University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Employing Machine Learning in the Policy-based Blockchain Selection Process

Ratanak Hy
Zürich, Switzerland
Student ID: 11-717-790

Supervisor: Eder John Scheid, Muriel Franco
Date of Submission: May 3, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Zusammenfassung

In den letzten Jahren gewann das Thema Blockchain, die zugrundeliegende Technologie von Kryptowährungen, zunehmend an Bedeutung und Aufmerksamkeit. Seit der Erfindung des Bitcoins in 2009 hat die Anzahl Kryptowährungen und verschiedener Blockchain Plattformen drastisch zugenommen. Bei einer solchen Vielzahl von Implementierungen und der daraus resultierende Unübersichtlichkeit wird es komplex, eine geeignete Blockchain für einen bestimmten Anwendungsfall auszuwählen. Kürzlich wurde ein policy-basierter Ansatz vorgeschlagen, um das Auswahlverfahren zu automatisieren, welcher Blockchain Implementierungen auf Basis von Transaktionsinformationen und vordefinierten Policies empfielt. Dieser Auswahlprozess wird durch einen einfachen Selektionsalgorithmus gesteuert, welcher verschiedene Filter anwendet. Das Ziel dieser Arbeit ist, neuartige Ansätze, wie maschinelles Lernen, zu erforschen und diese auf den Blockchain Auswahlprozess anzuwenden mit dem Ziel, deren Einbindung in das Policysystem. Dazu wurden verschiedene maschinelle Lernalgorithmen trainiert, eingesetzt und auf ihre Anwendbarkeit im Blockchain-Auswahlprozess evaluiert. Der entwickelte Prototyp erweitert die bestehende Lösung und bietet den Benutzern die Möglichkeit, zwischen dem herkömmlichen und dem auf maschinellem Lernen basierenden Algorithmus zu wählen, wobei sich die Parameter der Policies je nach Auswahl ändern. Die Resultate der Evaluierung des Prototyps zeigen, dass eine solche Kombination tatsächlich umsetzbar ist und genaue Ergebnisse liefern kann. Jedoch werden dabei auch Herausforderungen und notwendige Überlegungen hervorgehoben.

Abstract

In recent years the blockchain topic, the underlying technology of cryptocurrencies, has gained increasing importance and attention. Since the invention of Bitcoin in 2009, the number of cryptocurrencies and various blockchain platforms has drastically increased. With such a myriad of implementations and the resulting lack of transparency, it becomes complex to select a suitable blockchain for a specific use case. Recently, a policy-based management approach has been proposed to automate the selection process, which recommends blockchain implementations based on transaction information and pre-defined policies. This selection process is governed by a simple algorithm, which applies straightforward filtering. The goal of this thesis is the research of novel approaches, such as machine learning, and apply them to the blockchain selection process with the aim of integrating them into the existing solution. Therefore, various machine learning algorithms were trained, deployed and evaluated on their applicability in the blockchain selection process. The developed prototype extends the existing solution and provides users the option to choose between the conventional and the machine learning-based selection algorithm, with changing policy parameters depending on the selection. The results of the evaluation show that such a combination is in fact feasible and can deliver accurate results. But it also highlights pitfalls and necessary considerations.

Acknowledgments

I want to thank a number of people who have helped me throughout my work on this thesis. This would not have been feasible without their continuous support, directly or indirectly.

I would like to express my gratitude to my supervisor Eder John Scheid for his constant support, valuable feedback, and experienced advice throughout the thesis. I want to thank Prof. Dr. Burkhard Stiller for the opportunity to work on this thesis; Ivo, who was writing his thesis in parallel and was always available for a rant about current issues; and finally, Corina, for her encouragements.

Contents

Chapter 1

Introduction

Over the past years, several Blockchain (BC) implementations were created focusing on solving particular problems from early BC implementations, *e.g.,* low Transactions Per Second (TPS), lack of security, or non-sustainable consensus mechanism. The increase in the number of these BC implementations is evident when one verifies the CoinMarketCap website [19], where more than 4000 cryptocurrencies and BC implementations are listed. With such a myriad of implementations, selecting the most appropriate BC based on specific requirements is not a trivial task. Thus, there exists a need for specialized BC selection algorithms to be created with the aid of novel techniques, such as Machine Learning (ML).

Within the ML context, a decision tree algorithm can be trained and BC parameters required (e.g., TPS, block time, and supported data size) fit in such a tree to predict the BC that is the most suitable with these parameters. Furthermore, given a dataset disclosing choices of which BC to use for a given set of constraints, it is possible to define patterns and automate the decision process. In this context, dedicated ML techniques (e.g., supervised ML) can be applied given specific contexts (e.g., supply-chain, IoT transparency, or cryptocurrencies). Additionally, clustering algorithms can be applied to form groups of BC that share similar properties to provide a recommendation of alternative BC platforms. A couple of works [5, 27, 31, 116] propose approaches to decide the best suitable BC platform and type. However, they mostly follow a manual approach with diagrams and flowcharts, in this sense the inclusion of a new BC platform requires the revision of the flowchart. In contrast, with the employment of ML, the selection is automatic, and the algorithm readjustment straightforward.

The main focus of work of this Master thesis is the research of current ML algorithms that can be employed in the BC selection. In this context, the thesis explores a research aspect *(i)* that surveys the state-of-the-art ML application to BC, with the listing of underlying BC parameters that are crucial for the selection. The thesis further covers the practical aspect *(ii)*, with the implementation and integration of algorithms to an already developed BC selection framework (*i.e.,* PleBeuS [84]), and their evaluation to assess the feasibility in such combination. The objectives were to implement at least two fully functioning and in PleBeuS integrated ML algorithms that automatically select the best suitable BC given user-defined parameters with a detailed description and reasoning

of the implementation decisions. Ultimately this Master thesis evaluates and discusses the combination of technologies that cover a variety of dimensions. This also includes the analysis of the usability of the solution, security, and overall processing time.

Following Research Questions (RQ) drive the development of the thesis:

- **RQ1:** Can Machine Learning be useful in the BC selection? And which algorithms are most suitable?
- **RQ2:** How does a ML-based selection approach compare to a rule-based system?
- **RQ3:** Which features are important for the selection process?

1.1 Thesis Outline

The remainder of this thesis is structured as follows: Chapter 2 provides information on BC characteristics and BC implementations used for the ML algorithms. It introduces the BC selection framework, PleBeuS, explaining the key concepts of the framework. Moreover, it also discusses the concept of ML and the algorithms being used for the implementation of the ML-based BC selection solution. Chapter 3 discusses state-of-the-art BC selection approaches. The solution design of the ML-based BC selection is presented in Chapter 4. It further describes the Data Acquisition and Feature Engineering of the BC data and the resulting dataset that is used for the ML model training. The implementation of the ML models and their integration into the PleBeuS framework is discussed in Chapter 5. Chapter 6 provides an in-depth evaluation of the ML models and presents the results of the performance testing. Furthermore, it includes a use case analysis and provides a conclusion to the Research Questions. Finally, Chapter 7 draws a conclusion and summarizes the most important findings and provides an outlook for future work.

Chapter 2

Background

This chapter outlines and details the main concepts involved in this thesis. Section 2.1 provides a brief explanation of the BC technology along with a description of the BCs included in the ML-based BC selection solution and their corresponding characteristics. Followed by Section 2.2, which introduces the PleBeuS framework and its architectural design. Finally, Section 2.3 presents the term ML in more detail and the ML algorithms being tested for the BC selection process.

2.1 Blockchain Overview

The BC technology can be used in a wide variety of use cases where assets are managed or transactions occur. Its functional characteristics that facilitate transactions through trust, consensus and Smart Contracts (SC) can provide a secure chain of custody [98]. In the case of Bitcoin (*i.e.,* first BC implementation) [61], the BC is used to transparently record a ledger of payments, but it can be used beyond payment processing and money transfers, *e.g.,* it allows any data to be stored in an immutable and transparent way. From digital voting [49] and digital Identity [106] to medical records sharing [24] and supply chain monitoring [82], the range of its application is extensive and diverse.

Fundamentally, a BC is a distributed append-only immutable ledger [61]. It is organized as a list of ordered blocks that are immutable once they have been committed to the chain. It is replicated across all the nodes/participants of the network, and contrarily to traditional databases, there is no trusted central authority, allowing non-trusting nodes to verifiably interact with each other. The data of the BC is managed in blocks, which contain a pointer to the previous block, the transaction data, a cryptographically hashed value of a puzzle, and a timestamp.

