NOVA IMS Information Management School

MDSAA

Master Degree Program in

Data Science and Advanced Analytics

Improving Trustworthiness in DeFi: Anomaly Detection for Blockchain Oracle Price Feed Data

Lukas Gross

Master Thesis

presented as partial requirement for obtaining a Master's Degree in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Improving Trustworthiness in DeFi: Anomaly Detection for Blockchain Oracle Price Feed Data

by

Lukas Gross

Master Thesis presented as partial requirement for obtaining the Master's degree in Data Science and Advanced Analytics, with a specialization in Business Analytics

Supervised by

Fernando José Ferreira Lucas Bação

July 2024

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Lisbon, 27.06.2024

ABSTRACT

This thesis investigates the application of anomaly detection techniques to blockchain oracle data, specifically focusing on the Ethereum - US Dollar exchange rate from the Chainlink oracle. The study addresses a critical gap in the reliability of decentralized finance applications by exploring both traditional static machine learning algorithms and novel streaming methods for identifying irregularities in oracle data feeds. The research compares the performance of various anomaly detection algorithms, including Support Vector Machines (SVM), DBSCAN, K-Means, Self-Organizing Maps (SOM), K-Nearest Neighbors (KNN), and Isolation Forest for static methods, and Sliding Window KNN (SWKNN), Robust Random Cut Forest (RRCF), and Half-Space Trees for streaming methods. These algorithms are evaluated using F1 score and recall as primary metrics, with a focus on their ability to detect sudden price spikes. Furthermore, the study examines the incorporating financial indicators such as Relative Strength Index, Average True Range, and Exponential Moving Average to enhance anomaly detection capabilities. The research utilizes UMAP projections for initial visual analysis and conducts comprehensive comparisons between the originally fetched univariate dataset and different multivariate datasets enhanced with additional features in form of financial indicators. The results indicate that multivariate datasets generally improve F1 scores, for static as well as streaming methods, while the streaming methods achieve a higher baseline on univariate datasets, compared to the static methods.

This research contributes to the field by addressing the understudied aspect of anomaly detection in blockchain oracle data, providing insights into the effectiveness of various machine learning techniques in enhancing the security and reliability of DeFi applications.

KEYWORDS

Blockchain; Smart Contracts; Decentralized Finance; Blockchain Oracles; Anomaly Detection; Streaming Data

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AMM	Automated Market Maker
API	Application Programming Interface
ATR	Average True Range
BFT	Byzantine Fault Tolerant
CeFi	Centralized Finance
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DeFi	Decentralized Finance
DEX	Decentralized Exchange
DON	Decentralized Oracle Network
EMA	Exponential Moving Average
ETH	Ethereum
HST	Half-Space Trees
KNN	K-Nearest Neighbors
LMPC	Latest Mass Profile Counter
MACD	Moving Average Convergence Divergence
OCR	Off-Chain Reporting
PBFT	Practical Byzantine Fault Tolerance
PLF	Protocol for Loanable Funds
PoS	Proof of Stake
PoW	Proof of Work
RMPC	Reference Mass Profile Counter
RRCF	Robust Random Cut Forest
RSI	Relative Strength Index
SOM	Self-Organizing Maps
SVM	Support Vector Machines

SWKNN	Sliding Window KNN
TVL	Total Value Locked
TVS	Total Value Secured
UMAP	Uniform Manifold Approximation and Projection
USD	United States Dollar

1. INTRODUCTION

The blockchain technology has enabled the rise of decentralized finance (DeFi) platforms. These platforms rely heavily on blockchain oracles to fetch external data, primarily price feeds, for the execution of smart contracts. However, the integrity and accuracy of these feeds are susceptible to anomalies, either due to simple errors or deliberate manipulations.

1.1 Problem Statement and Research Gap

The reliability of blockchain oracles is critical for functional DeFi applications. While the current body of research points out the so-called oracle problem and describes that oracles represent a single point of failure, it largely overlooks the application of machine learning approaches in the form of anomaly detection to increase the robustness of DeFi environments. This gap exists despite its already established presence in traditional financial security markets (Koosha Golmohammadi & Osmar R. Zaiane, 2015), where anomaly detection approaches were examined regarding stock market data. Furthermore, while statistical deviation analyses have evaluated oracle price feeds by comparing them to their price sources (Ankit Gangwal et al., 2022; B. Liu et al., 2021), and theoretical discussions have explored integrating machine learning approaches into the data validation process of oracles (Zhao et al., 2022), practical applications of these techniques in the scope of blockchain oracles remain unexplored. This gap in research and practice presents a risk, as undetected anomalies can lead to significant financial losses and can potentially erode trust in DeFi applications. Recent incidents, such as the Compound liquidation of November 2020, where an inaccurate DAI/USDC price feed from CoinbasePRO led to over $85 million in wrongful liquidations (Caldarelli & Ellul, 2021), as well as the Synthetix incident in June 2019, where a corrupted API reported prices inflated by 1000 times, resulting in nearly $37 million in wrongful profits due to price arbitrage (Gu et al., 2021), further emphasize this risk.

1.2. Research Questions

Given the identified gaps in the application of anomaly detection techniques to blockchain oracles, this study aims to answer the following research questions:

- To what extent can machine learning techniques accurately identify irregularities in form of sudden price spikes in the Ethereum-US Dollar exchange rate within financial time series data sourced from the blockchain oracle Chainlink?
- How do traditional static machine learning techniques compare to novel data streaming methods in identifying anomalies in the Ethereum-US Dollar exchange rate from Chainlink?
- How does the extraction of new features, based on common financial indicators, impact the performance of anomaly detection techniques?

1.3. Research Objectives

This thesis aims to:

- Investigate the performance of various anomaly detection techniques applied to ETH/USD price feeds from the Chainlink blockchain oracle in assessing their ability to improve data reliability.
- Compare the performance of traditional machine learning algorithms with data streaming algorithms in detecting anomalies by evaluating their detection capabilities with the performance measures F1 score and recall.
- Compare the performance of anomaly detection techniques on the original univariate dataset versus datasets enhanced with extracted financial features.

1.4. Thesis Structure

The thesis is organized to systematically address the research topic through a series of interconnected chapters.

Chapter 1 introduces the research topic and outlines the structure of the thesis, providing an overview of the research objectives and the scope of the study.

Chapter 2 reviews the background necessary for understanding the research context. This includes an examination of blockchain technology, decentralized finance, and blockchain oracles. Additionally, it covers traditional and streaming anomaly detection techniques, establishing a foundation of relevant knowledge.

Chapter 3 discusses related work, focusing on previous research on anomaly detection in the scope of blockchain oracles, traditional finance as well as the decentralized finance domain, identifying existing research gaps and limitations. Furthermore, it presents the literature on the algorithms used in this study

Chapter 4 details the methodologies employed in the research. This chapter describes the experimental setups and methods used to model the various anomaly detection techniques.

Chapter 5 presents the research results, including a discussion of the findings and their relation to the research questions. This chapter provides insights and interpretations based on the data.

Chapter 6 concludes the thesis by summarizing the main findings and their implications. It also suggests directions for future research.

2. BACKGROUND

This background chapter provides the theoretical foundation necessary for understanding the key concepts and technologies involved in this research. It is separated from the related work to offer a clear, focused exploration of the fundamental principles underlying blockchain technology, smart contracts, decentralized finance, and anomaly detection. By providing this theoretical background, the chapter aims to provide the reader with the necessary knowledge to understand the context of this thesis.

2.1. BLOCKCHAIN TECHNOLOGY

Blockchain technology, first mentioned as a so-called timestamp server in the 2009 Bitcoin white paper by its anonymous creator Satoshi Nakamoto, is a decentralized append-only database of transactions. It executes and stores transactions without the need for a trusted central authority. In the financial domain, it can be used as an electronic payment system that is not based on trust but on the mutual cryptographic verification of the market participants (Nakamoto, 2008).

A central component of blockchain technology is the mechanism by which consensus is achieved among network participants. Consensus mechanisms are fundamental in ensuring that all participants in the network reach a common decision on valid transactions that get appended to the blockchain without the need for a central authority.

Proof of Work (PoW) relies on computational power to validate transactions and create new blocks. In this mechanism, miners compete to solve complex mathematical problems, with the first one to solve it gaining the right to add a new block to the blockchain. Consensus is reached when the newly proposed block is accepted by the majority of the network's nodes, ensuring that all nodes have the same updated version.

Proof of Stake (PoS), in contrast, uses the concept of staking to validate transactions and create new blocks. Participants are required to hold a certain amount of cryptocurrency as collateral, and the higher the stake, the higher their chances of being chosen to validate the next transaction and create a new block.

Practical Byzantine Fault Tolerance (PBFT), in comparison, employs a master node to generate new blocks, which are then verified by the other nodes in the network. Consensus is achieved when a majority of the nodes agree on the validity of the block, ensuring that the network can still reach a consensus even if some nodes act maliciously (C. Zhang et al., 2020).

While consensus mechanisms ensure agreement across the network, blockchain technology is defined by several key features blockchain technology is characterized by several key features that aim to enhance its functionality and trustworthiness. Decentralization is a core aspect of these, eliminating the need for a central authority, like a central bank, to validate

transactions. Instead, transactions are conducted directly between peers in a peer-to-peer network. This aims to eliminate the reliance on trust that is needed in centralized systems.

In addition to its decentralized nature, persistency is another important characteristic. The blockchain ensures the integrity and authenticity of data by providing a transparent and unalterable record of all transactions. Each block in a blockchain contains a cryptographic hash of the previous block, creating a chain of appended blocks. Any attempt to modify information in a block would require changing the hash of all preceding blocks, making it extremely challenging and detectable.

While ensuring data integrity, blockchain technology also offers a degree of anonymity. This feature allows users to interact with the network using randomly generated addresses. Users can have multiple addresses within the blockchain network to protect their identity. Since the blockchain operates on a decentralized system, there is no central authority monitoring or recording the user's private information.

Despite this anonymity, auditability is still maintained. This is achieved through the recording of all transactions in a digital distributed ledger, along with digital timestamps. Such a system enables anyone to access previous records by connecting to any node or running a node in the network themselves, ensuring transparency without compromising user privacy (Monrat et al., 2019).

2.2. SMART CONTRACTS

Building upon the foundational concepts of blockchain technology, smart contracts represent a significant advancement in decentralized systems. In 1994, Nick Szabo first introduced the concept of smart contracts as a computerized transaction agreement that automatically enforces the terms of a contract. Blockchain-based smart contracts are contract agreements written in computer code that automatically execute transactions when a list of pre-programmed conditions is met. They are embedded on the blockchain and enable the settlement of contracts. By being embedded on the blockchain, the transactions processed by smart contracts are stored in the same auditable and immutable fashion as regular peer-to-peer blockchain transactions (Zheng et al., 2019).

Smart contracts possess several distinctive characteristics that set them apart from traditional contracts. Firstly, they are machine-readable computer programs, typically written in programming languages like Solidity, and are stored on the blockchain. These contracts are designed to execute automatically when certain conditions are fulfilled and can be accessed and executed by anyone in the blockchain network.

Another key feature of smart contracts is their event-driven nature. They are triggered by specific events on the blockchain network, such as the completion of a transaction or the arrival of new data. Upon being triggered, the smart contract executes a set of predefined actions automatically. This automation extends further: once deployed on the blockchain

network, smart contracts operate autonomously, without the need for human intervention or third parties. Their autonomous nature ensures that once initiated, the contract will carry out its instructions independently.

Enhancing their security and reliability, smart contracts are stored on a distributed ledger, which means they are replicated across all participating nodes of the blockchain network. This distributed nature makes them highly secure and resistant to tampering, as any attempt to modify a smart contract would need approval from the majority of nodes on the network.

Despite their many capabilities, smart contracts have a notable limitation: their isolation. Smart contracts are self-contained within the blockchain network and cannot access external data on their own. This limitation necessitates the use of third-party data providers, known as blockchain oracles, to integrate external information into the contract.

To fully understand the functionality of smart contracts, it's essential to examine their key components. At the core of every smart contract is the Smart Contract Agreement, a digital agreement between two or more parties. This agreement forms the foundation of the contract, containing predefined functions that enable it to store information, process inputs, and generate outputs based on the specified terms and conditions.

Initializing the smart contract is the Constructor Function, a special function executed only once during the contract's creation. This function plays a crucial role in setting the initial values of the contract's states. These states, which can be either constant or writable, are variables that hold information related to the contract. Constant states remain immutable once set, while writable states can be modified and updated by the smart contract functions, providing flexibility in contract execution.

Within the smart contract, functions serve as the active components, capable of reading or modifying its states. These functions execute specific actions based on the contract's terms and conditions. They are categorized into two types: read-only functions that do not consume gas and write functions that require gas for their execution.

The concept of gas is integral to the efficient operation of smart contracts. Representing a unit of measurement for computational effort, gas is necessary for executing write functions. This mechanism serves a dual purpose: it prevents infinite execution loops in smart contracts and ensures that they are executed efficiently. Moreover, the gas concept acts as a safeguard against potential attacks on the network through resource-intensive operations, contributing to the overall security and stability of the blockchain ecosystem (S. N. Khan et al., 2021).

2.3. DECENTRALIZED FINANCE (DEFI)

Building on the foundation of smart contracts, decentralized finance (DeFi) represents an ecosystem of financial applications constructed on top of blockchain networks. DeFi aims to provide financial services traditionally offered by central institutions in a decentralized manner. By leveraging the open and transparent foundations established by blockchains, DeFi

allows for the development of financial contracts that can be accessed by participants, provided they comply with the network's rules.

DeFi presents several key characteristics that distinguish it from traditional centralized finance (CeFi) services, largely inherited from its blockchain origin.

Transparency is a fundamental feature, enabling users to examine and verify the specific rules of financial products. This is facilitated by the execution through smart contracts and the public storage of data on the blockchain. The open nature of DeFi protocols allows for continuous scrutiny by the community, potentially affecting the identification and resolution of issues or vulnerabilities.

DeFi also changes the way users interact with their assets. Users act as custodians of their own assets, which means these assets cannot be manipulated, transferred, or destroyed without their explicit consent. This approach differs from traditional financial systems, where intermediaries often hold and manage assets on behalf of users. The concept of self-custody in DeFi aligns with the broader blockchain principle of decentralization.

Additionally, DeFi aims to increase accessibility, potentially allowing individuals with basic computer knowledge and internet access to create and use DeFi products. This approach may expand financial services to a broader audience, addressing some barriers that exist in traditional finance (Qin et al., 2021).

This permissionless nature of DeFi platforms means that users can participate in complex financial activities without needing to meet traditional banking requirements or credit checks, potentially democratizing access to financial services.

The growth of DeFi is evident, with the total value locked (TVL) in DeFi protocols reaching USD 62 billion in February 2024 (DefiLlama). This figure suggests an increasing adoption of DeFi solutions. As the DeFi sector expands, it has begun to offer services traditionally associated with centralized Finance (CeFi). Among the various types of DeFi protocols, three main categories have emerged:

Decentralized exchanges (DEXs) are protocols that facilitate the exchange of digital assets without a centralized intermediary. By enabling peer-to-peer transactions, DEXs aim to reduce the need for traditional exchange platforms. Many DEXs utilize automated market maker (AMM) models, which employ liquidity pools and mathematical formulas to determine asset prices, as an alternative to traditional order books.

Protocols for loanable funds (PLFs) provide lending and borrowing services. In these protocols, lenders contribute their funds to a smart contract with a predefined interest rate. Borrowers can access loans by providing collateral that exceeds the borrowed amount, based on a predefined collateral multiplier. This system is designed to secure loans and protect lenders' funds in case of borrower default.

