengths** (with the longest paths spanning only 2-3 nodes) and a high number of **isolated nodes**. These structural properties limited the effectiveness of supervised Graph Neural Networks (GNNs), which rely on rich graph topology to learn task-specific patterns. Conversely, **unsupervised embedding techniques** and custom methods tailored to the data were more effective at capturing the underlying economic dynamics and transactional behaviors occurring within that specific 30-minute timeframe.

3. **Adaptability to Real-Time Data and Computational Constraints**
   Given that the algorithm was designed to process **real-time blockchain data** on a **laptop**, computational efficiency was a critical factor. **Unsupervised embedding techniques** generally require fewer computational resources compared to supervised GNNs, which involve iterative training on labeled data.

4. **Custom Embedding Techniques to Address Data Specificities**

To further address the limitations of the dataset and the unique structural characteristics of blockchain transaction graphs, I developed **custom unsupervised embedding techniques**. These methods were tailored to capture specific transactional patterns, such as **cash flow dynamics** and **node interaction temporal frequencies**, which traditional unsupervised methods might overlook. This customization enabled the generation of more **informative embeddings** that reflected the underlying behavior of the blockchain network.

In summary, the decision to employ **unsupervised graph embedding techniques** was driven by the limited availability of labeled data, the structural simplicity of the transaction graphs, and the need for computational efficiency in a real-time, resource-constrained environment. By leveraging both traditional and custom (task-based) unsupervised methods, I was able to extract meaningful representations from the data, enabling effective analysis of blockchain transaction dynamics.

### 2.1.2 Time Evaluation

Our approach, which relies on very simple yet effective feature engineering, enables the rapid generation of graph embeddings. This efficiency is particularly valuable in a live forecasting setting where timely predictions are crucial. The fast embedding computation allows our model to process and respond to new data in real time, ensuring that the forecasting system remains both responsive and practical even when deployed in dynamic market conditions.

## 2.2 Unsupervised Graph Embedding Methods

Our forecasting model leverages a comprehensive feature set that integrates three key categories of engineered features, each capturing critical aspects of our transaction data:

**Feature-Based Statistics** summarize the distribution of raw node attributes (such as transaction size, input and output values, and timestamps) by computing measures like the mean, standard deviation, median, minimum, maximum, and sum. These 24 features provide essential insights into the central tendencies and variability of the data, which are foundational for understanding the underlying dynamics.

**Structural Features** describe the topology of the network by quantifying aspects such as the number of nodes and edges, density, degree statistics (mean, standard deviation, minimum, and maximum), as well as the connectivity of the graph (number of connected components and the size of the largest component). These 9 features

are crucial for capturing the organization and interconnection of the network, which can significantly influence information flow and aggregation.

**Temporal Features** capture the dynamic behavior of transactions over time, including overall activity rates, variability in transaction timing, burstiness, local trends via moving averages and standard deviations, as well as trend indicators like the slope and rate change of transactions. This category contributes 28 features that reflect both the aggregate and local temporal patterns, providing signals about potential shifts or anomalies in the market.

In total, the feature set comprises 61 features (24 from Feature-Based Statistics, 9 from Structural Features, and 28 from Temporal Features). This multidimensional approach enables the model to account for both static properties and dynamic trends in the data, which is essential for making accurate and reliable forecasts.

#### 2.2.0.1 Descriptive Statistics of Node Features for Forecasting

At first a total of 24 features from the node feature matrix are extracted. For each of the four node features—**size**, **input_value**, **output_value**, and **timestamp**—the following six statistical measures are computed:

mean, standard deviation, median, minimum, maximum, and sum.

Below is a breakdown of these features, what they represent, and why they are important for forecasting:

**1. Size (Transaction Size in Bytes) (6 features)**

- **Mean:** Average size of transactions.
- **Standard Deviation:** Variability in transaction sizes.
- **Median:** Central tendency that is robust to outliers.
- **Minimum:** Smallest transaction size observed.
- **Maximum:** Largest transaction size observed.
- **Sum:** Total size of all transactions.

**Importance:** These metrics capture the distribution of transaction sizes, which may reflect network load, processing costs, and other operational characteristics crucial for predicting future trends.

**2. Input Value (Bitcoin Spent by Input Address) (6 features)**

- **Mean:** Average amount spent.
- **Standard Deviation:** Variability in spending.
- **Median:** Robust central value of spending.
- **Minimum:** Lowest amount spent.
- **Maximum:** Highest amount spent.
- **Sum:** Total amount spent.

**Importance:** These statistics reflect the outflow of Bitcoin, which is key for understanding market sentiment, liquidity, and potential price movements.

**3. Output Value (Bitcoin Received by Output Address) (6 features)**

- **Mean:** Average amount received.
- **Standard Deviation:** Variability in received amounts.
- **Median:** Central tendency of the received amounts.
- **Minimum:** Smallest amount received.
- **Maximum:** Largest amount received.
- **Sum:** Total amount received.

**Importance:** These features capture the inflow of Bitcoin, providing insight into accumulation trends and overall market demand which are vital for forecasting future market dynamics.

**4. Timestamp (Transaction Broadcast Time) (6 features)**

- **Mean:** Average time of transactions, indicating when most activity occurs.
- **Standard Deviation:** Variability in transaction times, showing how spread out the transactions are.
- **Median:** The middle transaction time, offering a robust measure of central tendency.
- **Minimum:** The earliest transaction time.
- **Maximum:** The latest transaction time.

- **Sum:** The aggregate of all transaction times (useful for certain aggregate analyses).

**Importance:** Temporal metrics reveal patterns in transaction timing, such as peak periods or irregularities, which are essential for time-series forecasting and understanding market dynamics.

#### 2.2.0.2 Descriptive Structural Features for Forecasting

Next, a total of 9 features are extracted. These features provide insight into the network's size, connectivity, and overall structure, which are important for forecasting tasks. Below is a list of the features, their meanings, and their significance:

1. **Number of Nodes:** The total count of nodes in the graph. This indicates the overall size of the network.
2. **Number of Edges:** The total count of edges in the graph. This reflects the level of interactions or connections among nodes.
3. **Density:** Calculated as

$$\text{density} = \frac{2 \times \text{num\_edges}}{\text{num\_nodes} \times (\text{num\_nodes} - 1)},$$

   it measures how closely the network approaches being fully connected. A higher density suggests a tightly knit network.
4. **Mean Degree:** The average number of connections per node, which gives an idea of typical node connectivity.
5. **Standard Deviation of Degree:** This quantifies the variability in node connectivity. A higher standard deviation indicates that some nodes are significantly more connected than others.
6. **Minimum Degree:** The smallest number of connections that any node in the graph has. It highlights the sparsest areas of the network.
7. **Maximum Degree:** The largest number of connections observed for any node, which can signal the presence of highly influential or central nodes.
8. **Number of Connected Components:** The count of distinct connected subgraphs within the network, reflecting how fragmented or cohesive the network is.

9. **Size of the Largest Connected Component:** The number of nodes in the largest connected subgraph, representing the main cluster of activity in the network.

#### 2.2.0.3 Engineered Temporal Features for Forecasting

Below is a list of the engineered temporal feature categories (14 in total) along with their mathematical formulas and the parameters used in their computation. In our implementation, we use the following parameter values:

- Snapshot Duration, $T = 30$ minutes
- Subwindow Size, $w_s = 5$ minutes
- Number of Bins for Entropy, $k = 6$
- Recency Window, $w_r = 10$ minutes
- Moving Window Size, $w_m = 10$ minutes
- Moving Step Size, $s_m = 5$ minutes

1. **Transactions Per Minute (TPM):**

$$\text{TPM} = \frac{N}{T},$$

where $N$ is the number of transactions (rows in the node feature matrix). This feature normalizes the overall transaction activity.

2. **Timestamp Standard Deviation:**

$$\sigma_t = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(t_i - \mu_t)^2}, \quad \mu_t = \frac{1}{N}\sum_{i=1}^{N} t_i.$$

This quantifies how spread out the transaction times $t_i$ (in minutes) are within the snapshot.

3. **Maximum Subwindow Count:**

$$\max_{w \subset [0,T]} \text{Count}(w),$$

where $w$ is any subwindow of length $w_s = 5$ minutes. It identifies the peak number of transactions in any such interval.