A consensus mechanism guarantees that each copy of the BC is identical, providing a secure tamper-proof ledger. It ensures that a newly added block is legitimate and prevents the network from attackers compromising and forking the chain [18]. There are various consensus algorithms, one of them being Proof-of-Work (PoW) which was proposed by Satoshi Nakamoto [61], where a computation-intensive cryptographic puzzle needs to be

solved in order to add a new block to the chain. Other common consensus algorithms are Proof-of-Stake (PoS) or Byzantine Fault Tolerance, among others [112], which have been proposed to overcome some of the weaknesses of PoW, such as excessive power consumption [72]. Beyond the distributed ledger functionality of the BC, implementations can vary in their technical details as well as capabilities.

The following subsections aim to provide an overview of all BC implementations included in the ML-based BC selection solution and their main characteristics.

2.1.1 Bitcoin

The first decentralized electronic cash system, named Bitcoin, was invented and introduced by Satoshi Nakamoto in 2009. Its main proposal is to provide a means of monetary transaction without the intervention of a Trusted Third Party, allowing two willing parties to directly transact with each other [61]. The main disadvantage of Bitcoin is the computation-intensive PoW mechanism which causes issues regarding power consumption, scalability and centralization. The expected block time of Bitcoin is 10 minutes, *i.e.,* every ten minutes, a new block is mined, and it allows a data storage of 80 bytes per transaction [52]. Further, Bitcoin does provide the ability to create minimal SCs tailored for the automatic transfer of funds.

2.1.2 Ethereum

Ethereum is an open-source platform for decentralized applications and was introduced in 2014. In contrast to Bitcoin, it is a turing-complete scripting language and enables complex SCs. A SC is a self-enforcing agreement in the form of a coded script that runs on top of the BC. It consists of rules under which the parties agree to interact with each other [8]. When the rules are met, the contract is automatically enforced. Due to these capabilities, Ethereum also allows developing distributed applications using SCs and creating additional cryptocurrencies. To successfully perform a transaction or execute a contract on the platform, a fee, known as Gas, is required. It can be bought with Ether, Ethereum's currency [13]. However, it also uses PoW as a consensus mechanism, making it prone to the same shortcomings as Bitcoin, which is why it is planned to change to a PoS consensus mechanism [26]. This mechanism is an efficient alternative to achieve consensus, where holders of the coin are encouraged to stake their holdings, and the creator of the next block is randomly selected based on the stake of coins. The expected block time of Ethereum is 15 seconds, and it reaches a throughput of approximately 15-25 TPS [52].

2.1.3 Stellar

Stellar aims to provide a platform that connects the world's financial systems on a single network. It focuses on reducing the cost and time required for cross-border payments. It uses its own consensus protocol, namely the Stellar consensus protocol, which enables a

high transaction throughput (claimed 3000+ TPS) and a block time of around 5 seconds while having low transaction fees [54].

2.1.4 EOS

EOS has a similar vision as Ethereum. It aims at providing an infrastructure for decentralized applications, calling itself the most performant BC platform.

EOS uses a combination of delegated Proof-of-Stake (dPoS) and asynchronous Byzantine fault tolerance (aBFT) as its consensus algorithm, allowing a high transaction throughput that has reached an all-time high of 3'996 TPS [23], as well as a block time of 0.5 seconds. Hence, it is more efficient and consumes less energy than PoW but comes with a higher cost of centralization, introduced by dPoS with only 21 block producers (delegates) validating and checking new transactions [25]. It allows up to 256 bytes of data storage per transaction [84].

2.1.5 IOTA

IOTA was designed to handle transactions between machines and devices in the internet of things ecosystem.

By definition, it is not a BC *per se*, but its properties are similar. Rather than using blocks, it makes use of an acyclic directed graph, known as "tangle", to hold the transactions. The confirmation of transactions happens within the network by the transaction issuers. Before being able to add a new transaction, two pre-existing transactions have to be confirmed, which add two new edges to the graph [32]. The amount of transactions IOTA can handle depends on the number of users; the more users, the more transactions.

IOTA has a fixed block time between 1 and 5 minutes and allows a throughput of 500-800 TPS [45]. IOTA allows a data storage size of 1300 bytes per transaction and does not natively support smart contracts, which means it is also not turing complete [84].

2.1.6 Hyperledger Sawtooth

Hyperledger Sawtooth is an enterprise BC platform under the Hyperledger umbrella for building private distributed ledger applications and networks. The platform is highly modular, allowing applications to choose permission rights, transaction rules, consensus algorithms and many more according to their needs and use cases. Moreover, Sawtooth's multi-language support to write SCs makes it attractive for developers [21].

With the ability to use Proof-of-Elapsed-Time (PoET), a lottery-based consensus algorithm, the platform can support large production networks and improve the efficiency of present solutions such as PoW. The consensus algorithm generates a random wait time for each node in the network. The node with the shortest wait time will be selected and

permitted to create the next block. Thus, the block time is dependent on the random wait time, but a target waiting time can be set with a default of 20 seconds [22].

2.1.7 Multichain

Multichain aims to provide a BC platform for creating and deploying private BCs in the financial sector. Similar to Hyperledger, it is highly configurable (*e.g.,* different consensus, block sizes and time, permissions for nodes can be implemented [35]). The target block time can be customized but is set to a default value of 15 seconds [60].

2.1.8 Corda

Corda is a BC platform that has been created by a consortium lead by R3. The platform was built to address the challenges of privacy and finality shortcomings in legacy BC technologies by enabling private transactions with immediate finality. Participants of the network must have verifiable identities using a public-key infrastructure. Instead of broadcasting each transaction to the network, the data is only shared with the nodes involved in the transaction on a "need-to-know" basis, ensuring a higher level of privacy. Consensus is reached at a transaction level, where each transaction points to a notary, a network service that checks the validity and uniqueness of the transaction [55]. Corda measures between 15 and 1678 TPS [20].

2.1.9 Stratis

Stratis is a BC development platform for enterprise businesses. Stratis seeks to facilitate the development process for creating BC applications and accelerates the development lifecycle for BC development projects. Businesses have the ability to deploy their BC applications on Stratis as private sidechains, allowing customization and taking advantage of the parent BC's properties. Stratis applications can be developed in pure C# and also make use of the Microsoft.NET framework while taking advantage of Stratis APIs and framework. Stratis also has an ICO launch platform that allows ICO's to be launched quickly. It uses PoS as a consensus algorithm and has a block time of 48 seconds [97]. Arbitrary data up to 40 bytes can be stored in a transaction [96]. The maximum throughput on the mainchain is between 33 and 67 TPS, while the maximum throughput on the sidechain is dependant on the configuration of the sidechain creator [95].

2.1.10 Cardano

Cardano is a decentralized public BC and cryptocurrency. Like Ethereum, it is designed as a platform on top of which SCs and decentralized applications (known as "Dapps") can be executed. It is based on the Ouroboros algorithm, which is the first peer-reviewed,

verifiably secure BC protocol based on PoS, that determines how consensus is achieved in the network [48].

The protocol proceeds in *epochs*; each epoch has its fixed number of units called *slots*, each lasting for one second. Currently, an epoch includes 432000 slots, which corresponds to five days. Not more than one block might be appended during every slot, also allowing slots, where no blocks are generated. In each slot, one stakeholder, known as the slot leader, is randomly selected from a pool of stakeholders and creates the next block. On average, every 20 seconds, one node is nominated [14]. The average transaction data size is around 500 bytes, which means it can reach 205 TPS [15].

2.1.11 NEO

NEO is an open-source BC platform on which decentralized applications and SCs can be run, similar to Ethereum and Cardano. Neo is also referred to as the "Chinese Ethereum" and was first released in 2014 under the name "Antshares". In 2017 the project was re-branded to "NEO". It aims at creating a smart economy to digitize assets and automating their management [103]. *Gas* is NEO's native currency which, much like Ether, is used to pay the fees (GAS) to utilize the platform. NEO uses a consensus algorithm known as delegated Byzantine Fault Tolerance (dBFT), which enables large-scale participation in consensus through proxy voting. The consensus algorithm takes up to 15 to 20 seconds to produce a block and allows around 1000 TPS. However, it could potentially support 10000 TPS [62].