Stablecoins are digital assets designed to maintain a stable value relative to a specific currency, addressing the challenge of price volatility in cryptocurrencies. One method of achieving price stability is through on-chain collateral, where the stablecoin is backed by a reserve of digital assets held in a smart contract on the blockchain. This mechanism allows stablecoins to offer loan solutions while aiming to maintain a stable value pegged to a currency (Werner et al., 2022).

2.4. BLOCKCHAIN ORACLES

Blockchain oracles serve as a critical link between smart contracts and the external world, providing the necessary data for contract execution. They bridge the gap between the isolated blockchain environment and real-world information sources, enabling smart contracts to interact with and respond to external events and data.

The nature of data sources leads to different types of blockchain oracles. Software oracles handle data from online sources, extracting real-time information such as currency exchange rates, stock prices, and commodity costs. They access APIs to fetch this information and deliver it to the blockchain in a format that smart contracts can utilize.

Moving from the digital to the physical realm, hardware oracles interact directly with the physical world to gather and verify data from external sources. They use a range of devices, including sensors, scanners, and internet of things devices, to collect real-world data such as temperature readings, physical movements, or logistical data.

While software and hardware oracles automate data collection, there are scenarios where human judgment is irreplaceable. This is where human oracles come into play. Human oracles rely on human input to provide data or verify its accuracy. They often use mechanisms like voting to aggregate responses, ensuring that the data provided to smart contracts reflects the consensus of a group rather than a single source. This approach is particularly useful in scenarios where automated systems cannot easily judge data, such as in subjective assessments or events not easily quantifiable by machines (Al-Breiki et al., 2020).

These oracle types can serve a wide range of applications, from automating payments based on financial indices to executing contracts based on sensor data in supply chain management. The choice of oracle type depends on the specific requirements of the application, including the need for speed, reliability, and the nature of the data required.

Ensuring the integrity and accuracy of the data provided to smart contracts is crucial for blockchain oracles. Different trust models address how oracles manage data sources and mitigate risks of data manipulation. Centralized oracles rely on data from a single source or node. While this setup offers high efficiency and fast data retrieval, it introduces significant risks, as the singular data source represents a single point of failure, making it susceptible to manipulation or errors. To address these vulnerabilities, decentralized or consensus-based oracles have emerged. These enhance security and reliability by using data from multiple

nodes. These nodes work together to verify data, which is then aggregated to form a median value. This consensus-based approach aims to reduce the risk of manipulation, as corrupting the data would require compromising multiple nodes.

Design patterns in blockchain oracles describe how data is stored, accessed, and transmitted. The immediate-read pattern stores data directly within their contracts on the blockchain, allowing other smart contracts to immediately access the data via request calls. An example of this is a university system where academic certificates are stored on the blockchain, allowing instant verification without external data sources. The publish-subscribe pattern enables oracles to act as a broadcast service, pushing updates as they become available. This pattern is suitable for data that frequently changes, such as financial market prices or weather updates, ensuring smart contracts receive timely and relevant data. In the request-response pattern, oracles respond to specific requests by delivering the needed data subset at the requested time. This method is efficient for managing large volumes of data while maintaining adequate performance (Beniiche, 2020).

Oracles engage in various interactions with the external world, either inserting data into the blockchain or delivering data from the blockchain. Inbound oracles bring external data, such as asset prices or weather conditions, onto the blockchain. These are used for contracts that depend on external events to trigger actions within the blockchain. Outbound oracles allow the blockchain to interact with external systems. For instance, a smart contract might automatically notify an external database or trigger a payment system outside the blockchain once certain conditions are met within the contract (Al-Breiki et al., 2020).

As oracle technology has evolved, decentralized oracles have demonstrated advantages and are increasingly used to provide data feeds for DeFi projects. While centralized oracles typically excel in data processing speed and are simpler to develop and implement, decentralized oracles receive data from multiple sources and use decentralized validation mechanisms. This approach decreases the potential impact of malicious data and manipulation attempts, thereby enhancing the security and reliability of the data provided.

Decentralized oracles can employ various methods for data validation and aggregation. Aggregation-based processing uses predefined rules and mechanisms, such as taking the median, to generate a result from multiple data providers, regardless of the individual quality or performance of the provided data. This method aims to neutralize the impact of single malicious data sources. Staking-based processing involves participants locking a specific amount of cryptocurrency as collateral to validate data. This mechanism encourages honest behavior by penalizing validators who provide inaccurate information. Reputation-based processing assigns reputation scores to data providers based on their historical performance in terms of the accuracy of the reported data. Providers with higher reputations then receive greater influence in the data validation process (Zhao et al., 2022).

2.5. ORACLE SOLUTIONS

This chapter examines contemporary oracle solutions, focusing on Chainlink 2.0, Chainlink Data Streams, and the Pyth Data Oracle. The analysis highlights their architecture and functionality. These specific oracles were selected based on several criteria: their total secured value (TVS), the number of protocols each oracle secures, the availability of an ETH/USD Exchange rate broadcast, and the accessibility of technical documentation provided by the oracle providers (DefiLlama/Oracles).

2.5.1. Chainlink 2.0

Chainlink 2.0 represents the second-generation oracle network from the provider Chainlink. It functions as a decentralized publish-subscribe oracle, integrating multiple components to enhance the security and accuracy of data feeds provided to consumer blockchains. Central to the Chainlink 2.0 network is the Decentralized Oracle Network (DON), which serves as the primary framework for data consensus and aggregation. The DON employs off-chain computing resources to optimize the efficiency of price feed transmissions to blockchain systems.

Several key components contribute to the functionality of Chainlink 2.0. Oracle Nodes, operated independently, are crucial for the DON's operation. They retrieve external data via APIs and adapters, which is then processed and transmitted to the DON. External APIs serve as the interface through which oracle nodes interact with external data sources, facilitating the retrieval and transmission of off-chain data to the blockchain. Adapters, standardized software components, streamline communication between APIs and oracle nodes.

The Aggregator Contract, a smart contract, receives data from multiple oracle nodes and combines it according to pre-programmed rules to calculate a consensus-based price feed. The Hybrid Smart Contract, a variant of a conventional smart contract, includes both an on-chain component (following programmed rules and logic) and an off-chain component (executing data acquisition and processing algorithms of the DON). Finally, the Storage Component serves as a repository for aggregated data, ensuring its persistence and accessibility for smart contract execution (Breidenbach et al., 2021).

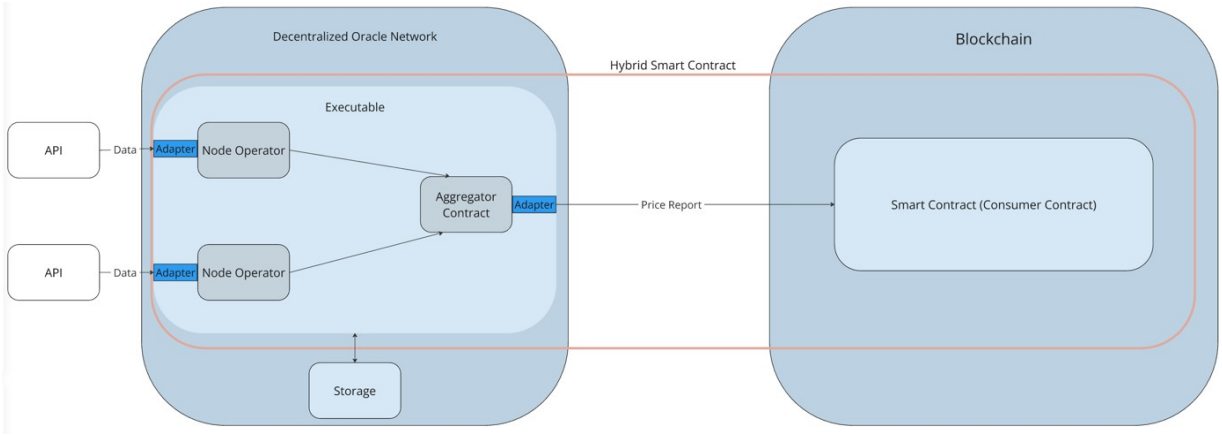

Figure 2.1- Chainlink 2.0 design architecture

A key aspect of this system is its consensus mechanism, known as Off-Chain Reporting (OCR). This mechanism facilitates the aggregation of oracle reports produced by DONs and is based on the Byzantine Fault Tolerant (BFT) protocol. The aim of said mechanism is to maintain the liveness and correctness of the price feeds even in the presence of $f < n/3$ arbitrarily faulty nodes, if the number of faulty nodes f is less than one-third of the total number of participating nodes n. Furthermore, OCR can maintain reliability and security properties in the presence of faulty nodes as long as at least $2f+1$ honest nodes participate in the aggregation.

The OCR process consists of three main phases: oracle report creation, verification, and price feed broadcasting. During creation, a leader node, selected by the network nodes' quorum, requests signed price observations from follower nodes. These observations are then aggregated into an oracle report. In the verification phase, this report is sent to the follower nodes for validation and signing, after which a final quorum report is created. The broadcasting phase involves a randomly selected node sending the final attested report to the aggregator contract, which verifies the signatures and calculates the aggregated median value.

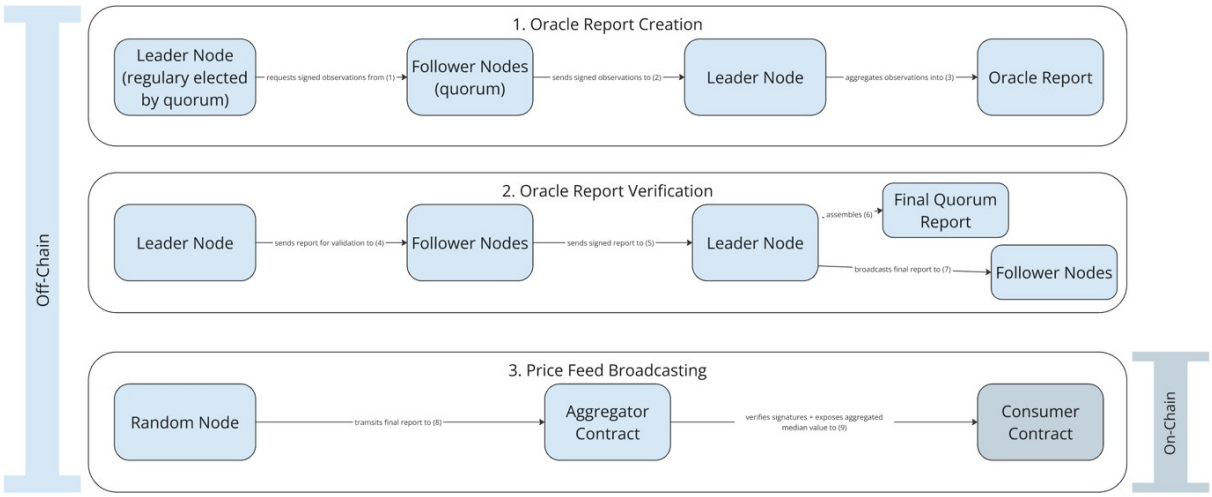

Figure 2.2 - Chainlink 2.0 Off-Chain Reporting execution flow

The pillars of Off-Chain Reporting (OCR) include several key concepts. All-or-Nothing Off-Chain Report Broadcasting ensures that the final attested report is either released to all participating nodes or to none, thereby promoting fairness and participation in the report broadcast.

Reliable Transmission guarantees that OCR reports and messages are transmitted reliably to the smart contract within a predefined time interval, even in the presence of faulty or malicious nodes.

Contract-Based Trust Minimization represents the employment of additional filtering mechanisms by the aggregation contract, to identify and discard potentially faulty OCR-generated reports, such as those with values that deviate significantly from other recently received reports. This approach provides an additional layer of correctness enforcement (Chainlink 2.0).

2.5.2. Chainlink Data Streams

Chainlink Data Streams builds upon the principles of Chainlink 2.0, aiming to provide price feeds on demand and reduce on-chain operations. While it shares the foundation of Decentralized Oracle Networks (DONs) with Chainlink 2.0, it introduces additional components to its structure.

The design of Chainlink Data Streams incorporates several interconnected elements. The Chainlink Data Engine stores signed reports from the DON and delivers them to the Chainlink Automation Registry until they are requested. This setup ensures that data is available when needed. The Automation Registry's role is to monitor and execute both on-chain and off-chain upkeep contracts, facilitating interactions and ensuring the correct execution of scheduled tasks. When specific events are triggered on consumer contracts, the Off-Chain Upkeep Contract monitors these events and requests price feeds from the DON as necessary. Once the price feeds are obtained, the On-Chain Upkeep Contract is responsible for deploying them onto the blockchain, ensuring that the data is accessible. To safeguard the authenticity and integrity of the data, the Verifier Contract operates on the target blockchain to cryptographically verify the signatures of incoming price reports (Chainlink Data Streams).

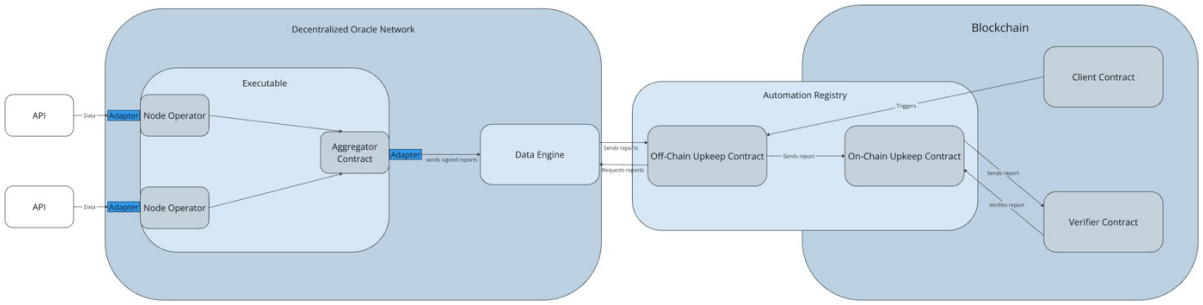

Figure 2.3 - Chainlink Data Streams design architecture

The data retrieval process follows a structured execution flow. When a user initiates an action on the blockchain, such as a trade on a broker platform, a trigger is activated within the client

contract. This trigger is then identified by the off-chain upkeep contract, which requests a price report from the data engine. Subsequently, the data engine produces a signed report and transmits it back to the off-chain upkeep contract, which forwards the report to the on-chain upkeep contract. The on-chain upkeep contract then calls the verifier contract, responsible for verifying the report and providing the output (Chainlink Data Streams - Execution Flow).

