Development of an Analytical Support System to drive Advanced Visualisations

By

Jaskeerat Singh

School of Electrical Engineering and Computer Science

The University of Queensland

Submitted for the degree of Master of Data Science
Capstone Project 2

5th June 2024

Abstract

The sudden growth and worldwide promotion of digital currencies have transformed blockchain currencies like Bitcoin into a viable alternative to traditional payment methods. Concurrently, this rapid advancement has also heightened their use in various forms of cybercrime. This project aims to develop a real-time analytical support system to monitor, detect, and present anomalous transactions to relevant authorities. Unlike existing solutions that rely on static datasets to learn about the nature of such transactions, this system employs real-time streaming components to create a modern application to help support advanced visualisation techniques. The system was subjected to performance testing to ensure reliability using extreme scenarios that simulate real-life conditions. Then, different anomaly detection methods like DBSCAN clustering, isolation forest and other statistical methods were tested to gauge anomalous behaviour. This integration helped demonstrate its capabilities in detecting anomalous transactions while handling real-time data and providing a low latency response to drive advanced visualisations.

In this report, I will first describe the motivation for developing the project. Next, I will introduce the problem statement and discuss the related works. Following that, I will provide information about the real-time data used to develop this project. Subsequently, I will present the outcomes and the results of the system built and discuss its limitations and improvements for the future. Eventually, I developed a support system that performs anomaly detection, data processing, and facilitation to relevant authorities in real time while following the constraints and requirements of the whole application.

Executive Summary

The development and swift incorporation of blockchain technologies into daily life have hugely benefited cryptocurrencies like Bitcoin. The fast adoption of cryptocurrencies has been a boon for crypto scammers (Prosser, 2022). Bitcoin features like anonymity, decentralisation and lack of any authority for regulation have provided an easy way to mask fraudulent activities (Zola et al., 2019). This increase in crypto usage prompts the need for a real-time system to monitor, detect, and visualise anomalous transactions. In this report, I will design and assess the performance of a backend support system that will help drive advanced visualisations. Through this project, I aim to develop a system capable of handling real-time data, incorporating unsupervised machine learning methods to detect anomalous transactions, rapid processing and storage of real-time data, and provisioning low latency data access.

In this report, I will first evaluate existing solutions capable of handling real-time data. Subsequently, I will explore different machine learning methods that effectively find anomalous transactions in the financial sector. Finally, viable database solutions that help in fast data access and retrieval will be considered. Ultimately, this application will help law enforcement agencies uncover patterns and visualise the provenance flow of a fraudulent transaction to its origin.

To test the system's capability, I conducted a six-month study on real-time incoming data by performing stress testing on the different components of the system. In the first month, I analysed the data source by measuring the variability present in incoming data. Next, I committed the next two months to finding and integrating different data services to architect a cohesive application capable of handling the project's requirements and working under the defined constraints. Subsequently, I dedicated the next two months to evaluating different unsupervised machine learning methods and their capabilities in detecting fraud or anomalies in the present data (T. Pham & S. Lee, 2016). Ultimately, I spent time incorporating different systems, refining the architecture, and finding ways to provide low-latency data flow to dependent members.

In the study, source analysis was conducted to assess the variability in the incoming data. This is a crucial step, as it affects the selection of different real-time platforms. I identified fluctuations in the incoming data's time and size using point estimate visualisations, notably in the volume of transactions present in a Bitcoin block at any given time, as these parameters do not follow any predefined range. Next, due to the scarcity of a large amount of labelled data, different statistical and unsupervised machine learning methods like DBSCAN were evaluated and incorporated into the system. These techniques helped discover patterns and fluctuations in the incoming data. Additionally, various database solutions were stress tested, and their performance was checked. The selection of the NoSQL database because of its ability to handle incomplete data presents us with a range of flexibilities that make it a suitable solution for handling the project's requirements. The culmination of this project presents us with an application developed to manage the project's requirements effectively while working under the constraints defined. Given its flexibility, the application is modern and can be modified for future enhancement.

Contents

Abstract

Executive Summary

1. Introduction
    1.1. Project Motivation and Methodology
    1.2. Project Objectives and Contribution
    1.3. Privacy and Ethics
2. Related Works
    2.1. Real Time Streaming
    2.2. Big Data Storage Solutions
    2.3. Anomaly Detection
3. Dataset Overview
    3.1. Feature Set
    3.2. Data Characteristics
4. Proposed Solution
5. Project Preliminary
    5.1. KAFKA Configuration
    5.2. Neo4j
    5.3. Anomaly Detection
    5.4. Data Preparation
6. Methodology
    6.1. Data Ingestion
    6.2. KAFKA Processing
        6.2.1. KAFKA Clean Consumer
        6.2.2. KAFKA Processed Consumer
    6.3. Storage Configuration
    6.4. Anomaly Detection
        6.4.1. Address Node Analysis
        6.4.2. Z-Score
        6.4.3. Grubbs Test
        6.4.4. Isolation Forest
        6.4.5. DBSCAN Clustering
    6.5. Rest API
7. Results
8. Conclusion

References

1 Introduction

The extensive adoption and swift integration of digital currencies on different platforms have been a boon for cryptocurrencies. This development has placed crypto in a unique position where it is now considered an alternative payment method. However, this acceptance has attracted cybercriminals, who aim to exploit features such as anonymity, decentralisation, and robust security to manipulate the system (Brown, 2016). The growth of cryptocurrencies for the purpose of cybercrime highlighted the requirement for a real-time system to monitor, detect and provide low-latency data access for law enforcement agencies. Specifically, helping the agencies visualise the path of the provenance of suspicious transactions from the origin of Bitcoin data (Nakamoto, 2008).

In this report, I will architect and develop a support system that will be able to handle the requirements of the visualisations. First, I will investigate the variabilities in the incoming data that can affect the performance of a real-time system. Second, different statistical and machine learning methods will be evaluated to detect suspicious transactions that deviate from the normal. Third, pre-processing and storage of the incoming data in a concise and structured format to facilitate low latency data access to drive advanced visualisations. This UI aims to provide a latency-free experience for visualising money flow. Therefore, the proposed solution will be capable of monitoring and detecting suspicious transactions while providing a low-latency method of traversing the complex web of Bitcoin transactions by creating a path to provenance (Ahmed et al., 2018).

I will begin by addressing the motivation for the project and the methodologies followed to develop the system. Then, I will outline the objectives and requirements and mention any ethical and privacy issues. Subsequently, an examination of related works and methods will be considered. Next, the efficiency of the final iteration of the proposed solution will be discussed and how capable it is in handling the load and requirements of the system. Finally, I will discuss the outcomes, limitations of the system and work needed for future improvement.

1.1 Project Motivation and Methodology

To assess the performance of different real-time data processing components, I will address the questions that motivated this project:

- What standard components can be used to build a real-time system?
- What factors affect the system, and how can these effects be subdued?
- How can different anomaly detection methods be incorporated?

To achieve this, I will discuss these questions by detailing how they will be evaluated to fulfil the requirements to drive advanced visualisations. First, I will directly consume the incoming data from the Bitcoin node and optimise the data structure required for building these visualisations. Second, stress testing will be performed, and information regarding variabilities like time, space, and system performance will be considered while designing the solution (Sanla & Numnonda, 2019). I will discuss methods to help mitigate the factors affecting the system's performance. The developed application's performance was refined based on iterative performance testing. Third, I will discuss the different statistical and machine-learning methods incorporated into the system to find anomalous data points. Due to the unavailability of labelled data, specific methods that work by finding deviations in the data point from the normal were selected. Silhouette score was considered for measuring the performance of different clustering methods (Rousseeuw, 1987). Furthermore, to verify the performance of the support system, I propose the following objectives for this project.

1.2 Project Objectives and Contribution

The entire project is designed to support the development of advanced visualisations for tracking the provenance of suspicious transactions. This project is a part of a larger project, which was divided into four parts. The team consisted of four members, each working on developing independent parts of the project. The whole project consisted of:

- Setup connection, retrieval in human-readable format and modification of Bitcoin data
- Process, storage, anomaly detection of incoming data and backend system services
- Development of Supervised machine learning models to produce alerts based on historical data
- Deployment of user UI to visualise the provenance flow of suspicious transactions

While the whole project aims to solve the issue of provenance flow, my individual part of the project aims to develop a real-time anomaly detection support system and evaluate the factors that affect the performance of the application. To achieve this, the following objectives were laid:

- Identify the set of real-time streaming components to build the application that helps data flow throughout the system's different components.
- Integrate different parts of the system to provide a seamless experience to the user.
- Devise strategies that mitigate the effects of factors that affect performance
- Implement anomaly detection methods to identify anomalous transactions
- Provide latency-free performance to the end user

1.3 Privacy and Ethics

The ethical concerns related to data privacy and reliability do not affect this project as Bitcoin data is fully anonymised due to the inherent nature of blockchain. The data is collected from a publicly hosted Bitcoin full-node and is openly available. No sensitive data related to any individual was included in the incoming data (Androulaki et al., 2013)The data was collected using a Bitcoin ETL process, which follows ethical standards for generating and collecting data.

In the subsequent section, I will outline the currently available solutions and present issues I aim to solve.

2 Related work

In this section, I will discuss the pre-existing solutions that have worked to solve the objectives related to this project. First, I will discuss methods and components that aid in processing real-time data. Second, I will discuss different storage solutions developed to solve issues relating to big data. Third, I will review some anomaly detection methods for identifying anomalous transactions. Following this, I will discuss the challenges that exist in the existing methods from the view of cryptocurrency.

2.1 Real-Time streaming

The technologies available in the market can be divided into two categories: messaging protocols and streaming protocols. Messaging protocols are generally used for establishing communication channels, messaging, and data transfer (Gruber). In contrast, streaming protocols are used for data collection, integration with varied components, and complex computations on the move (Saxena & Gupta, 2017).

In this project, I evaluated various real-time systems that successfully handled large amounts of transactional updates in the financial sector. Under messaging protocols, web socket technology is used to set up a duplex long-lived connection. Web sockets help reduce latency and overhead by preserving a bidirectional connection between the source and the client. It proved advantageous in developing applications that follow trends and patterns in financial data as they ensured prompt data delivery and better user experience (Yinka-Banjo & Esther, 2019). One of the drawbacks of such a system is the resource-intensive nature of the technology, requiring a highly scalable environment to provide reliability in case of high workloads because of its stateful nature (Alexeev et al., 2019). To tackle this issue and enhance the scalability and reliability of the system, a queue system was discussed. RabbitMQ Is used for decoupling services and improving reliability by reducing server load. RabbitMQ helps manage the data flow by managing a data queue that temporarily stores data when the server is busy. This property helps in developing a microservice architecture that, in turn, improves the performance and response time of the system (Ćatović et al., 2022). The RabbitMQ protocol struggles to handle a large number of messages and is not horizontally scalable, which affects the system's latency and reliability in the long run (Rostanski et al., 2014).

Streaming protocols like Apache Storm were evaluated to overcome the issues faced by messaging protocols. Apache Storm helps in real-time data computation as it flows through the system. For instance, Apache Storm was implemented when the NewsAsset platform was redesigned to process and analyse millions of records from different social media platforms (Requeno et al., 2019). Apache Storms focuses on real-time computation rather than data flow, making it a better application for low-latency computations. However, because it cannot store temporary data, it is inferior regarding integration with different systems and fault tolerance (Liu et al., 2017).

To mitigate all such issues and concerns, Kafka was evaluated based on the objectives defined for the project. KAFKA is built for high throughput, low latency, and fault-tolerant data processing and transfer. It is a pub-sub system that can integrate with a host of systems (Garg, 2013). For instance, Kafka was used to develop a system to detect fraud in real-time credit card transactional data. The system integrated various machine learning classifiers to detect and identify fraudulent transactions (Kumar et al., 2021)

Hence, my solution will focus on developing a system that uses KAFKA technology to mitigate any issues with the application's latency, reliability, and scalability.

2.2 Big Data Storage Solutions

In this section, I have evaluated various storage solutions as real-time systems require a fast, reliable, and scalable solution that can handle large amounts of data while maintaining minimal latency. Different forms of databases—relational, NoSQL, and in-memory—were compared and evaluated to select a solution that fulfils the project's objectives.

Different storage options' performance and feature sets were reviewed based on different use cases. Under in-memory solutions, the Redis in-memory store was evaluated for its performance (Kabakus & Kara, 2017). Redis has very low latency and high throughput because it uses RAM instead of physical storage space. Its use in real-time trading platforms helps achieve a low-level latency of 90 milliseconds (Jain et al., 2019). Using RAM increases the system's usage cost and decreases its scalability. The Redis system is single-threaded, so it cannot be distributed and needs to be vertically scaled to improve performance.

Relational database features like data integrity, data segmentation, and complex querying capabilities have been extensively used in the market. The relational database's ability to handle complex data queries is important for complex analytics. The support of transaction queries helps ensure consistency across the data. For instance, hospital systems use transactions to ensure that the data is updated accordingly and data consistency and integrity are maintained for use cases like successfully updating patient prescriptions (Xu, 2009). However, the project's requirement to maintain high data availability under a given latency period makes it beyond the capabilities of simple SQL databases when handling big data (Seda et al., 2018).

NoSQL databases like MongoDB were assessed to check their capabilities to tackle the issues of scalability and low-latency data access. For this purpose, MongoDB is examined for its incorporation into various real-time systems because of its flexibility, scalability and reliability in handling big data. Its ability to handle large volumes of unstructured data was demonstrated in a project where it worked under a distributed architecture and provided quick query responses to help perform complex data processing tasks (Kim et al., 2013). The data stored in MongoDB supports hierarchical data storage, which might not support traversal between data points because of slow join performance between two different MongoDB documents (Dragonfly, 2023).

In the project motivations, I defined a need to develop a fast data retrieval system to interconnect multiple data points. For this purpose, Neo4j (a graph NoSQL database) was selected for its ability to store joins as relationships in the database, eliminating the need to make joins before querying data. This Neo4j property was used to analyse call records to understand user behaviour and detect anomalies by efficiently automating analysis in handling large volumes of interconnected data (Geepalla et al., 2018).

2.3 Anomaly detection

The current methods used for anomalous transaction detection in the context of Bitcoin can be divided into two categories: network analysis and price pattern fluctuation. Anomaly detection using network analysis considers the node and its neighbourhood features. This approach uses the network structure to enhance anomaly detection capabilities in the transaction network (T.-B. Pham & S. Lee, 2016). In contrast, price pattern matching uses historical data to measure variation in the data and find anomalous transactions (Shi et al., 2019).

The use of network structure to create network-specific features like in-degree, out-degree, and centrality measures and their combination with machine learning methods like local outlier factor and clustering methods help detect anomalies in the Bitcoin network (T.-B. Pham & S. Lee, 2016). This approach utilised two graphs, one containing user connections and the second consisting of transactions. The method effectively spotlighted the patterns that helped find anomalous transactions. The process utilised LOF to find a deviation from the immediate neighbourhood, while the global outlier found anomalies with respect to the entire dataset. Alternate studies combined random walks and graph features produced using transaction nodes and their neighbourhood to detect anomalies. The random walk traverses the network, selecting a path using random steps and probabilities, which helps find anomalies (Liao et al., 2020). The algorithm benefitted from the node and network-specific features, which led to improved anomaly detection.

Anomaly detection methods use fluctuations in the data to find anomalies. Multiple algorithms identify unusual patterns like frequency, seasonal, and trend changes. Such methods help detect unusual patterns and outliers in the time series data, which can be used to detect fraud in the incoming data (Schmidl et al., 2022). The study uses forecasting methods like ARIMA to forecast and find deviations in the data. The use of the collective anomaly approach has proved to be effective in catching anomalous activities in the Bitcoin data. Such methods focus on detecting groups rather than individual transactions. Such transactions differ significantly and do not share collective behaviour with any group (Shayegan et al., 2022).

In the next section, I will briefly overview the dataset used in the development and performance testing of the analytical backend system.

3 Dataset Overview

In the following section, I will discuss the dataset used in developing the project. To fulfil the project's objectives, I have used publicly available data received in real time from the Bitcoin node. This node is a full node hosted on the development system, which extracts data in a nested JSON format (*Running a full node*, 2015). This block data tests the configured system's performance and helps develop the anomaly detection method.

3.1 Feature Set

The data received from the Bitcoin Node, which is in the form of nested JSON, represents high-dimensional data consisting of 32 columns. This high-dimensional data even contains information used for the blockchain's data management and organisation. Such features were removed from the incoming data, and a final feature list was prepared for performance testing and anomaly detection.

Table 1 Features used for the development of the analytical support system

Feature	**Definition**	**Data Type**
Block_hash	Unique identifier of block	String
Height	Block number	Integer
Mediantime	Median timestamp of last 11 blocks	Integer
nTx	Number of transactions in the block	Integer
Previousblockhash	Hash of the previous block	String
Block_size	Size of the block	Integer
Time	Block creation time	Integer
Weight	Impact of witness data on block	Integer
Txid	Id of the transaction	String
Fee	Fees earned by miners	Float
Hash	Unique identifier of transaction	String
Locktime	Earliest time transaction can be confirmed	Integer
Size	Size of the transaction	Integer
total_bitcoin_transacted	Total amount of bitcoin transferred	Float
Vin	List of Inputs	List of Dictionary
Vout	List of Outputs	List of Dictionary
Transaction_type	Type of the transaction	String
Value	Amount of bitcoin transacted per vin or vout	Float
Address	Address involved in the transaction	String

The “vin and vout” define the relationship connection for the movement of Bitcoin from one transaction address to another. A transaction has multiple vin’s and multiple vout’s. These vin and vout nodes represent the sub-transaction that divides the bitcoin value and passes it between the addresses.