2.1.12 Ripple

Ripple refers to both the cryptocurrency XRP and to its open-source, decentralized digital payment platform RippleNet which is widely used by a number of banks and financial institutions. The platform can settle near-instantaneous transfers of currency, independent of their type. It allows transactions in USD, Euro, Yen, Pound, Bitcoin, Ether, Litecoin or XRP [101]. XRP was built to act as a bridge currency that financial institutions can use to settle cross-border payments faster and less expensive than by using the usual global payment networks, which are slow and involve several intermediaries. Bitcoin could also be used for a bridge currency, but XRP can settle up to 1500 TPS, and the transaction fees are significantly lower. However, RippleNet, in practice, does not require a bridge currency to work [66].

Ripple is not using PoW or PoS mechanism to validate its transactions but instead, its own specific consensus protocol, which reaches consensus every 3 to 5 seconds [77]. Ripple can be categorized as permissioned BC because the company behind XRP, Ripple (Labs) Inc, determines who can act as a transaction validator on its network. However, the BC itself is considered public because it can be accessed by anyone. XRP consistently handles 1500 TPS and reaches consensus every 3-5 seconds [78].

2.1.13 QTUM

QTUM aims to provide a public BC platform for SC systems that are highly scalable. It is an UTXO-based (BC design of Bitcoin) SC system with a PoS consensus model and compatible with Bitcoin- and Ethereum ecosystems. It attempts to produce a variation of Bitcoin with Ethereum Virtual Machine (EVM) compatibility. Thus, providing an abstraction layer that translates the UTXO-based model to an account-based interface for the EVM [64].

It allows around 70 TPS and the block time is approximately 144 seconds [75]. A reasonable transaction size is 2 kb [74].

2.1.14 ICON

ICON is a BC technology, which is built to enable interoperability between independent BCs. The network is supported through their own cryptocurrency token, called ICX [41]. The foundation aims to provide universities, hospitals and financial institutions the ability to interact with each other across BCs [38]. It is based on a Loop Fault Tolerance consensus algorithm, which is an enhanced Byzantine Fault Tolerance [41].

ICON offers a transaction speed of around 500 TPS [42] and a block time of 2 seconds [39]. The maximum size of data in a transaction is 512 kb [37].

2.1.15 VeChain

VeChain, a leading public BC platform, was founded in 2015 with the goal of connecting BC technology to the real world by supplying businesses with tailored blockchain solutions based and building a trust-free and distributed business ecosystem platform for business value [107]. VeChain makes use of two tokens, the VeChain Token (VET) and the VeChainThorEnergy (VTHO). The former, to transfer value across VeChain's network, the latter to power SC transactions, which is similar to Ethereum's Ether or NEO's Gas. VeChain uses a Proof of Authority consensus protocol, where only a designated number (101 master nodes) of VET holders are given the power to validate transactions on the network [29]. Vechain reaches 165 TPS and has an average block time of 10 seconds [114].

2.1.16 Wanchain

Wanchain is a cross-chain public BC intended for the financial industry to promote asset transfers and host decentralized applications. Wanchain's vision is to provide a cross-chain DeFi platform, where assets and information will be able to flow freely and securely between any type of BCs without any centralized third party [110]. As of now, it has been integrated with Bitcoin, Ethereum, EOS, and other ERC-tokens. It also has its own token called WAN and uses PoS as its consensus algorithm [57].

2.1.17 Summary

Table 2.1 shows the summary of the BCs used for this project, including their fundamental properties. For BCs where no information was found regarding the supported data size (Ripple, VeChain and Wanchain), an arbitrary amount of 150 bytes was defined. The information about BC implementations that are already supported by PleBeuS (see Section 2.2) have been adopted from [52] for the most part.

Table 2.1: Summary of BCs and their characteristics

BC	Type	TPS	Time (s)	Data	Smart Contracts	Turing-complete
Bitcoin	public	4 - 7	600	80 bytes	No	No
Ethereum	public	15 - 25	15	46 kbytes	Yes	Yes
Stellar	public	3'000	5	28 bytes	Yes	No
EOS	public	250 - 3'996	0.5	256 bytes	Yes	Yes
IOTA	public	500 - 800	60	1300 bytes	No	No
Hyperledger	private	variable	variable	20 bytes	Yes	Yes
Multichain	private	variable	variable	80 bytes	No	No
R3 Corda	private	variable	variable	100 bytes	Yes	Yes
Stratis	private	variable	variable	40 bytes	Yes	No
Neo	public	1'000	15	255 bytes	Yes	Yes
Cardano	public	205	10	500 bytes	No	No
Ripple	public	1500	3.5	150 bytes	No	No
QTUM	public	70	120	2 kbytes	Yes	Yes
ICON	public	500	2	512 kbytes	Yes	No
VeChain	public	165	10	150 bytes	Yes	No
Wanchain	public	15	15	150 bytes	Yes	Yes

2.2 PleBeuS

PleBeuS is a policy-based BC selection framework that allows users to define requirements based on their needs. These requirements are captured in the form of policies, which are used by the framework to automatically determine the most suitable BC technology to store incoming data. The proposed BC selection process, described in [84], first applies basic filtering, following a divide and conquer approach and removing BCs that do not meet the requirements. For the filtering, PleBeuS uses the previously defined policy to reduce the input size of suitable BCs for the selection algorithm. The selection algorithm either uses an algorithm that minimizes transaction costs or an algorithm that prioritizes the transaction speed, which depends on the configuration of the user for the Cost Profile in the policy. The framework allows two types of policy parameters, BC-specific and externally driven parameters. The former includes key BC implementation characteristics that are static and only change if a hard fork occurs. The latter contains non-static parameters that are prone to external factors, such as the time of the day. The configurable parameters are outlined in Table 2.2.

The framework supports seven BC implementations at the time of writing, but further

Table 2.2: Policy parameters in [84]

Policy Parameters	
BC-specific	Public vs Private
	BC Throughput
	Block Time
	Data Size
	Turing Completeness
Externally Driven	Cost Thresholds
	Cost Interval
	Cost Profile
	Transaction Split
	Time Frame
	Preferred Blockchain

BC integrations are anticipated. Table 2.3 presents all available BC, including their key characteristics.

Table 2.3: BCs and characteristics supported by PleBeuS [52]

BC	**Type**	**TPS**	**Time (s)**	**Data**	**Turing**	**Fees**
Bitcoin	public	4 - 7	600	80 bytes	No	variable
Ethereum	public	15 - 25	15	46 kbytes	Yes	variable
Stellar	public	1'000 - 4'000	5	28 bytes	No	base fee
EOS	public	250 - 3'996	0.5	256 bytes	Yes	variable
IOTA	public	500 - 800	60	1300 bytes	No	none
Hyperledger	private	variable	20 (default)	20 bytes	Yes	none
Multichain	private	variable	15 (default)	80 bytes	No	none

2.2.1 Policy-Based Network Management (PBNM)

PleBeuS uses a PBNM architecture as a building block. PBNM is an already widely used technology and provides a means to facilitate the complex task of managing networks and distributed systems in a mostly automated fashion. It allows to manage a network or distributed system in a flexible and simple manner by utilizing predefined policies which govern its behavior [109]. The rules (*i.e.,* policies) are defined in an Event-Condition-Action format. The applicability of these principles is not only limited to a distributed systems and network management context and can be used in other contexts too. PleBeuS has successfully shown that these principles could be abstracted and applied in the decision process on which BC to store data based on a set of rules [84].

The PBNM architecture is based on RCF 3060 [58] and can be divided into four components:

- **Policy Management Tool (PMT)**, which is used to define and manage policies.
- **Policy Repository (PR)**, which serves as the storage for the defined policies.
- **Policy Decision Point (PDP)**, which retrieves policies from the repository and decides which policy will be used and enforced from the next component.
- **Policy Enforcement Point (PEP)** is responsible for the execution of the policy that has been chosen from the PDP.

2.3 Machine Learning

ML is about understanding and extracting knowledge from data. In its simplest sense, ML uses algorithms that learn by analyzing input data and optimize their operations to make predictions within an acceptable range [105]. These algorithms have a wide range of applications, including detection of fraudulent credit card transaction, identification of handwritten letters, medical diagnosis, customer segmentation and recommender systems [59].