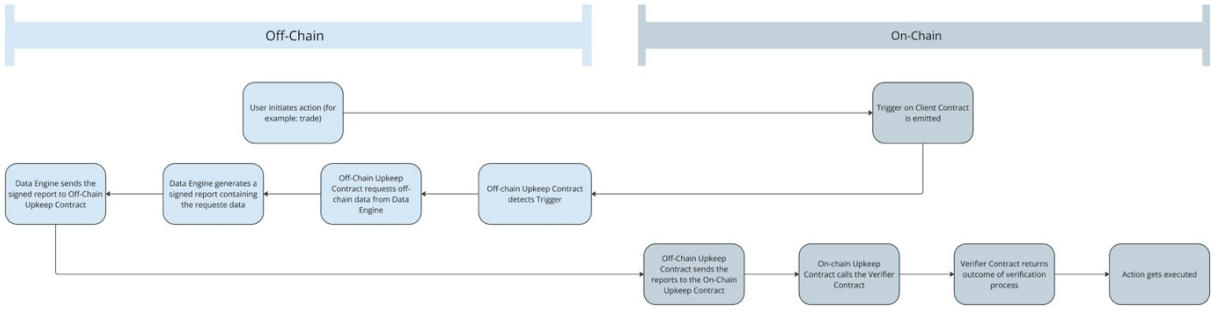

Figure 2.4 - Chainlink Data Streams data retrieval execution flow

Chainlink Data Streams supports various trigger types for data retrieval. Time-based triggers execute functions on a predefined schedule, suitable for regular updates. Custom logic triggers allow function execution based on specific conditions written in Solidity syntax, offering flexibility for complex scenarios. Log triggers respond to specific blockchain events, enabling real-time data retrieval (Chainlink Data Streams - Trigger).

2.5.3. Pyth Data Oracle

Pyth Data Oracle also operates as a request-response oracle, transmitting data only upon request. The design includes several key participants that collaborate to collect, validate, and transfer price data between blockchains in real-time. At the core of the oracle is the pythnet blockchain, which receives price data from various Publisher APIs via its validator nodes. These APIs transfer off-chain data collected from independent publishers to the validator nodes on the pythnet blockchain. Validator Nodes then perform the task of validating and verifying the received price data before it is sent to the aggregation mechanism. The Aggregation Mechanism is responsible for combining the data feeds from the validator nodes into a median based on predefined conditions. Once the data is aggregated, the Message Buffer Program creates a message on the pythnet blockchain encapsulating the aggregated price data, ready for cross-chain transmission. The Cross-Chain Messaging Protocol, utilizing networks like the Wormhole network, transmits the aggregated price data from the pythnet blockchain to the target blockchain. Finally, the Consumer Smart Contract on the target blockchain receives and integrates the aggregated price data into its operations.

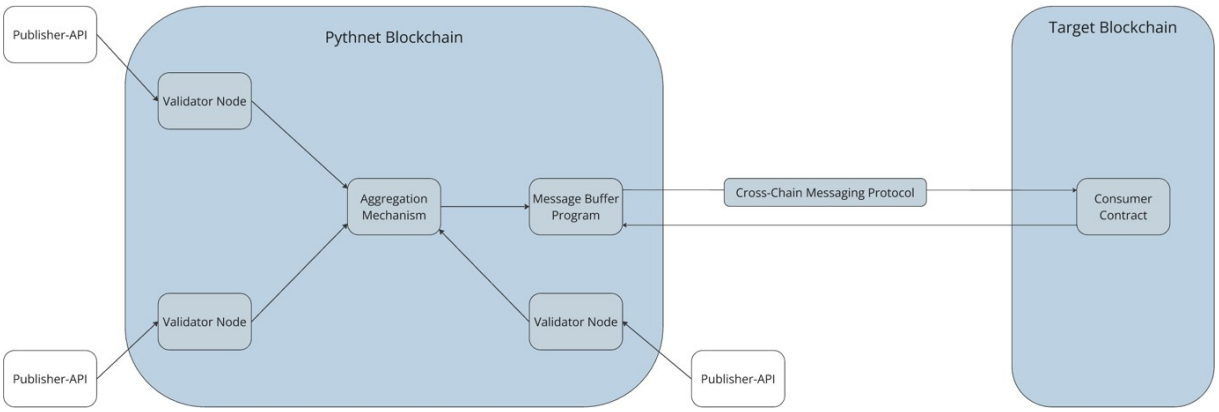

Figure 2.5 - Pyth Oracle design architecture

A key component of the Pyth protocol is its aggregation algorithm. This algorithm processes data in the form of price feeds and confidence intervals from individual publishers. For example, a publisher might submit a confidence interval of ±5€ along with their reported price of 2500€ for ETH/USD. The algorithm uses a weighted median that accounts for the stake-weights of the respective publishers. Each publisher is given three votes: one for the reported price and the other two for their reported negative and positive confidence intervals.

For this it uses the following objective function:

$$\frac{1}{3}\sum_i s_i|R-p_i| + \frac{2}{3}\sum_i s_i \max(|R-p_i| - c_i, 0)$$

In the first step, the algorithm calculates the weighted median of the price using an objective function. The first term is the sum of the absolute differences between the aggregate price *R* and the reported prices of individual publishers *pi*, weighted by the stake-weight of each publisher *si*. This term aims to ensure that the aggregate price is more influenced by publishers with higher stake-weights. The second term reflects the varying levels of accuracy of the publishers and their confidence intervals, contributing to the determination of the aggregate price. It ensures that the aggregate price lies within the confidence intervals of the publishers, reflecting the collective information provided by the publishers while accounting for their varying levels of accuracy. In a second step the algorithm calculates the weighted 25th and 75th percentiles of the median and sets the higher of both values as the confidence interval for the aggregated price (Pyth Data Association, 2023).

2.6. ANOMALY DETECTION

Anomaly detection is a process used to identify data points or patterns that differ significantly from defined normal behavior. This section examines various anomaly detection techniques, comparing their functionalities.

Anomalies, as described by (Prasad et al., 2009) are data points or patterns that deviate substantially from what is considered normal behavior. To illustrate this concept, Figure 2.6 presents a 2-dimensional dataset where the majority of data points form a dense cluster. Points labeled 1, 2, and 3 in this figure are notably distant from this cluster, potentially indicating anomalies.

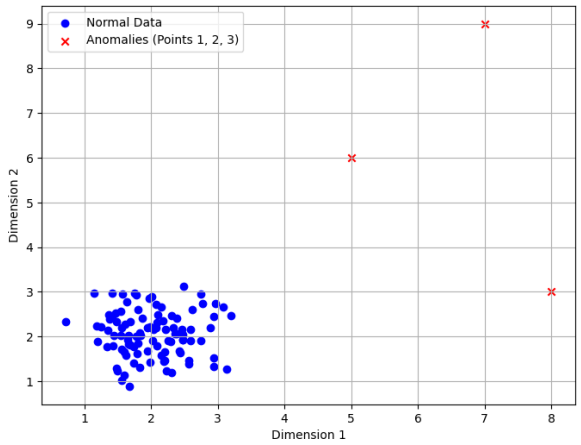

Figure 2.6 - Anomalies in a 2-dimensional dataset, where points 1, 2, and 3 are significantly different from the majority of data points, indicating that they are anomalies.

Anomalies can be classified into several categories, each with distinct characteristics. Point anomalies refer to individual data instances that are considered anomalous when compared to the rest of the data. Contextual anomalies are data instances that are deemed anomalous in one specific context but not in another, highlighting their context-dependent nature. Collective anomalies occur when a group of related data instances is considered anomalous when evaluated within the context of the entire dataset (Choi et al., 2021).

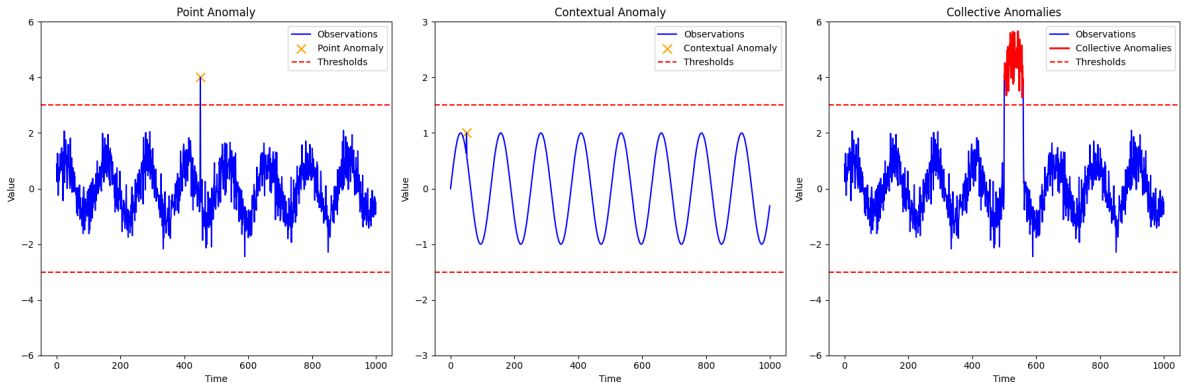

Figure 2.7 - Types of Anomalies in Time Series Data. From left to right, the subplots illustrate point anomaly, contextual anomaly, and collective anomalies.

2.6.1. Classification of Traditional Anomaly Detection Techniques

Anomaly detection techniques rely on algorithms and models that learn from data and recognize patterns without explicit rules or instructions. These techniques can be categorized based on the availability and use of labeled data.

Supervised anomaly detection methods involve training models to differentiate between normal and anomalous data using labeled training data containing both normal and anomalous instances. This approach is particularly useful when there's a clear understanding of what constitutes an anomaly in the system.

Unsupervised anomaly detection identifies anomalies without the need for labeled training data. The model learns the normal patterns in the data independently and detects deviations as anomalies. This method is valuable in systems where new types of anomalies may emerge over time.

Semi-supervised anomaly detection techniques combine elements of supervised and unsupervised methods by using both labeled and unlabeled data to identify anomalies. This approach can be beneficial when there's limited labeled data available, which is often the case in evolving systems (Nassif et al., 2021).

2.6.2. Classification Based Methods

Classification-based anomaly detection techniques use labeled data instances to train a model and classify test instances as either normal or anomalous. They can be categorized into multi-class and one-class approaches.

Multi-class techniques train the model on data instances labeled with multiple normal classes to distinguish them from anomalies. This approach is useful when there are distinct categories of normal behavior in the data.

One-class techniques assume that all training instances belong to a single normal class and learn a discriminative boundary around these instances to identify anomalies. This method is particularly effective when anomalies are rare and diverse.

The effectiveness of classification-based methods depends on the availability of labeled data for both normal and anomalous instances. Obtaining labeled anomalous data can be challenging in some scenarios, which can limit the applicability of these methods (Prasad et al., 2009).

2.6.2.1. One Class Support Vector Machines (SVM)

Support Vector Machines (SVM) find the optimal hyperplane that separates different classes in the feature space with a maximum margin. One-Class SVM learns to identify normal behavior and flag deviations as potential anomalies.

A kernel function maps the input features into a higher-dimensional space where linear separation of the data is possible. The algorithm finds the hyperplane that separates the classes and maximizes the distance between the hyperplane and the nearest data points from each class, known as support vectors (Schölkopf et al., 2000). SVM algorithm aims to solve an optimization problem to find the smallest hypersphere containing most of the data points. Points falling outside this hypersphere are considered anomalies.

The One-Class SVM algorithm solves the following optimization problem:

$$\min_{\mathbf{w},\rho,\xi_i} \left(\frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho \right)$$

Subject to:

$$(\mathbf{w} \cdot \phi(\mathbf{x}_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n$$

Where:

- w is the normal vector to the hyperplane.
- $\phi(xi)$ is the kernel function mapping the inputs to a higher-dimensional space.
- p is the offset which adjusts the position of the hyperplane relative to the origin.
- ξi is the slack variables that allow for some data points to be on the wrong side of the hyperplane; these points are considered anomalies.
- v is a parameter that controls the trade-off between maximizing the margin (distance from the hyperplane to the origin) and minimizing the fraction of outliers (points on the wrong side of the hyperplane).
- n is the number of observations.

The decision function is given by:

$$f(\mathbf{x}) = (\mathbf{w} \cdot \phi(\mathbf{x})) - \rho$$

Where:

- $f(x) \geq 0$ indicates that points are normal.
- $f(x) < 0$ indicates that points are anomalies.

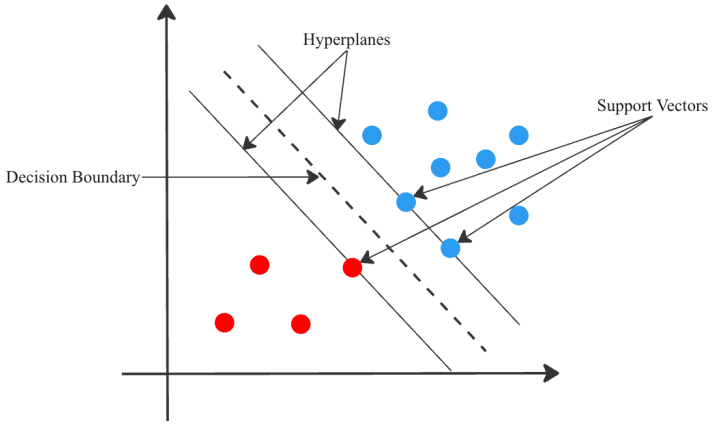

Figure 2.8 - Illustration of the concept of Support Vector Machines (SVM) with a decision boundary, hyperplanes, and support vectors. The red and blue points represent different classes.

2.6.3. Clustering / Density Based Methods

Cluster-based anomaly detection techniques are classified as unsupervised machine learning, techniques and involve grouping similar data instances into clusters.

2.6.3.1. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

The DBSCAN algorithm requires two key parameters: *Eps,* representing the radius for neighboring point search, and MinPts, indicating the minimum number of points required to form a cluster. These parameters are crucial for identifying three types of points in the dataset: core points, border points, and noise points.

Core points are those data points that have at least MinPts within their Eps neighborhood, indicating they are in dense regions. Border points have fewer than MinPts within their Eps neighborhood but are reachable from core points, thus belonging to a cluster. Noise points do not meet the criteria to be classified as core or border points and are considered outliers.

The algorithm initiates by randomly selecting a core point and iteratively expanding clusters from these core points. It assigns border points to the appropriate clusters, continuing this process until all data points are classified. This allows DBSCAN to effectively separate clusters in data of varying densities (K. Khan et al., 2014). Points that get classified as a noise point will be flagged as an anomaly.

The distance between data points can be measured in various distance metrics. In the context of the *Euclidean distance*, the distance between two points is defined by:

$$\text{distance}(p, q) = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2}$$

Where:

- *p* and *q* are points in *d*-dimensional space.
- *pi* and *qi* are the *i-th* coordinates of points *p* and *q*.
- *d* is the number of dimensions.

The neighborhood of a point is defined as:

$$N_\epsilon(p) = \{q \in \mathbb{R}^d \mid \text{distance}(p, q) \leq \epsilon\}$$

Where:

- *Nε(p)* is the set of all points *q* whose distance from *p* is less than or equal to *ε*.
- *ε* is the radius defining the neighborhood around point *p*.

Core Points get identified with the following approach:

$$\text{Core}(p) = \begin{cases} \text{True} & \text{if } |N_\epsilon(p)| \geq \text{MinPts} \\ \text{False} & \text{otherwise} \end{cases}$$

Where:

- *Core(p)* is a Boolean condition indicating whether point *p* is a core point.
- $|N\epsilon(p)|$ is the number of points in the *ε-neighborhood* of *p*.
- *MinPts* is the minimum number of points required to form a dense region.