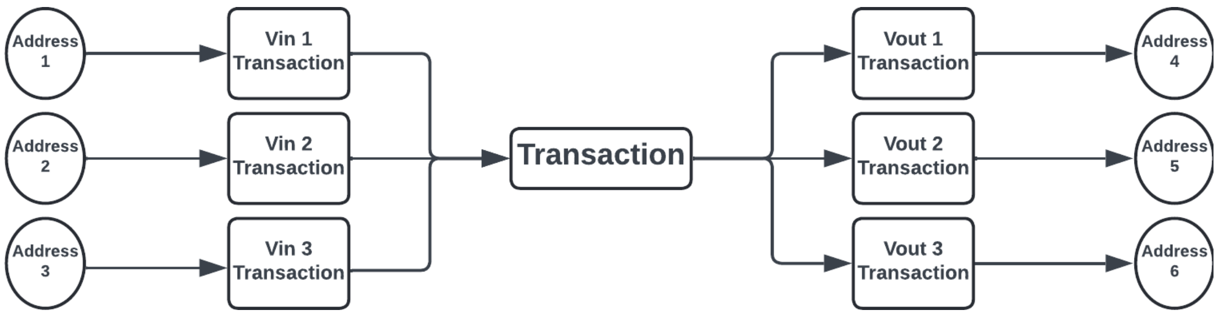


Figure 1 Single transaction representation

3.2 Data Characteristics

In this section, I will delve into the characteristics of the incoming block data. This is necessary to ensure the system's robustness. Various dataset attributes, like transaction velocity, transaction volume, and transaction veracity, are vital in identifying the variables affecting the system's performance. For the analysis, 100 consecutive blocks of Bitcoin data were collected over 24 hours. Through this analysis, I was able to define the constraints under which the proposed solution will be developed.

Size Variabilities

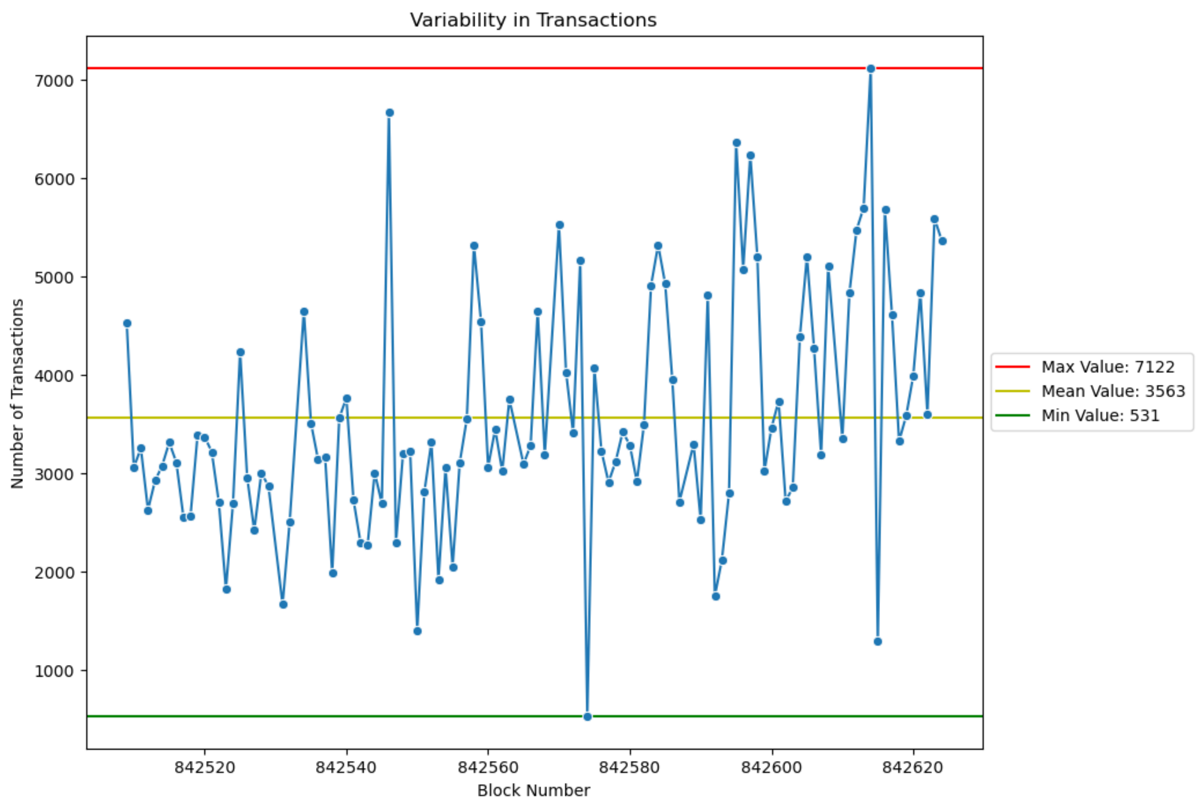


Figure 2 Variability in the number of transactions

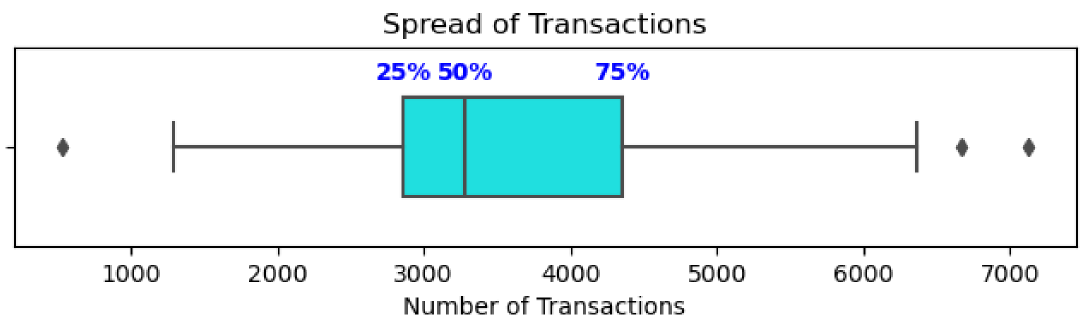


Figure 3 Box Plot representing the transaction quantity distribution

Through analysis, we can see that there does not exist a defined range in which the number of transactions occurs. In our sample, we can see that –

- The average number of transactions in a block lies close to 3500.
- The maximum number of transactions in a block equal approximately double the average value and stands at 7100. However, blocks with such many transactions are rare and comprise only 4% of the total blocks analysed.
- The smallest block received during analysis is close to 500 transactions.

Time Variabilities

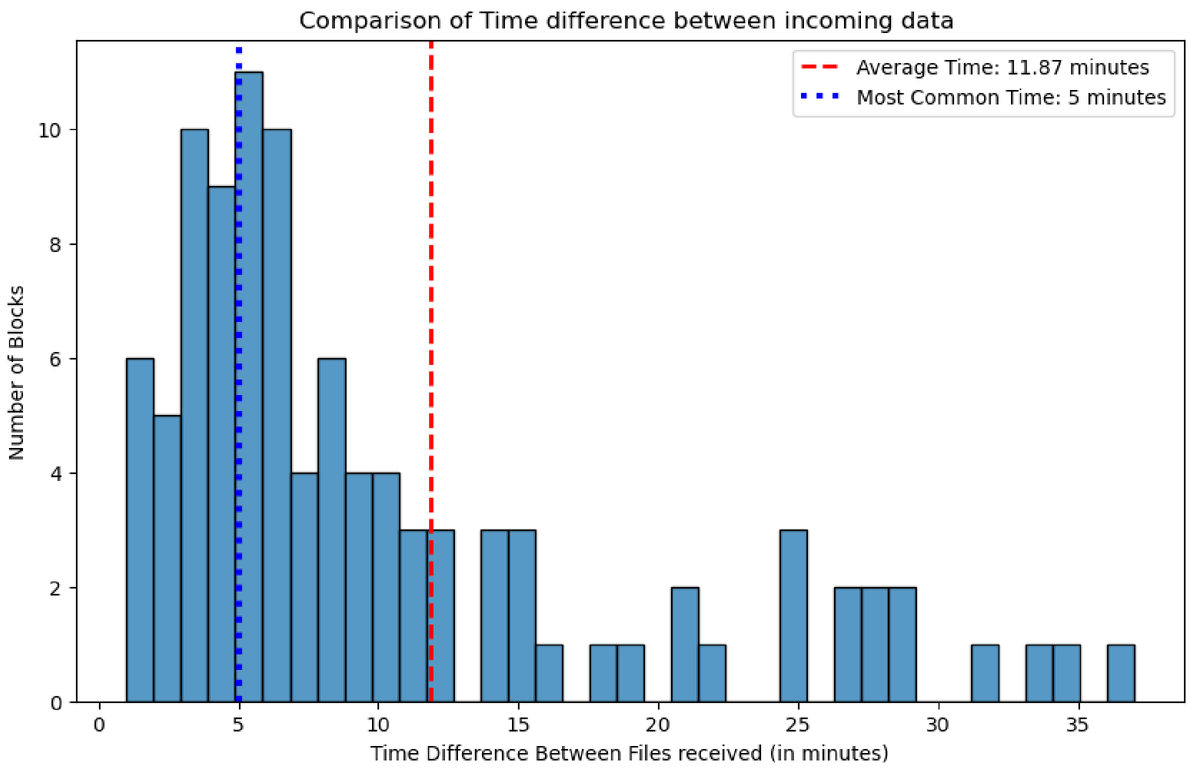


Figure 4 Time difference between each incoming block

In this analysis, the histogram was graphed to measure the time difference between incoming blocks of Bitcoin data. From the analysis, we can see a –

- Right-skewed histogram suggesting that most of the data is concentrated on the left.

- Most of the blocks were received at a time difference of 5 min.
- The average time elapsed between each incoming block is approximately close to 12 min.

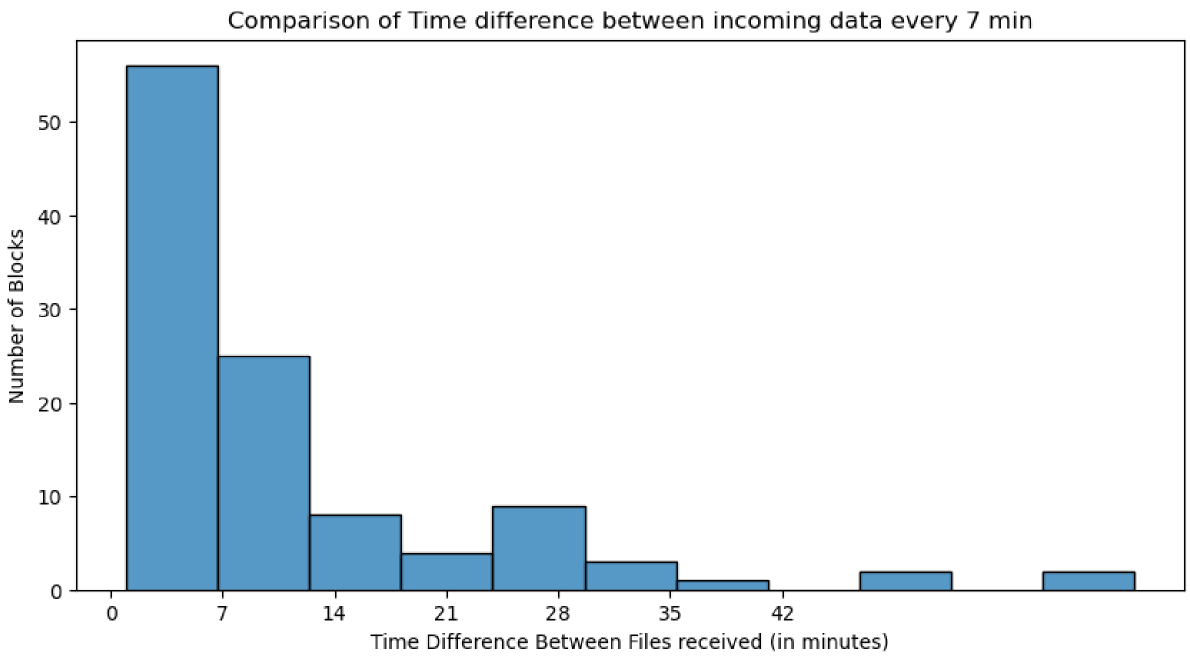


Figure 5 Most of the blocks are received every 7 min

Through a time-measured analysis of 7 min, we can see a right-skewed histogram suggesting that a large number of blocks arrive within a short time span of the previous block. Furthermore, more than half (50%) of the bitcoin blocks were received every 7 min.

Data Quality Variabilities

The data received from the Bitcoin node, presented in a structured nested JSON format, is of high quality. There are no data discrepancies in the incoming data. The data in the incoming block are free from nulls, and all the features have a pre-defined data type. The only instance of nulls in the system occurs when the Bitcoin value is used up, and nothing can be passed to any vout address. In such cases, no relationship is defined while processing and storing the data. This high data quality ensures the reliability of our findings for our audience of data analysts, researchers, and stakeholders.

In the following section, I will present an overview of the proposed support system solution, which aims to fulfil the project motivation while following the project requirements and working under the constraints defined.

4 Proposed Solution

In this section, I will discuss the solution proposed for handling the project's requirements. This architecture was carefully defined after many rounds of discussion, considering the constraints of the data source.

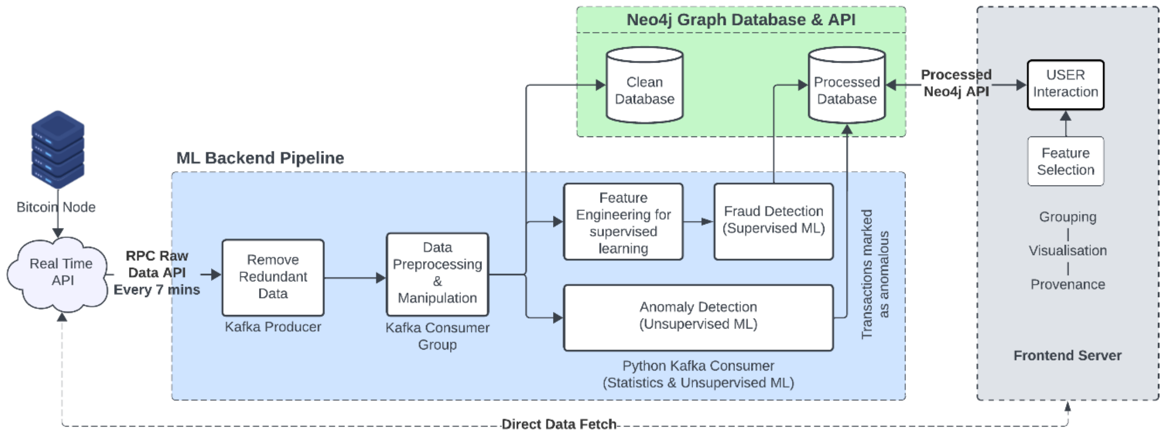

Neo4j Graph Database & API
Clean Database
Processed Database
Processed Neo4j API
USER Interaction
Feature Selection
Grouping
Visualisation
Provenance
Frontend Server
Bitcoin Node
Real Time API
RPC Raw Data API Every 7 mins
ML Backend Pipeline
Remove Redundant Data
Kafka Producer
Data Preprocessing & Manipulation
Kafka Consumer Group
Feature Engineering for supervised learning
Fraud Detection (Supervised ML)
Anomaly Detection (Unsupervised ML)
Transactions marked as anomalous
Python Kafka Consumer (Statistics & Unsupervised ML)
Direct Data Fetch

Figure 6 Proposed architecture

The whole project was divided into four parts, and each team member handled individual components of the system. This integrated pipeline handles the system's requirements by allowing data to flow from one component to the other. In this proposed solution, a real-time API serves as a point of contact with the Bitcoin node. The Bitcoin node will publish data every 7 minutes. The data will be fetched using the Kafka producer, temporarily stored, and distributed accordingly to the consumers subscribed to the Kafka producer.

The Kafka consumer subscribes to the Kafka producer and consumes data directly from the producer. Before the data is passed using the Kafka producer, the data will be pre-processed, and only the relevant features required will be passed. The number of Kafka consumers doesn’t affect the data distribution, as each consumer will receive the same data without any data discrepancy. This method will pass data to a Kafka-clean consumer, supervised machine learning Kafka-processed consumer and anomaly detection consumer. These processes will receive the data in parallel, thus providing a robust, reliable and scalable application.

After receiving the data from the producer, fully cleaned data will be stored in a Neo4j database based on the storage schema defined. The same pre-processed data will be passed to the supervised machine learning model, which generates alerts for incoming transactions in the block. Here, a probability score will be marked, and the pruned information regarding the incoming transaction required to build the advanced visualisations will be stored in the processed neo4j database. The processed information is saved according to the schema created for fast data retrieval and data flow to the end user.

In the third process, the data is passed to the anomaly detection methods, where incoming transactions are used to find anomalous transactions that deviate much from the normal. Here, unsupervised machine learning methods and statistical methods are used to find fluctuations and patterns in the historical data. By temporarily keeping historical data available for a defined range, different patterns and fluctuations can be measured and transactions falling out of the normal can be marked as anomalous. The final processed data is made available to the end user using a host of API endpoints.

Goals

In this section, I will list objectives based on the proposed support system solution. These objectives aim to aid the development of advanced visualisations.

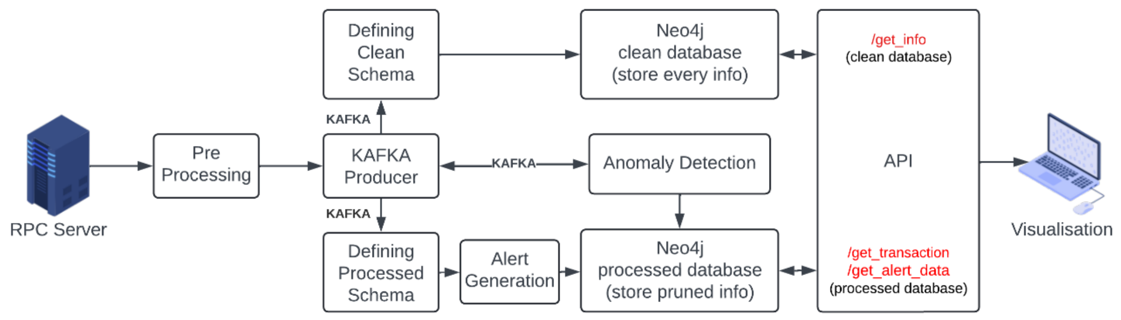


Figure 7 Support system architecture

- Retrieve data from Bitcoin RPC API and pre-process the incoming data
- Reduce the time from when new data becomes available and when it is retrieved by the system
- Determine the Kafka configuration for partitioning, replicating and distributing the data
- Define Kafka producer to forward the data to all the subscribed consumers
- Develop schema for storing full data available in neo4j
- Perform Anomaly detection to find anomalous transactions that deviate from the normal
- Integrate supervised machine learning models into the system to produce alerts for incoming transactions
- Develop schema to store processed data with only relevant features
- Host API to share data with the frontend USER UI system and create filters for faster reading of data

In the upcoming section, I will discuss the background knowledge required to configure and design the proposed solution. Here, I will outline the project requirements and how each component works to address those requirements.