One of the most promising and successful kinds of ML algorithms is *supervised learning*, where the decision-making process is automated by generalizing from known data. Given an input, which is unknown to the ML algorithm, it can generate an output without human interaction. In general, algorithms that learn from labeled datasets are called supervised learning. The labeled data acts as the supervisor to the algorithm, informing them about the desired output for each input. Another type of ML algorithm is *unsupervised* learning. In contrast to supervised learning, only the input (unlabeled data) is given to the algorithm, and hence no supervision takes place.

Two groups of categories of algorithms come under the umbrella of supervised learning [56]:

- **Classification**: This is a fundamental problem to analytics, patterns recognition and ML. Since it categorizes data from observed samples, it is regarded as a supervised learning technique. The samples are associated with two or more classes known as Binary classification and Multiclass classification. The class of each instance is determined by finding patterns and combining features from the training data. A classification in ML is accomplished in two phases. In the first step, a classification algorithm, such as a decision tree, is applied to the training data. Secondly, the model is extracted and validated against a test dataset to evaluate the performance and accuracy of the model [91].
- **Regression**: The data is labeled with a real value instead of a class (such as time-series data, stock prices over time, daily sales volume, etc.). The algorithm is used to predict continuous values.

Two main types of problems that unsupervised learning tries to solve:

- **Clustering**: The goal is to identify different groups within the data and to separate data into groups (clusters) according to their similarity and other measures. Common clustering algorithms are K-means, Hierarchical Clustering and Gaussian Clustering Model [79].
- **Dimensionality Reduction**: Reduces the dimensionality, number of features (columns) present in the dataset, which decreases the model's complexity and the chance of overfitting [80].

With regards to the BC selection process, the task at hand is to present users with a suitable BC that will fulfill their requirements for their use case. This problem can be associated with a multiclass classification problem, where each sample (*i.e.,* requirements of BC characteristics) is exactly assigned to one BC from a set of BCs. Thus, a supervised learning algorithm can be applied to solve this problem. Common learning algorithms for such problems include, among others, Decision tree, Random forest, Support vector machine and Naïve Bayes that are discussed in the following subsections.

2.3.1 Decision Tree

Decision Tree (DT) belongs to one of the few models which are straightforward to comprehend why a particular decision was made. It is among the most popular classification algorithms being used in ML. Its main concept is to assist in finding the most suitable features to split the tree into sub-parts [51]. There are different impurity metrics that are used to determine the decision variables at each node. One of them is Entropy/Information Gain and the other the Gini index [76]. These splitting strategies are used to build an appropriate decision tree efficiently.

Entropy is a metric to measure disorder or unpredictability/uncertainty in a system. It is measured between 0 and 1. It characterizes the impurity of an arbitrary collection of data points. The data is said to be pure if there is only one class. Contrarily, the data is impure if there is more than just one class. Thus, the impurity is the degree of randomness. So the higher the Entropy, the higher the level of disorder, *i.e.,* high level of impurity. If the data is either completely pure or impure, the randomness equals zero.

The mathematical expression for the Entropy is described in Equation 2.3 [100]:

$$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{2.1}$$

where,

p_i: Frequentist probability of class "i" in data
$E(S)$: Entropy of the entire dataset

The algorithm computes the Entropy for every attribute after each split, and as the splitting continues, it selects the best feature and starts splitting accordingly.

The goal of this approach is to minimize the degree of uncertainty. A concept called **Information Gain** (IG) is used. This is a metric to quantify how much a piece of information reduces the uncertainty. It represents the decrease in Entropy after a dataset is split based on some attribute. While creating a decision tree, it is important to find an attribute that returns the highest IG. It decides which particular attribute should be selected as the decision node. Formally the IG from X on Y can be described as follows [100]:

$$IG(Y, X) = E(Y) - E(Y|X) \tag{2.2}$$

To obtain the IG of attribute X, *i.e.,* how much did additional attribute X reduced certainty about Y, the Entropy of Y given X is subtracted from the Entropy of Y. The greater the IG, the more information is gained about Y from X.

However, another widely used metric to measure impurity is the **Gini Index**, also known as Gini impurity. It measures how often a randomly chosen element would be incorrectly identified, thus making an particular attribute with a lower Gini Index more preferable. It is computationally less heavy than Entropy, because it uses simple probabilities instead of the log base 2 of the probabilities. The Gini Index can take on values between 0 and 0.5. The mathematical notation is shown below [115].

$$Gini = 1 - \sum_{i}^{n} p_i^2 \tag{2.3}$$

Advantages

DTs are relatively straightforward to understand and interpret compared to other ML algorithms. The resulting DT can be visualized and inspected to comprehend why the classifier has made a particular decision. Furthermore, DTs can be applied to both classification and regression problems. Data preparation requires less effort during pre-processing as multiple data types such as numeric, nominal and categorical are supported. Additionally, it uses different measures such as Entropy, IG and the Gini index to find the best split attribute [105].

Limitations

Despite being one of the most popular and useful ML algorithms, DTs have some disadvantages as well. DTs are less appropriate for estimation tasks where continuous values are predicted. They can be subject to overfitting and underfitting, specifically when a small dataset is used. This can lead to poor generalizability and robustness of the resulting model. Another shortcoming is that the algorithm cannot branch if some attribute for a non-leaf node is missing [105].

2.3.2 Random Forest

A Random Forest (RF) is an ensemble classifier made up of many DTs [10]. They belong to the most popular and used ensemble methods. Ensemble methods make use of many learners to increase the performance of any single one of them individually. It takes a group of learners together in order to create a better, aggregated one. It combines the simplicity of DTs with the power of an ensemble model.

Instead of using the entire dataset to train a single DT, the RF method randomly picks a sample of the dataset to train each DT of the RF. The use of different samples for training decreases the chance of overfitting. A single DT might be prone to overfitting, but as each DT in the RF is trained with a different training set, the aggregated combination of the trees can generalize better to new data [105].

Advantages

As RF ultimately consists of multiple DTs, it generally has the same advantages. Moreover, since RF takes the average value from the outcomes of its constituent DTs, the chances of overfitting are lower compared to a single DT. Empirically this ensemble method performed better than an individual DT. RF also scales well for large datasets with higher dimensionality [105]. Furthermore, the model outputs the importance of variables or attributes for the classification or regression task.

Limitations

RF is generally more complex, which makes it less interpretable than a single DT. It is also computationally more expensive as it involves training multiple DTs and retaining the information from these trees. Therefore, the algorithm slows down as the number of trees increases. It might be less vulnerable to overfitting than a single DT, but nevertheless, it can still occur [105].

2.3.3 Naïve Bayes

Naïve Bayes (NB) is another supervised ML algorithm that is widely used to solve classification problems due to its simplicity. It assumes that features are independent of each other and that there is no correlation between them, which is rarely the case in real life. This naive assumption is the reason for the algorithm's name. Moreover, it is a probabilistic classifier based on Bayes Theorem for calculating probabilities and conditional probabilities [105]. It uses these probabilities to predict the class of new data instances.

Bayes theorem can be mathematically described by the following expression [81]:

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)} \tag{2.4}$$

where,

- $P(A)$ is the probability of class A.
- $P(A|B)$ is the conditional probability of class A given predictor B (features).
- $P(B)$ is the prior probability of a predictor.
- $P(B|A)$ is the conditional probability of predictor B given class A.

There are different types of NB Classifiers: *Gaussian*, which is used for classification tasks where the features follow a normal distribution. *Multinomial*, which is suitable for classification with discrete features, and *Bernoulli* that is a binomial model used for feature vectors that are binary [86].

Advantages

It is a relatively fast algorithm for classification problems and requires short computational time for training, as it only requires one scan of the training data for the probability generation. NB can also be applied for both binary and multiclass classification problems. It also does not require a large dataset. When the independence assumption holds, NB performs well and better compared to other approaches [91].

Limitations

If the independence assumption does not hold, the performance of an NB model can be very low. Moreover, it requires a vast number of records to obtain good results. For categorical values that have not been observed during training, the model will be unable to make a prediction [105].

2.3.4 Support Vector Machines

Support Vector Machine (SVM) is mainly applied to binary classification problems, dividing the data point either in 1 or 0. In the case of multiclass classification, the same concept is used. The multiclass problem is decomposed into multiple binary classification instances, also known as one-vs-one. Each class is compared to every other class, such that there is a classifier for each pair of classes. Considering n classes, the number of classifiers necessary for one-vs-one multiclass classification can be determined with the following formula: $\frac{K(K-1)}{2}$ binary classifiers [3, 34]. With this approach, every classifier separates points of two different classes, and comprising them, generates a multiclass classifier.