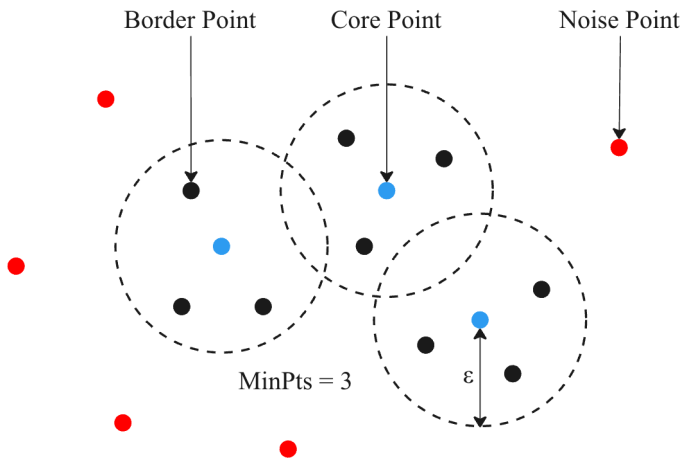

Figure 2.9 - Illustration of the DBSCAN clustering algorithm. It highlights core points, border points, and noise points. The core points have at least a minimum number of points within a specified distance (ε). Border points are within ε distance of a core point but have fewer than MinPts. Noise points do not meet either criterion.

2.6.3.2. K-Means

The K-Means algorithm's objective is to partition a dataset into k clusters by minimizing the sum of squared distances between data points and their respective cluster centroids. After an initialization of the centroids, based on a chosen initialization technique, the algorithm iterates through two main steps.

An assignment Step where each data point is assigned to the nearest centroid based on euclidean distance.

This is represented as:

$$c_i = \arg\min_j \text{distance}(x_i, \mu_j)$$

Where:

- *ci* is the cluster assignment for point x_i.
- μj is the *j-th* cluster center.

An Update Step, where the centroids are recalculated as the mean of all data points assigned to that cluster.

This update is calculated using:

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

Where:

- μj is the updated cluster center for cluster *j*.
- *Cj* is the set of points assigned to cluster *j*.
- |*Cj*| is the number of points in cluster *j*.

This process continues iteratively until convergence, where the centroids no longer change significantly, or a predefined number of iterations is reached (Ahmed et al., 2020). Points with a distance to the centroid, that is higher than a set threshold will be classified as anomalies.

2.6.3.3. Self-Organizing Map (SOM)

Self-Organizing Maps are an algorithm used for data visualization and clustering. The idea behind SOM is to organize data in a topographic manner, allowing for the visualization of complex high-dimensional data in a simplified way. The SOM algorithm consists of two main stages: the competitive stage and the cooperative stage. In the competitive stage, the algorithm sets the initial weights of the neurons and then selects the best matching neuron, also known as the "winner," based on the smallest distance. In the cooperative stage, the weights of the winning neuron and its neighboring neurons are adjusted to better represent

the input data, based on a neighborhood function. The neighborhood function defines the extent to which neighboring neurons of the winning neuron will be updated when adapting their weights. This process is iterated with a gradually decreasing adjustment of the weight vectors based on the learning rate until the SOM stabilizes. (Van Hulle, 2012). Points that are significantly distant from their BMU, exceeding a predefined threshold, are classified as anomalies.

The selection of the BMU is based on the minimum distance criterion:

$$\mathrm{BMU}(x) = \arg\min_{w_j} \mathrm{distance}(x, w_j)$$

Where:

- *BMU*(*x*) is the index of the neuron with the smallest distance to x.
- *wj* is the weight vector of neuron *j*.

Updating the weight vectors of the BMU and its neighbors follows this approach:

$$w_j(t+1) = w_j(t) + \theta_{j,\mathrm{BMU}}(t) \cdot \alpha(t) \cdot (x - w_j(t))$$

Where:

- *x* is the input vector.
- *wj*(*t*) is the weight vector of neuron *j* at time t.
- $\theta j, BMU(t)$ is the neighborhood function, decreasing with the distance from the BMU.
- $\alpha(t)$ is the learning rate, which typically decreases over time.

2.6.4. Nearest Neighbor / Proximity Based Methods

Nearest neighbor-based anomaly detection techniques assume that normal data instances are grouped close together, while anomalies are far from their nearest neighbors.

2.6.4.1. K-Nearest Neighbors (k-NN)

The idea behind kNN is to classify a data point based on the majority class of its k-nearest neighbors. For every new data point that needs to be classified, its distance to all other data points is calculated based on one of various distance metrics. After the k-nearest neighbors are selected based on this distance, the data point is labeled based on the majority class of the neighbors. (Wong MA & Tom Lane, 1982). Points with an average distance to their nearest neighbors, that is higher than a set threshold will be classified as anomalies.

The average distance gets calculated as follows:

$$\text{AvgDistance}(x) = \frac{1}{k}\sum_{i=1}^{k}\text{distance}(x, y_i)$$

Where:

- *k* is the number of nearest neighbors considered for calculating the average distance.
- *distance(x,yi)* is the distance between the data point x and each of its *k* nearest neighbors y_i.

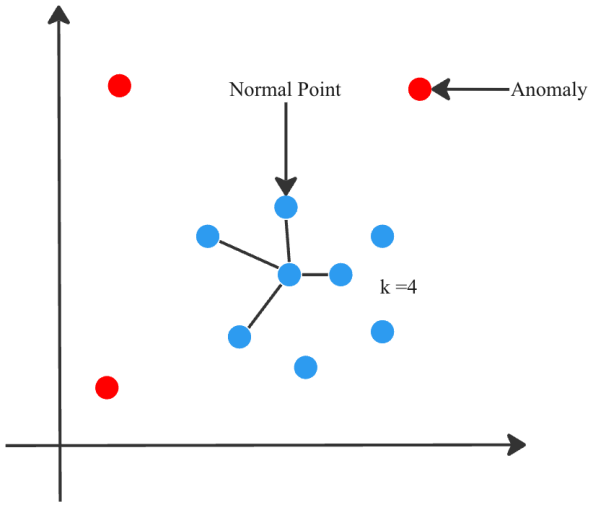

Figure 2.10 - Illustration of the k-NN algorithm. It shows normal points and anomalies in a dataset. The normal point has its nearest neighbors (k=4) indicated. Anomalies, with less than 4 neighbors are marked red.

2.6.5. Isolation Forest

The fundamental concept of Isolation Forest involves separating anomalous instances from the majority of the data. In theory, anomalies are rare and therefore easily distinguishable from the rest of the data. In Isolation Forest, a forest of random trees is built by recursively partitioning the data points until each instance is isolated. Due to their differentiating values and lower occurrence, anomalies will result in, on average, shorter tree path lengths. This enables the calculation of an anomaly score for each data point based on the average path length of that instance across all trees in the forest. (F. T. Liu et al., 2008). Points with an anomaly score higher than a set threshold will be classified as anomalies.

The average path length of a data point across all trees in the forest is calculated by the formula:

$$\bar{h}(x) = \frac{1}{t}\sum_{i=1}^{t} h_i(x)$$

Where:

- t is the number of trees in the forest.
- $hi(x)$ is the path length of sample *x* in the *i-th* tree.

The anomaly score is computed using the formula:

$$s(x) = 2^{-\frac{\bar{h}(x)}{c(n)}}$$

Where the normalization factor to adjust the path length relative to the size of the tree is calculated as:

$$c(n) = 2\ln(n-1) + \gamma - \frac{2(n-1)}{n}$$

Where:

- n is the number of external nodes.
- γ is Euler's constant, approximately 0.5772.

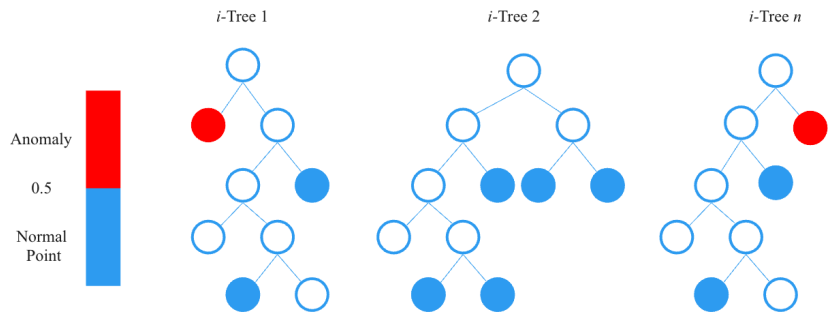

Figure 2.11 - Illustration the Isolation Forest algorithm using multiple decision trees. Anomalies that require fewer splits to isolate are marked in red and to normal points are marked blue. The color bar indicates the anomaly threshold,

2.7. Static Algorithms vs. Streaming Algorithms

Anomaly detection techniques can be broadly categorized into two main types based on how they process data: static (offline) algorithms and streaming (online) algorithms. This section explores the characteristics and applications of these two approaches.

Static algorithms, also referred to as offline or traditional machine learning algorithms in this thesis, operate on complete datasets that have been fully collected and remain unchanged during processing. These algorithms are typically used in scenarios where comprehensive analysis of the entire dataset is required to produce results. When new data becomes available, static algorithms often require reprocessing and retraining of the entire dataset, which may present challenges in situations demanding real-time processing.

In contrast, streaming algorithms, also known as online algorithms, process data sequentially, one data point at a time, without needing access to the entire dataset simultaneously. These algorithms are designed to update their output continuously as new data becomes available. Streaming algorithms often have lower memory requirements compared to static algorithms, as they typically store only a constant amount of memory rather than entire large datasets. Additionally, streaming algorithms have the potential to adapt to new trends or changes in data patterns as they occur (Lu et al., 2023).

2.7.1. Sliding Window k-Nearest Neighbor (SWkNN)

The Sliding Window k-Nearest Neighbors (SWkNN) algorithm is a variation of the traditional k-Nearest Neighbors (kNN). SWkNN uses a sliding window approach to maintain a recent subset of the data. This window only includes the most recent observations and moves over time, discarding older data points. For each incoming data point, the algorithm calculates the distance to other points in the sliding window. It then selects the k closest neighbors based on these distances (L. Zhang et al., 2017). Data points are considered anomalies if they are far away from their nearest neighbors.

The sliding window of SWkNN gets defined as:

$$W_t = \{x_{t-n+1}, x_{t-n+2}, \ldots, x_t\}$$

Where:

- *xt* is the data point at time *t.*
- *n* is the window size.

2.7.2. Robust Random Cut Forest (RRCF)

Robust Random Cut Forest (RRCF) is a collection of independent Robust Random Cut Trees (RRCTs). The process involves recursively partitioning the data based on random dimensions and cuts. At each node of the tree, a random dimension is selected from the available dimensions of the data, and a random cut point along that dimension is chosen to split the data into two subsets. When a new data point arrives in the data stream, the RRCF algorithm evaluates the anomaly score of this point by considering its impact on each tree in the ensemble. The anomaly score of the new point is calculated based on its average depth across

the ensemble of trees (Guha et al., 2016). Points that deviate significantly from the expected depth are then flagged as anomalies.

2.7.3. Half-Space Trees (HST)

Half-Space Trees (HST) organize the dataset into a tree structure of smaller sub-spaces. Each node in the tree represents a specific sub-space, known as the node's workspace. When a node has children, its workspace is split into halves with the same volume of data points along a hyperplane of a randomly chosen dimension. The data stream is then divided into two windows of a fixed length: the latest window and the reference window. The reference window is initialized with a set of historical data points and serves as the baseline for comparison with the incoming data. As new data points arrive in the data stream, the algorithm slides the window forward. The latest window is updated with the new data point, and the oldest data point in the window is removed to maintain a fixed window size.

The data points in the latest window are compared to the data points in the reference window with the help of so-called mass counters. When the tree is constructed, each node is equipped with a reference mass profile counter (RMPC) and a latest mass profile counter (LMPC). The RMPC tracks the flow of data points from the reference window. It stores the count of data points that have passed through the node in the previous time frame. The LMPC tracks the flow of data points from the latest window. It stores the count of data points that have passed through the node in the current time frame. As a new data point moves through the tree from the root to a leaf node, the counters at each node are updated (Tan et al., 2011). The counts stored in the reference and latest mass profile counters at each node are then used to calculate the anomaly score for each data point.

The anomaly score for HS-Trees gets calculated as follows:

$$\text{Score}(x, T) = \sum_{i=1}^{h} \frac{r_i + l_i}{r_i - l_i}$$

Where:

- *Score(x,T)* is the anomaly score for streaming point *x* in HS-Tree T.
- *h* is the maximum depth of the tree.
- *ri* is the mass of the node in the reference window at depth *i*.
- *li* is the mass of the node in the latest window at depth *i*.

2.8. Challenges of Streaming Data

Streaming data, as described by (Tan et al., 2011), refers to information that is continuously generated by various sources and processed in real-time. This approach to data processing presents several challenges:

One primary consideration in streaming data is velocity. The high speed at which data flows into the system requires real-time processing capabilities. This involves quick ingestion and analysis of the data, which poses challenges in terms of computational requirements and system latency.

Closely related to velocity is the challenge of volume. The potentially infinite amount of data generated continuously in a stream can be overwhelming for offline learning algorithms due to their memory requirements, as they typically need to load the entire dataset into the main memory to perform anomaly detection. Furthermore, handling large volumes of data requires a robust infrastructure and optimized data pipelines (Ahmad & Purdy, 2016).

As data streams continuously, another challenge emerges: evolving data distribution. The data distribution underlying streaming data can evolve over time, a phenomenon also known as concept drift. This can gradually diminish the efficacy of offline learning algorithms in accurately detecting anomalies as the data shifts, as the algorithms cannot adapt to these shifts due to their static design (Iglesias Vázquez et al., 2023).

3. RELATED WORK

This chapter presents a comprehensive review of the current state of research regarding blockchain oracles and anomaly detection, highlighting key findings and their relevance to this thesis. It explores the challenges faced by blockchain oracles in DeFi applications, including vulnerabilities and past incidents that have undermined trust in these systems.

The reliability of blockchain oracles is crucial for the effective functioning of DeFi applications. Oracles, which provide external data to smart contracts, are often considered a single point of failure due to the potential of unintended centralization when third-party data providers are introduced (Beniiche, 2020). As noted by (Kaleem & Shi, 2021), there is a growing demand for price data feeds, driven by the expanding role of DeFi as an alternative to centralized finance (CeFi) services. However, this increased reliance on oracles, often referred to as the "oracle problem", exposes DeFi protocols to several attack vectors. These vectors include front running, where early access to data is exploited to execute advantageous trades, and Sybil attacks, which involve creating fake nodes to manipulate data feeds. Technical malfunctions can lead to incorrect data transmission, intentional tampering with data during transmission can lead to manipulation through bad actors. Additionally, flash loans can be used to manipulate market prices within a single transaction block (Caldarelli & Ellul, 2021).

In the past, several severe oracle incidents have undermined the trust in DeFi protocols. One example is the the Compound liquidation from November 2020 that led to over $85 million in wrongful liquidations because of an inaccurate DAI/USDC price feed from CoinbasePRO (Caldarelli & Ellul, 2021). Similarly, in June 2019, the Synthetix incident involved a corrupted API that reported prices inflated by 1000x, resulting in nearly $37 million in wrongful profits due to price arbitrage (Gu et al., 2021).

To address these vulnerabilities, there has been a transition from centralized oracle solutions to a decentralized architecture, which aims to mitigate these attack vectors through various data validation mechanisms. (Zhao et al., 2022) introduce oracles deployed in real-life scenarios They further outline the mechanisms they employ, which mostly involves the construction of a median value of multiple data sources. They highlight the lack of machine learning techniques in the data validation process and argue that incorporating machine learning could enhance the trustworthiness of data feeds provided by oracles, but do not specify particular techniques.

Despite these advancements, (Liu et al., 2021) mention deviations between real-time market prices and oracle-reported prices for MakerDAO, Compound, Synthetix, and AmpleForth, as well as disparities between the oracles and their respective price sources. Additionally, (Ankit Gangwal et al., 2022) come to similar conclusions regarding the discrepancies between price sources and oracle output prices for the Chainlink oracle. These deviations reveal potential security risks in DeFi platforms.