5 Project Preliminary

In this section, I will briefly outline the background knowledge needed to build the analytical pipeline to satisfy the project's requirements. This will include the configuration settings of different components like Kafka and Neo4j, the method devised for performance testing as well as the metrics employed for measuring the performance of different statistical and machine learning methods.

5.1 KAFKA Configuration

In this segment, I will discuss the workflow of Kafka, which will help us design the system's architecture and fulfil the project's requirements.

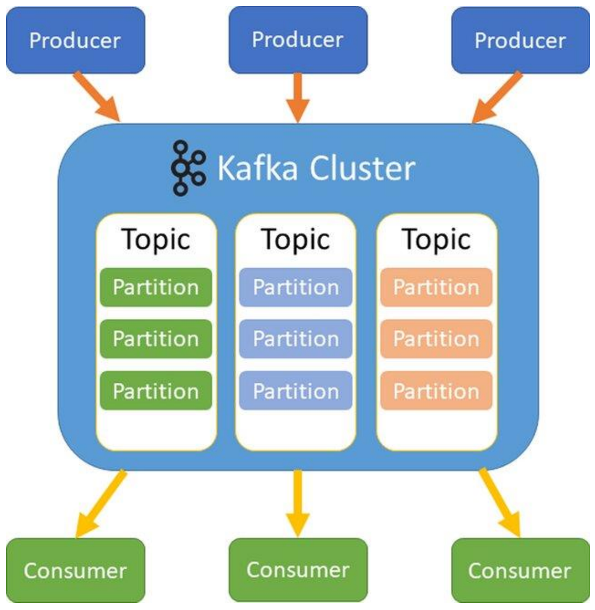


Figure 8 Kafka operation architecture (Liu et al., 2023)

The architecture of Kafka is distributed to handle large amounts of data in an efficient and reliable manner. Kafka runs on a cluster of brokers, where each broker is a server that handles requests between the producer and consumer and acts as temporary data storage for a configurable time period. Kafka brokers use the Kafka topics to partition the data and create data flow to the Kafka clients. In the later sections, I will describe the different combinations of configurations available in the application to serve the system's requirements.

The 3 main types of configurations that affect the performance of the system are –

- Partitioning: Reduce the processing time of the incoming data by introducing parallelism
- Replication: In the event of failure, ensure durability and availability by replicating a copy of the incoming data on multiple brokers (cluster)
- Retention: Management of data for a specified range of time, allowing for a period where data can be processed again in case of failure

As a brief overview, partitioning uses Kafka topics on which the producer produces the data. The consumer subscribes to such topics to retrieve the data. The number of partitions configured in the topic bifurcates the data accordingly and maintains a one-to-one mapping between the partition and the consumer part of the consumer group. Therefore, the consumer group, which is a group of consumers, jointly help in data consumption in parallel. This architecture helps improve the horizontal scalability of the system and introduces load balancing into the system by constant optimisation and rebalancing to divide the data (Vohra, 2016).

Multiple brokers help maintain the durability and availability of data across the system by replicating the data across each broker. If one broker fails, other brokers with replicas take over, ensuring high availability and no data loss (Wu, 2019). Kafka uses this property to temporarily store data for a defined period of time. When the time-based policy attached to the data expires, the old data is removed from the system. This feature helps manage system resources and allows data re-processing in case of failure.

As a result, I have chosen Kafka as the central part of my project because of its feature set, which helps reduce the time to process and ensures reliability in case of failure.

5.2 Neo4j

In this segment, I will discuss the workings of Neo4j. Neo4j, a graph NoSQL database, is designed to traverse a complex network of transactional data. It efficiently handles a complex web of nodes, relationships between the nodes, and properties associated with the nodes and relationships. These properties store the data using a key-value pair.

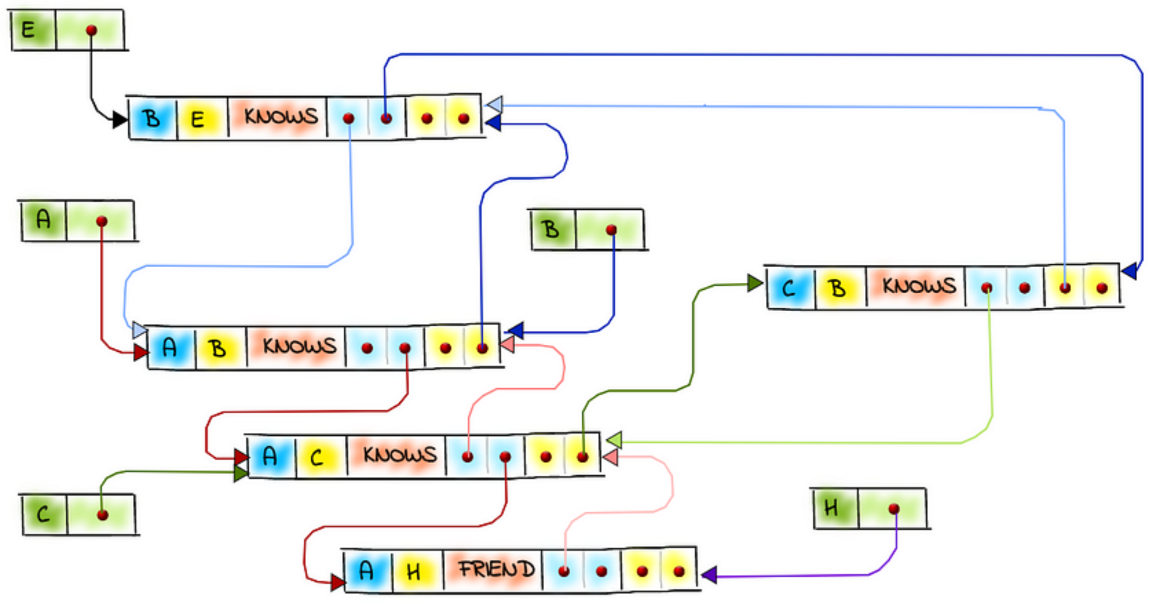


Figure 9 Neo4j uses a doubly linked list (Singh, 2022)

The nodes and relationships are assigned unique identifiers, making it easier for Neo4j to refer to them in the database. The backend system in Neo4j uses a doubly linked list to connect the nodes and relationships. Each element in Neo4j has a consistent data management format consisting of graph ID and element ID. Every unique node maintains a pointer to its associated relationships and vice versa. This data structure eliminates the need for costly table joins between huge datasets by focusing on using direct pointers for traversal with a constant time complexity of O(1).

As a result of this architecture, I have chosen Neo4j as my storage solution, as the system's performance during complex query execution will not be bogged down by costly joins.

5.3 Anomaly Detection

In this segment, I will discuss the theory of different statistical and unsupervised machine-learning methods for detecting anomalies in real-time incoming data. Methods like Z-score analysis, Box plot, and unsupervised methods like clustering and isolation forest are available for detecting anomalous transactions.

In a brief overview, statistical methods like Z-score and boxplot using IQR help isolate data points that deviate significantly from the dataset's normal behaviour. These methods use point estimates like standard deviation and mean to find patterns and fluctuations in incoming real-time data.

Z-score: The process consists of standardisation and threshold setting. Using the standardisation technique, a score is calculated, and the absolute score value above the set threshold is marked as anomalous. Here, x is the data point, μ is the mean of the sample population, and σ represents the standard deviation. Any value lying beyond ± 2*standard deviation is taken as the anomaly (Rousseeuw & Hubert, 2018). This threshold is determined by domain knowledge.

$$Z = \frac{x-\mu}{\sigma}$$

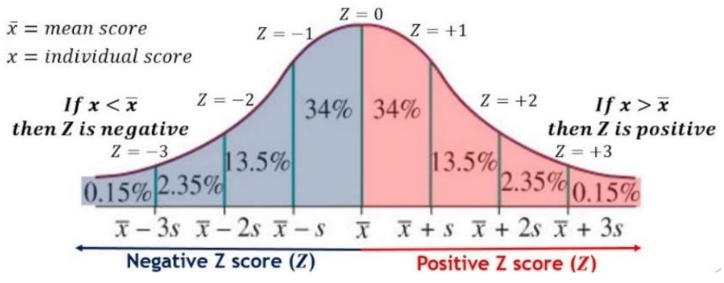


Figure 10 Z-score formula and standard deviation distribution (Kumar, 2023)

Grubbs Test: The method is used to detect outliers in a dataset that follows a normal distribution. The test statistic for the Grubbs test is calculated using:

$$G_{\max} = \frac{|X_{\max} - \bar{X}|}{s} \qquad G_{\min} = \frac{|X_{\min} - \bar{X}|}{s}$$

Figure 11 Grubbs Test formula

Here, X_{max} and X_{min} represent the dataset's largest and smallest values, respectively. $\bar{x}$ represents the mean of the dataset, and s represents the standard deviation. The greater G value will be taken as the threshold of the system where G equals the greater of G_{max} and G_{min} (Grubbs, 1969).

When a large amount of labelled data is unavailable, unsupervised techniques play a vital role in detecting anomalies that deviate from normal patterns. Some unsupervised techniques, like clustering and isolation forests, help label outliers without the presence of training datasets.

Clustering: The process involves grouping objects to form clusters (Rui & Wunsch, 2005). The method identifies similar objects and keeps them together. An object in a cluster will be more similar to objects in its cluster than objects of other clusters. The clustering techniques can be divided into three common types – k-means, hierarchical and DBSCAN. In K-means clustering, the algorithm divides the full dataset into a pre-defined number of clusters that do not overlap. The algorithm places data points iteratively into the cluster and tries to minimise the distance of a point from the centroid in each cluster created (Sharma, 2023). In hierarchical clustering, a tree of clusters is created from the data points. The process is computationally expensive when working with large datasets, but a pre-defined number of clusters is not required to create the dataset (Abdullahabrar, 2023). The method is also sensitive to outliers in the dataset.

In DBSCAN, the algorithm adds data points incrementally and then finds the cluster's core points, border points, and noise. The clusters are created based on the values of the selected hyperparameters, and such values are assigned to the data points.

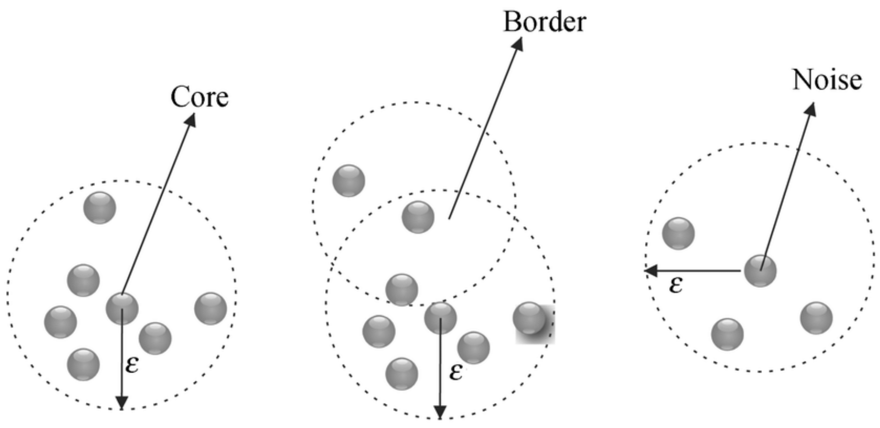


Figure 12 DBSCAN: core, border and noise points (Amini et al., 2014)

DBSCAN works well as the number of pre-defined clusters is not decided, and the value of the algorithm is computationally fast and inexpensive. The method group points that are closely packed (Thailappan, 2021). This property helps in identifying outliers that lie in low-density regions.

Isolation Forest: The algorithm works well where the dimensionality of the dataset is quite high and is created to isolate anomalies in the data (Akshara, 2021). The process involves building multiple decision trees to isolate data points from each other. A random feature is selected in every individual tree, and a random value is chosen to split the data. This process is repeated until all the data points are isolated from each other. The idea is such that the anomalies will have a shorter path when calculated from the root node and get isolated earlier in comparison to regular data points whose paths will be bigger and require more splits in the feature selected. A scoring function converts path lengths into anomaly scores and the value of the threshold decides if a certain data point can be defined as an anomaly.

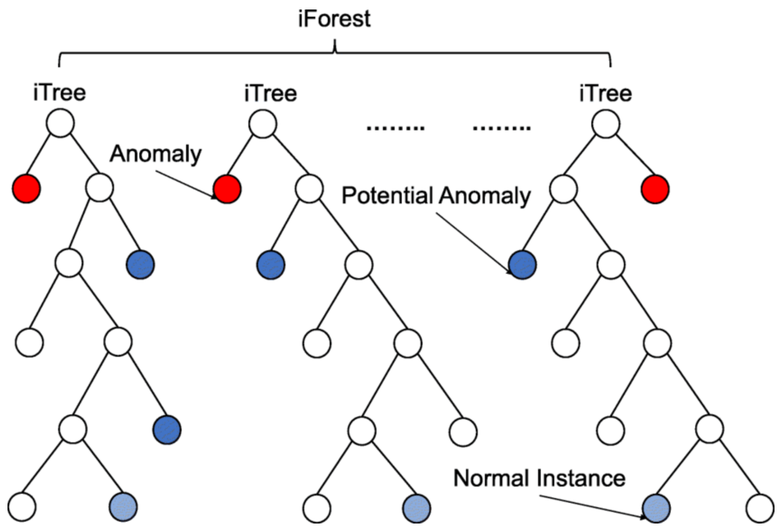


Figure 13 Isolation forest representation (Regaya et al., 2021)

5.4 Data Preparation

As introduced in Section 3, The real-time dataset used in the development of this project consists of 32 features. While the dataset is categorised as high dimensional data, many of the features are redundant and contain hashed values. Features like chainwork, hash, , previousblockhash, hex, coinbase and txinwitness do not serve any purpose but affect the performance of the pipeline. For this reason, such features are removed before passing through the Kafka pipeline. Any null value in the incoming data is replaced with an empty string and features like transaction value are standardised before analytical processing.

In the following section, I will demonstrate the implementation of the different system components and discuss the optimal solution finalised for creating the support system.

6 Methodology

In this section, I will present the project's experiment outcomes and discuss how the results were calculated. I will identify and outline the problem statement for each of the system's individual components. Then, each section will be followed by the method devised for performance testing. The performance testing results will be evaluated, and the best configuration will be selected to fulfil the system's objectives.

6.1 Data Ingestion

In this segment, I will discuss the first step of the analytical support system pipeline that needs to be optimised to fetch and consume real-time incoming data. I will also discuss the connection to the data source and how it is configured to provide a low-latency service.

Through analysis, we know from section 3 that a gap of 5 minutes exists between each incoming block of data. At least 50% of the incoming blocks are received with a 7-minute time difference. Using this arrival time logic, if the data source is polled every 7 minutes, a time delay of approximately 3.5 minutes is introduced between when the data became available on the data source side and when it was ingested by the pipeline.

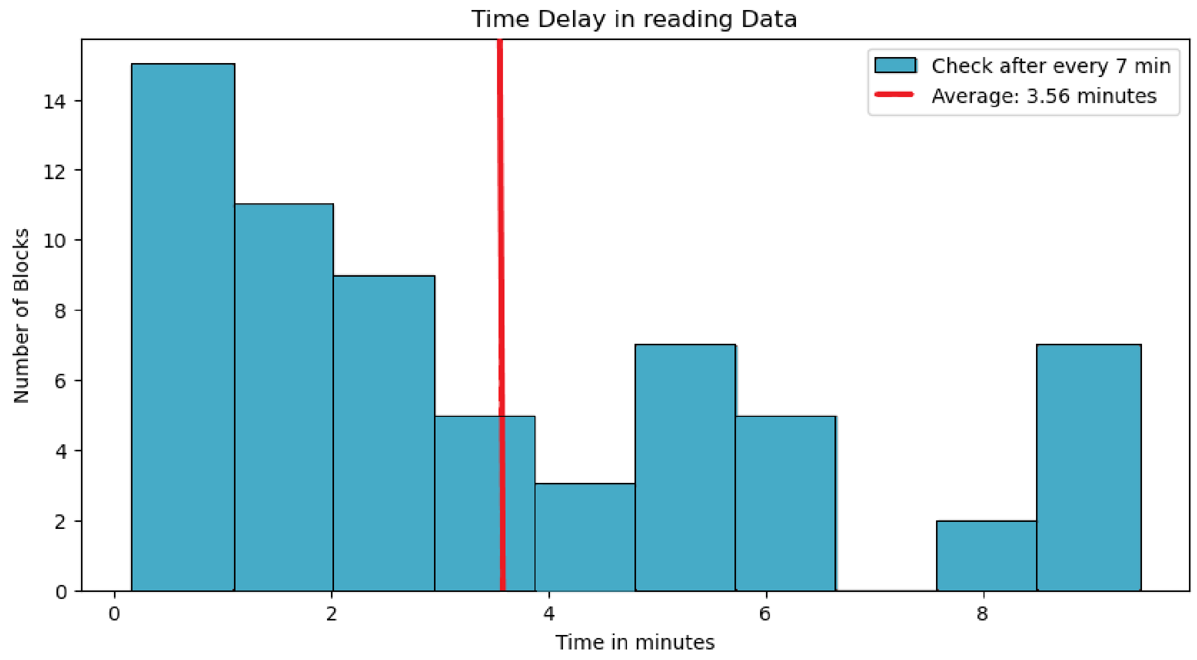


Figure 14 Time difference between creation and retrieval

The project aims to provide a fast and latency-free experience where the data becomes available to the end user. A time difference of approximately 3.5 min reduces the system's ability to produce results fast. Two methods were employed to reduce the time difference and overcome this bottleneck: long polling and constant data fetch. On average, the project’s data source takes approximately 7 minutes (most commonly close to 4 min) to produce full data with additional features required for supervised learning and the pointers required to connect data points. Such data fetch requests constantly produce a load on the source data system.