The data points are first mapped into an n-dimensional feature space, where n is the number of features. Subsequently, the hyperplane is identified, which separates the data points into their potential classes. This is done by maximizing the marginal distance (distance between the hyperplane and the datapoint of the class) for both classes and minimizing the classification errors [105]. For the minimization, a penalty term C is

introduced. It is a regularization parameter that controls the magnitude of the penalty regarding how many data points have been wrongly assigned. SVMs are also termed kernelized SVM due to their kernel, which transforms the input data space into a higher dimensional space. The most popular kernel functions are linear, polynomial, radial basis and sigmoid functions [36]. In terms of the BC selection process, the SVM algorithm would train classifiers for each pair of BCs , one to discriminate between Bitcoin and Ethereum, another between Bitcoin and Stellar, another for Ethereum and Stellar, and so on.

Advantages

SVM can handle multiple feature spaces, and the chance of overfitting is generally lower than DTs [105]. SVM can also be applied for both regression and classification problems. It has an excellent generalization capability and is robust to outliers. It also allows a flexible selection of kernels for nonlinearity, and it has a general good generalization ability [91].

Limitations

For large and complex datasets, the model training is computationally more expensive than other approaches. The impact and selection of variables of the SVM algorithm are rather difficult. Due to its complexity, the resulting model is hard to understand and interpret compared to DTs for example [105].

Chapter 3

Related Work

Since the release of Bitcoin in 2009 [61], the popularity of BC technology has rapidly increased. It received widespread attention from all kinds of interest groups over the last few years. A wide variety of different BC implementations and platforms have meanwhile emerged, making it almost impossible to maintain an overview, much less acquire knowledge about all various BC technologies. Professionals struggle with the decision if a BC should be adopted and integrated into their existing workflow. With the extreme emerging trend, various solutions came up, differing in features and configurations, ranging from cost efficiency, performance and storage, to access restrictions and decentralization.

In this sense, it became cumbersome to choose a suitable solution for a specific use case. Thus, it is necessary to gather domain-specific knowledge about the BC selection process while providing the most suitable BC for a given use case. There exist quite a few decision schemes already, all with the aim to ease the decision-making process.

This chapter provides an overview of state-of-the-art BC selection approaches. Section 3.1 presents manual approaches that propose solutions to guide users through the selection process. Whereas Section 3.2 discusses semi or fully automated approaches, providing solutions where a BC is automatically selected based on specific user requirements.

3.1 Manual Approach

[53] discusses the fact that there is no clear guideline in the literature that helped to evaluate if BC technology is suitable for a particular use case. Thus, they proposed an evaluation framework. They narrowed down the evaluation process to specific questions that need to be answered. With this framework, one can derive if a BC would be suitable or if a conventional database should be used instead. They further used the framework to assess four different use cases. They concluded that supply chain management, electronic health records and identity management systems are promising areas for BC-based applications.

In [67], a decision framework in the form of a flowchart is presented. The flowchart focuses on whether or not to use a BC in an Internet-of-Things (IoT) setting and defines which

setting is the most appropriate. The framework manually guides a potential user through the decision-making process.

In [50], a framework is provided that follows a questionnaire form. It is used to evaluate the applicability of BC technology on specific use cases. In the first step, the particular use case is being identified. If the use case is considered suitable for BC, a use case canvas can be applied for an in-depth analysis to gain an understanding of how it could profit from a BC. The main limitation of these two approaches is that they only provide the answer to whether a BC technology is suitable or not. They do not suggest what type of BC could be appropriate.

Wüst and Gervais present a flowchart approach in [116]. The framework helps to determine if a BC makes sense for a specific application. They compare permissionless and permissioned BCs to conventional centralized Database systems while taking the consensus mechanism, throughput, latency, number of readers, number of untrusted writers and central management into consideration. They further describe several application use cases, such as Supply Chain Management or Interbank and International Payments, and evaluate what implications a BC solution might have on these cases. They concluded that BC is well suited for financial applications to ease the process of interbank payments, which involves multiple complex steps as long transaction confirmation time and the associated costs. It additionally requires a lot of trust for the system to work. In fact, some central banks already elaborate distributed ledger technology for interbank payments [1].

In [71], the authors developed a catalog of criteria focusing on software quality. Besides BC specific criteria, the catalog also includes *(i)* software quality such as usability, maintainability, portability or modifiability, *(ii)* open-source software quality, and *(iii)* software maturity.

Another flowchart based approach was proposed in [5]. It guides users through the definition of *when* to use BC as a technology, providing them with an answer if a BC adoption would represent a good solution and if so, suggesting user *which* type of BC is the most appropriate.

3.2 Semi-automated and fully automated approach

In [27], a decision model was designed with the help of the knowledge of domain experts to assist decision-makers in the BC selection process. Users can prioritize BCs based on their desired features using the Must, Should, Could, Won't (MoSCoW) technique. The model offers a ranked list of feasible BC platforms to the users based on their specific preferences and requirements.

In [2], a neural network-based decision scheme is presented, which not only allows users to set their requirements based on a fixed predefined set of answers but provide them with the opportunity to fine-grain their preferences beyond the proposed answers. Users are allowed to specify proportional weights according to their preferences between characteristics.

These two decision models can be considered semi-automated, as they automatically recommend feasible BC implementations based on user requirements. However, they do not provide the ability to react to changes in an automated manner, such as the framework proposed in [30], where users can define their demands based on a set of metrics and weights which are used to calculate and automatically select an appropriate BC. Moreover, it enables a switch over to another BC at runtime and copying data from one BC to the other. In this way, users are not being tied to one particular BC. Similarly, PleBeuS [84], which was introduced in Section 2.2, also provides a means to automatically store, retrieve and copy data from various BCs without requiring any changes in the application code [84].

3.3 Discussion

As presented, most decision schemes follow a manual approach and are not automated, forcing users to manually perform the selection process. Although [27] and [2] automatically suggest appropriate BC implementations, they are not fully automated and do not have the ability to store data on the BC or provide BC interoperability support as [30] or [84] do.

In summary, there are various means that aim to ease the BC selection process, which follows either a manual, a semi-automated or a fully automated approach. Table 3.1 summarizes the presented comparison of the state-of-the-art BC selection approaches.

Table 3.1: Comparison of related work

Work	Approach	Automated	Interoperability	Required Knowledge
[53]	Criteria-based	No	No	Basic
[67]	Flowchart-based	No	No	Basic
[50]	Questionnare-based	No	No	Basic
[116]	Flowchart-based	No	No	Basic
[71]	Criteria-based	No	No	Basic
[27]	Experts-based	Semi	No	Basic
[5]	Flowchart-based	No	No	Advanced
[2]	Neural network based	Semi	No	Advanced
[30]	Weighted metrics-based	Yes	Yes	Advanced
PleBeuS [84]	Policy-based	Yes	Yes	Advanced

Chapter 4

ML-based BC Selection

This chapter provides a high-level view of the steps involved in the ML-based BC selection solution and the data acquisition process that led to the final dataset. It starts with the solution design in Section 4.1, where all steps involved in the process of model training and deployment are illustrated. Section 4.2 contains a detailed description of all the features included in the dataset.

4.1 Solution Design

The building process of the ML model follows the workflow depicted in Figure 4.1. It involves two main artifacts: *Data* and *Model*, which are generated in three different phases: *Data Acquisition, Data Processing* and *Model Engineering*. In a first step, relevant data is gathered and integrated from various sources, such as external APIs, to fetch information about the block time of a BC. Besides collecting raw data, the *Data Acquisition* phase includes identifying and generating features, as well as data labeling. The result of this data collection is the initial dataset, which will be the primary focus in Section 4.2. Next, the data is further processed in an intermediate *Data Processing* phase, which aims at preparing the data for model training. As raw data typically cannot be used directly for model training, it needs to be processed and transformed into a numeric representation. The processed data is then divided into a training and a test set. This split is necessary to estimate the performance of the ML models because it allows to make prediction on data that was not used to train the model. In the *Model Engineering* phase, different ML algorithms are applied to the training data to obtain the final ML models for the BC selection. Finally, in the last stage, the *Deployment* phase, the models are deployed through a REST API and integrated into the PleBeuS framework, where the framework asks the model for predictions by passing feature values through an API call.