Given these ongoing issues, employing traditional machine learning techniques, such as anomaly detection, can be effective in flagging these discrepancies. In traditional finance, anomaly detection has already been used to detect manipulations in stock market data, as demonstrated with k-NN by (Koosha Golmohammadi & Osmar R. Zaiane, 2015).

Expanding on the use of machine learning for financial data, (Fu et al., 2001) discuss the application of the Self-Organizing Maps (SOM) algorithm for pattern discovery in financial time series data in the Hang Seng Index of Hong Kong Stock Exchange. By using SOM, similar patterns in the time series can be grouped together. This pattern recognition abilities can also be used for anomaly detection, identifying deviations from expected data behavior.

(Huang et al., 2019) present a hybrid algorithm for forecasting financial time series data, utilizing different versions of the DBSCAN algorithm for stock market data from the Shanghai and Shenzhen Stock Exchanges. While their primary focus lays on price prediction, an approach based on DBSCAN can also be adapted for an anomaly detection scenario.

A further application of anomaly detection in financial networks is demonstrated by Pham and (Pham & Lee, 2016). While they focus on the DeFi domain rather than traditional finance, they also detect anomalies using unsupervised learning methods. Their approach employs K-Means clustering and Support Vector Machines on transaction data from the Bitcoin network.

Similarly, (Kaufman & Iaremenko, 2022) focus on detecting anomalies in Ethereum transaction data. Their study explores various anomaly detection algorithms, including Isolation Forest to identify potentially fraudulent activities in real-time. Expanding further on the use of Isolation Forest, (Feremans et al., 2020) address the challenge of identifying anomalies in univariate, and multivariate mixed-type data. They propose a method that uses Isolation Forest for anomaly detection on various real-world time-series datasets.

In addition to these studies, numerous other papers have mentioned these algorithms as suitable for anomaly detection applications without applying them directly. These papers are summarized in the following Table 3.1.

Table 3.1 - Traditional Algorithms and the Associated Literature

Technique	Literature
SVM	(Koosha Golmohammadi & Osmar R. Zaiane, 2015), (Nassif et al., 2021)
DBSCAN	(Braei & Wagner, 2020)
K-Means	(Koosha Golmohammadi & Osmar R. Zaiane, 2015), (Schmidl et al., 2022)
SOM	(Prasad et al., 2009)
k-NN	(Koosha Golmohammadi & Osmar R. Zaiane, 2015)
Isolation Forest	(Schmidl et al., 2022), (Nassif et al., 2021)

While these algorithms are frequently referenced in the literature and have been applied to time series datasets in various closely related fields of traditional and decentralized finance, the continuous data provision by blockchain oracles to smart contracts also exhibits attributes of streaming data. This aligns with (Tan et al., 2011) description of streaming data as information that is continuously generated and processed in real-time. Furthermore, (Ahmad & Purdy, 2016) describe that the potentially infinite volume of streaming data can overwhelm traditional machine learning algorithms due to their memory requirements.

In contrast streaming algorithms, process data that arrives in sequential order, without needing to access the entire dataset. According to (Lu et al., 2023), these algorithms are designed to update their output as new data becomes available. Further benefits of streaming algorithms they describe include lower memory requirements, as they do not need to store large datasets fully but only a constant amount of memory, as well as their adaptability to changes in underlying data patterns.

Regarding streaming data algorithms, (Iglesias Vázquez et al., 2023) highlight SWKNN as the most popular approach for outlier detection in streaming data. SWKNN's implementation of the k-NN algorithm within a sliding window framework is considered well-suited for handling streaming data and detecting anomalies in real-time.

(Guha et al., 2016) introduce Robust Random Cut Forest (RRCF) as an algorithm specifically engineered to address the challenges introduced by dynamic data streams, making it suitable for real-time anomaly detection tasks where the data distribution is changing over time.

Finally, (Tan et al., 2011) present the concept of Streaming Half-Space Trees (HS-Trees) as a fast one-class anomaly detector for evolving data streams, where HS-Trees capture the data points within specific subspaces of the data stream and use a mass measure to rank anomalies.

While research has explored anomaly detection in related fields such as traditional finance (Koosha Golmohammadi & Osmar R. Zaiane, 2015) and DeFi (Pham & Lee, 2016; Kaufman & Iaremenko, 2022), its application regarding blockchain oracles remains largely unexplored. The current body of research acknowledges the "oracle problem" and identifies oracles as potential single points of failure (Beniiche, 2020), but it generally overlooks the application of machine learning approaches, particularly anomaly detection, to enhance the robustness of oracle environments.

Some studies have conducted statistical deviation analyses of oracle price feeds by comparing them to their price sources (Ankit Gangwal et al., 2022; B. Liu et al., 2021), and theoretical discussions have explored the potential integration of machine learning approaches into the data validation process of oracles (Zhao et al., 2022). However, practical applications of these techniques in real-world blockchain oracle environments remain unexplored.

Furthermore, while streaming algorithms have shown promise in handling continuous data flows (Iglesias Vázquez et al., 2023; Guha et al., 2016; Tan et al., 2011), there is limited research on their application to blockchain oracle data. This gap is significant given the continuous nature of data provision by blockchain oracles to smart contracts, which aligns with the characteristics of streaming data as described by Tan et al. (2011).

This research gap presents a risk, as undetected anomalies in oracle data can lead to significant financial losses and potentially erode trust in DeFi applications. The severity of this risk is underscored by incidents such as the Compound liquidation of November 2020 and the Synthetix incident in June 2019 (Caldarelli & Ellul, 2021; Gu et al., 2021).

4. RESEARCH METHODOLOGY

This chapter details the methodology for evaluating the anomaly detection techniques on the Ethereum to U.S. Dollar price feed from the Chainlink blockchain oracle. It explains the data retrieval process, the parameters of the utilized anomaly detection algorithms and the experimental steps followed. Additionally,the performance metrics, as well as the software tools used for data processing and algorithm implementation are described.

4.1. EXPERIMENTAL DATA

The experimental data is represented by the historical broadcast prices of the Chainlink oracle that was fetched directly from the Ethereum blockchain. To connect to the Ethereum blockchain, the Web3 Python library alongside an Infura endpoint was used. The Web3 library enables interactions with Ethereum's blockchain. Infura provides a gateway to the Ethereum network by offering API access to an Ethereum node, thereby eliminating the need to run a personal node. The contract address of the aggregator contract was then used to establish a connection to the contract on the blockchain and fetch the historical price feed data.

Table 4.1 - Dataset Information (Preprocessed)

Dataset	Number of Features	Total Datapoints	Anomaly Datapoints	Anomaly Percentage
Chainlink Price Feed	8	17310	173	1.0

4.2. ALGORITHM PARAMETERS

This section discusses the parameters used for the respective machine learning algorithms. Table 4.2 provides an overview of the anomaly detection methods and their associated parameters, categorized into static and streaming algorithms.

Table 4.2 - Anomaly Detection Algorithms and their Parameters

Category	Method/Algorithm	Parameters
Static	SVM	• Kernel Type • Gamma • Nu
	DBSCAN	• Epsilon • Minimum Samples
	K-Means	• Number of Clusters • Initialization Method • Number of Runs • Maximum Iterations
	SOM	• Grid Size (x, y) • Learning Rate • Sigma • Iterations
	KNN	• Number of Neighbors • Algorithm
	Isolation Forest	• Number of Trees • Maximum Samples • Contamination • Maximum Features
Streaming	SWKNN	• Window Size • Number of Neighbors • Algorithm
	RRCF	• Number of Trees • Tree Size • Shingle Size
	Half Space Trees	• Number of Trees • Height • Window Size

SVM: In SVM, the Kernel Type defines the function used to map data into a higher-dimensional space, Gamma controls the influence of individual training examples, and Nu regulates the algorithm's sensitivity to outliers.

DBSCAN: This algorithm requires the specification of two key parameters: Epsilon, which defines the neighborhood radius, and Minimum Samples, which denotes the minimum number of points needed to form a dense region.

K-Means: K-Means clustering necessitates setting the Number of Clusters, which indicates how many clusters the data should be partitioned into, Initialization Method, which determines the starting positions of cluster centroids, Number of Runs for stability, and Maximum Iterations to limit the computational cost.

SOM: For SOM, the Grid Size (x, y) determines the dimensions of the map, Learning Rate controls how quickly the SOM adapts to the input data, Sigma defines the neighborhood function's initial radius, and Iterations determine the number of epochs for training.

k-NN: This method requires specifying the Number of Neighbors, which is the number of nearest neighbors considered for a point, and the Algorithm used to compute the nearest neighbors.

Isolation Forest: This method involves setting the Number of Trees, which determines the ensemble's size, Maximum Samples that controls how many samples each tree can use during training, Contamination to define the proportion of outliers, and Maximum Features to control the fraction of features used.

SWKNN: The primary parameters for SWKNN are Window Size, which defines the number of consecutive data points treated as input, Number of Neighbors (K), and the Algorithm.

RRCF: For RRCF, essential parameters include the Number of Trees, Tree Size, which limits the number of points a tree can hold, and Shingle Size, which determines the number of consecutive data points treated as input.

HST: HST involves specifying the Number of Trees, Tree Height, which limits the capacity of each tree, and Window Size to determine the scope of data points for analysis.

4.3. PREPROCESSING

The preprocessing steps undertaken in this study include resampling the data to a consistent hourly frequency, applying linear interpolation to handle missing values, and inserting artificial anomalies in order to create a dataset for evaluating the performance of various detection algorithms.

4.3.1. Resampling

The initial preprocessing step involved transforming the temporal granularity of the Ethereum to USD exchange rate data from irregular time intervals to a consistent hourly frequency. This transformation was accomplished through resampling. Resampling involves aggregating the data points within each one-hour interval into a single data point, which was chosen as the mean value of the ETH/USD Rate within that hour. Resampling to an hourly frequency was crucial because the timestamps in the Chainlink data were irregular. This irregularity can complicate anomaly detection, making it harder to learn patterns or changes over time. By standardizing timestamps to an hourly interval, the uniformity of the dataset was increased, aiming to enhance the potential results of the anomaly detection approaches.

Resampled data points are calculated using the formula:

$$R_t = \frac{1}{|D_t|} \sum_{d \in D_t} d$$

Where:

- $|Dt|$ represents the number of data points within the *t-th* hour.
- d represents an individual data points value in this case the ETH/USD Rate at a specific timestamp.
- $\sum_{d \in Dt} d$ is the summation of all exchange rates during the *t-th* hour.

4.3.2. Linear Interpolation

To address the problem of missing values, linear interpolation was implemented using adjacent data points for estimation. This technique assumes a linear progression between consecutive data points and fills missing values by creating a linear path between two consecutive available points.

Linear interpolation calculates the missing value as:

$$M_t = M_{t-1} + \frac{M_{t+1} - M_{t-1}}{2}$$

Where:

- $Mt-1$ is the ETH/USD Rate one hour before the missing value.
- $Mt+1$ represents the ETH/USD Rate one hour after the missing value.

4.3.3. Artificial Anomaly Insertion

Most datasets in the real world are not labeled, and obtaining accurate and representative labels, especially for the anomaly class, is a challenging and time-consuming undertaking (Prasad et al., 2009). The insertion of artificial anomalies into datasets is a common practice to avoid this limitation, as mentioned by multiple papers [(Koosha Golmohammadi & Osmar R. Zaiane, 2015);(Schmidl et al., 2022)]. Therefore, this thesis made use of the insertion of artificial anomalies.

The initial step involved determining the total number of anomalies to be introduced. This quantity was set at 1% of the dataset's overall size. This specific proportion was chosen to ensure that the inserted anomalies were substantial enough to test the anomaly detection techniques effectively without distorting the fundamental structure of the original dataset. After determining the total number of anomalies, indices from the dataset's datetime index were randomly chosen for the insertion of these anomalies. This method ensures that

anomalies are evenly distributed over the entire timeframe, preventing any clustering within specific periods.

For each selected index, a random spike ±10-30% in the ETH/USD Rate was generated. This randomness in the spikes aims to ensure that no algorithm can detect a pattern and therefore overfits to the data. The dataset was then updated with these new values, and the modified entries were marked in the 'Anomaly' column as 1.

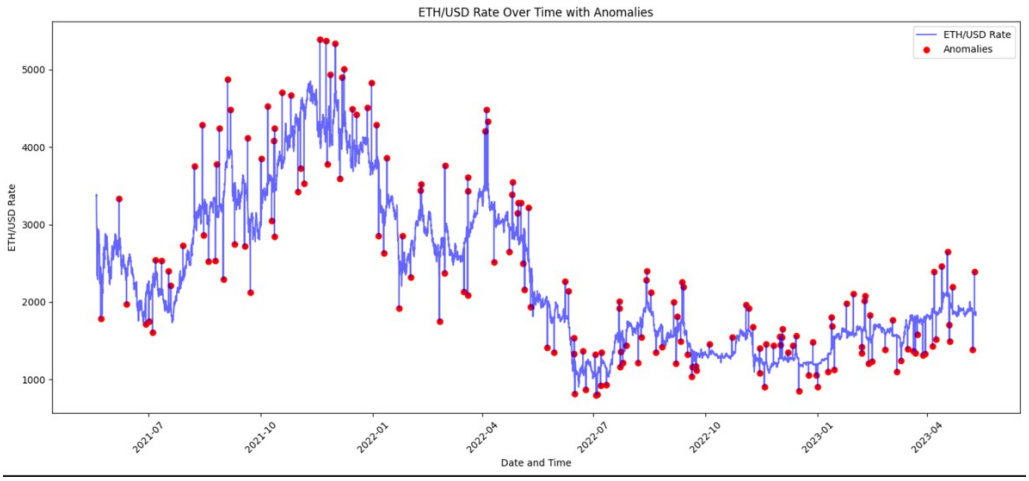

Figure 4.1 - Illustration of the ETH/USD Rate over time in blue, with the artificial anomalies marked in red.

4.4. Feature Engineering

This chapter discusses the feature engineering process. The features were derived from the ETH/USD Rate and datetime data and are based on common financial indicators mentioned in (Agrawal et al., 2021; Di, 2014; Manuel R. Vargas et al., 2018). The features aim to improve the ability of the algorithms to distinguish between normal and anomalous data points. Table 4.3 details the list of the engineered features.

Table 4.3 – List and Description of the Engineered Features

Feature	Description
7-Day Moving Average	Smoothens the price data over a 7-day period, highlighting trends.
7-Day Moving Standard Deviation	Measures the volatility of the price over the past 7 days. Higher values indicate higher volatility.
Percentage Change	Shows the rate of change in price from the previous day to the current day, indicating the direction and magnitude of price movements.
Relative Strength Index (RSI)	Oscillates between 0 and 100 and indicates overbought or oversold conditions.
Bollinger Bands	Consists of a middle band and two outer bands (2 standard deviations away from the middle band). Measures market volatility and potential buy/sell signals.
Moving Average Convergence Divergence (MACD)	A trend-following momentum indicator showing the relationship between two moving averages. Indicates buy or sell signals.
Exponential Moving Average (EMA)	Similar to a simple moving average but gives more weight to recent prices, making it more responsive to new information.
Average True Range (ATR)	Measures market volatility by decomposing the entire range of an asset price for that period. Higher ATR values indicate higher volatility.

For all the mentioned features, the formulas can be found in Appendix A.