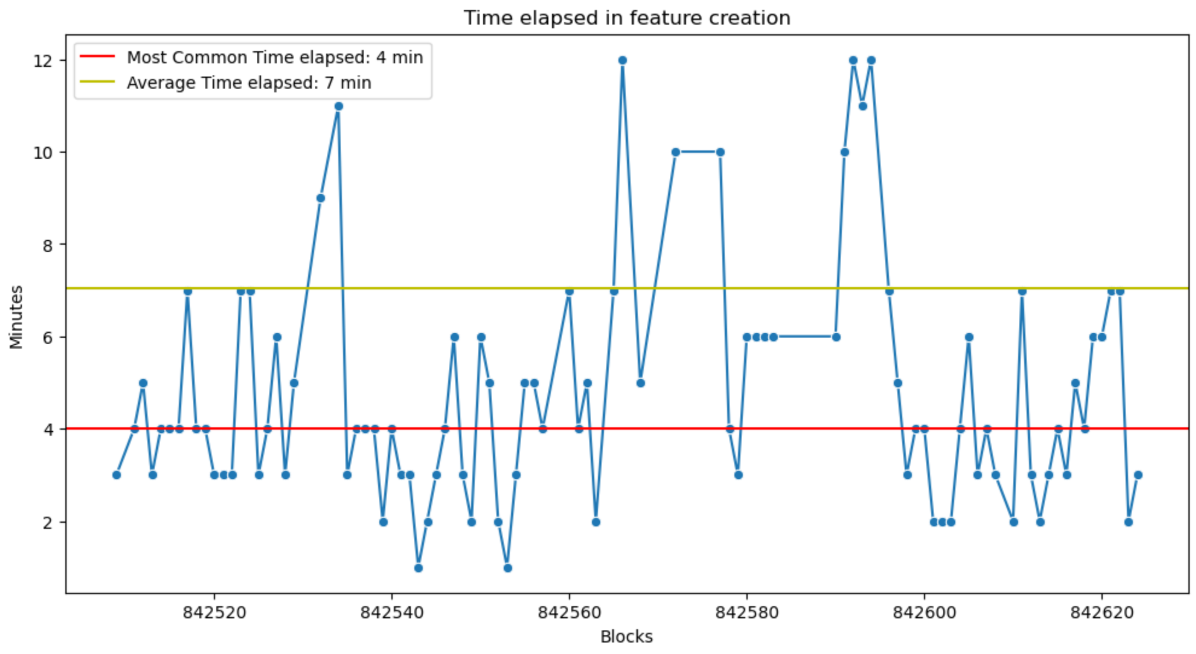


Figure 15 Time taken to create additional features

Methods like event-based polling and custom calls were considered to mitigate system resource issues on the server side and prevent multiple data file creation every time the source API is called. In event-based polling, the concept of webhooks was implemented, where the source API would push notifications to the data consumer (FastApi, 2024). The arrival of the notifications will trigger an automatic script that will fetch the data from the source API. The limitation of such a system outweighs the benefits of the system. First, the failure on the source side meant that consumers would not receive any events and, in turn, would not receive any alerts regarding the unavailability of the incoming data. Second, the system is not scalable as the source data point becomes the liaison and changes will have to be implemented on the source side rather than the client side in case of an increase in the data consumers.

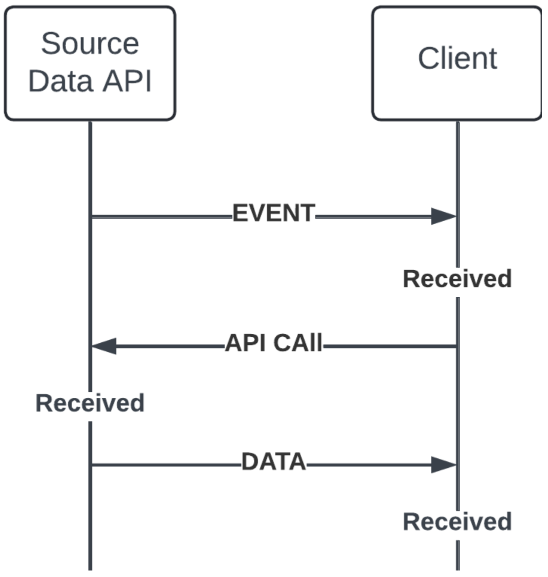


Figure 16 Event-based data fetch

To overcome such issues, a custom and flexible system was implemented that retrieves the data from the source while taking care of the constraints and the bottlenecks. This script retrieves the data by checking every 1 min. If the same data is received, the system goes on standby, and the data check is done again after 30 sec. If a new block is retrieved from the source API, the data will be read and passed to the source API. This cycle goes on until manually interrupted.

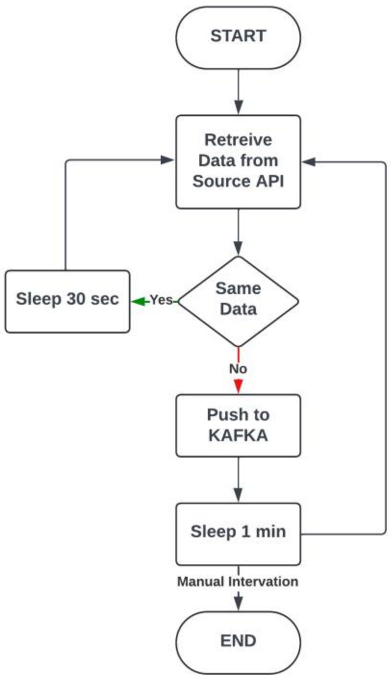


Figure 17 Custom data fetch flowchart

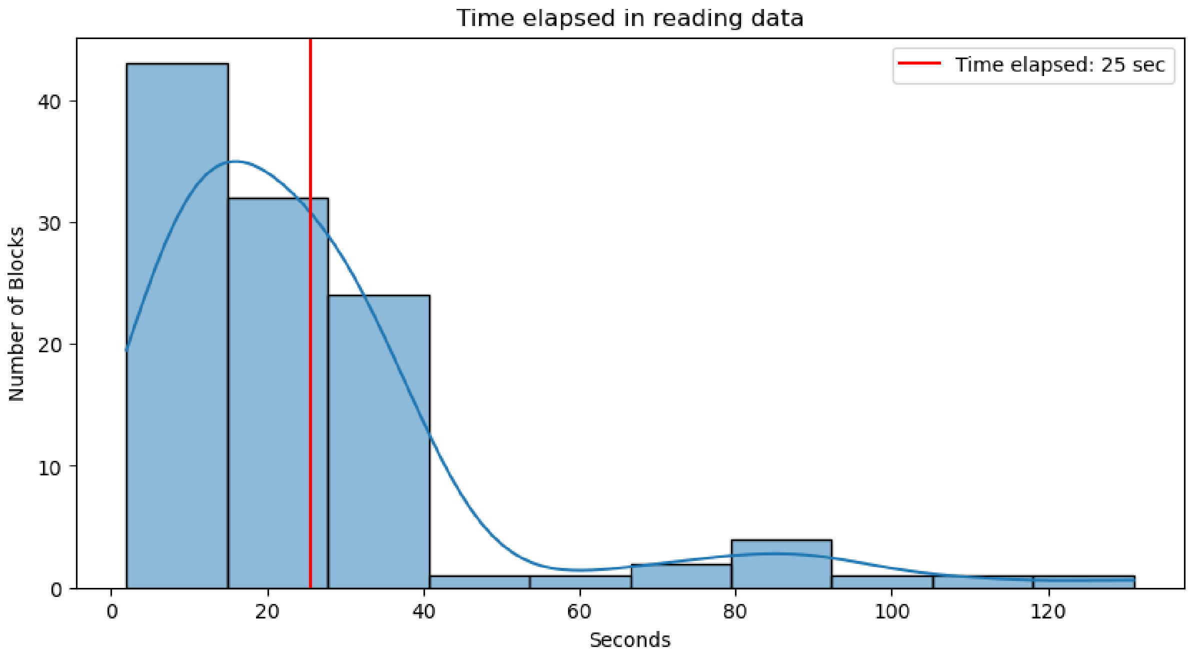


Figure 18 Time difference reduced to 25 sec

Through this custom script, I was able to reduce the time between when the data became available by the data source API and when it was read by the system to an average of 25 sec. The average time elapsed while reading the data was reduced from 3.5 min to 25 sec. Most of the incoming block data, close to 95%, were received in a time difference of 40 sec. The right-skewed shape suggests that some blocks were received, which took more than 1 minute. Such blocks (taking more than 90 sec) can be treated as outliers, and this behaviour can be explained by the large data size and connection issues between the source and the consumer.

After receiving the data from the data source API, it must be processed before being sent to the Kafka system for analysis. The block data received from the API contains a large amount of hex and hashed data produced by the source system for data management. This data must be removed before passing it through the central Kafka system, as the data size produces latency and slows down the system's performance (Wu et al., 2020).

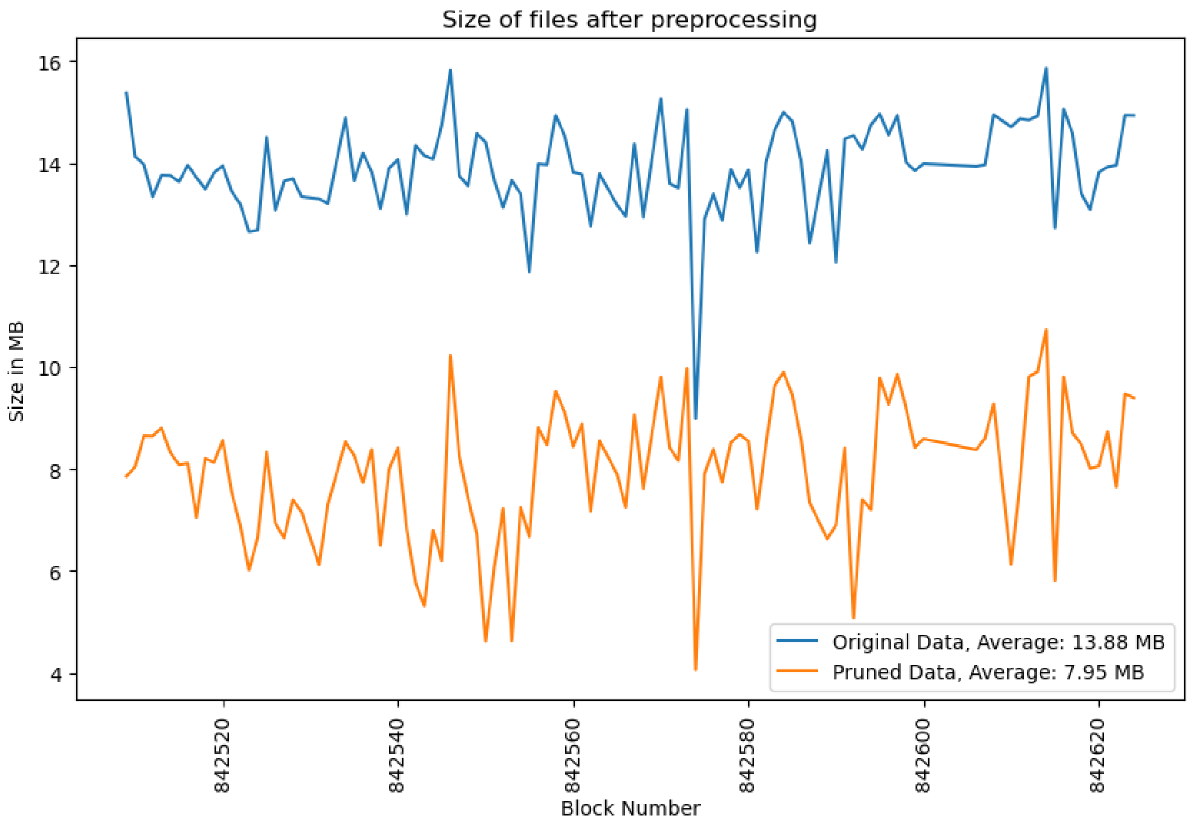


Figure 19 Size of block data before and after preprocessing

Through preprocessing, I was able to reduce the size of the data before passing it through the central Kafka system. The average size of the block data was reduced from approximately 14 MB to 8 MB. This step helps in reducing the load on the Kafka system implemented.

6.2 KAFKA Processing

In this section, I will discuss the second step of the analytical support system pipeline, where the central Kafka system has to be optimised to provide scalable, fast, and reliable handling of incoming data. In this step, after the data has been retrieved from the source API, it will be pushed into the Kafka producer for pre-processing and later passed onto the Kafka consumers subscribed to the producer.

In the proposed solution, Kafka was chosen to improve parallelisation and act as a link to serve the 3 processes. The block of data will be consumed by the Kafka producer and processed by the producer before making it available for further analysis. In the support system, the data needs to be stored in a clean database with full information, passed along for anomaly detection, and used to produce alerts using supervised machine learning methods before saving the processed data in a pruned state in the database.

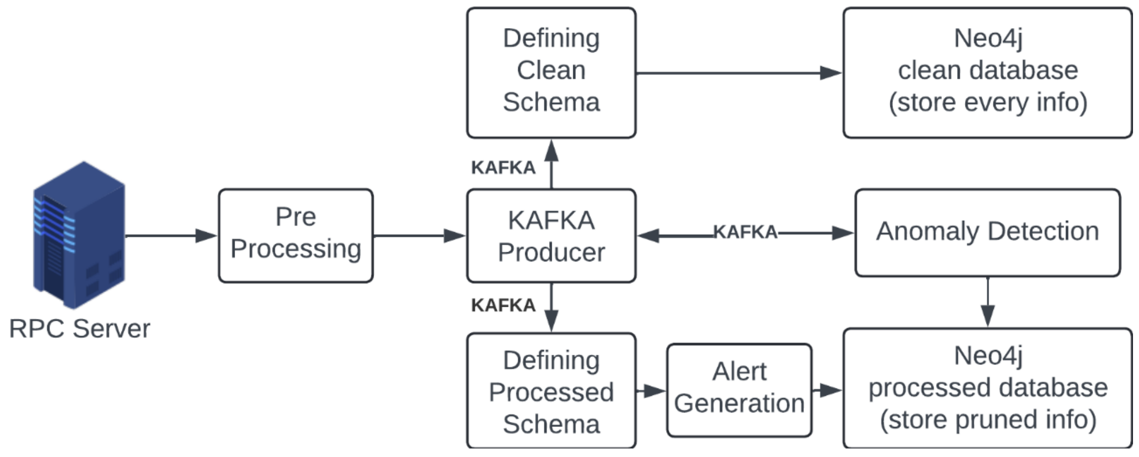

Defining Clean Schema
Neo4j clean database (store every info)
KAFKA
Pre Processing
KAFKA Producer
KAFKA
Anomaly Detection
KAFKA
RPC Server
Defining Processed Schema
Alert Generation
Neo4j processed database (store pruned info)

Figure 20 Stage 2: Kafka Process

Kafka works on a pub-sub model, where the Kafka producer makes the data available to all the subscribed consumers. The Kafka producer uses a Kafka topic to pass this data along to the different systems. The producer in our system will preprocess the data before publishing the data for the consumers. The three different systems will subscribe to the topic and receive the data using the topic created. The configurations regarding the reliability of the data, the persistence of the data, size allowance of the data, and partitioning of the data are all handled by the Kafka topic. This data will be partitioned into 2 and temporarily stored by the zookeeper before automatically overriding it with new data.

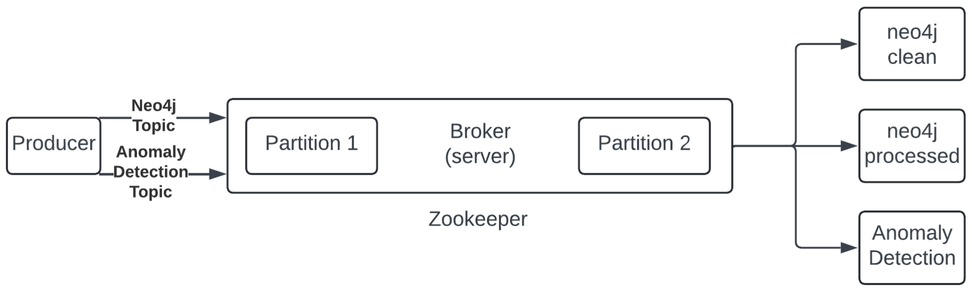


Figure 21 Final Kafka system implemented

6.2.1 KAFKA Clean consumer

In this segment, I will discuss the effect of Kafka configurations on the system's resources. Passing the data through the system while taking care of the constraints, size variance of the block data, time variance in the incoming data, and a static non-scalable system causes a lot of overhead work that the system needs to manage. After the pre-processed data is received from the Kafka producer, it must be stored in a clean database for analysis and visualisation.

From the data source analysis done in section three, most incoming blocks become available every 7 min. Keeping this constraint in mind, performance testing was done to check the system's resource consumption. For testing purposes, the block with the largest number of transactions, close to 7300 transactions, was considered and information regarding the system resources was noted. At ideal, a machine comprising 8 cores and 16 GB of RAM consumed 9 GB of RAM and had only a 10% CPU usage. For the base run, the CPU usage hovered around 40% with 9.5 GB RAM usage and it took close to 6 minutes to complete the consumption of the data after retrieving the data.

Given that new data arrives every 7 minutes and the base run requires 6 minutes to complete, the time remaining to handle other extreme cases is quite limited. To reduce the time taken to process a new block of data, Parallel processing was introduced by incorporating a Kafka consumer group. To incorporate parallelisation, a Kafka topic with a parallelisation factor of 4 and a replication factor of 3 was set to handle cases of failures. The results were calculated over an average of 5 runs.