4. **Entropy of Time Distribution:**

$$H = -\sum_{j=1}^{k} p_j \ln(p_j),$$

where the interval $[0, T]$ is divided into $k = 6$ equal bins and $p_j$ is the fraction of transactions in bin $j$. This measures the uniformity versus clustering of transactions.

5. **Inter-Arrival Time Features:** For the inter-arrival times $\Delta t_i = t_i - t_{i-1}$ (for $i = 2, \ldots, N$):

   - **Mean IAT:** Mean IAT $= \dfrac{1}{N-1} \displaystyle\sum_{i=2}^{N} \Delta t_i$.
   - **Std IAT:** Standard deviation of $\{\Delta t_i\}$.
   - **Min IAT:** $\min\limits_{2 \le i \le N} \Delta t_i$.
   - **Max IAT:** $\max\limits_{2 \le i \le N} \Delta t_i$.
   - **Median IAT:** The median value of $\{\Delta t_i\}$.
   - **Skew IAT:** The skewness of the inter-arrival times.

   These features describe the rhythm and regularity of transactions.

6. **Burstiness Index:**

$$B = \frac{\sigma_{\Delta t} - \mu_{\Delta t}}{\sigma_{\Delta t} + \mu_{\Delta t}},$$

where $\mu_{\Delta t}$ and $\sigma_{\Delta t}$ are the mean and standard deviation of the inter-arrival times. This indicates the degree of clustering in transactions.

7. **Recency Features:**

   - **Recency Proportion:**

$$R = \frac{\text{Transactions in last } w_r \text{ minutes}}{N}.$$

   - **Mean Timestamp:** $\mu_t = \dfrac{1}{N} \displaystyle\sum_{i=1}^{N} t_i$.

   These features reflect the concentration of transactions towards the end of the snapshot.

8. **Moving Average TPM:** Calculated over sliding windows of size $w_m = 10$ minutes with a step $s_m = 5$ minutes, it captures local average transaction rates.

9. **Moving Standard Deviation of TPM:** The standard deviation of TPM within each moving window, reflecting local volatility.

10. **Slope of TPM:** Obtained by fitting a linear regression to transaction indices versus a constant TPM contribution:

$$y = \beta x + \alpha, \quad \text{Slope} = \beta.$$

This indicates the overall trend in transaction activity over the snapshot.

11. **Rate Change of TPM:**

$$\Delta\text{TPM} = \text{TPM}_{\text{last}} - \text{TPM}_{\text{first}},$$

representing the difference in TPM between the last and first subwindow.

12. **Transaction Density:** Essentially the TPM itself, representing the density of transactions in the snapshot.

13. **Normalized Transaction Density:**

$$\text{Normalized Density} = \frac{\text{TPM}}{\text{TPM}_{\text{max}}},$$

where $\text{TPM}_{\text{max}}$ is a reference maximum (e.g., 100 TPM). This scales density to a 0–1 range.

14. **Transaction Timing Symmetry:** Evaluates the balance of transactions around the median time. One formulation is:

$$S = \left| \mu_{\text{left}} - \frac{T}{2} \right| + \left| \mu_{\text{right}} - \frac{T}{2} \right|,$$

where $\mu_{\text{left}}$ and $\mu_{\text{right}}$ are the means of transactions before and after the median, respectively.

# 3. FEATURE TRANSFORMATION AND DIMENSIONALITY REDUCTION

## 3.1 Dual-Source Feature Integration and Transformation

In our forecasting framework, we utilize two distinct sources of features:

1. **Blockchain-Derived Features (61 dimensions):**
   This dataset is generated from the blockchain and consists of 61 features engineered from detailed transaction-level and structural data. It captures multiple facets of blockchain activity—including node-based statistics, network structural properties, and temporal transaction dynamics—which are critical for understanding the underlying processes.

2. **Historical Market Data (11 dimensions):**
   This dataset includes key market indicators such as:

   - Total Volume (24h)
   - Current Price
   - Market Cap
   - 24h High
   - 24h Low
   - Price Change (24h)
   - Price Change Percentage (24h)
   - Market Cap Change (24h)
   - Market Cap Change Percentage (24h)
   - All-Time High (ATH)
   - ATH Change Percentage

   These features provide essential context regarding market performance and sentiment over the recent 24-hour period.

We apply the same transformations—such as differencing to achieve stationarity, standardization to normalize scales, and Principal Component Analysis (PCA) for dimensionality reduction—to both datasets. This uniform preprocessing ensures that the features from both sources are comparable and that the forecasting model can reliably capture the underlying trends and patterns.

## 3.2 Differencing

**Stationarity:** A time series is *stationary* if its statistical properties—specifically, its mean, variance, and autocorrelation—remain constant over time [7] . Formally, a series $\{y_t\}$ is stationary if for all $t$:

$$E[y_t] = \mu, \quad \text{Var}(y_t) = \sigma^2, \quad \text{and} \quad \text{Cov}(y_t, y_{t-k}) = \gamma_k,$$

where $\mu$, $\sigma^2$, and $\gamma_k$ are constant.

**Importance:** Stationarity is critical for forecasting models because these models assume that the underlying data do not change their behavior over time. When a series is stationary, past behavior is a reliable guide to future behavior, leading to more stable predictions and valid statistical inference. Non-stationary data, on the other hand, can result in false relationships and unreliable forecasts.

**Testing for Stationarity:** A common method to test for stationarity is the *Augmented Dickey-Fuller (ADF) test*. The ADF [7] test estimates the regression:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta y_{t-i} + \epsilon_t,$$

where:

- $\Delta y_t = y_t - y_{t-1}$ is the first difference of the series.
- $\alpha$ is a constant.
- $\beta t$ captures a deterministic trend.
- $\gamma$ is the coefficient of the lagged level of the series.
- $p$ is the number of lagged difference terms included.
- $\epsilon_t$ is an error term.

The null hypothesis $H_0 : \gamma = 0$ implies the series has a unit root (i.e., is non-stationary). If the test statistic is less than the critical value or if the p-value is below a chosen significance level (commonly 0.05), we reject $H_0$ and conclude that the series is stationary.

**Implementation via Differencing:** For any feature that is determined to be non-stationary, we apply *first-order differencing* to remove trends and stabilize the mean. The first-order difference is computed as:

$$\Delta y_t = y_t - y_{t-1}.$$

By replacing the original series with its differences, we reduce or eliminate trends, making the series more stationary and suitable for forecasting.

**Summary:** Stationarity ensures that a time series has consistent statistical properties over time, which is essential for reliable forecasting. We test each feature for stationarity (using tests like the ADF test) and, if necessary, apply first-order differencing:

$$\Delta y_t = y_t - y_{t-1},$$

to achieve stationarity. This process removes trends and stabilizes the series, leading to more robust and accurate forecasting models.

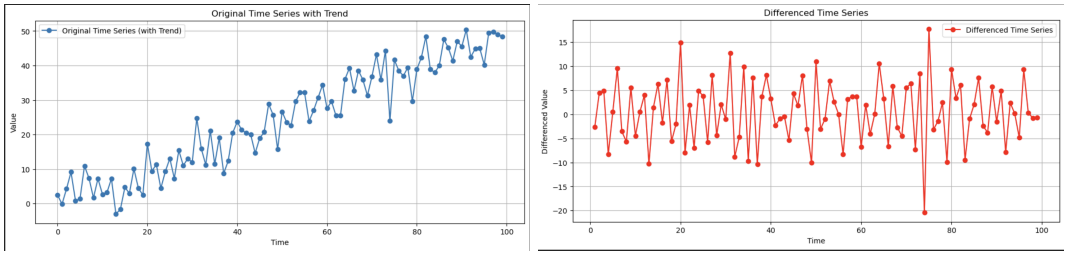



**Figure 3.1: Effect of differencing transformation.**

## 3.3 Standardization

Standardization is a data transformation technique that rescales each feature so that it has a zero mean and unit variance [8]. Mathematically, for a given feature value $x$, the standardized value $z$ is computed as:

$$z = \frac{x - \mu}{\sigma},$$

where $\mu$ is the mean of the feature and $\sigma$ is its standard deviation.

**Importance:**

Standardization ensures that all features contribute equally to the analysis by placing them on a common scale. This is especially important for methods such as Principal Component Analysis (PCA), which aims to capture the directions of maximum variance. Without standardization, features with larger scales might dominate the variance, leading to a distorted view of the data's underlying structure. In addition, many machine learning algorithms assume that data are centered and scaled, which can result in improved convergence and more reliable predictions.