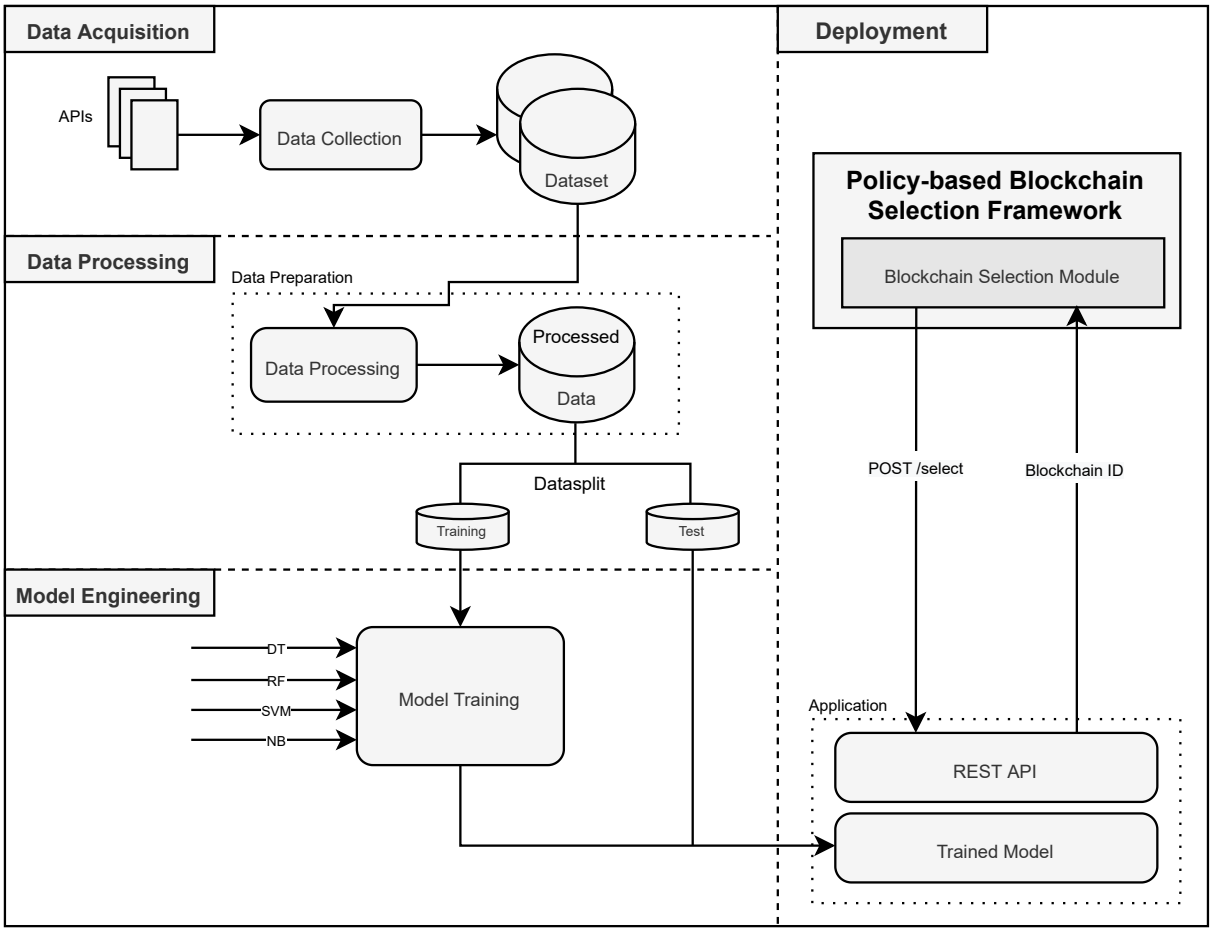


Figure 4.1: Solution design

4.2 Features

A feature typically represents an individual, measurable attribute or characteristic, which is commonly depicted by a column in a dataset. It generally represents an attribute plus its value (*e.g.,* "Popularity = High"). Many people use the words *feature* and *attribute* interchangeably [34]. The features included in the dataset will be used as input to build a ML model and predict target values on them. Thus, being the building blocks of a dataset for model training, features are of utter importance and ultimately impact the quality of the ML model. The BC selection process can be regarded as a classification-based task that falls into supervised ML, as described in Section 2.3. The main goal is to predict a suitable BC based on multiple input data that consist of different attributes or features. This section provides an overview of the data acquisition and all features being used for model training.

4.2.1 Popularity measure

The popularity of a BC platform can be an essential aspect and potentially a requirement for businesses considering adopting BC technologies in their day-to-day operations. A popularity measure could be an indicator of how technically mature a certain BC is in terms of development and community support. To generate such a measure, various information from different sources was used. In particular, the number of Twitter followers, number of monthly Google searches as well as the number of conference papers mentioning the platform have been collected and collated in order to classify a platform's popularity.

Twitter

Twitter has expeditiously increased in popularity since its launch in 2006. A prominent example to demonstrate its reach and power was on January 15, 2019, when a Twitter post broke the news of a flight crash faster than traditional media outlets. Twitter has approximately 330 million monthly active users and 145 million daily users. In total, around 1.3 billion accounts have been created, and 83% of the world's leaders have a Twitter account [92]. Due to these remarkable statistics, Twitter seems to be a brilliant source to get insights into how people perceive certain topics, which is why it is used as one source to measure the popularity of a specific BC.

Following parameters were gathered through Twitter's API [104]:

- Followers
- Friend count, *i.e.,* number of users one is following
- Number of tweets

As the friend count and the number of tweets were insignificant compared to the number of followers, which can be observed in Figure 4.2, these values have been neglected for the popularity measure. Instead, the followers' count was used solely for that purpose. The distribution of the number of followers as a boxplot can be seen in Figure 4.3. The 33^{rd} and 66^{th} percentiles are approximately 15200 and 356000 followers, respectively.

Finally, the number of followers has been grouped into three categories with an ordinal scale: low, medium or high. A percentile-based approach has been used to categorize them into one of these three categories. Numbers below the 33^{rd} percentile were ranked low, numbers between the 33^{rd} and 66^{th} percentiles medium, and the ones above the 66^{th} percentile high, which leads to the categorization shown in Table 4.1.

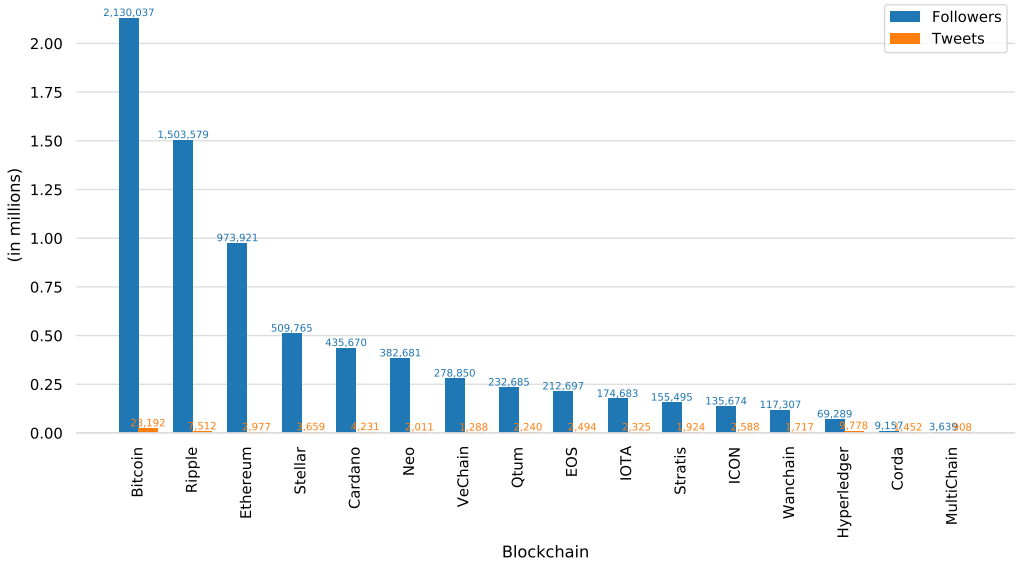


Figure 4.2: Twitter followers and number of tweets as of April 26th 2021

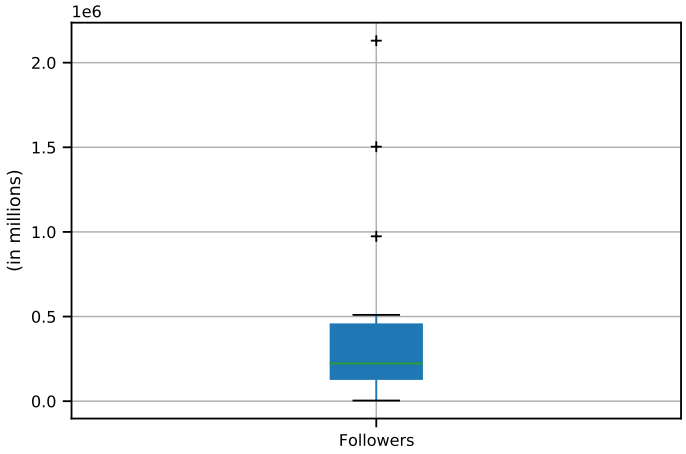


Figure 4.3: Distribution of followers

Google Searches

The Google search engine is by far the most popular in the world, with over 90 percent of the market share [17]. Therefore, Google search data can be very valuable in providing insights into how the public perceives a particular topic.