Table 2 System resource consumption based on Consumer Group size

Consumer Group	Time	CPU Usage	RAM Usage
1	6 min	40%	9.5GB
2	2 min 50 sec	60%	9.6GB
4	1 min 30sec	75%	9.8GB

6.2.2 KAFKA Processed consumer

In this segment, I will discuss the effect of Kafka configurations on the system's resources and timing while incorporating the supervised model developed by other team members. The factors affecting the Kafka-clean consumer and Kafka-processed consumer are the same. Following the constraints of 7 min data arrival, size variance of the block data, time variance in the incoming data, and a static non-scalable system cause a lot of overhead work that needs to be managed by the system. After the pre-processed data is received from the Kafka producer, it must be passed through a supervised machine learning model to mark alerts in the incoming data before passing it to a processed database for analysis and path visualisation.

As we know from the data source analysis done in section three, the majority of the incoming blocks, on average, become available every 7 min. Keeping this constraint in mind, performance testing was done to check the resource consumption of the system. Again, an extreme case of incoming data, the block containing close to 7300 transactions, was considered and information regarding the system resources was noted. At ideal, a machine comprising 8 cores and 16 GB of RAM consumed 9 GB of RAM and had only a 10% CPU usage. For the base run, the CPU usage hovered around 40% with 9.5 GB RAM usage and took close to 5 min 30 sec to complete the consumption of the data after retrieving the data.

Like the analysis conducted for the clean process, we know that new data arrives every 7 minutes, and the base run requires 5 min 30 sec to complete. The time remaining to handle other extreme cases is quite limited. Parallel processing was introduced for processed database by incorporating Kafka consumer groups to reduce the time taken to process a new data block. The results were calculated over an average of 5 runs by creating a Kafka topic with a parallelisation factor of 4, and a replication factor of 3 was set to handle cases of failures.

Table 3 System resource consumption based on Consumer Group size

Consumer Group	Time	CPU Usage	RAM Usage
1	5 min 30 sec	40%	9.6GB
2	3 min	65%	9.6GB
4	1 min 40 sec	85%	9.6GB

Through analysis, I have constructed a consumer group of 2 consumers, substantially reducing the load on the system's resources. Both Kafka clean consumers and Kafka processed consumers will consume data from the Kafka topic, which partitions the data into two and feeds it to the consumers in the consumer group. The CPU usage hovers around 65% and the processing time is reduced to 3min. There is not a lot of effect on the RAM usage as once the data is consumed by the consumers, the zookeeper overwrites the data with new incoming blocks.

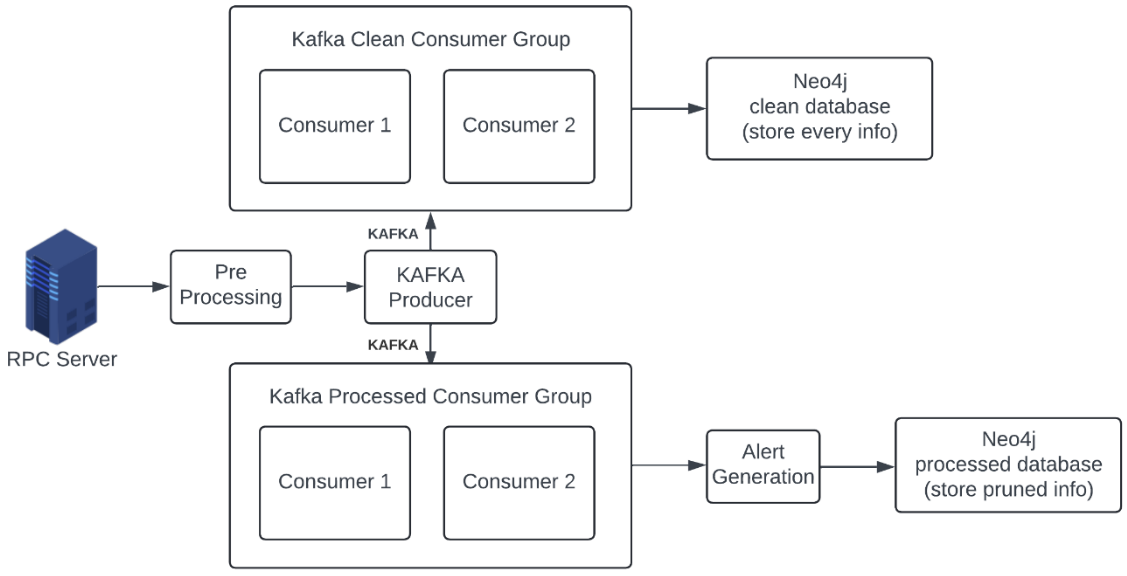


Figure 22 Final Kafka implementation

6.3 Storage Configuration

In this section, I will discuss the third step of the analytical support system pipeline that needs to be optimised to provide fast and accurate storage of real-time incoming data. In this step, different storage solutions and their configurations were discussed to develop a solution that provides low latency performance in terms of read and write capabilities.

After the pre-processed data has been received by the Kafka consumer, it needs to be stored to drive advanced visualisations. For the purpose of our project, I needed to choose a solution that is capable of providing a fast read and write performance while handling real-time incoming data. The scale of this data qualifies it as big data, and to handle such a vast amount of data, a need for a database solution arose that is scalable, reliable and handles data updates without much headwork. I considered different database solutions like Relational Database (MySQL) and NoSQL database (neo4j), and these were compared to handle the system's constraints and the project's requirements.

Table 4 Relational vs NoSQL database

Feature	SQL Database (MySQL)	NOSQL Database (Neo4j)
Data Structure	Utilisation of individual tables in grid format to store structured data	Uses nodes to store structured data and edges to represent the relationship between the nodes
Read Performance	Data is stored in individual tables. As the number of joins between tables increases, the read performance decreases	Data is not stored in individual tables. Joins are not required as joins are stored as relationships directly in the neo4j schema
Write Performance	Provides fast writing of a large number of transactions in individual table	As the complexity of the relationship between different entities increases, write performance decreases
Schema Update	Rigid Schema to enforce consistency and requires significant work before making schema changes	Highly flexible schema that allows for dynamic updates without much headwork
Join Queries	As the complexity of joins between individual tables increases, the join performance decreases	Joins are not required as the connection between the entities is stored in the database
Scalability	Incorporates principals of vertical scalability, allowing for updates to the system resources like CPU and memory	Incorporates principals of Horizontal scalability where multiple entities can join together to distribute workload

Consequently, when deciding the optimal solution for use in the handling and storage of the data, I had to consider the performance of the join while retrieving the data. In our dataset, the data will be stored across three types of nodes or tables. This will be the transaction node containing information on the main Bitcoin transaction, vin and vout. Vin refers to the sub-transaction carried out by an address to serve as the input for the main transaction, while vout refers to the sub-transaction directed towards an address to serve as the output of the main transaction. In our system, A transaction can have multiple vins and vouts. The join performance and the retrieval of joined data in a structured manner are affected by the number of joins required in big data.

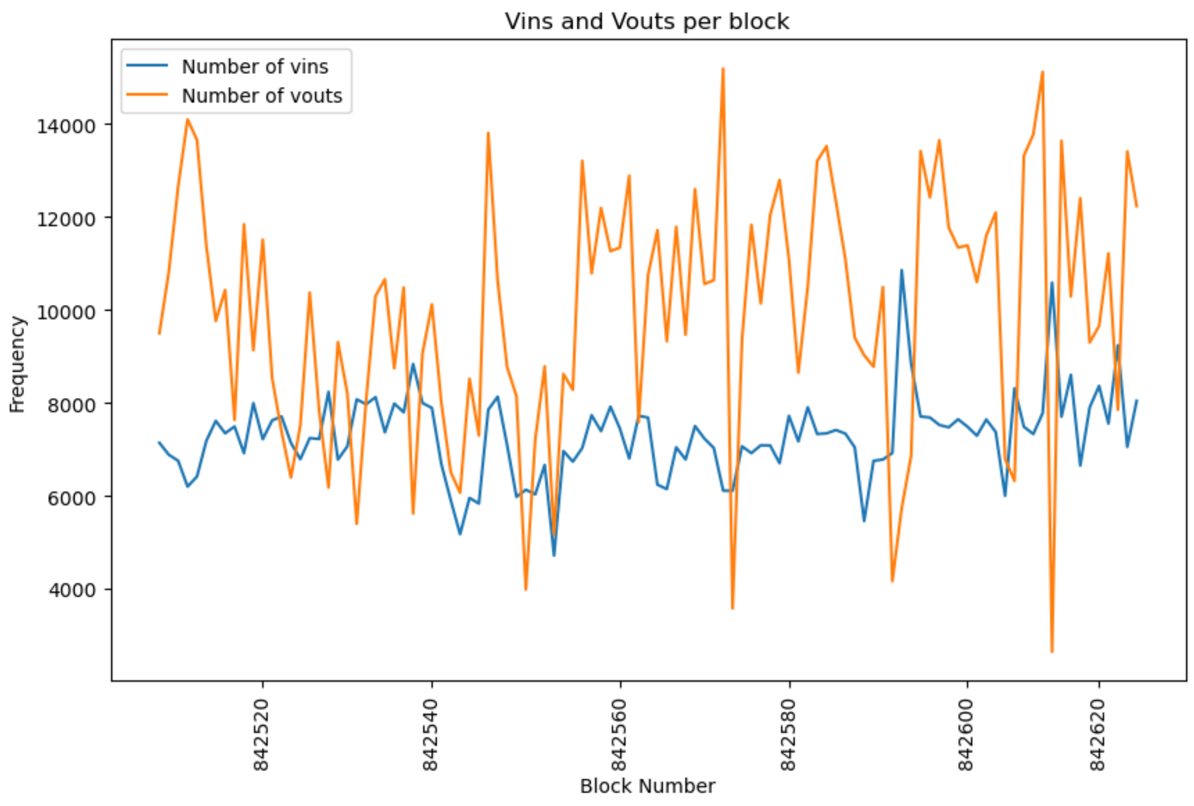


Figure 23 Number of vin and vout per block

From the analysis, we can infer that the number of vins and vouts per block varies significantly. This can be attributed to the fact that the size of the block, as well as the number of transactions in the block, are not consistent. Considering the number of transactions in the block to be the average of 3500, we can say that the average number of vins and vouts per transaction is around 2 vins and 3 vouts. The number of vouts is always more than the number of vins as the transaction creates some UTXO sub-transactions that contain information regarding the unspent Bitcoin data. Based on the analysis of the study in the figure below, we can see that the number of average vins and vouts per transaction in a block also varies and cannot be defined as a constant value.

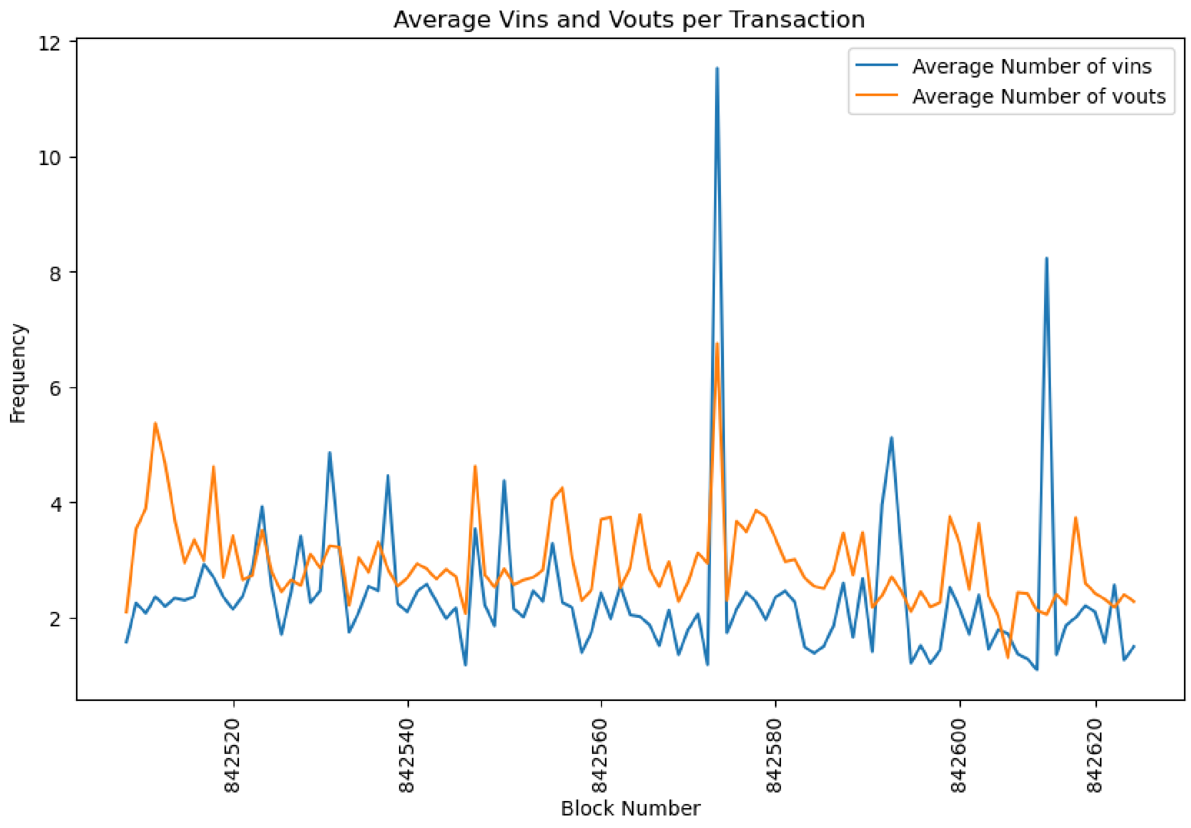


Figure 24 Average number of vin and vout per transaction per block

Based on the analysis of the joins required to present the full data, performance testing of the database with respect to the read performance of relational and NoSQL databases is performed. Here, 100 consecutive Bitcoin data blocks are loaded into the system to simulate a real scenario. From these blocks, a transaction with a large number of vins and vouts will be read. The transaction with 229 vins and 456 vouts represents an extreme case that will be read using the user's visualisation. I will try to simulate the response the end system requires, which is structured data in a specific JSON format. The two metrics present the data retrieval time from an indexed and non-indexed system. The indexed system maintains a ledger of the location where the data is available. These results represent the average of 5 runs created by clearing the cached memory after each run.

Table 5 Read performance comparison: Neo4j vs MySQL

Database	Indexed	Non-Indexed
Neo4j	8 milliseconds	2.8 seconds
MySQL	16 milliseconds	3.6 seconds

Even though the performance of a relational database is close to that of a graph database, this performance will be impacted by the size of the data stored in the two databases. The added advantage of a flexible schema with the ability to add specific data fields to certain entities in the graph database makes it a perfect choice for my analytical support system.

6.4 Anomaly detection

In this section, I will discuss the fourth step of the analytical support system pipeline, where different statistical methods are implemented to find fluctuations in the incoming data, as well as the use of unsupervised machine learning methods to isolate data points that differ greatly from the available set of data points. In this step, supervised machine learning methods were not implemented because of the unavailability of a large amount of labelled continuous dataset. Readily available labelled datasets contain information about the data points that are not continuous in nature, i.e. they don't contain data points that are received one after the other. They contain information on data points that have been identified as fraud and have huge time gaps between consecutive data points when ordered by time. Such datasets will not accurately help develop methods to find anomalous transactions in real-time incoming data. The unavailability of labelled datasets where transactions in a block are classified as regular and anomalous pushes us towards using unsupervised methods.

6.4.1 Node Address Analysis

In this segment, I implemented a real-time address analysis method that considers the number of transactions an address has made. A Bitcoin address is like an account number shared to receive Bitcoin. These addresses are generated using a user's public key, and the holder of the private key is authorised to make transactions using this address (Davis, 2024). A Bitcoin wallet can make transactions using multiple addresses linked to it.

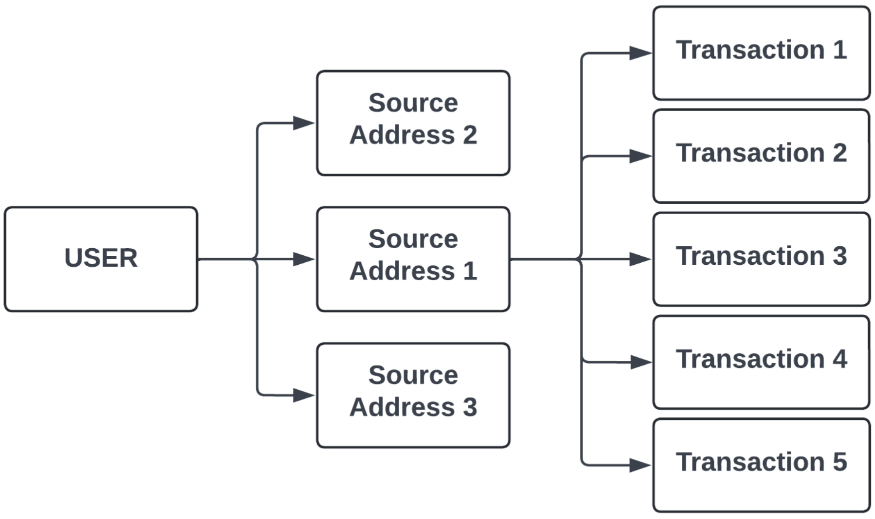


Figure 25 Address Transaction link

Through this analysis, I can maintain a running list. If a certain transaction is identified as fraud, then using a sliding window of a particular time range, I can get the list of other transactions that have been created using the address. This address will then act as a marker to help identify other transactions that could have been used for fraud. Through this, a cluster of transactions could be identified and help visualise anomalous data flow.

6.4.2 Z-score

In this segment, I implemented a z-score method for anomaly detection that compares new incoming block data to the data received in a running time window of 30 minutes. This time window is flexible and can be changed to incorporate more data. Normally, 10,000 independent transactions are received in 30 minutes, giving us a large sample size to compare data points.