**Global vs. Local Standardization:**

There are two main approaches to standardizing data:

- **Global Standardization:** The mean and standard deviation are computed over the entire collection of datasets (or a combined dataset) and these global statistics are then used to standardize each dataset. This approach is ideal when the datasets come from the same source, as it ensures consistent scaling across all datasets.
- **Local Standardization:** Each dataset is standardized independently by computing its own mean and standard deviation. While this method can be useful if datasets originate from different sources or contexts, it may lead to inconsistencies when comparing or integrating the data.

Given that our datasets are derived from the same source, we choose global standardization. This unified approach guarantees that all datasets are scaled consistently, which is particularly beneficial when applying dimensionality reduction techniques such as PCA.

### 3.4 Principal Component Analysis (PCA) for Dimensionality Reduction

Principal Component Analysis (PCA) is a statistical method [9] used to transform a dataset $X \in \mathbb{R}^{n \times p}$ into a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. Mathematically, PCA finds an orthogonal transformation matrix $P \in \mathbb{R}^{p \times p}$ so that

$$Y = XP,$$

where $Y$ is the transformed data. The columns of $P$ are the eigenvectors of the covariance matrix of $X$, and the corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_p$ represent the amount of variance captured by each principal component.

A key concept in PCA is the *variance threshold*. This threshold determines the number of principal components $k$ to retain such that the cumulative explained variance meets or exceeds a pre-specified level. Mathematically, we select the smallest $k$ for which

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{p} \lambda_j} \geq \theta,$$

where $\theta$ is the variance threshold (e.g., 90% or 95%). This ensures that the transformed dataset retains most of the original variability while reducing the dimensionality.

In our application, we have two distinct sources of features:

- A 61-dimensional feature set derived from blockchain data.

- A 11-dimensional feature set of historical market data.

We apply PCA globally to each dataset. Global PCA means that the transformation is computed on the entire dataset rather than on individual subsets. Because our datasets originate from the same source, global standardization and PCA ensure consistent scaling and that the variance captured is comparable across all observations.

The transformed dataset consists of linear combinations of the original features. These linear combinations retain the predictive value of the original features while reducing the dimensionality. This reduction is crucial for us, as we have limited data. Specifically, for the blockchain data, we set a variance threshold of 90%, which resulted in a reduction to 27 transformed features. For the historical market data, we used a variance threshold of 95%, yielding 7 transformed features.

### 3.5 Summary of Global Transformations Applied to Datasets

Our forecasting pipeline applies three key transformations to both our blockchain-derived dataset and our historical market data. These transformations are performed globally on each dataset to ensure consistent scaling and uniform treatment across all observations.

**1. Differencing:** Differencing is applied to remove trends and achieve stationarity in non-stationary features. By computing the difference between successive observations (i.e., $\Delta y_t = y_t - y_{t-1}$), we stabilize the mean of the time series. This step is crucial because many forecasting models assume that the underlying data is stationary.

**2. Standardization:** After differencing, all features are globally standardized so that each feature has a zero mean and unit variance. The standardized value is computed as:

$$z = \frac{x - \mu}{\sigma},$$

where $\mu$ and $\sigma$ are the global mean and standard deviation computed over the entire dataset. Global standardization is used because both datasets originate from the same source, ensuring that their scales are consistent and comparable, which is essential for downstream analyses.

**3. Principal Component Analysis (PCA):** Finally, PCA is applied globally to reduce the dimensionality of the datasets while preserving as much variance as possible. By finding linear combinations of the original features, PCA transforms the data into a lower-dimensional space. For example, the blockchain data (initially 61-dimensional) is reduced to 27 principal components at a 90% variance threshold, and the historical market data is reduced to 7 principal components at a 95%

variance threshold. This reduction is particularly important given our limited data, as it helps to mitigate overfitting while retaining the predictive value of the original features.

Together, these global transformations—differencing, standardization, and PCA—ensure that the data fed into our forecasting models is stationary, consistently scaled, and of reduced dimensionality, thereby enhancing model robustness and predictive accuracy.

# 4. EXPERIMENTAL MODELS AND EVALUATION

## 4.1 RNN Architecture

Given that our forecasting task involves time series data that is relatively straightforward in complexity, we opted for a Recurrent Neural Network (RNN) [10] architecture. The decision to use RNNs—specifically, LSTM and GRU variants—is based on the following considerations:

- **Temporal Dependency Capture:** RNNs are inherently designed to model sequential data. They efficiently capture temporal dependencies by maintaining a hidden state that evolves over time, making them well-suited for our forecasting task.
- **Simplicity and Efficiency:** Our data does not exhibit highly complex patterns or intricate seasonality. A simple RNN model is sufficient to capture the essential temporal patterns without the need for more elaborate architectures, such as transformers or attention-based models. This simplicity helps prevent overfitting, particularly given our limited dataset.
- **Interpretability:** The relative simplicity of RNNs like LSTM and GRU not only offers strong performance but also facilitates interpretability. This allows us to better understand how past observations influence future predictions.
- **Global Consistency:** Since our datasets originate from the same source, using a consistent, global RNN architecture ensures that the model leverages uniform temporal representations across all data, further stabilizing the forecasting process.

Overall, the choice of an RNN architecture strikes a balance between capturing the necessary temporal dynamics and maintaining model simplicity and efficiency, making it an ideal baseline for our forecasting application.

### 4.1.1 Fine-Tuning RNN Architecture for Forecasting

To select the optimal neural network for our forecasting task, we performed a comprehensive hyperparameter grid search over 128 model configurations. Our search space included the following parameters:

- **Batch Size:** Tested with 8 and 16, which affects how many samples are processed before updating the model.

- **Window Size:** Sequence lengths of 4 and 6 time steps, determining how much historical context is used.
- **RNN Type:** Comparison between Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures.
- **RNN Units:** The number of neurons per RNN layer, set to either 64 or 128, controlling model capacity.
- **Dropout Rate:** Values of 0.2 and 0.3 to mitigate overfitting by randomly deactivating neurons.
- **Weight Decay:** Fixed at 0, meaning no additional L2 regularization was applied.
- **Learning Rate:** Tested with 0.001 and 0.0005, which governs the step size during optimization.
- **Bidirectional:** Both unidirectional and bidirectional configurations were evaluated, where bidirectional RNNs process sequences in both forward and reverse directions.

All models were trained using MSE and the maximum number of training epochs was set to 50. Then, the weights corresponding to the lowest validation loss were selected (early stopping).

The fine-tuning process allowed us to systematically evaluate all 128 combinations and select in each case an RNN architecture that balances model complexity and generalization capability. In essence, the chosen model effectively captures the underlying temporal dynamics without overfitting, which is critical given our limited data.

### 4.1.2 Weighted Mean Squared Error Calculation

In our evaluation process, we compute the Mean Squared Error (MSE) for each of our six validation datasets separately. For a given dataset $d$ containing $n_d$ samples, the MSE is calculated as [10]:

$$\text{MSE}_d = \frac{1}{n_d} \sum_{i=1}^{n_d} \left( y_{\text{true},i}^{(d)} - y_{\text{pred},i}^{(d)} \right)^2 .$$

Since the number of samples varies across datasets (especially with different window sizes), we aggregate the errors using a weighted average. The overall (weighted)

MSE is computed by weighting each dataset's MSE by its number of samples, and then dividing by the total number of samples across all datasets:

$$\text{Weighted MSE} = \frac{\sum_{d=1}^{6} \text{MSE}_d \times n_d}{\sum_{d=1}^{6} n_d}.$$

This method ensures that datasets with a larger number of observations contribute more to the final error metric, thereby providing a more accurate and representative measure of the forecasting performance across all validation datasets.

### 4.1.3 Additional Error Metrics in Forecasting

In our evaluation framework, in addition to the Mean Squared Error (MSE) described previously, we also compute three other error metrics to obtain a comprehensive assessment of our forecasting performance. These metrics capture different aspects of the prediction error, which is crucial for understanding the strengths and weaknesses of our model [10].

#### Mean Absolute Error (MAE)

The Mean Absolute Error is defined as:

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

where $y_i$ is the true value and $\hat{y}_i$ is the predicted value. MAE provides a linear measure of error and represents the average absolute difference between predicted and actual values. This metric is particularly useful because it is less sensitive to outliers compared to MSE, offering a more intuitive understanding of the typical forecast error.