Most keyword research tools provide limited statistics about search volumes. `Ahrefs.com`, a common toolset for Search Engine Optimization (SEO) analysis, provides a keyword explorer, where the global search volume (sum of searches across all countries) can be

checked for a specific keyword. The search volume represents the average number of monthly searches on Google (12-month averages) and is an essential and frequently used metric, especially in SEO [93]. For that reason, monthly Google searches have been included in the final popularity score as well. The monthly search results are depicted in Figure 4.4.

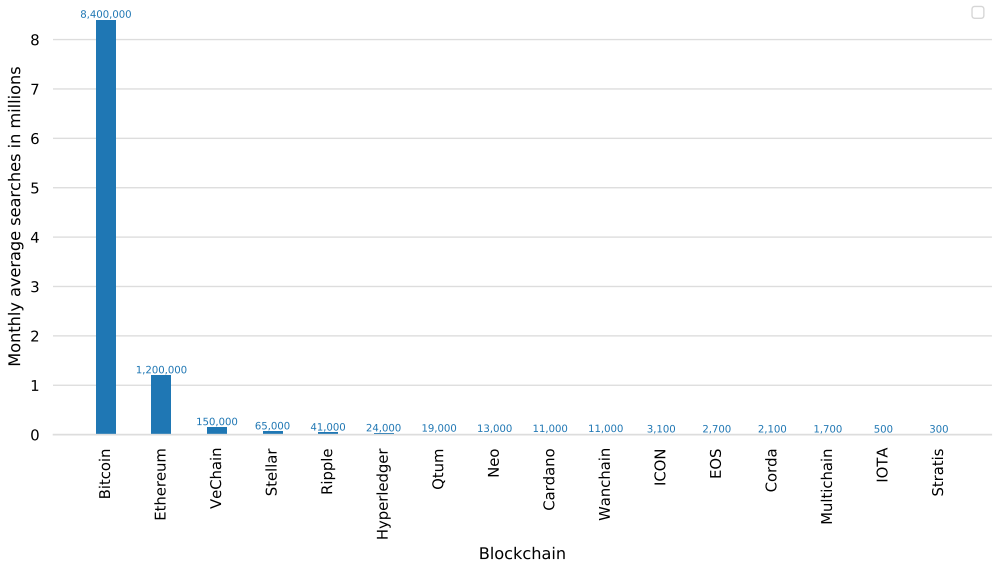


Figure 4.4: Monthly Google searches as of January 27th 2021

As with the Twitter followers, a percentile-based approach has been followed to categorize the data. The 33$^{\text{rd}}$ and 66$^{\text{th}}$ percentiles for the number of monthly Google searches are approximately 3080 and 23500, respectively. The final categorization is shown in Table 4.1.

Conference Papers

To include a metric for the popularity of a BC among academia and research, the number of conference papers mentioning the BC was exploited. Therefore, IEEE Xplore was used as the primary resource. IEEE Xplore is a digital library for discovering scientific and technical content published by the Institute of Electrical and Electronics Engineers and its partners [43]. An overview of the total number of conference papers is presented in Figure 4.5. Equivalent to the Twitter followers and Google searches, a percentile-based approach was used for the final categorization. The 33$^{\text{rd}}$ and 66$^{\text{th}}$ percentile conference papers are 3 and and 12, respectively. The categorization is shown in Table 4.1.

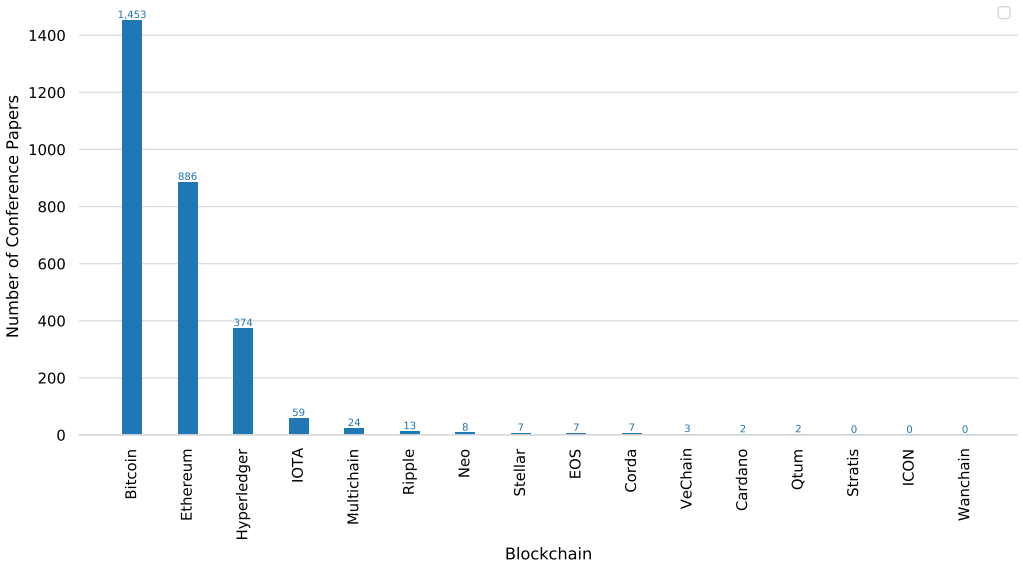


Figure 4.5: Number of conference papers as of January 27th 2021

Popularity Score Calculation

The final popularity score presented in Table 4.1 is calculated using Equation 4.1. Each variable is multiplied by an arbitrary weight ($\{W_1, W_2, W_3\}$) defined by the user. For example, if the user defines that Google searches impact in 50% of the score (*i.e.,* 0.5), then $W_2 = 0.5$ and W_1 and W_3 must sum to 0.5, *e.g.,* $W_1 = 0.25$ and $W_3 = 0.25$, being 25% of impact for each.

$$PopularityScore = N_{follow} \times W_1 + G_{search} \times W_2 + C_{papers} \times W_3 \tag{4.1}$$

where,

$$W_1 + W_2 + W_3 = 1.0$$

The weights used to calculate the overall score in Table 4.1 were the following. Twitter followers impact 15% of the score ($W_1 = 0.15$). This factor is chosen low because private BCs tend to have a significantly lower number of followers, but that does not mean they are not popular among enterprises. Google searches impact 50% ($W_2 = 0.5$) of the score as it indicates clear popularity among regular users and developers. Finally, academic papers have more impact than Twitter followers because they show the popularity among academia and researchers. Therefore, its factor is set to $W_3 = 0.35$.

Table 4.1: Final popularity score

Blockchain	Nº of Followers	Google Searches	Conference Papers	Overall score
Bitcoin	High	High	High	High
Ethereum	High	High	High	High
Stellar	High	High	Medium	High
EOS	Medium	Low	Medium	Medium
IOTA	Medium	Low	High	Medium
Hyperledger	Low	High	High	High
Multichain	Low	Low	High	Medium
Corda	Low	Low	Medium	Low
Stratis	Medium	Low	Low	Low
NEO	High	Medium	Medium	Medium
Cardano	High	Medium	Low	Medium
Ripple	High	High	High	High
QTUM	Medium	Medium	Low	Medium
ICON	Low	Medium	Low	Medium
VeChain	Medium	High	Low	Medium
Wanchain	Low	Medium	Low	Medium

4.2.2 Platform Transaction Speed

Another measure that is included in the feature space is the platform transaction speed. For this measure, the block time of a BC was considered. The block time defines the time that it takes to produce a new block in the BC network [47].