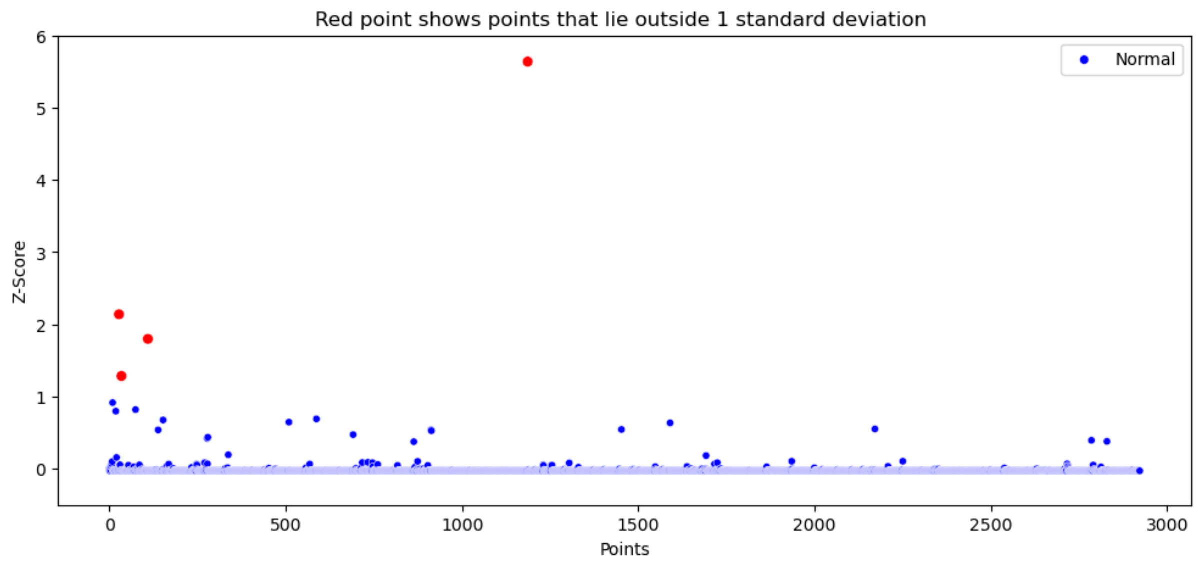


Figure 26 Outliers (marked in red) in an incoming block

In the above analysis, the points with a z-score greater than 1 are marked as anomalous. Here, the blocks received in the past 30 minutes are used to calculate the data's mean and standard deviation. These point values are used to calculate the z-score of the new block of data. For example, based on the threshold set of the z-score of 1, if a data point is received with a calculated z-score of more than 1, then such a data point is marked as an anomaly. This method was rejected as the z-score works assume that the data distribution follows a natural or bell curve distribution (*What is a Z-score? What is a P-value?*, 2024). Plotting the values on a histogram shows a left curve and not even a skewed normal distribution. For this reason, z-score was not incorporated for anomaly detection.

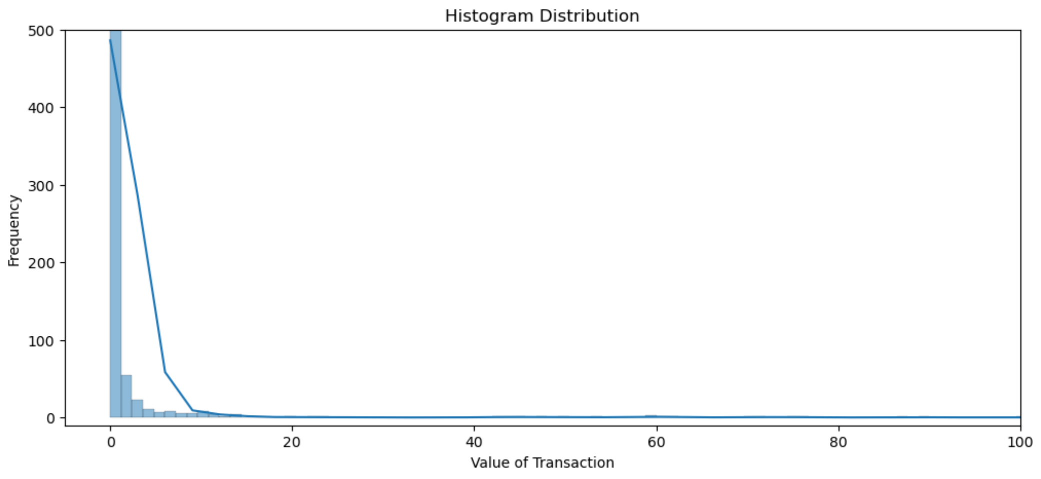


Figure 27 Data distribution

6.4.3 Grubbs Test

In this segment, I implemented the Grubbs test statistical method to detect outliers in a given dataset. The test looks for the most extreme values in the dataset. The test uses 2 major point estimators: the mean of the dataset and the standard deviation. It then calculates the G-value of each data point and compares it to the $G_{critical}$ value in the system. If the G-value exceeds the $G_{critical}$ value, then such data points are marked as anomalies. The Grubbs test assumes that the data follows a normal distribution.

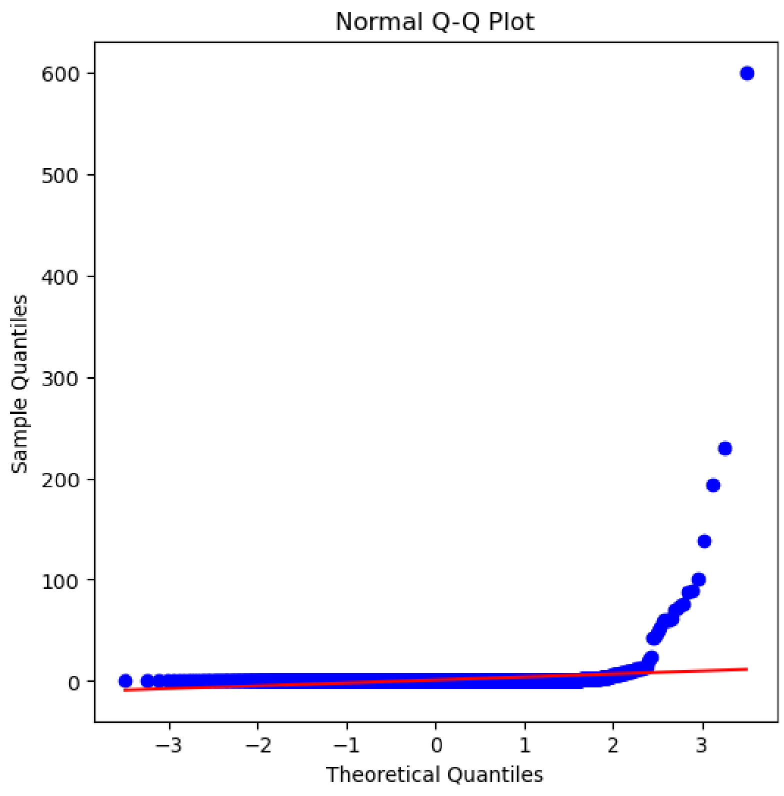


Figure 28 Normal Q-Q plot

The Q-Q plot is used to infer the probability distribution by plotting the sample Quantile against the theoretical quantile. The data is well-modelled if the scatter points lie on a straight line. The line will be angled at a 45-degree angle for a normal distribution (*Normal QQ plot and general QQ plot—ArcMap | Documentation*, 2021)For the purpose of the analysis, incoming data was collected from an average-sized block of data containing 3500 transactions, and the values were sorted in ascending order. These sorted values were used to plot the Q-Q plot.

From the plot, deviation from the line indicates that normality is not followed. The slope is surprisingly flat, almost near 0 degrees, suggesting that the normal distribution is not followed. The existence of the upper right tail suggests the presence of outliers in the system. Based on this analysis, the Grubbs test was not incorporated for anomaly detection.

6.4.4 Isolation Forest

In this segment, I will discuss implementing the isolation forest unsupervised machine learning model for detecting anomalous data points in the incoming batched data. It uses the principle of specifically isolating anomalous data points rather than learning about regular data points. Isolation forests help isolate data points based on the contamination factor. This contamination factor acts as the threshold and isolates a certain percentage of data points.

The method retrieves data from the kafka producer as batched data where the whole block is passed to the isolation forest kafka consumer. The number of transactions identified as fraudulent in the year 2023 hovered in the 0.3% to 0.4% of the transactions (Leising, 2024). Based on this study, I will use a contamination factor of 0.004 to identify anomalous transactions. An average block of 3500 transactions will isolate 15 of the available data points. The features used for training the model will consist of 5 features – the value of the transaction, fee for confirming the transaction in the Bitcoin blockchain, in-degree specifying the number of inputs for a transaction, out-degree specifying the number of outputs for a transaction and influence of the transaction in the blockchain. The isolated data points will differ significantly from the rest of the data points in the block.

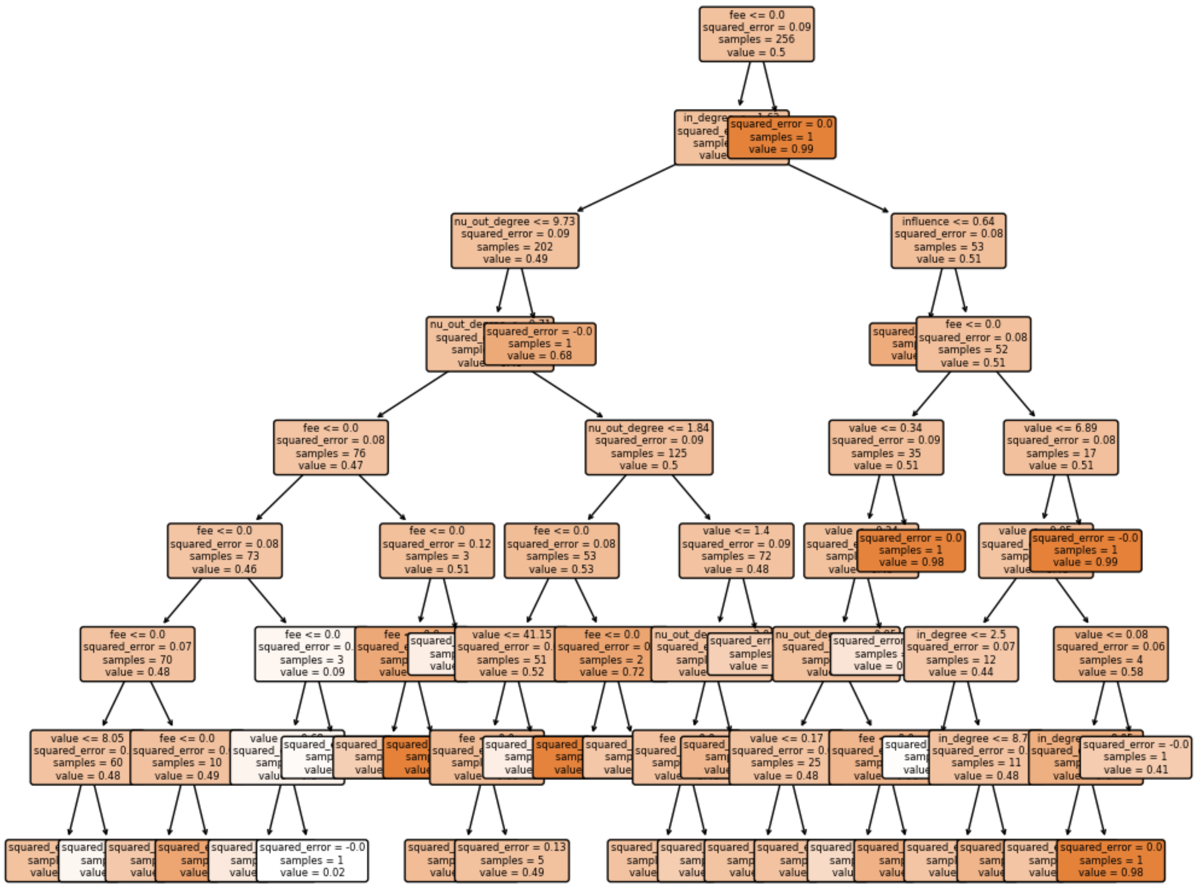


Figure 29 One decision tree, part of the isolation forest

6.4.5 DBSCAN Clustering

In this section, I will implement DBSCAN clustering, a type of unsupervised machine-learning method that aims to club together objects that are more similar to each other than other objects. The unavailability of a large amount of labelled data and the constraints of time and size variability present a good case for implementing a clustering-based algorithm for anomaly detection. DBSCAN is a density-based clustering method that clusters based on closely packed points. This property helps us identify outlier points that lie in low-density regions.

The algorithm employs two hyper-parameters, epsilon and minimum points, to develop density regions. The low number of hyper-parameters needed to tune helps in dealing with the outliers, forming clusters of arbitrary shapes to group data and automatically creating clusters on its own without any need to mention the number of clusters to create. To incorporate such a system, hyperparameter tuning is required to decide the optimal epsilon and minimum points value. For these reasons, a block with an average number of transactions (3500) is selected, and a grid search method is employed (Revag, 2023). A silhouette score, whose value lies in the range of [-1, 1], is employed to select the best combination of hyperparameters. The silhouette score defines how well the object fits inside a cluster; the higher the score, the better the fit of the clusters created (Rousseeuw, 1987).

The features used for the purpose of the analysis include the value of the transaction, fees used to confirm the transaction in the blockchain, in-degree specifying the input of the transaction, out-degree specifying the output of the transaction, the influence of the transaction node in the graph, degree balance of the transaction and z-score of the value. To define the range of values from which the optimal values will be selected. We looked to plot the K-distance plot that is created between a point and K other points closest to it. The value of K is set to (2*Number_of_features -1) in the dataset (Sander et al., 1998). This plot will present a range under which the optimal value of the EPS will be defined.

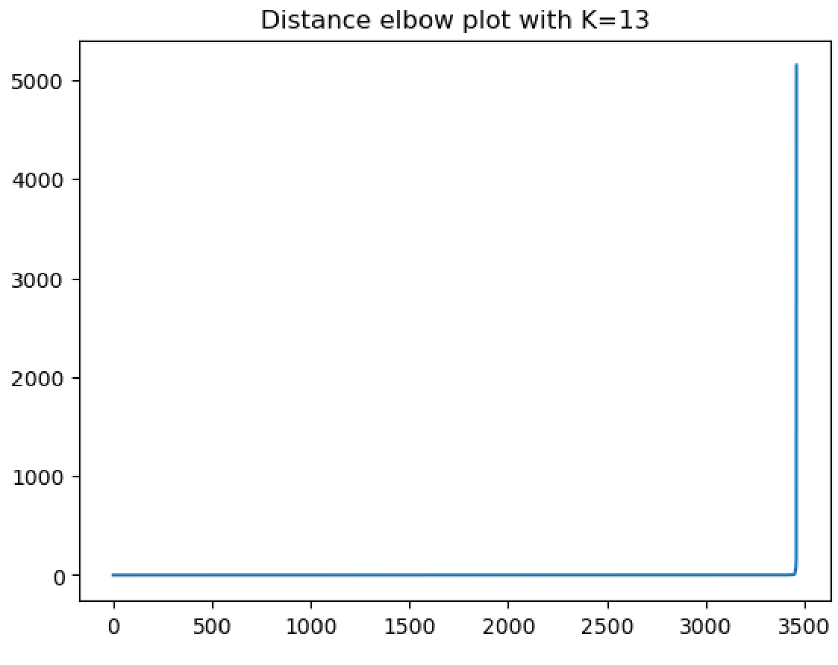


Figure 30 K-Distance graph

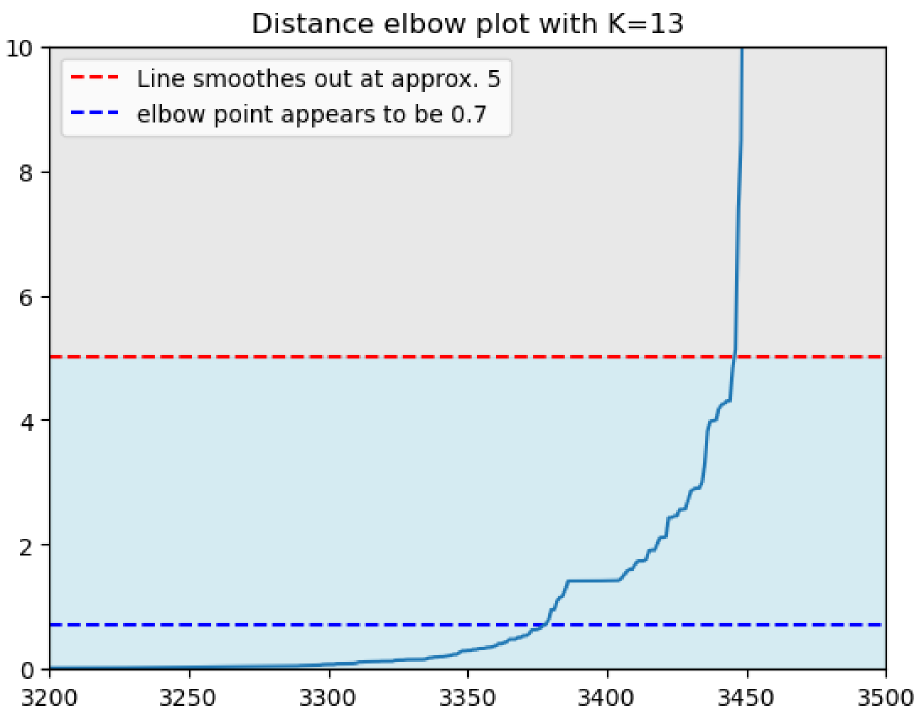


Figure 31 K-Distance graph (zoomed in)

The distance plot is used to visualise the distances between different pairs of objects. The x-axis and y-axis represent the same data points in the same order. The above plot shows that the line smooths out after the value on the y-axis of 5. I chose the range of [0,5] as the range for grid search as after value 5, it suggests a transition from dense to sparse clusters. For minpoints, a range spanning between [number of features + 1, number of features * 2] is chosen based on the studies where DBSCAN has been implemented (Sander et al., 1998; Valeti, 2021). After the range of values for both hyperparameters is defined, the grid search value is plotted, and the silhouette score is noted to assess the quality of the clusters created. This will define how similar an object is to the objects within its own cluster as compared to the objects present in other clusters.