4.5. Feature Selection

The feature selection approaches aimed to identify the features that contributed most to successful anomaly detection. Two primary methods were employed for feature selection: correlation analysis and an embedded method utilizing Decision Tree classifiers for feature importance evaluation.

By determining the correlation coefficients, the features were ranked by their correlation with the anomaly status, focusing on those with the strongest correlations, whether positive or negative.

Furthermore, a Decision Tree Classifier was trained to calculate the importance of each feature. This importance was based on how much each feature contributed to the decision-making process of the tree. It considers how often a feature was used to split data and how much it improved the decision tree's predictions. Features that are more frequently involved in key decisions or significantly improved the model's performance are marked as more important.

4.6. Experimental Procedure

The experimental procedure was designed to evaluate the performance of the different anomaly detection algorithms. Initially, the dataset was split into a training set and a test set, with a standard division of 80% for training and 20% for testing. To account for the temporal nature of the data, Time Series Cross-Validation (TSCV) was implemented instead of traditional k-fold cross-validation. TSCV was applied to the training set, creating 5 train- validation splits that respect the time-based structure of the data.

For each TSCV fold, Standard Scaler was applied to the training data, ensuring each feature had a mean of zero and a standard deviation of one. The scaling parameters from the training data were then applied to the corresponding validation set and to the test set, to ensure that no single feature dominated the model due to its scale.

In the hyperparameter tuning phase, different strategies were employed for supervised and unsupervised learning methods. For supervised learning methods, GridSearchCV was used with the TSCV folds, iterating through a predefined grid of parameters to optimize for the highest F1 score across all folds. For unsupervised methods, a manual iteration over a range of possible parameter values was performed, again focusing on maximizing the F1 score across the TSCV folds.

After determining the optimal parameters, the model training phase differed based on the type of algorithm. Supervised models were retrained on the entire training dataset using the optimal parameters. Unsupervised and statistical methods bypassed this retraining phase, as they don't rely on labeled data for learning. All models were then evaluated on the held-out test set, which represents future, unseen data. This step provides an unbiased estimate of each model's performance on new data. Finally, the performance of all models on the test set was compared using the F1 score as the primary metric, with Recall as a secondary consideration.

This procedure ensures that the evaluation respects the time-dependent nature of the data, prevents data leakage, and provides a robust estimate of each model's performance on future, unseen data.

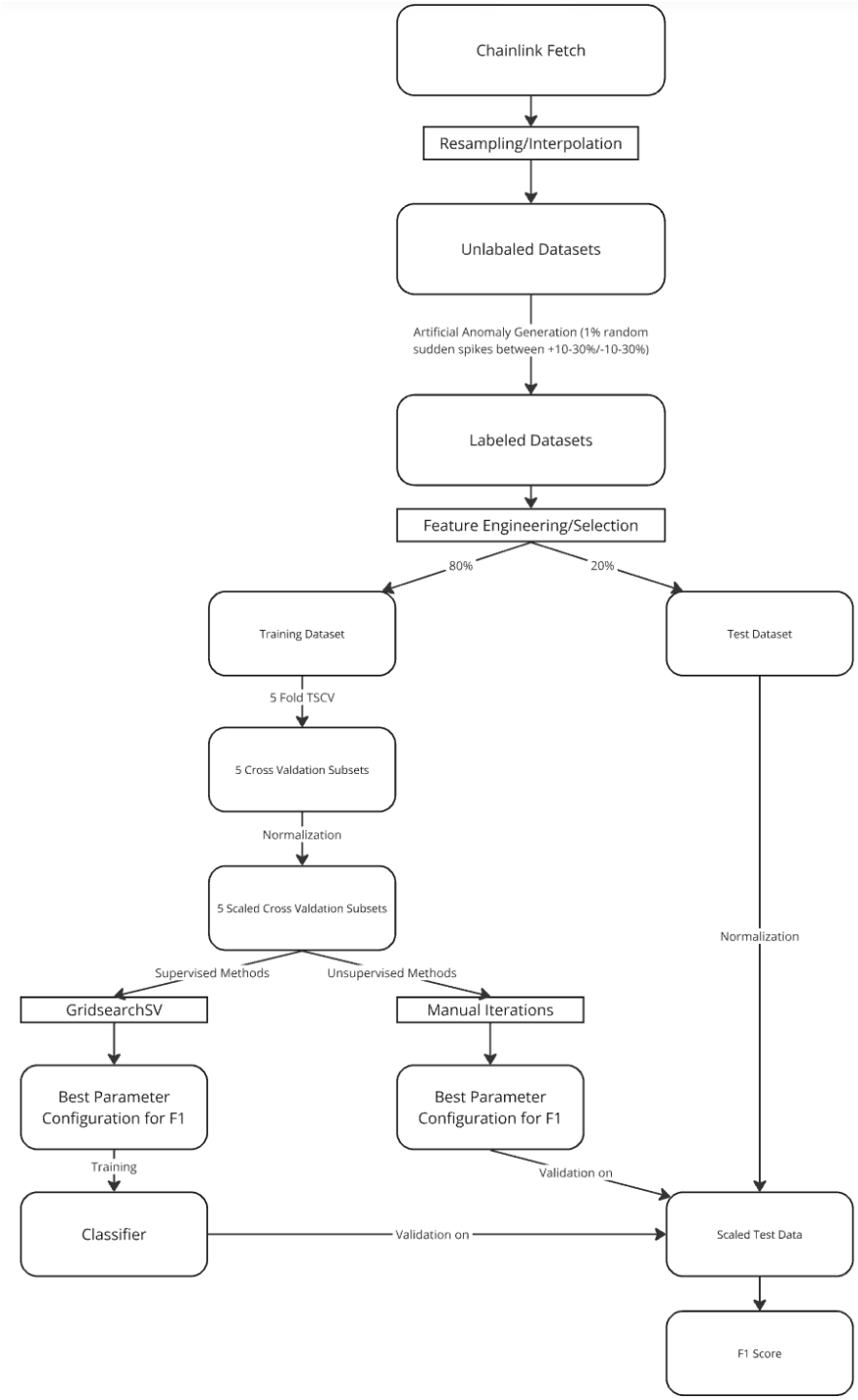

Figure 4.2 - Experimental Procedure

4.7. Performance Metrics

In anomaly detection, selecting appropriate performance evaluation metrics is crucial for assessing the effectiveness of different algorithms and their practical applicability in real-world scenarios. This is particularly important in the context of blockchain oracle price feeds, where inaccurate anomaly detection can have significant financial implications. Multiple

performance evaluation metrics can be used, with Precision, Recall, and F1 score being among the most common ones.

Precision measures the proportion of correctly identified anomalies among all instances that are flagged as anomalies. It is calculated as the ratio of true positives to the sum of true positives and false positives.

Formula:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Where:

- TP (True Positives) is the number of correct predictions that an instance is an anomaly.
- FP (False Positives) is the number of incorrect predictions that a non-anomalous instance is an anomaly.

Recall represents the proportion of true anomalies identified from all actual anomalies. It is calculated as the ratio of true positives to the sum of true positives and false negatives.

Formula:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Where:

- FN (False Negatives) is the number of anomalies that were not detected.

F1 score combines Precision and Recall into a unified metric, creating a balanced evaluation of algorithm performance. It ensures a compromise between identifying as many anomalies as possible (Recall) and ensuring that the detections are indeed anomalies (Precision).

The F1 score is calculated as the harmonic mean of Precision and Recall:

$$\text{F1-Score} = 2 \times \left(\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \right)$$

This thesis primarily uses the F1 score, with Recall as a secondary metric. This choice reflects the need to balance different considerations in the context of this kind of applications. A High Recall is crucial to minimize missed anomalies, which could lead to incorrect smart contract executions or financial transactions. However, focusing solely on Recall can result in low Precision, potentially overwhelming the system with false alarms. The F1 score provides a balanced measure, ensuring critical anomalies are caught while minimizing disruptive false alarms. This approach aligns with the broader literature on anomaly detection, where the F1 score is widely recognized as a robust metric for balanced performance evaluation. It is

particularly suitable for the sensitive environment of blockchain oracles, where both missed anomalies and false alarms can have significant consequences.

By prioritizing the F1 score, this study aims to evaluate anomaly detection algorithms in a way that best reflects the practical utility of blockchain oracles, balancing the need for accurate anomaly detection with the operational requirements of the underlying DeFi systems. This balanced approach ensures that the evaluation process considers both the ability to detect genuine anomalies and the necessity to maintain system efficiency by minimizing false alarms, providing a more comprehensive assessment of each algorithm's performance in the context of blockchain oracle price feed anomaly detection.

4.8. SOFTWARE IMPLEMENTATION

The research employed several Python libraries for data analysis and algorithm implementation. Pandas and NumPy were used for data manipulation and preprocessing, while SciPy provided statistical functions. Matplotlib and Seaborn were utilized for data visualization, creating both basic plots and more complex graphical representations. Scikit-learn served as the primary library for machine learning tasks, including feature selection, model training, anomaly detection algorithm implementation, and model evaluation. For specific algorithms not available in Scikit-learn, additional libraries were incorporated. The minisom library provided the implementation for Self-Organizing Maps (SOM).

To address anomaly detection with streaming algorithms, the river library was used for implementing Half-Space Trees, and the RRCF library for the Random Cut Forest algorithm.

This combination of software tools facilitated the data analysis, algorithm implementation, and performance evaluation processes throughout the study.

5. RESULTS

To address the research question regarding the impact of extracted features based on common financial indicators, feature selection methods as described in Chapter 4.4, were employed. After applying these methods, the highest-ranking features were identified, as visualized in figure 5.1. Based on these scores, three distinct feature sets were constructed:

1. RSI and ATR, representing the two best-performing indicators
2. RSI, ATR, and ETH/USD Rate, representing the three top features
3. RSI, ATR, ETH/USD Rate, and EMA, encompassing the four highest-ranking features

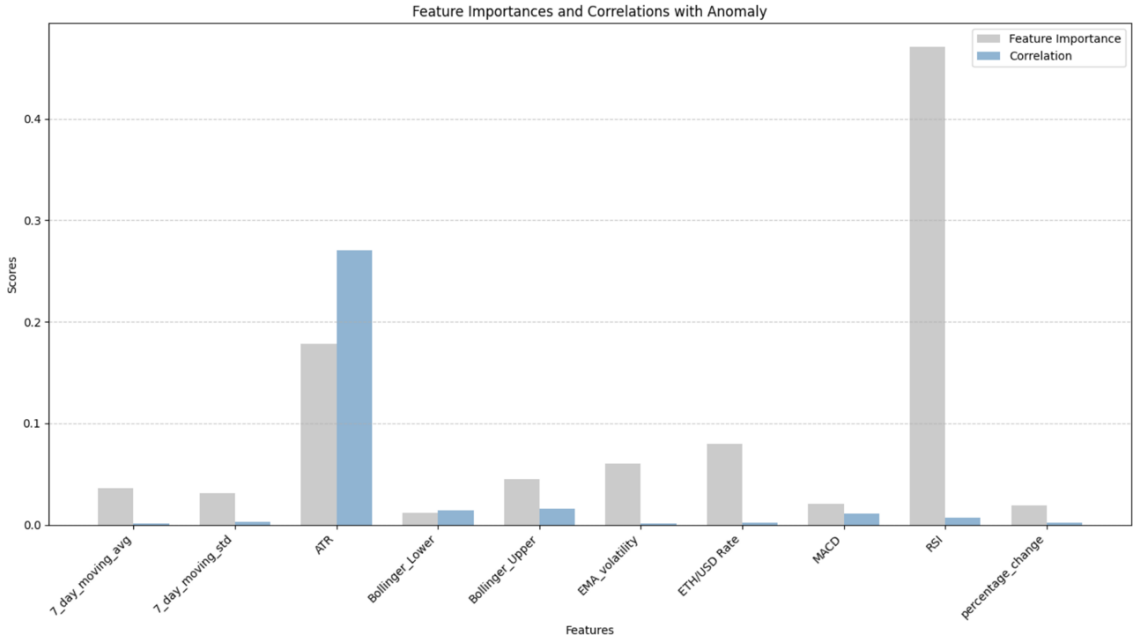

Figure 5.1 - Feature importance and correlation analysis. The grey bars show the feature importance from the decision tree modeling, and the blue bars indicate the correlation coefficients with the anomaly variable.

Based on the high scores obtained from the feature selection methods, the RSI and ATR set was chosen for further analysis without additional evaluation. This selection was made directly from the feature importance rankings, reflecting the strong performance of these indicators in the initial assessment. To gain additional insights into how different feature combinations might perform in distinguishing anomalies, Uniform Manifold Approximation and Projection (UMAP) was employed to visualize two of the feature sets into a two-dimensional space. The UMAP visualization allows for a qualitative assessment of how well the chosen features separate anomalous data from normal data in a lower-dimensional space.

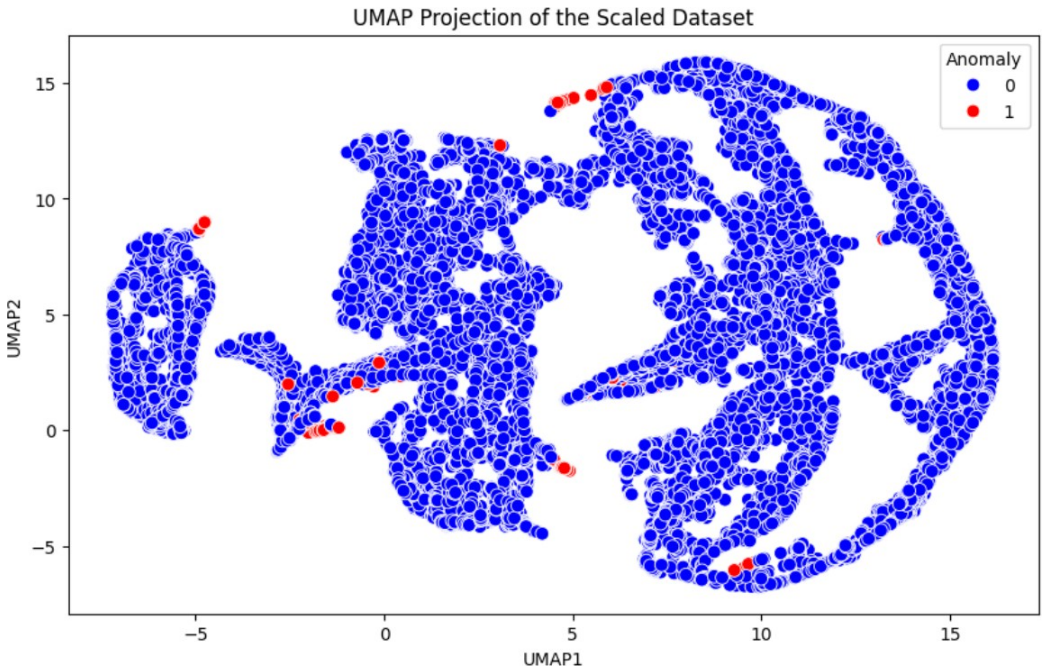

Figure 5.2 - Umap Projection of the RSI, ATR, ETH/USD Rate feature set, where blue points represent normal data points and red points indicate anomalies.