#### Mean Absolute Percentage Error (MAPE)

MAPE is given by:

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| .$$

This metric expresses the forecast error as a percentage of the true values, making it scale-independent and easy to interpret. It allows us to understand the error in relative terms, although it may be sensitive when actual values are close to zero.

**Symmetric Mean Absolute Percentage Error (sMAPE)**

The symmetric MAPE is defined as:

$$\text{sMAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2}.$$

sMAPE addresses some of the limitations of MAPE by symmetrically considering the absolute values of both the true and predicted values in the denominator. This formulation helps to mitigate issues when the true values are near zero and provides a more balanced view of the error.

**Importance in Forecasting:**

Each error metric offers unique insights:

- **MSE** is sensitive to large errors, penalizing them more heavily, which is useful when large deviations are particularly undesirable.
- **MAE** provides a straightforward average error that is robust to outliers, giving a clear measure of typical error magnitude.
- **MAPE** and **sMAPE** express errors as percentages, making them interpretable across different scales and allowing for easy comparison across various forecasting scenarios.

Together, these metrics ensure that our model's performance is evaluated from multiple perspectives, leading to a more robust and comprehensive assessment of its forecasting accuracy.

### 4.1.4 Data Preparation for (RNN) architecture

In order to select the best neural network architecture, we evaluated our models on six unseen datasets, each spanning 12 hours. For each scenario, we prepared the data using a sliding window approach. In the initial stages, each training sample consisted of a feature vector constructed from the engineered features and the true value from this time step. In the final step of data preparation—designed to simulate a live forecasting environment—we replaced the true target value with the average of the previous 3 or 5 true values (depending on the chosen window size).

This strategy reflects the real-world setup of our live algorithm, where the last observed values are used as inputs to predict future behavior. By using sliding windows and averaging recent observations, we ensure that the model receives a robust, temporally smoothed input, which is critical given the limited data and the need for accurate, real-time forecasts.

## 4.2 Baseline Models for Differenced Time Series Forecasting

In our forecasting framework, we consider several naive baseline models applied to a differenced time series. Differencing is intended to remove trends and stabilize the mean—often reducing it close to zero—so that the remaining fluctuations are largely random. Simple baselines in this context serve as reference points against which more sophisticated models can be compared. We discuss three baselines: the Last Observed Value, Live Forecasting (Cumulative Mean), and the Moving Average, providing their mathematical formulations and rationale.

### 1. Last Observed Value (Naive Forecast)

The simplest baseline is the **Last Observed Value** method [11], which predicts that the next value in the differenced series will be identical to the most recent observed value. Mathematically, if the differenced series is represented as $\{d_1, d_2, \ldots, d_t\}$, then the forecast for the next time step is:

$$\hat{d}_{t+1} = d_t.$$

**Rationale:** For a differenced series that is expected to hover around zero (if the original series was well-trended), using the last observed difference as the forecast is a natural baseline. It assumes no change in the difference, serving as a simple reference.

This model achieved a validation MSE of 0.000297, with additional performance metrics as follows:

- **Mean Absolute Error (MAE):** 0.010806
- **sMAPE:** 0.033001 (approximately 3.30%)
- **MAPE:** 3.318401%

### 2. Live Forecasting (Cumulative Mean)

In the **Live Forecasting** approach [11], the forecast at time $t+1$ is the running mean of all observed differences up to time $t$. This is expressed as:

$$\hat{d}_{t+1} = \frac{1}{t} \sum_{i=1}^{t} d_i.$$

**Rationale:** Since differencing often produces a series with a mean close to zero, the cumulative mean is a robust estimate of the central tendency. This baseline

adapts as new data arrives and is especially useful if there is any slight drift in the differenced series.

This model achieved a validation MSE of 0.001016, with additional performance metrics as follows:

- **Mean Absolute Error (MAE):** 0.050366
- **sMAPE:** 0.15353824 (approximately 15.35%)
- **MAPE:** 15.499464%

### 3. Moving Average

The **Moving Average** method uses a fixed-size window [11] of the most recent $w$ observations to forecast the next difference, in our case 4 and 6. Specifically, for a window of size $w$, the forecast is given by:

$$\hat{d}_{t+1} = \frac{1}{w} \sum_{i=t-w+1}^{t} d_i.$$

**Rationale:** The moving average smooths out short-term fluctuations by considering only the most recent observations. This is particularly useful in a differenced series where the signal might be noisy. It provides a local estimate of the central tendency, which can be compared against the global cumulative mean.

Each of these baselines is simple and interpretable. They serve as good starting points because they assume that the differenced series, ideally devoid of trends, will exhibit random fluctuations around a stable mean. Given that our series is differenced, these methods help to verify if more complex models truly capture additional predictive power beyond what these naive approaches provide.

This model achieved a validation MSE of 0.000157, with additional performance metrics as follows:

- **Mean Absolute Error (MAE):** 0.008095
- **sMAPE:** 0.024695 (approximately 2.46%)
- **MAPE:** 2.493405%

## 4. Best RNN Architecture Using Only Market Data

In order to evaluate the added predictive value of blockchain transaction graphs, we established a baseline model that relies solely on historical market data. This baseline is implemented using a Recurrent Neural Network (RNN) architecture that forecasts future market indicators based exclusively on past market data. By using only this historical market data, the baseline model sets a performance benchmark against which we can assess whether incorporating additional blockchain transaction graphs leads to improved forecasting accuracy.

Fine Tuning Results for Market Data Baseline Model

In this section, we present the fine tuning results for our fourth baseline model, which is the best RNN architecture using only historical market data. To evaluate the model's performance, we considered four evaluation metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), Symmetric Mean Absolute Percentage Error (sMAPE), and Mean Absolute Percentage Error (MAPE). The following four matrices summarize the top 5 model configurations for each metric. These results were obtained from a grid search over hyperparameters, and all configurations were tested globally on our market data.

**Table 1: Top 5 Models by MSE**

| Batch | Window | RNN | Units | Dropout | Bidir | MSE |
|---|---|---|---|---|---|---|
| 8 | 6 | LSTM | 128 | 0.3 | True | 0.000170 |
| 8 | 4 | LSTM | 64 | 0.3 | True | 0.000172 |
| 8 | 6 | LSTM | 128 | 0.2 | False | 0.000173 |
| 16 | 6 | GRU | 64 | 0.3 | False | 0.000177 |
| 8 | 4 | LSTM | 128 | 0.3 | True | 0.000193 |

**Table 4.1: Top 5 model configurations based on MSE.**

| Batch | Window | RNN | Units | Dropout | Bidir | MAE |
|---|---|---|---|---|---|---|
| 8 | 6 | LSTM | 128 | 0.2 | False | 0.009134 |
| 8 | 6 | LSTM | 128 | 0.3 | True | 0.009258 |
| 16 | 6 | GRU | 64 | 0.3 | False | 0.009290 |
| 8 | 4 | LSTM | 64 | 0.3 | True | 0.009651 |
| 16 | 4 | LSTM | 128 | 0.3 | False | 0.010285 |

**Table 4.2: Top 5 model configurations based on MAE.**

| Batch | Window | RNN | Units | Dropout | Bidir | sMAPE |
|---|---|---|---|---|---|---|
| 8 | 6 | LSTM | 128 | 0.2 | False | 0.026043 |
| 8 | 6 | LSTM | 128 | 0.3 | True | 0.027065 |
| 16 | 6 | GRU | 64 | 0.3 | False | 0.027872 |
| 8 | 4 | LSTM | 64 | 0.3 | True | 0.029493 |
| 16 | 4 | LSTM | 128 | 0.3 | False | 0.030762 |

**Table 4.3: Top 5 model configurations based on sMAPE.**

**Table 2: Top 5 Models by MAE**

**Table 3: Top 5 Models by sMAPE**

**Table 4: Top 5 Models by MAPE**

**Selected Model and Performance Overview**

Based on our comprehensive hyperparameter search, we selected the model with the best Mean Squared Error (MSE) as our fourth baseline using only historical market data. This model is our top performer and demonstrates the lowest error, or of the lowest errors, across all evaluation metrics. Its configuration is as follows:

- **Batch Size:** 8
- **Window Size:** 6
- **RNN Architecture:** LSTM
- **RNN Units:** 128
- **Dropout Rate:** 0.3
- **Learning Rate:** 0.0010

| Batch | Window | RNN | Units | Dropout | Bidir | MAPE |
|---|---|---|---|---|---|---|
| 8 | 6 | LSTM | 128 | 0.2 | False | 2.794361 |
| 16 | 6 | GRU | 64 | 0.3 | False | 2.836120 |
| 8 | 6 | LSTM | 128 | 0.3 | True | 2.839655 |
| 8 | 4 | LSTM | 64 | 0.3 | True | 2.923284 |
| 16 | 4 | LSTM | 128 | 0.3 | False | 3.122168 |

**Table 4.4: Top 5 model configurations based on MAPE.**

- **Bidirectional:** True

This model achieved a validation MSE of 0.000170, with additional performance metrics as follows:

- **Mean Absolute Error (MAE):** 0.009258
- **sMAPE:** 0.027065 (approximately 2.71%)
- **MAPE:** 2.84%

## 4.3 Final Model Architectures

In the final phase of our study, we developed two distinct forecasting models: one that uses only the blockchain-derived features and another that utilizes a concatenation of market data with blockchain data. These models were built using the best RNN architecture identified during our hyperparameter tuning process. The blockchain-only model captures the micro-level dynamics within the blockchain, while the combined model integrates both the structural and temporal nuances of blockchain data with the macro-level insights from historical market indicators.

Subsequently, we constructed several ensemble models that integrate the forecasts of the two models, the model that uses only the blockchain-derived features and the model that uses only the market-derived features. The ensemble approach aims to leverage the strengths of both individual forecasters, with the goal of improving overall prediction accuracy and robustness. This combination is particularly valuable, as it allows us to assess whether blockchain transaction graphs add significant predictive power over traditional market data alone.

### 4.3.1 Fine-Tuning Results for Blockchain Data Model

Below are four compact tables summarizing the top 5 model configurations for our blockchain-only baseline model based on different evaluation metrics. These results demonstrate that a similar model configuration consistently appears among the top performers.

**Table 1: Top 5 Models by MSE (Blockchain Data)**

| Batch | Window | RNN | Units | Dropout | Bidir | MSE |
|---|---|---|---|---|---|---|
| 24 | 8 | GRU | 128 | 0.2 | False | 0.000169 |
| 28 | 8 | GRU | 128 | 0.3 | False | 0.000175 |
| 46 | 8 | LSTM | 128 | 0.3 | False | 0.000180 |
| 8 | 8 | LSTM | 128 | 0.2 | False | 0.000181 |
| 5 | 8 | LSTM | 64 | 0.3 | True | 0.000191 |

**Table 4.5: Top 5 model configurations based on MSE for blockchain data.**

**Table 2: Top 5 Models by MAE (Blockchain Data)**

| Batch | Window | RNN | Units | Dropout | Bidir | MAE |
|---|---|---|---|---|---|---|
| 28 | 8 | GRU | 128 | 0.3 | False | 0.009140 |
| 24 | 8 | GRU | 128 | 0.2 | False | 0.009242 |
| 46 | 8 | LSTM | 128 | 0.3 | False | 0.009348 |
| 9 | 8 | LSTM | 128 | 0.2 | True | 0.009779 |
| 8 | 8 | LSTM | 128 | 0.2 | False | 0.009894 |

**Table 4.6: Top 5 model configurations based on MAE for blockchain data.**

**Table 3: Top 5 Models by sMAPE (Blockchain Data)**

**Table 4: Top 5 Models by MAPE (Blockchain Data)**

### 4.3.2 Selected Model and Performance Overview for Blockchain Data

After evaluating 128 model configurations on the blockchain data, we selected the model with the best Mean Squared Error (MSE) as our baseline. This model, which

| Batch | Window | RNN | Units | Dropout | Bidir | sMAPE |
|---|---|---|---|---|---|---|
| 24 | 8 | GRU | 128 | 0.2 | False | 0.027603 |
| 28 | 8 | GRU | 128 | 0.3 | False | 0.028029 |
| 46 | 8 | LSTM | 128 | 0.3 | False | 0.028949 |
| 9 | 8 | LSTM | 128 | 0.2 | True | 0.029219 |
| 16 | 8 | GRU | 64 | 0.2 | False | 0.030135 |

**Table 4.7: Top 5 model configurations based on sMAPE for blockchain data.**

| Batch | Window | RNN | Units | Dropout | Bidir | MAPE |
|---|---|---|---|---|---|---|
| 28 | 8 | GRU | 128 | 0.3 | False | 2.780350 |
| 46 | 8 | LSTM | 128 | 0.3 | False | 2.822991 |
| 24 | 8 | GRU | 128 | 0.2 | False | 2.847462 |
| 9 | 8 | LSTM | 128 | 0.2 | True | 2.990134 |
| 32 | 8 | LSTM | 64 | 0.2 | False | 2.997248 |

**Table 4.8: Top 5 model configurations based on MAPE for blockchain data.**

also ranks among the top 5 in MAE, sMAPE, and MAPE, has the following configuration:

- **Batch Size:** 24
- **Window Size:** 8
- **RNN Type:** GRU
- **RNN Units:** 128
- **Dropout Rate:** 0.2
- **Learning Rate:** 0.0010
- **Bidirectional:** False

This model achieved a validation MSE of 0.000169, with additional performance metrics as follows:

- **Mean Absolute Error (MAE):** 0.009242
- **sMAPE:** 0.027603 (approximately 2.76%)
- **MAPE:** 2.84%

The consistent performance across all metrics confirms that this configuration is robust and effective for forecasting using only blockchain data.

### Fine-Tuning Results for Combined Market and Blockchain Data Baseline Model

Below are the compact tables summarizing the top 5 configurations for our combined model. Each table shows the configuration parameters and the corresponding performance metric.

### Table 1: Top 5 Models by MSE

| Batch | Window | RNN | Units | Dropout | Bidir | MSE |
|---|---|---|---|---|---|---|
| 8 | 4 | GRU | 128 | 0.3 | False | 0.000181 |
| 8 | 6 | GRU | 128 | 0.2 | False | 0.000190 |
| 8 | 6 | GRU | 128 | 0.3 | False | 0.000194 |
| 8 | 4 | LSTM | 64 | 0.2 | True | 0.000196 |
| 8 | 6 | LSTM | 128 | 0.3 | True | 0.000222 |

**Table 4.9: Top 5 model configurations based on MSE for combined data.**

### Table 2: Top 5 Models by MAE

| Batch | Window | RNN | Units | Dropout | Bidir | MAE |
|---|---|---|---|---|---|---|
| 8 | 4 | GRU | 128 | 0.3 | False | 0.009745 |
| 8 | 6 | GRU | 128 | 0.2 | False | 0.009747 |
| 8 | 6 | GRU | 128 | 0.3 | False | 0.010578 |
| 8 | 4 | LSTM | 64 | 0.2 | True | 0.010626 |
| 8 | 6 | LSTM | 128 | 0.3 | True | 0.010816 |

**Table 4.10: Top 5 model configurations based on MAE for combined data.**

### Table 3: Top 5 Models by sMAPE

### Table 4: Top 5 Models by MAPE

### Selected Model and Performance Overview for Combined Data

Based on our fine-tuning results for the combined market and blockchain data model, the configuration with the lowest Mean Squared Error (MSE) is considered the best. In our MSE table, the best-performing model has the following configuration:

| Batch | Window | RNN | Units | Dropout | Bidir | sMAPE |
|---|---|---|---|---|---|---|
| 8 | 4 | GRU | 128 | 0.3 | False | 0.029821 |
| 8 | 6 | GRU | 128 | 0.2 | False | 0.029529 |
| 8 | 6 | GRU | 128 | 0.3 | False | 0.031826 |
| 8 | 6 | LSTM | 128 | 0.3 | False | 0.032364 |
| 8 | 4 | LSTM | 64 | 0.2 | True | 0.032705 |

**Table 4.11: Top 5 model configurations based on sMAPE for combined data.**

| Batch | Window | RNN | Units | Dropout | Bidir | MAPE |
|---|---|---|---|---|---|---|
| 8 | 4 | GRU | 128 | 0.3 | False | 3.006108 |
| 8 | 6 | GRU | 128 | 0.2 | False | 2.995781 |
| 8 | 6 | GRU | 128 | 0.3 | False | 3.209159 |
| 8 | 4 | LSTM | 64 | 0.2 | True | 3.251320 |
| 8 | 6 | LSTM | 128 | 0.3 | True | 3.317647 |

**Table 4.12: Top 5 model configurations based on MAPE for combined data.**

- **Batch Size:** 8
- **Window Size:** 4
- **RNN Type:** GRU
- **RNN Units:** 128
- **Dropout Rate:** 0.3
- **Learning Rate:** 0.0010
- **Bidirectional:** False

This model achieved a validation MSE of 0.000181, which is the lowest among the top 5 configurations. Additionally, its performance on other metrics is competitive:

- **MAE:** 0.009745
- **sMAPE:** 0.029821 (approximately 2.98%)
- **MAPE:** 3.006108%

The model's performance across all metrics confirms that this configuration is effective for forecasting using both market and blockchain data. But, the combined model, which uses both market and blockchain data, shows performance very similar to

the blockchain-only model. One plausible explanation for this is that the combined model has a higher number of parameters due to the increased input dimensionality. Given that our available data is limited, the additional complexity may lead to overfitting or simply insufficient learning capacity, meaning that the model isn't able to fully leverage the extra information from the market data. As a result, the added complexity doesn't translate into improved performance, and the model's predictive ability remains comparable to that of the simpler, blockchain-only model.