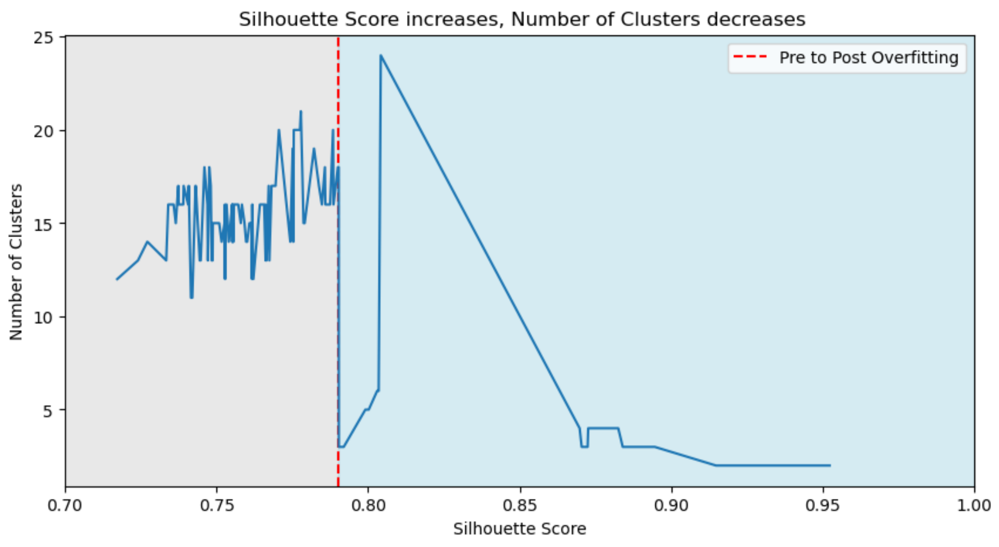


Figure 32 Silhouette score increases, the number of clusters decreases

The line chart was constructed to show the effects of the overfitting during the cluster creation. From the graph, we can infer that the number of clusters created decreases as the silhouette score increases. This shows that as the silhouette score increases to 1, all the data points cluster to become part of one cluster. As the silhouette score increases past approximately 0.8, there is a sharp downturn in the value of the number of clusters created.

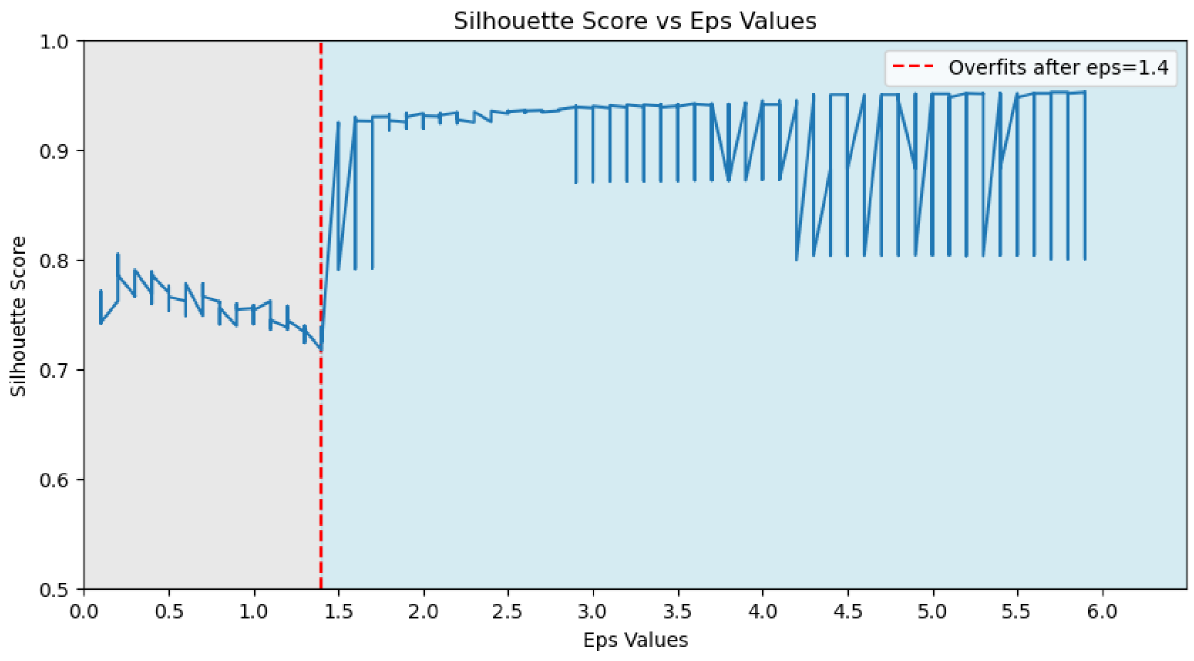


Figure 33 As the EPS value increases, the silhouette score also increases

When plotting the above analysis of EPS from the defined combinations in a grid search, we can see that after the EPS value of 1.5, the silhouette score increases sharply. As the value of the EPS increases, which essentially means that as the radius for forming the cluster increases, the silhouette score increases to 1. This represents potential overfitting, as different data points become part of one cluster when the radius for forming the cluster increases, thus leading to overfitting.

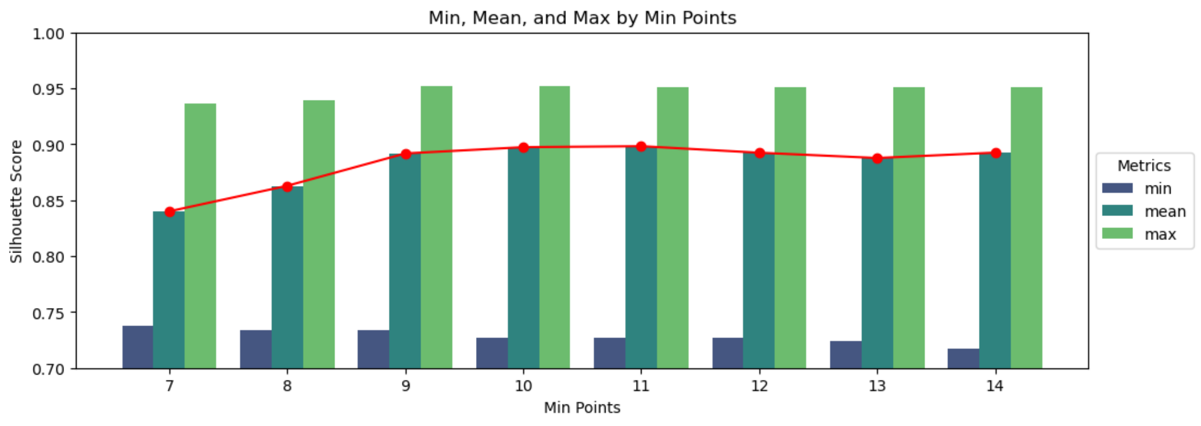


Figure 34 As the Minpoints value increases, the silhouette score also increases

When finding the optimal value of the minimum points required to form a dense region, we plot the mean, maximum, and minimum silhouette scores for the available values of minimum points in the grid search. We can clearly see that after a value of 8 minimum points, the mean and the max silhouette score increase and become close to 1. As the value of the silhouette score increases, the model overfits, and more points become necessary to create a cluster, which in turn means the number of clusters decreases.

Based on the analysis's results, the hyperparameter values of min points 8 and an EPS value of 0.6 give the best silhouette score while forming a large number of clusters (15-20 clusters). The dimensionality reduction technique of t-SNE is used to visualise the clustering results by converting the high-dimensional features into low-dimensional features (van der Maaten & Hinton, 2008).

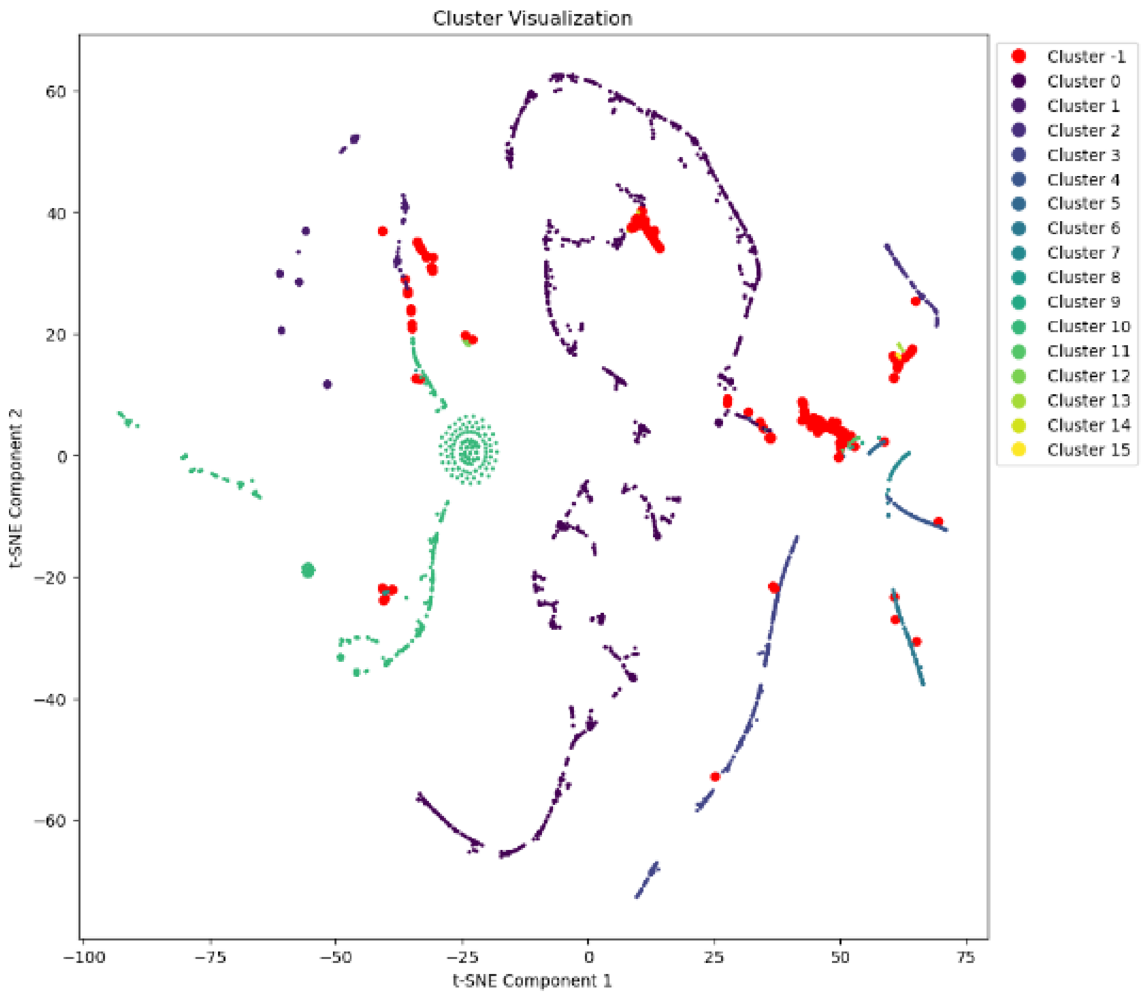


Figure 35 Cluster creation and visualisation

The cluster visualisation shows the distribution of the points according to the top 2 low-level features created using t-SNE. The red points represent data points that are part of low-density regions. All the data points that are part of cluster -1 and marked as red are the points that are not part of any cluster based on the hyperparameters set. Cluster formation was mostly affected by the features in descending order of influence of the transaction node, in-degree, transaction confirmation fees, value of the transaction, and nu-out-degree.

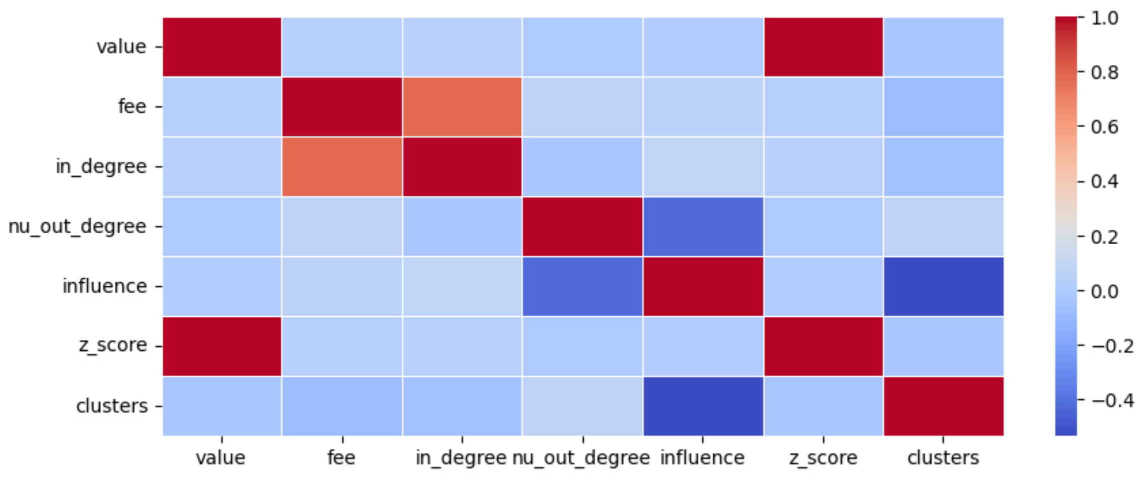


Figure 36 Correlation Heatmap between cluster creation and features

6.5 Rest API

In this section, I will discuss the API implementation built to exchange data with the user's UI. The user UI will make API calls to retrieve data from neo4j. Here, the system will retrieve the data from the clean and processed databases. The clean database contains information regarding the transaction's structure and its features, while the processed database contains information about the transaction being anomalous as well as the fraud probability produced using the integrated supervised machine learning model developed by the other team members. In the user UI, the homepage shows the transactions that have been marked as anomalous using the anomaly detection methods. Clicking to view the transaction shows the visualisations that expand to show the transaction's structure and fraud probability. In each case, an API call on the routes defined will be triggered that returns the data in a structured format as required by the UI.

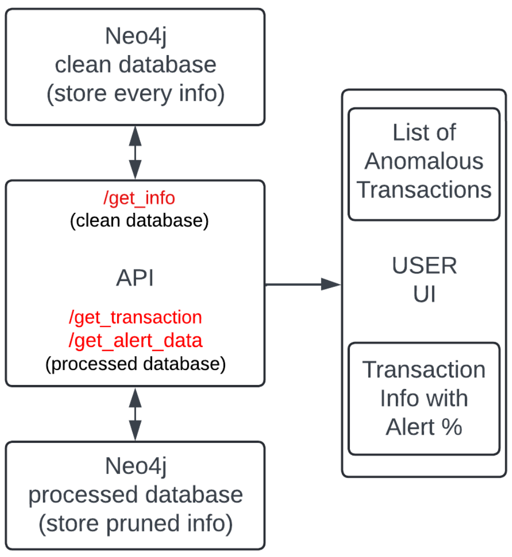


Figure 37 API Implementation with separate routes

7 Results

In this section, I will discuss the final setup of the support system, which is built to tackle the project's requirements while working within the defined constraints. For this project, I have utilised the real-time block data from a Bitcoin core full node. As previously described, I defined the constraints regarding the incoming data's size, time and quality. The time constraint, where the time in which the new block of data becomes available, on average, is close to 12 min. Most of the blocks become available in 7 min. The number of transactions coming in a block also varies, where the number of transactions can range from 500 to 7100. Based on these constraints, a system incorporating Kafka, neo4j, anomaly detection methods and rest API was developed.

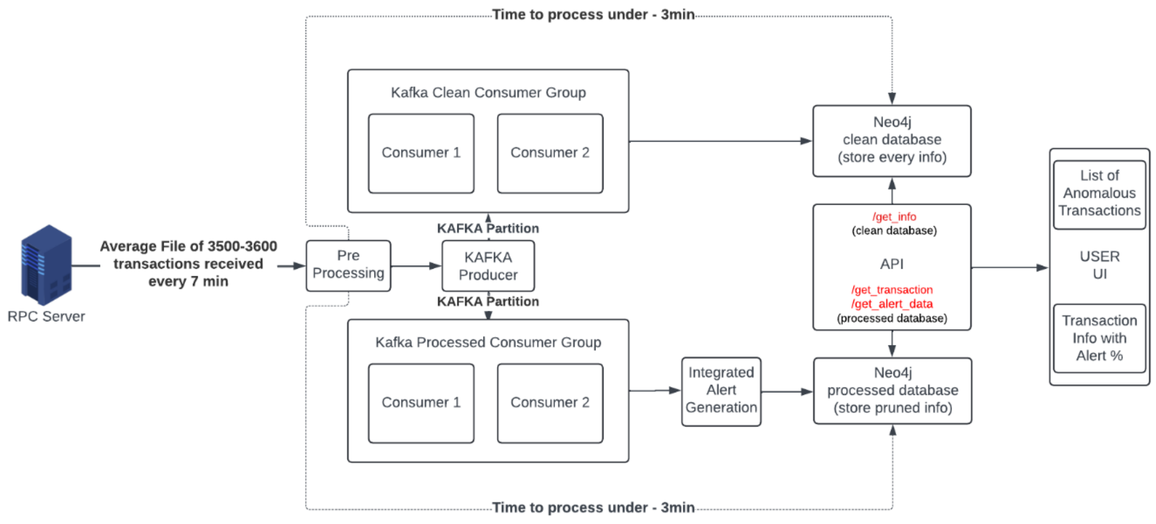
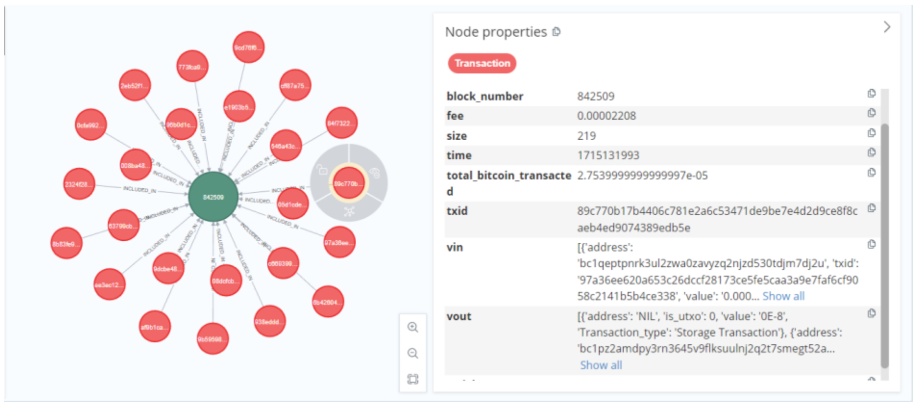

Time to process under - 3min
Kafka Clean Consumer Group
Consumer 1
Consumer 2
Neo4j clean database (store every info)
List of Anomalous Transactions
USER UI
Transaction Info with Alert %
/get_info (clean database)
API
/get_transaction
/get_alert_data
(processed database)
RPC Server
Average File of 3500-3600 transactions received every 7 min
Pre Processing
KAFKA Producer
KAFKA Partition
KAFKA Partition
Kafka Processed Consumer Group
Consumer 1
Consumer 2
Integrated Alert Generation
Neo4j processed database (store pruned info)
Time to process under - 3min

Figure 38 Final support system Implementation

Performance testing was done to reduce the load as well as the response time of the system implemented. Specifically, through iterative testing, I developed an integrated system that processes incoming data, detects anomalies and stores the data in a required schema format under 3 min. The Kafka central system was optimised through performance testing, where three separate consumer groups work in parallel to distribute the data across different project processes. A consumer group consisting of 2 consumers consume data in parallel, which helps to provide low-latency data processing while following the rules of scalability and reliability. Furthermore, I implemented two neo4j schemas that allow for faster read and write speeds, giving users latency-free responses by handling requests from the API quickly and concisely. Simple schemas were designed to reduce the time required for data storage as complex relationships hamper the write performance of the system. Reducing the number of individual nodes in the Neo4j provides better write performance when dealing with big data.