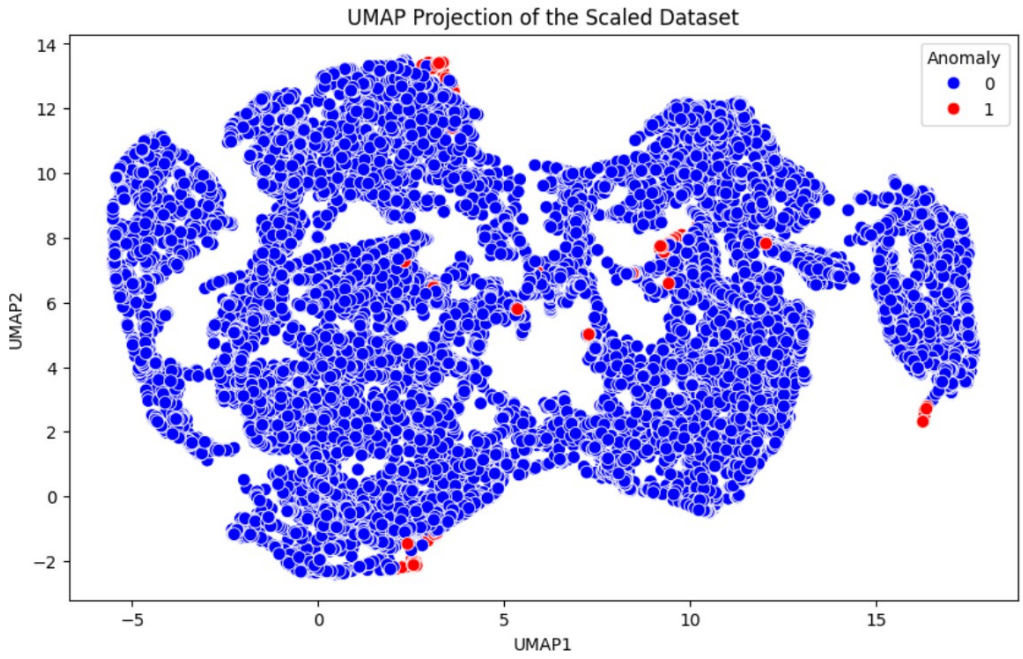

Figure 5.3 - Umap Projection of the RSI, ATR, ETH/USD Rate, EMA feature set, where blue points represent normal data points and red points indicate anomalies.

Comparing the two UMAP projections, it appeared that the RSI, ATR, and ETH/USD Rate feature set (Figure 5.2) might be more effective in isolating anomalies. In this projection, the anomalies seemed to be more distinctly clustered and separated from the normal data points,

particularly along the edges of the projection. This suggests that this three-feature set could potentially provide better discrimination between normal and anomalous data points.

In contrast, the four-feature set including EMA (Figure 5.3) shows a slightly more dispersed distribution of anomalies throughout the projection. While some clustering of anomalies is still visible, the separation between normal and anomalous points appeared less pronounced compared to Figure 5.2.

To further enhance this analysis, DBSCAN clustering was employed on top of the UMAP projection of the RSI, ATR, and ETH/USD Rate feature set. Figure 5.4 shows the results of this combined approach.

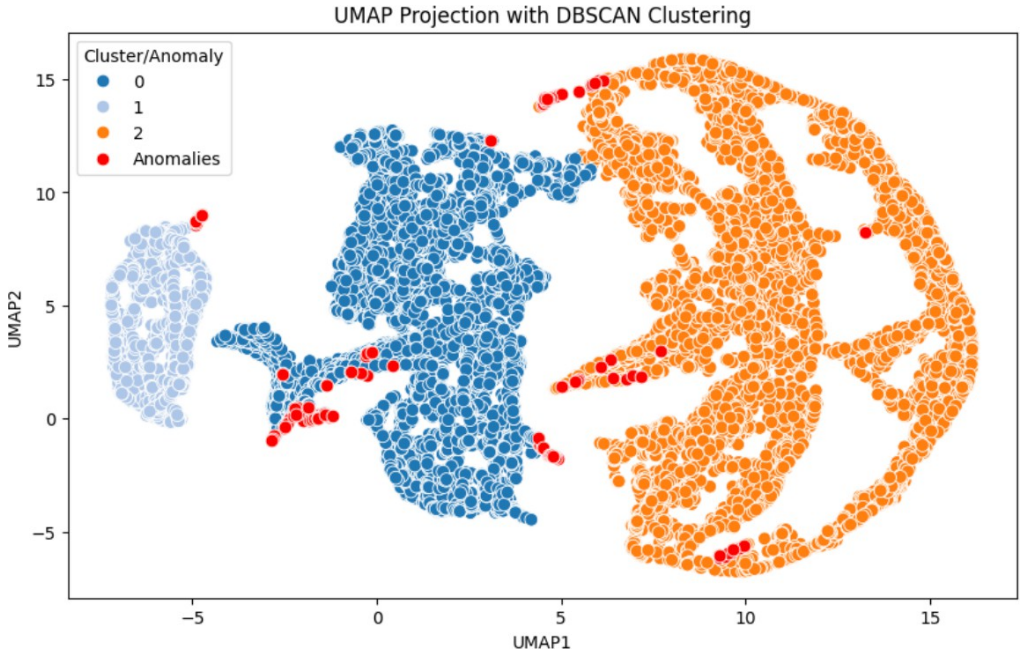

Figure 5.4 - UMAP projection with DBSCAN clustering, where three clusters were identified colored light blue (0), dark blue (1), and orange (2) and the red points representing anomalies.

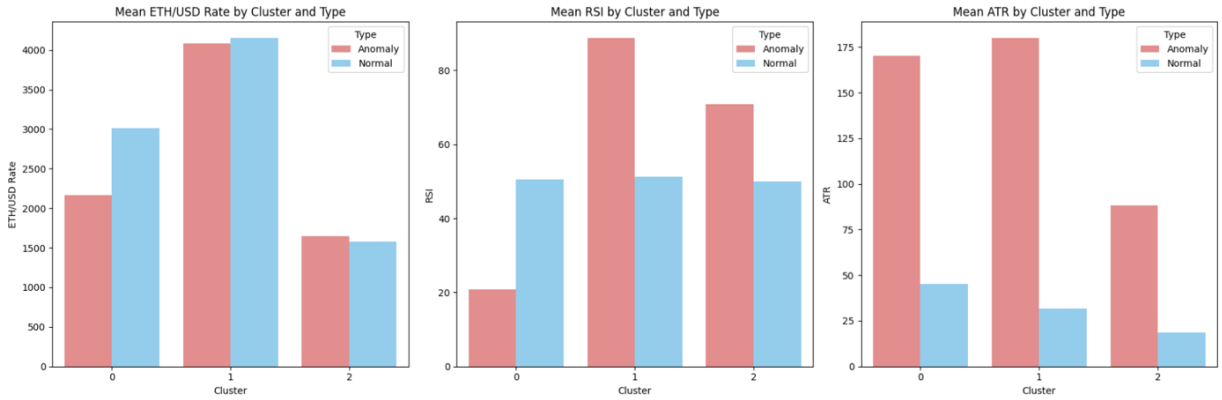

Figure 5.5 - Mean ETH/USD Rate, RSI, and ATR by cluster and anomaly type, with blue representing normal data points and red anomalous data points.

In the next step the mean values of the different features per cluster were examined, as visualized in figure 5.5, where the ETH/USD Rate proved to be the least effective feature overall. While it showed a clear distinction in Cluster 0, with anomalies having lower rates than normal data points, Clusters 1 and 2 displayed minimal distinctions. RSI demonstrated improved distinctions than the ETH/USD Rate, with significant deviations between normal and anomalous instances in Clusters 1 and 2, where anomalies had significantly higher RSI values than normal data points. Interestingly, Cluster 0 exhibited an opposite trend with lower RSI for anomalies. ATR emerged as the most effective feature for anomaly detection. It consistently showed higher values for anomalies across all clusters.

Following the comprehensive feature analysis, an initial F1 score comparison was conducted for the two most promising feature sets: RSI and ATR, and RSI, ATR, and ETH/USD Rate. Figure 5.6 presents the F1 scores for these two feature sets across different anomaly detection algorithms. The performance varied across different methods, with some algorithms showing improved F1 scores with the inclusion of the ETH/USD Rate, while others perform better with just RSI and ATR. The best-performing feature combinations from each method were then utilized for the final evaluation. Table 5.1 presents a comparison of the best combinations and the originally fetched univariate dataset. This final evaluation includes both F1 scores and recall.

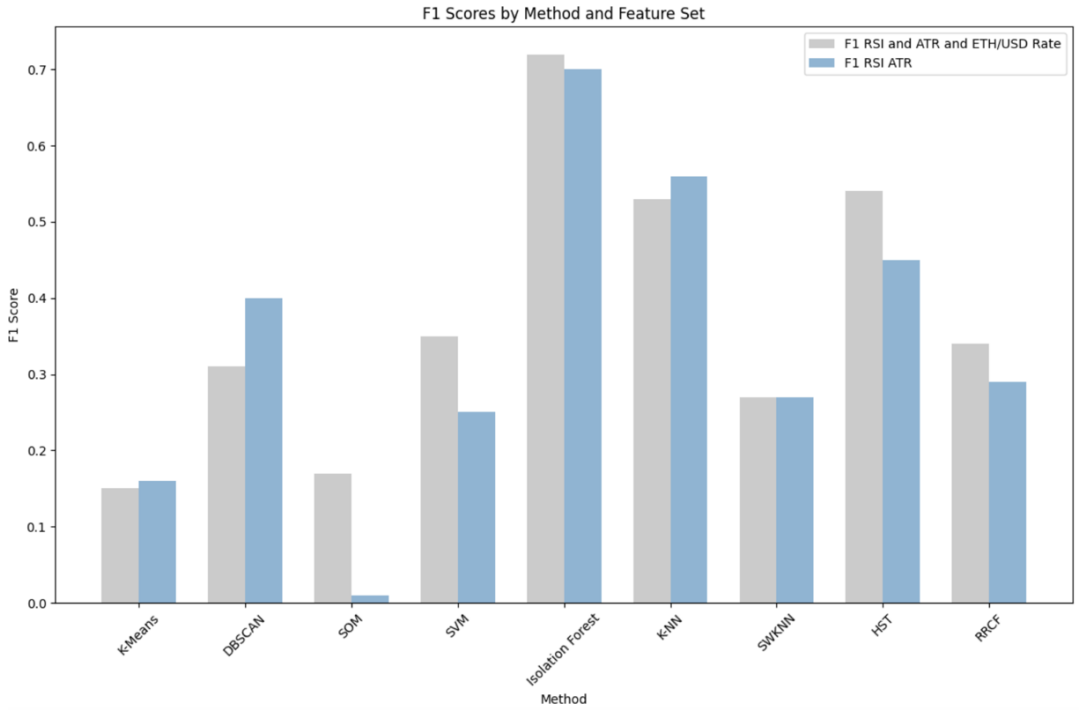

Figure 5.6 - Initial F1 score comparison of algorithms and feature sets, with grey representing the F1 score of the RSI, ATR and ETH/USD Rate feature set and blue representing the RSI and ATR feature set.

Table 5.1 - Performance metrics of the anomaly detection techniques on the original dataset (univariate) compared to the multivariate datasets, with the best performing combinations in bold.

Type	Method	F1 Score		Recall	
		Univariate	Multivariate	Univariate	Multivariate
Static	SVM	0.11	0.35	0.05	0.68
	DBSCAN	0.11	0.40	**0.17**	0.26
	K-Means	0.01	0.15	0.11	0.29
	SOM	0.02	0.17	0.07	0.34
	K-nn	0.08	0.56	0.11	**0.94**
	Isolation Forest	**0.11**	**0.72**	0.06	0.74
Streaming	SWKNN	0.25	0.27	**1.0**	**1.0**
	RRCF	**0.34**	0.29	**1.0**	0.86
	Half Space Trees	**0.34**	**0.54**	**1.0**	0.54

Among the static methods, Isolation Forest and k-NN showed the highest overall performance. Isolation Forest achieved an F1 score of 0.72 on the multivariate dataset, the highest among all methods, including streaming methods. It also achieved a recall of 0.74, indicating a strong balance between precision and recall by identifying most true positives while maintaining a relatively low rate of false alarms. In contrast, k-NN achieved an F1 score of 0.56 on the multivariate dataset, the second highest among static algorithms, with a recall of 0.94—the highest among all static methods—highlighting its high sensitivity in detecting anomalies. However, this high sensitivity comes with a trade-off in precision, potentially flagging more false positives compared to Isolation Forest.

A noticeable trend among the static methods is the significantly higher performance of scores achieved on the multivariate dataset compared to the univariate dataset. For example, Isolation Forest's F1 score improved significantly from 0.11 (univariate) to 0.72 (multivariate), while its recall increased from 0.06 to 0.74. Similarly, k-NN improved its F1 score from 0.08 (univariate) to 0.56 (multivariate), and its recall from 0.11 to 0.94.

In the streaming methods category, Half Space Trees achieved the best scores F1 score with 0.54 and a recall of 0.54 on the multivariate dataset. SWKNN demonstrated an F1 score of 0.27 with a perfect recall of 1.0. Interestingly, RRCF scored second best in this category but was the only algorithm among both static and streaming approaches that achieved a better F1 score on univariate data. It achieved an F1 score of 0.34 and a recall of 1.0 on the univariate dataset, compared to an F1 score of 0.29 and a recall of 0.86 on the multivariate dataset.

Generally, the Streaming algorithms showed a similar pattern of higher F1 scores on multivariate data, with the already explained exception of RRCF. However, the difference is not as pronounced as in the static methods. They generally maintained a higher baseline of F1 scores on univariate data, compared to the static algorithms. For example, Half Space Trees'

F1 score improved from 0.34 (univariate) to 0.54 (multivariate). Similarly, SWKNN's F1 score increased from 0.25 (univariate) to 0.27 (multivariate).

In terms of recall, two streaming algorithms saw a decrease when moving from univariate to multivariate data: Half Space Trees decreased from 1.0 to 0.54, and RRCF decreased from 1.0 to 0.86. SWKNN's recall remained the same, maintaining a perfect recall of 1.0 for both univariate and multivariate data. It's worth noting that all streaming algorithms exhibited perfect recall (1.0) when applied to univariate data, indicating their strong capability to detect anomalies in less complex datasets.

6. CONCLUSION

This study investigated the effectiveness of static and streaming machine learning techniques in detecting anomalies within the Ethereum-US Dollar (ETH/USD) exchange rate from the Chainlink blockchain oracle. Additionally, it examined the impact of extracting new features in the form of financial indicators. The findings reveal several key insights for anomaly detection applications in this context.

The results demonstrated significant variations in performance across the tested methods, both in terms of F1 score and recall. Isolation Forest emerged as a strong performer, achieving an F1 score of 0.72 and a recall of 0.74. This indicates Isolation Forest's relative effectiveness in identifying anomalies while maintaining a good balance between precision and recall, showing its ability to detect true anomalies while relatively minimizing false alarms.

In contrast, the streaming algorithm Half Space Trees achieved an F1 score of 0.54, which represents moderate performance. The comparison between static and streaming methods revealed that static methods like Isolation Forest and k-NN showed significant performance improvements with multivariate data compared to univariate data. This highlights their enhanced capabilities when using richer feature sets. The use of multivariate data allows these models to capture a broader range of relationships within the data, providing a more comprehensive view of the underlying patterns.

On the other hand, the streaming algorithms performed notably better on univariate data. This can be attributed to their design, which often prioritizes real-time processing and efficiency over depth of analysis. The complexity of multivariate data can potentially introduce noise and complicate the model's ability to detect anomalies. This was particularly notable for RRCF, where both the F1 score and recall decreased when applied on multivariate data. For Half Space Trees, the transition to higher-dimensional data resulted in an improved F1 score, but at the cost of decreased recall. These observations underscore the importance of balancing multidimensional feature sets with model simplicity for effective implementation of streaming data algorithms.