## 4.4 Ensemble Models

In this section, we introduce our ensemble approach, which combines the predictions of two distinct forecasting models. One model is based solely on blockchain data, while the other uses only historical market data. The goal of the ensemble is to leverage the strengths of both individual forecasters in order to improve overall prediction accuracy.

All models' parameters were trained in the training data, and later, evaluated on the 6 validation datasets.

### 4.4.1 Simple Averaging

In this approach, we construct an ensemble model by performing a simple averaging of the predictions from those two individual models. This simple averaging method assumes that each model contributes equally to the final forecast. It is an effective baseline for ensemble methods, especially when the individual models capture complementary aspects of the data.

### 4.4.2 Grid Search for Optimal Weights

In this approach, we seek to improve our ensemble predictions by finding the optimal weight combination for different models. The grid search procedure evaluates various candidate weight combinations and selects the one that minimizes the forecasting error in the training data, measured here by the Mean Squared Error (MSE).

Mathematically, if we have $m$ models with predictions $\hat{y}_i^{(j)}$ for $i = 1, \ldots, n$ (for each time step) and $j = 1, \ldots, m$, and a candidate weight vector $\mathbf{w} = [w_1, w_2, \ldots, w_m]$, the ensemble prediction is computed as:

$$\hat{y}_i = \sum_{j=1}^{m} w_j \, \hat{y}_i^{(j)}.$$

The grid search procedure systematically tests each weight combination in a predefined weight grid.

### 4.4.3 Linear Regression for Weight Optimization

In this approach, we train a linear regression model [12] to determine the optimal weights for combining the predictions of different models. Given a list of predictions from $m$ models, let the prediction of the $j$th model for the $i$th observation be $\hat{y}_i^{(j)}$. We form a feature matrix $X \in \mathbb{R}^{n \times m}$ by column-stacking these predictions, where each column corresponds to a model's predictions and $n$ is the number of observations. The true values are represented by the vector $y \in \mathbb{R}^n$.

The linear regression model aims to find a weight vector $\mathbf{w} \in \mathbb{R}^m$ and an intercept $b$ such that:

$$y \approx X\mathbf{w} + b.$$

The optimal weights are obtained by minimizing the mean squared error between the combined predictions and the true values. Mathematically, the optimization problem is:

$$\min_{\mathbf{w},b} \frac{1}{n} \sum_{i=1}^{n} \left( y_i - \left( \sum_{j=1}^{m} w_j \, \hat{y}_i^{(j)} + b \right) \right)^2 .$$

After training, the learned coefficients $w_1, w_2, \ldots, w_m$ represent the optimal weights for model averaging, and $b$ is the bias term.

This method is advantageous because it uses a data-driven approach to determine the contribution of each model.

### 4.4.4 XGBoost for Optimal Weighting

In addition to the linear regression ensemble, we explored an alternative ensemble method using XGBoost, a gradient boosting framework that is capable of modeling complex, non-linear interactions among the predictions of individual models.

**XGBoost** (eXtreme Gradient Boosting) is a highly efficient, scalable implementation of gradient boosting that builds an ensemble of decision trees [13]. It works by sequentially adding trees that correct the errors made by previous ones, optimizing a regularized objective function to reduce overfitting. XGBoost is well-known for its speed and performance, and it has been successfully applied to a wide range of machine learning problems, including regression and classification tasks.

**Key Hyperparameters:**

- **n_estimators:** This parameter determines the number of boosting rounds (i.e., the total number of trees to be built). Increasing this value can improve the model's performance, but too many trees may lead to overfitting. This value was set to 100.
- **learning_rate:** Also known as the shrinkage factor, the learning rate scales the contribution of each new tree. A smaller learning rate typically requires more trees but can lead to better generalization by preventing any single tree from having too much influence, here it was set to 0.05.
- **max_depth:** This parameter specifies the maximum depth of each decision tree. Deeper trees can capture more complex patterns but also risk overfitting the data. A moderate depth helps balance model complexity and generalization, in this case, where we have to combine 2 values, the maximum depth was set to 6, a small value .
- **objective:** For regression tasks, we use `reg:squarederror`, which means that the model minimizes the squared error between the predictions and the true values.

This approach is beneficial because it allows the ensemble model to learn non-linear combinations of the individual model predictions, potentially capturing complex interactions that a simple linear model might miss.

### 4.4.5 Residual Learning Ensemble Approach

In this approach, we use the blockchain model's predictions as a baseline and then correct these predictions by adding a residual term predicted by a separate market model [14].

Mathematically, if $\hat{y}^{(b)}$ denotes the predictions from the blockchain model and $\hat{r}$ represents the residuals predicted from the market features $X_{\text{market}}$ by the trained market model, the final corrected prediction is given by:

$$\hat{y} = \hat{y}^{(b)} + \hat{r}.$$

This strategy works as follows:

- The market model is trained to predict the residuals, that is, the differences between the true target values and the blockchain model predictions.
- During inference, the market model generates $\hat{r}$ using $X_{\text{market}}$, and this residual is added to the blockchain predictions $\hat{y}^{(b)}$ to obtain the final forecast.

By using residual learning, we are able to leverage the strengths of both models. The blockchain model captures the primary signal, while the market model compensates for systematic errors, potentially leading to more accurate overall forecasts.

### 4.4.6 Experimenting with Ensemble Models Based on Different Window Sizes

In this section, we describe our experimental approach to constructing and evaluating ensemble models using two different window sizes. Specifically, we built two types of ensemble models: one where both the blockchain and market models use a window size of 4, and another where both models use a window size of 6.

For each ensemble, we selected the best-performing blockchain-only model and the best-performing market-only model (as determined by our fine-tuning function) corresponding to the specific window size. The ensembles were then constructed by combining the predictions of these two models using our ensemble methods.

#### Ensemble Experiment for Window Size 4

For our ensemble experiments with a window size of 4, we constructed all ensemble models by combining two models:

1. A blockchain-only model with the following configuration:
    - **Batch Size:** 8
    - **Window Size:** 4
    - **RNN Type:** GRU
    - **RNN Units:** 128
    - **Dropout Rate:** 0.2
    - **Weight Decay:** 0.0
    - **Learning Rate:** 0.001
    - **Bidirectional:** False
2. A market-only model with the following configuration:
    - **Batch Size:** 8
    - **Window Size:** 4
    - **RNN Type:** LSTM
    - **RNN Units:** 64
    - **Dropout Rate:** 0.3

- **Weight Decay:** 0.0
- **Learning Rate:** 0.001
- **Bidirectional:** True

These configurations were determined to be the best-performing for each data source, based on our fine-tuning experiments. The market-only model architecture is also used in the residual learning model, but this time, it is trained using the residuals for the blockchain model's predictions.