Table 6 Write performance for an average file when Neo4j loaded with 100 blocks

Type	Time (Indexed)	Time (Not Indexed)
Simple	1 min 30 sec	2 min 50 sec
Complex	5 min 20 sec	9 min 40 sec



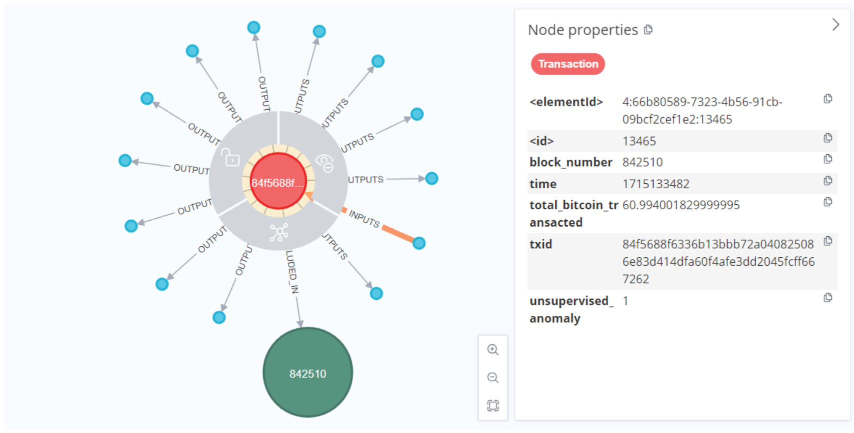
Node properties

Transaction

block_number	842509
fee	0.00002208
size	219
time	1715131993
total_bitcoin_transacted	2.7539999999999997e-05
txid	89c770b17b4406c781e2a6c53471de9be7e4d2d9ce8f8caeb4ed9074389edb5e
vin	[{'address': 'bc1qeptpnrk3ul2zwa0zavyzq2njzd530tdjm7dj2u', 'txid': '97a36ee620a653c26dccf28173ce5fe5caa3a9e7faf6cf9058c2141b5b4ce338', 'value': '0.000... Show all
vout	[{'address': 'NIL', 'is_utxo': 0, 'value': '0E-8', 'Transaction_type': 'Storage Transaction'}, {'address': 'bc1pz2amdpy3rn3645v9flksuulnj2q2t7smegt52a... Show all

Figure 39 Simple Neo4j schema with vin and vout saved as JSON string

Ultimately, I incorporated the anomaly detection methods of isolation forest and DBSCAN clustering. For implementation in a real-time system, hyperparameters were tuned by following the rules of the grid search method. First, the elbow point was graphed to gain a sense of the EPS hyperparameter. Second, based on domain knowledge, the range of values for the minpoint hyperparameter was also defined. Third, Silhouette's score was taken, representing the fit of the data points in a cluster. Based on the score and the number of clusters created to mitigate chances of incorrect modelling, the best combination of the hyperparameter values was selected to define the clusters and mark data points as anomalous that lie in low-density regions. An EPS value of 0.6 and a minpoint value of 8 were chosen because the model could form many clusters while providing a sufficiently high silhouette score. In the end, an end-to-end support system was developed that processes the data from start to finish in under 3 min, provided that an average block was received from the Bitcoin blockchain.



Node properties

Transaction

<elementId>	4:66b80589-7323-4b56-91cb-09bcf2cef1e2:13465
<id>	13465
block_number	842510
time	1715133482
total_bitcoin_transacted	60.994001829999995
txid	84f5688f6336b13bbb72a040825086e83d414dfa60f4afe3dd2045fcff667262
unsupervised_anomaly	1

Figure 40 Simple schema for the processed database containing anomaly alerts

Consequently, I have provided the results of the expected project outcomes. First, I underlined the constraints that impact the system's development. Second, I was able to work with the constraints and the requirements defined for the project and develop a support system using Kafka and Neo4j. The full system was performance-tested and fine-tuned to handle the load of the real-time incoming data. Furthermore, I incorporated the anomaly detection clustering methods to find anomalous data. In the end, a support system was developed capable of handling user load and driving advanced visualisations.

8 Conclusion

In conclusion, advancements in the use of cryptocurrencies for criminal activities highlight the need for a system to track the money flow from its origin. In this project, I deep-dived into the issues that became apparent while building a support system to handle the requirements for driving advanced visualisations. I closely followed the requirements and objectives discussed during the project planning phase. A particular project outcome to develop the application using open-source software has led to the creation of a support system capable of providing reliability and scalability under heavy workloads. Notably, I have identified the potential pitfalls that affect the system's performance when real-time data is passed through the individual components that work together to provide a latency-free experience. To address such challenges, I have refined and correctly configured the central Kafka system. The implemented framework not only managed to meet the expected performance but also significantly exceeded it. Additionally, I have detected the issues that come with big data storage. To alleviate such issues, I designed a storage schema that provides latency-free read and write capabilities while processing and storing the data. Finally, I have implemented a clustering mechanism to detect anomalous data points in the incoming data. After conducting numerous rounds of testing, I have refined and implemented the algorithm to separate the most distinct data points in incoming data. As a result, a cohesive system is established that is flexible and dependable in handling issues related to real-time anomaly detection. My attempt to create a scalable, reliable, flexible and robust support system is extremely crucial. An application with such capabilities has the potential to detect and visualize the money trail of suspicious transactions. This will help relevant authorities in catching cybercriminals who use blockchain features to commit fraud. Hence, by addressing the challenges that come up in the field of cryptocurrency transaction monitoring, the project manages to provide a framework that contributes to fairness and safety in the blockchain.

References

Abdullahabrar. (2023, 2023/07/12). *Exploring the versatility of hierarchical clustering: Applications, working, methods, and benefits*. https://medium.com/@abdullahabrar/exploring-the-versatility-of-hierarchical-clustering-applications-working-methods-and-benefits-5bae54cb9e7b

Ahmed, M., Shumailov, I., & Anderson, R. (2018). Tendrils of Crime: Visualizing the Diffusion of Stolen Bitcoins. GraMSec@FLoC,

Akshara. (2021, 2024/01/10). *Anomaly detection using isolation forest - A complete guide*. https://www.analyticsvidhya.com/blog/2021/07/anomaly-detection-using-isolation-forest-a-complete-guide/

Alexeev, V. A., Domashnev, P. V., Lavrukhina, T. V., & Nazarkin, O. A. (2019). The Design Principles of Intelligent Load Balancing for Scalable WebSocket Services Used with Grid Computing. *Procedia Computer Science*, *150*, 61-68. https://doi.org/https://doi.org/10.1016/j.procs.2019.02.014

Amini, A., Wah, T., & Saboohi, H. (2014). On Density-Based Data Streams Clustering Algorithms: A Survey. *Journal of Computer Science and Technology*, *29*, 116-141. https://doi.org/10.1007/s11390-013-1416-3

Androulaki, E., Karame, G. O., Roeschlin, M., Scherer, T., & Capkun, S. (2013). Evaluating User Privacy in Bitcoin. Financial Cryptography,

Brown, S. D. (2016). Cryptocurrency and criminality. *The Police Journal: Theory, Practice and Principles*, *89*, 327 - 339.

Ćatović, A., Buzađija, N., & Lemes, S. (2022). Microservice development using RabbitMQ message broker. *Science, Engineering and Technology*, *2*(1), 30-37. https://doi.org/10.54327/set2022/v2.i1.19

Davis, M. (2024, 2024/01/24). *How are bitcoin wallet addresses generated?* https://www.doubloin.com/learn/bitcoin-wallet-address-generated

Dragonfly. (2023). *How does MongoDB join performance impact database operations?* https://www.dragonflydb.io/faq/mongodb-join-performance

FastApi. (2024). *OpenAPI Webhooks*. https://fastapi.tiangolo.com/advanced/openapi-webhooks/

Garg, N. (2013). *Apache kafka*. Packt Publishing Birmingham, UK.

Geepalla, E., Abuhamoud, N., & Abouda, A. (2018, 2018//). Analysis of Call Detail Records for Understanding Users Behavior and Anomaly Detection Using Neo4j. 5th International Symposium on Data Mining Applications, Cham.

Grubbs, F. E. (1969). Procedures for Detecting Outlying Observations in Samples. *Technometrics*, *11*(1), 1-21. https://doi.org/10.1080/00401706.1969.10490657

Gruber, L. Comparative Study of Messaging Protocols Used in Mobile Software Architecture.

Jain, V., Saini, D., & Ahluwalia, A. (2019). Real-time autonomous trading system. *Journal of Statistics and Management Systems*, *22*(2), 403-413. https://doi.org/10.1080/09720510.2019.1582881

Kabakus, A. T., & Kara, R. (2017). A performance evaluation of in-memory databases. *Journal of King Saud University - Computer and Information Sciences*, *29*(4), 520-525. https://doi.org/https://doi.org/10.1016/j.jksuci.2016.06.007

Kim, M., Han, S., Cui, Y., & Lee, H. (2013). Design and Implementation of MongoDB-based Unstructured Log Processing System over Cloud Computing Environment. *Journal of Internet Computing and Services*, *14*(6), 71-84. https://doi.org/10.7472/jksii.2013.14.6.71

Kumar, A. (2023, 2023/04/19). *Z-score or Z-statistics: Concepts, formula & examples*. https://vitalflux.com/z-score-z-statistics-concepts-formula-examples/

Kumar, A., Prusti, D., Purusottam, I. S., & Rath, S. K. (2021, 6-8 July 2021). Real time SOA based credit card fraud detection system using machine learning techniques. 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT),

Leising, M. (2024). Crypto crime fell to 0.34% of all blockchain use in 2023; FTX $8.7B theft counted as fraud: Chainalysis. *Decential Media*. https://www.decential.io/articles/crypto-crime-fell-to-034-of-all-blockchain-use-in-2023-ftx-87b-loss-counted-as-fraudulent-chainalysis

Liao, Q., Gu, Y., Liao, J., & Li, W. (2020). Abnormal transaction detection of Bitcoin network based on feature fusion. *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, *9*, 542-549.

Liu, j.-c., Hsu, C.-H., Zhang, J.-H., Kristiani, E., & Yang, C.-T. (2023). An event-based data processing system using Kafka container cluster on Kubernetes environment. *Neural Computing and Applications*, 1-18. https://doi.org/10.1007/s00521-023-08326-1

Liu, X., Harwood, A., Karunasekera, S., Rubinstein, B., & Buyya, R. (2017, 14-17 Aug. 2017). E-Storm: Replication-Based State Management in Distributed Stream Processing Systems. 2017 46th International Conference on Parallel Processing (ICPP),

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Normal QQ plot and general QQ plot—ArcMap | Documentation. (2021, 2021/09/20). https://desktop.arcgis.com/en/arcmap/latest/extensions/geostatistical-analyst/normal-qq-plot-and-general-qq-plot.htm#GUID-B7DE7111-5DE3-4627-BC7F-987785A359C5

Pham, T.-B., & Lee, S. (2016). Anomaly Detection in the Bitcoin System - A Network Perspective. *ArXiv*, *abs/1611.03942*.

Pham, T., & Lee, S. (2016). Anomaly Detection in Bitcoin Network Using Unsupervised Learning Methods.

Prosser, D. (2022). Growing crypto fraud threat. *Public Finance*(5/6), 26-27. https://www.proquest.com/docview/2919861094?accountid=14723&sourcetype=Trade%20Journals

Regaya, Y., Fadli, F., & Amira, A. (2021). Point-Denoise: Unsupervised outlier detection for 3D point clouds enhancement. *Multimedia Tools and Applications*, *80*, 1-17. https://doi.org/10.1007/s11042-021-10924-x

Requeno, J. I., Merseguer, J., Bernardi, S., Perez-Palacin, D., Giotis, G., & Papanikolaou, V. (2019). Quantitative Analysis of Apache Storm Applications: The NewsAsset Case Study. *Information Systems Frontiers*, *21*(1), 67-85. https://doi.org/10.1007/s10796-018-9851-x

Revag. (2023, 2023/08/08). *DBSCAN — an easy clustering algorithm and also how to optimize it using grid search*. https://medium.com/@revag2014/dbscan-an-easy-clustering-algorithm-and-also-how-to-optimize-it-using-grid-search-69a382b63e85

Rostanski, M., Grochla, K., & Seman, A. (2014, 2014//). Evaluation of Fairness in Message Broker System Using Clustered Architecture and Mirrored Queues. Information Sciences and Systems 2014, Cham.

Rousseeuw, P. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*, 53-65. https://doi.org/10.1016/0377-0427%2887%2990125-7

Rousseeuw, P. J., & Hubert, M. (2018). Anomaly detection by robust statistics. *WIREs Data Mining and Knowledge Discovery*, *8*(2), e1236. https://doi.org/https://doi.org/10.1002/widm.1236

Rui, X., & Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, *16*(3), 645-678. https://doi.org/10.1109/TNN.2005.845141

Running a full node. (2015, 2015/03/16). Bitcoin - Open source P2P money. https://bitcoin.org/en/full-node

Sander, J., Ester, M., Kriegel, H.-P., & Xu, X. (1998). Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. *Data Mining and Knowledge Discovery*, *2*(2), 169-194. https://doi.org/10.1023/A:1009745219419

Sanla, A., & Numnonda, T. (2019). A Comparative Performance of Real-time Big Data Analytic Architectures. *2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, 1-5.

Saxena, S., & Gupta, S. (2017). *Practical Real-time Data Processing and Analytics: Distributed Computing and Event Processing using Apache Spark, Flink, Storm, and Kafka*. Packt Publishing. https://books.google.com.au/books?id=9JlGDwAAQBAJ

Schmidl, S., Wenig, P., & Papenbrock, T. (2022). Anomaly detection in time series: a comprehensive evaluation. *Proc. VLDB Endow.*, *15*(9), 1779–1797. https://doi.org/10.14778/3538598.3538602

Seda, P., Hosek, J., Masek, P., & Pokorny, J. (2018, 22-25 April 2018). Performance testing of NoSQL and RDBMS for storing big data in e-applications. 2018 3rd International Conference on Intelligent Green Building and Smart Grid (IGBSG), Sharma, N. (2023, 2023/08/08). *K-means clustering explained*. https://neptune.ai/blog/k-means-clustering

Shayegan, M. J., Sabor, H. R., Uddin, M., & Chen, C.-L. (2022). A Collective Anomaly Detection Technique to Detect Crypto Wallet Frauds on Bitcoin Network. *Symmetry*, *14*, 328.

Shi, F.-B., Sun, X.-Q., Gao, J.-H., Xu, L., Shen, H.-W., & Cheng, X.-Q. (2019). Anomaly detection in Bitcoin market via price return analysis. *PLOS ONE*, *14*(6), e0218341. https://doi.org/10.1371/journal.pone.0218341

Singh, B. (2022, 2022/09/12). *Graph data model and graph native storage & processing*. https://medium.com/noon-academy/graph-data-model-and-graph-native-storage-processing-b6b86f12a69a

Thailappan, D. (2021, 2022/08/26). *Understand the DBSCAN clustering algorithm!* https://www.analyticsvidhya.com/blog/2021/06/understand-the-dbscan-clustering-algorithm/

Valeti, D. (2021, 2021/10/18). *DBSCAN algorithm for fraud detection & outlier detection in a data set*. https://medium.com/@dilip.voleti/dbscan-algorithm-for-fraud-detection-outlier-detection-in-a-data-set-60a10ad06ea8

van der Maaten, L., & Hinton, G. (2008). Viualizing data using t-SNE. *Journal of Machine Learning Research*, *9*, 2579-2605.

Vohra, D. (2016). Using Apache Kafka. In D. Vohra (Ed.), *Pro Docker* (pp. 185-194). Apress. https://doi.org/10.1007/978-1-4842-1830-3_12

What is a Z-score? What is a P-value? (2024, 2024/04/10). https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/what-is-a-z-score-what-is-a-p-value.htm

Wu, H. (2019). Research Proposal: Reliability Evaluation of the Apache Kafka Streaming System. *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 112-113.

Wu, H., Shang, Z., Peng, G., & Wolter, K. (2020, 12-15 Oct. 2020). A Reactive Batching Strategy of Apache Kafka for Reliable Stream Processing in Real-time. 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE),

Xu, Z. w. (2009, 14-16 Aug. 2009). Maintaining database consistency and integrity in HIS with transactions. 2009 IEEE International Symposium on IT in Medicine & Education,

Yinka-Banjo, C., & Esther, O. (2019). Financial stock application using websocket in real time application. *International Journal of Informatics and Communication Technology (IJ-ICT)*, *8*, 139. https://doi.org/10.11591/ijict.v8i3.pp139-151

Zola, F., Eguimendia, M., Bruse, J. L., & Urrutia, R. O. (2019, 14-17 July 2019). Cascading Machine Learning to Attack Bitcoin Anonymity. 2019 IEEE International Conference on Blockchain (Blockchain),