Despite these challenges, the high recall scores of the streaming algorithms on the univariate data are an interesting discovery. This indicates their potential for ensuring that no true anomalies are missed, which is particularly important in financial applications, where unidentified anomalies could cause immense damage to customers. Regardless of these promising results it has to be noted, that models with these scores are not ready for an application in real-life scenarios yet. Future work needs to focus on improving the overall reliability to increase their suitability for practical use.

While the features in this thesis were all extracted solely from the exchange rate of Ethereum to US Dollar, future research could explore additional features that may be able to provide more context. Potential features could include the transaction volume, which can indicate market activity levels. Other valuable features might be on-chain metrics like the number of active addresses and the transaction fees. Incorporating a broader range of features could

lead to more robust models. However, as previously mentioned, it remains crucial to balance increasing the dimensionality of the data with maintaining model efficiency.

Future work could also address the computational efficiency of different methods. This consideration is important for real-time data analysis, where rapid processing is essential for a useful application. A comparative study of computational speeds across different algorithms could provide further insights into their practicality.

Lastly, another area of promising research could lie in the application of window sizing techniques to static algorithms, particularly in regard to the top performers like Isolation Forest. This approach could potentially enhance the performance of static methods when processing streaming data.

In conclusion, this study has shed a light on the potential use of machine learning techniques in identifying anomalies in blockchain oracle data price feeds. The comparative analysis of static and streaming methods, along with the exploration of feature engineering, has revealed both interesting directions as well as challenges for the application of anomaly detection into blockchain oracles, which, in the past, has generally received limited attention compared to other areas of finance and blockchain security.

BIBLIOGRAPHICAL REFERENCES

Agrawal, M., Shukla, P. K., Nair, R., Nayyar, A., & Masud, M. (2021). Stock prediction based on technical indicators using deep learning model. *Computers, Materials and Continua*, *70*(1), 287–304. https://doi.org/10.32604/cmc.2022.014637

Ahmad, S., & Purdy, S. (2016). *Real-Time Anomaly Detection for Streaming Analytics*. http://arxiv.org/abs/1607.02480

Ahmed, M., Seraj, R., & Islam, S. M. S. (2020). The k-means algorithm: A comprehensive survey and performance evaluation. In *Electronics (Switzerland)* (Vol. 9, Issue 8, pp. 1–12). MDPI AG. https://doi.org/10.3390/electronics9081295

Al-Breiki, H., Rehman, M. H. U., Salah, K., & Svetinovic, D. (2020). Trustworthy Blockchain Oracles: Review, Comparison, and Open Research Challenges. *IEEE Access*, *8*, 85675–85685. https://doi.org/10.1109/ACCESS.2020.2992698

Ankit Gangwal, Rahul Valluri, & Mauro Conti. (2022). *Analyzing Price Deviations in DeFi Oracles*. Cryptology and Network Security 329-338 https://link.springer.com/chapter/10.1007/978-3-031-20974-1_17

Beniiche, A. (2020). *A Study of Blockchain Oracles*. http://arxiv.org/abs/2004.07140

Braei, M., & Wagner, S. (2020). *Anomaly Detection in Univariate Time-series: A Survey on the State-of-the-Art*. http://arxiv.org/abs/2004.00433

Breidenbach, L., Cachin, C., Chan, B., Coventry, A., Ellis, S., Juels, A., Koushanfar, F., Miller, A., Magauran, B., Moroz, D., Nazarov, S., Topliceanu, A., Tramèr, F., & Zhang, F. (2021). *Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks*. https://research.chain.link/whitepaper-v2.pdf

Caldarelli, G., & Ellul, J. (2021). The blockchain oracle problem in decentralized finance—A multivocal approach. In *Applied Sciences (Switzerland)* (Vol. 11, Issue 16). MDPI AG. https://doi.org/10.3390/app11167572

Chainlink 2.0. (n.d.). Retrieved February 29, 2024, from https://docs.chain.link/architecture-overview/off-chain-reporting

Chainlink Data Streams. (n.d.). Retrieved February 29, 2024, from https://docs.chain.link/data-streams

Chainlink Data Streams - Execution Flow. (n.d.). Retrieved February 29, 2024, from https://docs.chain.link/data-streams

Chainlink Data Streams - Trigger. (n.d.). Retrieved February 29, 2024, from https://docs.chain.link/chainlink-automation/concepts/automation-concepts#upkeeps-and-triggers

Choi, K., Yi, J., Park, C., & Yoon, S. (2021). Deep Learning for Anomaly Detection in Time-Series Data: Review, Analysis, and Guidelines. In *IEEE Access* (Vol. 9, pp. 120043–120065). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2021.3107975

DefiLlama. (n.d.). Retrieved January 29, 2024, from https://defillama.com/

DefiLlama/Oracles. (n.d.). Retrieved January 29, 2024, from https://defillama.com/oracles

Di, X. (2014). *Stock Trend Prediction with Technical Indicators using SVM*. https://cs229.stanford.edu/proj2014/Xinjie%20Di,%20Stock%20Trend%20Prediction%20with%20Technical%20Indicators%20using%20SVM.pdf

Feremans, L., Vercruyssen, V., Cule, B., Meert, W., & Goethals, B. (2020). *Pattern-Based Anomaly Detection in Mixed-Type Time Series*. Machine Learning and Knowledge Discovery in Databases 240-256. https://link.springer.com/chapter/10.1007/978-3-030-46150-8_15

Fu, T.-C., Chung, F.-L., Ng, V., & Luk, R. (2001). *Pattern Discovery from Stock Time Series Using Self-Organizing Maps. https://www.researchgate.net/publication/228771755_Pattern_discovery_from_stock_time_series_using_self-organizing_maps*

Gu, W. C., Raghuvanshi, A., & Boneh, D. (2021). *Empirical Measurements on Pricing Oracles and Decentralized Governance for Stablecoins*. https://ssrn.com/abstract=3611231

Guha, S., Mishra, N., Roy, G., & Schrijvers, O. (2016). *Robust Random Cut Forest Based Anomaly Detection on Streams*. ICML'16: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 Pages 2712 – 2721. https://dl.acm.org/doi/10.5555/3045390.3045676

Huang, M., Bao, Q., Zhang, Y., & Feng, W. (2019). A hybrid algorithm for forecasting financial time series data based on DBSCAN and SVR. *Information (Switzerland)*, *10*(3). https://doi.org/10.3390/info10030103

Iglesias Vázquez, F., Hartl, A., Zseby, T., & Zimek, A. (2023). Anomaly detection in streaming data: A comparison and evaluation study. *Expert Systems with Applications*, *233*. https://doi.org/10.1016/j.eswa.2023.120994

Kaleem, M., & Shi, W. (2021). *Demystifying Pythia: A Survey of ChainLink Oracles Usage on Ethereum*. Financial Cryptography and Data Security 115-123. https://link.springer.com/chapter/10.1007/978-3-662-63958-0_10

Kaufman, E., & Iaremenko, A. (2022). *Anomaly Detection for Fraud in Cryptocurrency Time Series*. http://arxiv.org/abs/2207.11466

Khan, K., Rehman, S. U., Aziz, K., Fong, S., Sarasvady, S., & Vishwa, A. (2014). DBSCAN: Past, present and future. 5th International Conference on the Applications of Digital Information and Web Technologies, ICADIWT 2014, 232–238. https://doi.org/10.1109/ICADIWT.2014.6814687

Khan, S. N., Loukil, F., Ghedira-Guegan, C., Benkhelifa, E., & Bani-Hani, A. (2021). Blockchain smart contracts: Applications, challenges, and future trends. *Peer-to-Peer Networking and Applications*, *14*(5), 2901–2925. https://doi.org/10.1007/s12083-021-01127-0

Koosha Golmohammadi, & Osmar R. Zaiane. (2015). *Time Series Contextual Anomaly Detection for Detecting Market Manipulation in Stock Market*. https://ieeexplore.ieee.org/document/7344856

Liu, B., Szalachowski, P., & Zhou, J. (2021). A first look into DeFI oracles. *Proceedings - 3rd IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS 2021*, 39–48. https://doi.org/10.1109/DAPPS52256.2021.00010

Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation forest. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 413–422. https://doi.org/10.1109/ICDM.2008.17

Lu, T., Wang, L., & Zhao, X. (2023). Review of Anomaly Detection Algorithms for Data Streams. In *Applied Sciences (Switzerland)* (Vol. 13, Issue 10). MDPI. https://doi.org/10.3390/app13106353

Manuel R. Vargas, Carlos E. M. dos Anjos, Gustavo L. G. Bichara, & Alexandre G. Evsukoff. (2018). *Deep Learning for Stock Market Prediction Using Financial Indicators and Financial News Articles.* https://ieeexplore.ieee.org/document/8489208

Monrat, A. A., Schelén, O., & Andersson, K. (2019). A survey of blockchain from the perspectives of applications, challenges, and opportunities. In *IEEE Access* (Vol. 7, pp. 117134–117151). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2019.2936094

Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf

Nassif, A. B., Talib, M. A., Nasir, Q., & Dakalbab, F. M. (2021). Machine Learning for Anomaly Detection: A Systematic Review. In *IEEE Access* (Vol. 9, pp. 78658–78700). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2021.3083060

Pham, T., & Lee, S. (2016). *Anomaly Detection in Bitcoin Network Using Unsupervised Learning Methods*. http://arxiv.org/abs/1611.03941

Prasad, N. R., Almanza-Garcia, S., & Lu, T. T. (2009). Anomaly Detection: A Survey. *Computers, Materials and Continua*, *14*(1), 1–22. https://doi.org/10.1145/1541880.1541882

Pyth Data Association. (2023). *Pyth Network: A First-Party Financial Oracle*.

https://pythdataassociation.com/whitepaper.pdf

Schmidl, S., Wenig, P., & Papenbrock, T. (2022). Anomaly Detection in Time Series: A Comprehensive Evaluation. *Proceedings of the VLDB Endowment*, *15*(9), 1779–1797. https://doi.org/10.14778/3538598.3538602

Schölkopf, B., Platt, J. C., Solla, S., Leen, T., Williamson, R., Smola, A., Shawe-Taylor, J., Platt, J., & Holloway, R. (2000). *Support Vector Method for Novelty Detection*. MIT Press. https://www.researchgate.net/publication/221619107

Tan, S. C., Ting, K. M., & Liu, T. F. (2011). Fast Anomaly Detection for Streaming Data.

IJCAI'11: Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Two Pages 1511 – 15.
https://www.ijcai.org/Proceedings/11/Papers/254.pdf

Van Hulle, M. M. (2012). *Self-Organizing Maps*. Handbook of Natural Computing 585-

622. https://link.springer.com/referenceworkentry/10.1007/978-3-540-92910-9_19

Werner, S., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., & Knottenbelt, W. (2022). *SoK: Decentralized Finance (DeFi)*. AFT '22: Proceedings of the 4th ACM Conference on Advances in Financial Technologies 30 – 46. https://doi.org/10.1145/3558535.3559780

Wong MA, & Tom Lane. (1982). A Kth Nearest Neighbour Clustering Procedure. Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface. https://link.springer.com/book/10.1007/978-1-4613-9464-8

Zhang, C., Wu, C., & Wang, X. (2020). Overview of blockchain consensus mechanism. *ACM International Conference Proceeding Series*, 7–12. https://doi.org/10.1145/3404512.3404522

Zhang, L., Lin, J., & Karim, R. (2017). Sliding Window-Based Fault Detection From High- Dimensional Data Streams. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *47*(2), 289–303. https://doi.org/10.1109/TSMC.2016.2585566

Zhao, Y., Kang, X., Li, T., Chu, C. K., & Wang, H. (2022). Toward Trustworthy DeFi Oracles: Past, Present, and Future. IEEE Access, 10, 60914–60928. https://doi.org/10.1109/ACCESS.2022.3179374

APPENDIX A

1) 7-Day Moving Average: Calculated as the mean of the ETH/USD Rate over a rolling window of 7 days, multiplied by 24 hours to achieve hourly granularity. This feature aims to smooth short-term fluctuations, highlighting longer-term trends in price movement.

$$\text{7-Day Moving Average} = \frac{1}{7}\sum_{i=0}^{6} P_{t-i}$$

Where:

- *Pt is the price at time t.*

2) 7-Day Moving Standard Deviation: Represents the standard deviation of the Rate over a 7-day rolling window, aiming to provide a measure of volatility.

Formula:

$$\text{7-Day Moving Std Dev} = \sqrt{\frac{1}{7}\sum_{i=0}^{6}(P_{t-i} - \text{7-Day Moving Average})^2}$$

4) Percentage Change: The percentage change in the ETH/USD Rate from one hour to the next, multiplied by 100 to transform it into a percentage.

Formula:

$$PercentageChange = \left(\frac{P_t - P_{t-1}}{P_{t-1}}\right) \times 100$$

Where:

- *Pt* represents the ETH/USD Rate at the current hour *t.*
- *Pt–1* is the ETH/USD Rate at the previous hour.

3) Relative Strengh Index (RSI): A momentum oscillator calculated by using the average gains and losses over a 14-hour window. It aims to identify overbought or oversold conditions in the trading of ETH/USD.

The RSI is calculated using the following approach:

$$RSI = 100 - \frac{100}{1 + RS}$$

Where *RS* is the Relative Strength, defined by:

$$RS = \frac{AverageGain}{AverageLoss}$$

4) Bollinger Bands: Calculated using simple moving average (SMA) of the ETH/USD Rate, plus as well as minus two standard deviations to create upper and lower bands. These bands show the price movement aiming to indicate volatility.

Formula:

$$\text{Upper Band} = MA + (k \times \sigma)$$
$$\text{Lower Band} = MA - (k \times \sigma)$$

.

Where:

- *MA* is the Moving Average
- *σ* is the standard deviation
- *k* is typically set to 2

5) Exponential Moving Average (EMA): Similar to a simple moving average but gives more weight to recent prices, making it more responsive to new information.

Formula:

$$EMA_t = P_t \times \left(\frac{2}{N+1}\right) + EMA_{t-1} \times \left(1 - \frac{2}{N+1}\right)$$

Where:

- *N* is the number of days in the EMA
- *EMAt−1* is the EMA of the previous period

6) Moving Average Convergence Divergence (MACD): A trend-following momentum indicator showing the relationship between two exponential moving averages. Indicates buy or sell signals.

Formula:

$$MACD = EMA_{12} - EMA_{26}$$
$$\text{Signal Line} = EMA_9(MACD)$$

Where:

- *EMA12* is the 12-period exponential moving average of the ETH/USD
- *EMA26* is the 26-period exponential moving average of the ETH/USD Rate.

7) Average True Range (ATR): The Average True Range (ATR) is a technical indicator that aims to measure market volatility by decomposing the entire range of an asset price for a period.

The Average True Range is calculated using the following approach:

$$ATR = \frac{1}{n}\sum_{i=0}^{n-1} TR_i$$

Where:

$$TrueRange = \max\left(High_t - Low_t,\ High_t - Close_{t-1},\ Low_t - Close_{t-1}\right)$$

- *High$_t$* is the highest price at time *t*.
- *Low$_t$* is the lowest price at time *t*.
- *Close$_{t-1}$* is the closing price at time *t−1*.

NOVA Information Management School
Instituto Superior de Estatistica e Gestao de Informa ao
Universidade Nova de Lisboa