### Score Matrix for Ensemble Models with window size 4

Below is the score matrix summarizing the performance of our ensemble models evaluated using all four error metrics.

| Model | MSE | MAE | MAPE | sMAPE |
|---|---|---|---|---|
| Simple Averaging | 0.000148 | 0.008053 | 2.466 | 0.024638 |
| Linear Regression Ensemble | 0.000154 | 0.008819 | 2.717 | 0.026839 |
| XGBoost Ensemble | 0.000142 | 0.007949 | 2.420 | 0.024272 |
| Grid Search Weighted Ensemble | 0.000146 | 0.008079 | 2.473 | 0.024858 |
| Residual Learning Ensemble | 0.000663 | 0.021891 | 6.783 | 0.064616 |

**Table 4.13: Ensemble Models Performance Metrics**

**Interpretation:**
Among the ensemble methods, the XGBoost Ensemble achieved the lowest MSE (0.00014175) and the lowest MAE (0.00794855), along with competitive MAPE and sMAPE scores. The Linear Regression and Grid Search Weighted ensembles also performed well, with similar error metrics. In contrast, the Residual Learning Ensemble yielded noticeably higher error values across all metrics, suggesting that the added complexity from residual correction may not provide additional predictive value given our limited data. Overall, these results indicate that simple ensemble strategies—particularly those using XGBoost or linear regression for optimal weighting, offer robust forecasting performance on our combined dataset.

### Ensemble Experiment for Window Size 6

For our ensemble experiments with a window size of 6, we constructed all ensemble models by combining two models:

1. A blockchain-only model with the following configuration:
    - **Batch Size:** 8
    - **Window Size:** 6
    - **RNN Type:** LSTM
    - **RNN Units:** 128
    - **Dropout Rate:** 0.3
    - **Weight Decay:** 0.0
    - **Learning Rate:** 0.0005
    - **Bidirectional:** False
2. A market-only model with the following configuration:
    - **Batch Size:** 8
    - **Window Size:** 6
    - **RNN Type:** LSTM
    - **RNN Units:** 128
    - **Dropout Rate:** 0.3
    - **Weight Decay:** 0.0
    - **Learning Rate:** 0.001
    - **Bidirectional:** True

These configurations were determined to be the best-performing for each data source, based on our fine-tuning experiments. The market-only model architecture is also used in the residual learning model, but this time, it is trained using the residuals for the blockchain model's predictions.

**Score Matrix for Ensemble Models with window size 6**

Below is the score matrix summarizing the performance of our ensemble models evaluated using all four error metrics.

**Interpretation:**
Among the ensemble methods with window size 6, the XGBoost Ensemble achieved the lowest MSE (0.00014175) again, while the lowest MAE (0.00794855) was achieved by simple averaging model. Grid Search Weighted ensembles also performed well, with similar error metrics. In contrast, the Residual Learning Ensemble yielded again noticeably higher error values across all metrics.

| Model | MSE | MAE | MAPE | sMAPE |
|---|---|---|---|---|
| Simple Averaging | 0.000151 | 0.008199 | 2.508 | 0.024796 |
| Linear Regression Ensemble | 0.000173 | 0.009282 | 2.846 | 0.028421 |
| XGBoost Ensemble | 0.000149 | 0.008255 | 2.513 | 0.025230 |
| Grid Search Weighted Ensemble | 0.000154 | 0.008428 | 2.578 | 0.025368 |
| Residual Learning Ensemble | 0.006651 | 0.076761 | 23.542 | 0.207768 |

**Table 4.14: Ensemble Models Performance Metrics**

## 4.5 Final Score Matrix for All Models

The table below summarizes the performance of all the models we experimented with, evaluated on four error metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), Symmetric Mean Absolute Percentage Error (sMAPE), and Mean Absolute Percentage Error (MAPE). This comprehensive score matrix allows us to compare the following models:

| **Model** | **MSE** | **MAE** | **sMAPE** | **MAPE** |
|---|---|---|---|---|
| Naive Forecast (Last Observed) | 0.000297 | 0.010806 | 0.033001 | 3.318401 |
| Live Forecasting (Cumulative Mean) | 0.001016 | 0.050366 | 0.153538 | 15.499464 |
| Moving Average | 0.000157 | 0.008095 | 0.024695 | 2.493405 |
| Best RNN (Market Only) | 0.000170 | 0.009258 | 0.027065 | 2.840000 |
| Blockchain-only RNN | 0.000169 | 0.009242 | 0.027603 | 2.840000 |
| Combined Model (Market + Blockchain) | 0.000181 | 0.009745 | 0.029821 | 3.006108 |
| **Ensemble (Window 4):** | | | | |
| Simple Averaging | 0.000148 | 0.008053 | 0.024638 | 2.466367 |
| Linear Regression Ensemble | 0.000154 | 0.008819 | 0.026839 | 2.716774 |
| XGBoost Ensemble | 0.000142 | 0.007949 | 0.024272 | 2.420492 |
| Grid Search Weighted Ensemble | 0.000146 | 0.008079 | 0.024858 | 2.473311 |
| Residual Learning Ensemble | 0.000663 | 0.021891 | 0.064616 | 6.782863 |
| **Ensemble (Window 6):** | | | | |
| Simple Averaging | 0.000151 | 0.008199 | 0.024796 | 2.508000 |
| Linear Regression Ensemble | 0.000173 | 0.009282 | 0.028421 | 2.846000 |
| XGBoost Ensemble | 0.000149 | 0.008255 | 0.025230 | 2.513000 |
| Grid Search Weighted Ensemble | 0.000154 | 0.008428 | 0.025368 | 2.578000 |
| Residual Learning Ensemble | 0.006651 | 0.076761 | 0.207768 | 23.542000 |

**Table 4.15: Final evaluation metrics (6 decimals) for all models.**

### Model Selection:

Based on the overall performance metrics, the **XGBoost Ensemble with a window size of 4** is the best performing model. It yields the lowest MSE, MAE, sMAPE, and MAPE among all ensemble methods, indicating that combining predictions through non-linear weighting using XGBoost effectively leverages the strengths of the un-

derlying models.

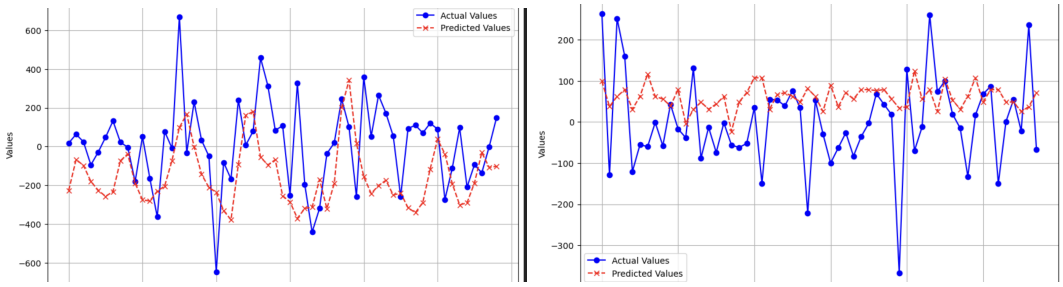



**Figure 4.1: XGBoost ensemble forecasts on validation dataset 1 and dataset 2.**

# 5. FINAL EVALUATION: TESTING AND TRADING SIMULATION

In this final chapter, we present a comprehensive evaluation of the best of our forecasting models. Our analysis is conducted on four distinct testing datasets to ensure robustness across varying market conditions. In addition, we implement a trading simulation to assess the practical impact of our forecasts on trading strategies. This chapter not only reports on the quantitative performance metrics of our models but also demonstrates their potential application in real-world trading scenarios. By combining model performance with simulated trading outcomes, we aim to provide a holistic view of the models' predictive and operational value.

## 5.1 XGBoost Ensemble Model Evaluation on Test Datasets

- In testing, the XGBoost ensemble achieved an aggregated MSE of

$$0.0005163425,$$

  computed over 4 test datasets. The per-dataset MSE is therefore:

$$\text{Per-dataset MSE} = \frac{0.0005163425}{4} \approx 0.00008605708.$$

- In validation, an aggregated MSE of 0.000142 was computed over 6 datasets, yielding a per-dataset MSE of:

$$\text{Per-dataset MSE} = \frac{0.000142}{6} \approx 0.00002366667.$$

- In testing, the XGBoost ensemble achieved an aggregated MAE of

$$0.004583,$$

  computed over 4 test datasets. The per-dataset MAE is therefore:

$$\text{Per-dataset MAE} = \frac{0.006583}{4} \approx 0.004583.$$

- In validation, an aggregated MAE of 0.007949 was computed over 6 datasets, yielding a per-dataset MAE of:

$$\text{Per-dataset MSE} = \frac{0.007949}{6} \approx 0.001325.$$

Moreover, the Mean Absolute Percentage Error (MAPE) is approximately 2.392621%, demonstrating that the model's forecasts are, on average, about 2.5% away from the true values. The Symmetric Mean Absolute Percentage Error (sMAPE) is about 0.028996, which reflects a balanced error measurement that treats over- and under-predictions equally.

Overall, these results suggest that the XGBoost ensemble model is highly accurate and robust.

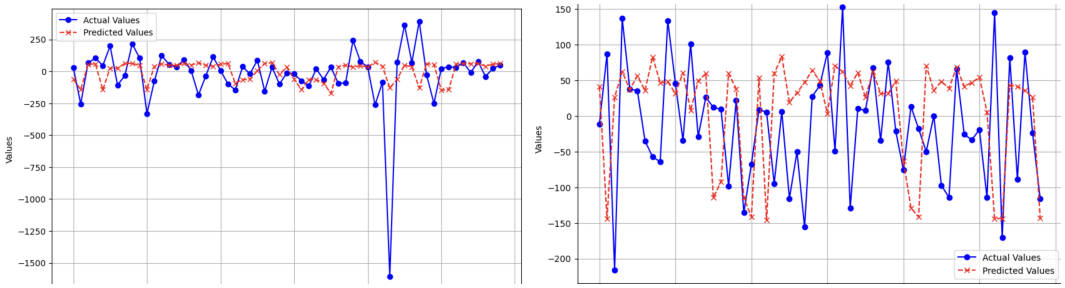



**Figure 5.1: XGBoost ensemble forecasts on testing dataset 1 and dataset 2.**

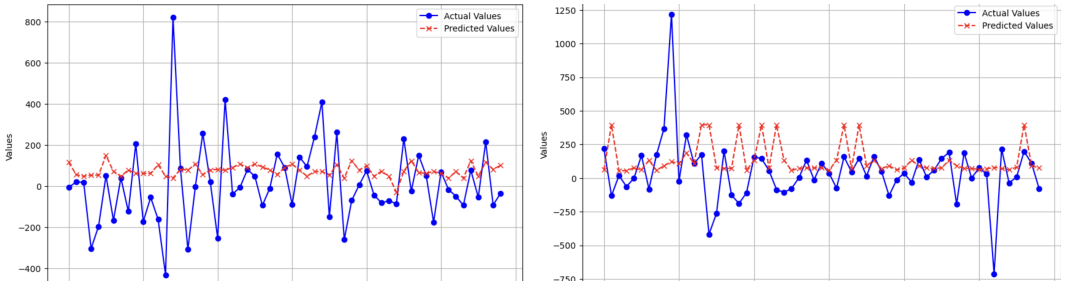



**Figure 5.2: XGBoost ensemble forecasts on testing dataset 3 and dataset 4.**

## 5.2 Trading Simulation

### 5.2.1 Buy and Hold Strategy

Buy and hold is a passive investment strategy where an investor purchases an asset and retains it over a long period, relying on its long-term appreciation rather than frequent trading [15].

For our trading simulation, we first evaluated a simple buy-and-hold strategy across four validation datasets. In each dataset, the strategy achieved the following profit multipliers:

- Dataset 1: $1.0004145446422157$

- Dataset 2: $0.9998319135772231$
- Dataset 3: $1.0032604231640307$
- Dataset 4: $0.9999233935560922$

To obtain the overall profit multiplier, we multiply these four values:

Overall Profit Multiplier $= 1.0004145446422157 \times 0.9998319135772231 \times 1.0032604231640307 \times 0.99992$

Evaluating this product gives approximately:

$$1.003430.$$

This result indicates that, over the evaluation period, the buy-and-hold strategy would have yielded an overall profit multiplier of about 1.00343, corresponding to a net profit increase of approximately 0.343%.

### 5.2.2 Moving Average Crossover Strategy

The moving average crossover strategy is a popular technical analysis method used to identify potential buy and sell signals. In this strategy, two moving averages are computed: one over a short (fast) window and another over a long (slow) window. A buy signal is typically generated when the fast moving average crosses above the slow moving average, indicating upward momentum, while a sell signal is generated when the fast moving average crosses below the slow moving average [15] .

For our experiment, we used a small window of 4 and a large window of 10, with a trading fee of 0.01% per trade. The strategy was evaluated on four datasets, achieving the following profit multipliers:

- Dataset 1: $0.897219$
- Dataset 2: $0.940046$
- Dataset 3: $0.966565$
- Dataset 4: $0.976200$

These profit multipliers, all below 1, indicate that the moving average crossover strategy resulted in losses across the datasets.

### 5.2.3 Threshold-Based Trading Strategy

In our threshold-based trading strategy, trades are executed based on predefined price change thresholds. In this approach, a buy signal is triggered when the predicted price change exceeds a positive threshold and a sell signal is initiated when the price change falls below a negative threshold. For my experiments, a trading fee of 0.01% per trade was applied to simulate realistic transaction costs.

After fine-tuning the thresholds, the strategy achieved the following performance on the test data:

- **Final Sell Price:** $67,642.32
- **Final Balance After Sell:** $9,985.32
- **Best Buy Threshold:** 0.0233
- **Best Sell Threshold:** -0.0500
- **Final Portfolio Value:** $10,099.74 (from an initial $10,000)

These results indicate that the threshold-based strategy based on the final forecasting model was able to generate a modest profit, increasing the portfolio value by approximately 0.9974% relative to the initial investment. This approach serves as an alternative trading strategy that relies on dynamic market thresholds, complementing the other strategies explored in our study.

# 6. DISCUSSION AND FUTURE WORK

## 6.1 Discussion of Results

In this project, we successfully developed a predictive live forecasting model for Bitcoin using limited and poor-quality data. Through a demanding process of feature engineering techniques, I extracted features from both blockchain transaction graphs and I combined them with historical market data. Despite the inherent data limitations, my ensemble approaches, particularly those based on XGBoost and linear regression for weight optimization, achieved competitive performance across multiple error metrics.

## 6.2 Future Work

While our current model shows promising results, there is significant margin for future improvements. With access to higher-quality data, more complex model architectures, such as advanced Graph Neural Networks (GNNs), could be exploited to capture deeper structural patterns in the blockchain data. Additionally, incorporating social media data, which plays a critical role in Bitcoin price fluctuations, may further enhance the model's predictive capabilities by providing insights into market sentiment and emerging trends. Future research should also explore the integration of additional data sources and the development of more sophisticated ensemble techniques to fully harness the rich, multifaceted nature of cryptocurrency markets.

# APPENDIX A. CODE AVAILABILITY

The complete code for this project, including data collection, preprocessing, model development, and evaluation, is publicly available in the following like. Interested readers can access, review, and run the code for further verification or experimentation at:

**Google Drive Repository:** bit.ly/4bhUMsT

This appendix provides the necessary details and instructions for reproducing the experiments described in this thesis. For any information contact me at george.grecas@gmail.com

# REFERENCES


[1] J. Reis and M. Housley, *Fundamentals of Data Engineering*. O'Reilly Media, 2022.

[2] Blockchain.info, "Blockchain.info websocket api," 2024. [Online]. Available: https://www.blockchain.info/api

[3] CoinGecko, "Coingecko api documentation," 2024. [Online]. Available: https://www.coingecko.com/en/api

[4] M. Fowler, *Designing APIs for Modern Applications*. Addison-Wesley, 2020.

[5] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, 2nd ed. New York: Dover Publications, 1998.

[6] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[7] J. D. Hamilton, *Time Series Analysis*. Princeton University Press, 1994.

[8] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, 1st ed. Springer, 2013.

[9] I. T. Jolliffe, "Principal component analysis," *Springer Series in Statistics*, 2002.

[10] A. C. Ian Goodfellow, Yoshua Bengio, *Deep Learning*. MIT Press, 2016.

[11] S. Makridakis, S. C. Wheelwright, and R. J. Hyndman, *Forecasting: Methods and Applications*, 3rd ed. John Wiley Sons, 1998.

[12] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*, 6th ed. Wiley, 2021.

[13] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 785–794, 2016. [Online]. Available: https://arxiv.org/abs/1603.02754

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. [Online]. Available: https://arxiv.org/abs/1512.03385

[15] E. P. Chan, *Algorithmic Trading: Winning Strategies and Their Rationale*, 1st ed. Wiley, 2013.