Table 4.2: Block time sources

Blockchain	API Source
Bitcoin	BitInfoCharts [7]
Ethereum	BitInfoCharts [7]
Stellar	Blockchair [9]
EOS	EOS Network Monitor [23]
NEO	Neoscan [63]
Cardano	Blockchair [9]
Ripple	Blockchair [9]
QTUM	Qtum.info [73]
ICON	ICON Blockchain Explorer [40]
VeChain	VeChain Explorer [108]
Wanchain	Wanchain Block Explorer [102]

The platform transaction speed has also been grouped into three categories with an ordinal scale: low, medium or high. For the categorization of the public BCs into these categories, the block time was fetched from various external APIs that provide information about the current state of the BC. In Table 4.2, the different sources are listed with the respective

name of the BC. Overall, 40 GET requests were sent to these endpoints, and the average block time was computed based on these values, which are depicted in Table 4.3. Finally, the platform transaction speed was determined based on the percentiles. Numbers below the 33rd percentile (5.30 seconds) were assigned a *high*, numbers between the 33rd and 66th percentile a *medium*, and numbers above the 66th percentile (19.10 seconds) a *low* transaction speed. The average block time and the corresponding percentiles are illustrated in Figure 4.6.

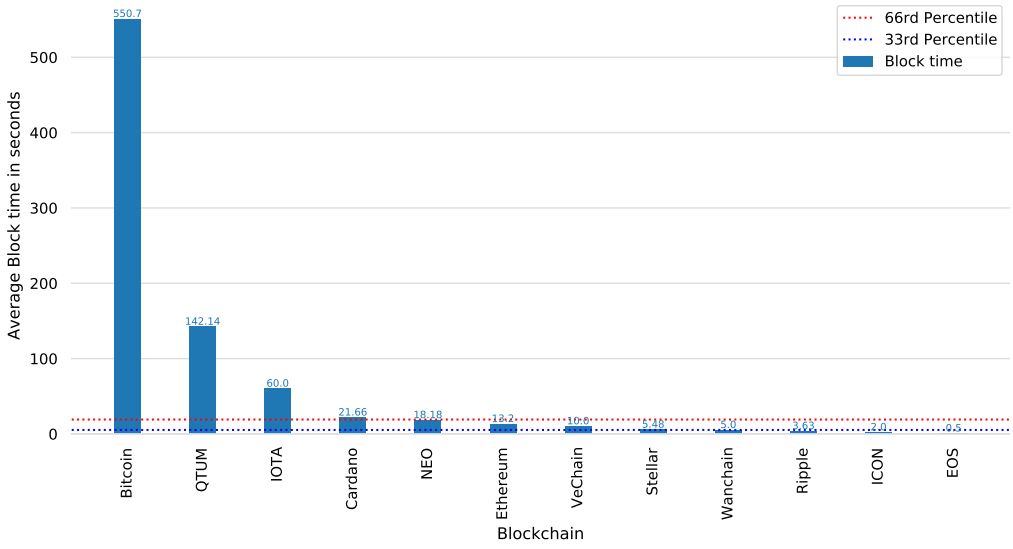


Figure 4.6: Average block time

For private BC, such as Hyperledger, Multichain, Corda or Stratis, which are highly configurable in terms of consensus mechanism and transactions rules, the block time can not be determined *per se*, as it depends on how the consensus mechanism is configured. But due to the limited amount of network participants in private BCs, it comes at no surprise that they have higher throughput and can process more transactions than public BCs. In [69], for example, a performance analysis of Ethereum and Hyperledger has been performed, where the authors measured the execution time, latency and throughput of both BCs in varying scenarios. They observed that Hyperledger outperformed Ethereum across all scenarios. Therefore, all supported private BCs were assigned a high platform transaction speed. The final classification is presented in Table 4.3.

Table 4.3: Platform Transaction Speed categorization

Blockchain	Average Block Time [s]	Platform Transaction Speed
Bitcoin	550.71	Low
Ethereum	13.20	Medium
Stellar	5.48	Medium
EOS	0.5	High
IOTA	60	Low
Hyperledger	-	High
Multichain	-	High
Corda	-	High
Stratis	-	High
NEO	18.18	Medium
Cardano	21.66	Low
Ripple	3.63	High
QTUM	142.14	Low
ICON	2.00	High
VeChain	10.00	Medium
Wanchain	5.00	High

4.2.3 Other features

In addition to the Popularity measure and the Platform Transaction Speed described in previous sections, the following features were included in the dataset.

BC Type

This attribute describes the type of the BC, which can be either *public* or *private*, which are also referred to as permissionless and permissioned, respectively. The type is classified based on its write and read access. It determines the accessibility of the network. In public BCs, access is not restricted and anyone can participate. They are decentralized and completely distributed, which is why there is no single entity that controls the network. Moreover, the data on the chain is publicly accessible and can be read by anyone. In private BCs, only authorized entities are allowed to participate and control the network. Consequently, they are not entirely decentralized. In the dataset, 12 public and four private BC have been included.

Smart Contract Support

This feature indicates whether or not the BC platform supports SCs. SCs are digital self-enforcing contracts/agreements in the form of computer code, typically deployed on and secured by BC. This enables a tamper-proof contract, with which network participants can agree to interact with each other without the need of a trusted third party. When the

predefined triggering conditions of the SC are met, the contract is automatically executed and submitted in the transaction, which is broadcasted to the network [8].

Turing Completeness

This feature specifies if the BC implementation is turing complete and supports complex SC. Bitcoin was the first cryptocurrency that supported SCss. But its scripting language is very limited and not turing complete, which is why complex logic cannot be designed [8]. However, in certain use cases, users wish to implement complex SCs on the BC and therefore require turing completeness.

Data Size

This feature indicates how much data (in Bytes) the BC supports in a single transaction. Transaction focused BCs (such as Bitcoin) are not intended to store a large amount of data, whereas SC-based BCs usually allow a higher amount.

4.3 Features Summary and Dataset

Table 4.4 summarizes all features, including their values and description. In addition, the 16 BCs associated with their corresponding features are presented in Table 4.5, which further served as the dataset.

Table 4.4: Overview of model features

Feature	Values	Description
Type	Public Private	The type of the BC
Smart Contracts	Yes No	Whether the BC supports Smart Contracts.
Turing Completeness	Yes No	Whether the platform supports Turing Completeness.
Platform Transaction Speed	Low Medium High	Indication on how fast transaction are settled on the BC
Popularity	Low Medium High	The popularity of the BC platform
Data Size	Data size values	Supported data size in one transaction

Table 4.5: Dataset

Name	Type	Smart Contracts	Turing Completeness	Platform Transaction Speed	Popularity	Data Size (in Bytes)
Bitcoin	Public	No	No	Low	High	80
Ethereum	Public	Yes	Yes	Medium	High	46000
Stellar	Public	Yes	No	Medium	High	28
EOS	Public	Yes	Yes	High	Medium	256
IOTA	Public	No	No	Low	Medium	1300
Hyperledger	Private	Yes	Yes	High	High	20
Multichain	Private	No	No	High	Medium	80
R3 Corda	Private	Yes	Yes	High	Low	100
Stratis	Private	Yes	No	High	Low	40
NEO	Public	Yes	Yes	Medium	Medium	255
Cardano	Public	No	No	Low	Medium	500
Ripple	Public	No	No	High	High	150
QTUM	Public	Yes	Yes	Low	Medium	2000
ICON	Public	Yes	No	High	Medium	512000
VeChain	Public	Yes	No	Medium	Medium	150
Wanchain	Public	Yes	Yes	High	Medium	150

Chapter 5

Implementation

This chapter describes all the steps necessary for the model development and deployment as well as the integration into PleBeuS. Section 5.1 discusses the data processing and model engineering. Section 5.2 contains a detailed description of how the models were deployed. The integration of the ML-based BC selection solution into the different components of PleBeuS and their joint use is described in Section 5.3.

5.1 ML Model Implementations

After the features were gathered and determined for all supported BCs (see Section 4.2), the resulting dataset was stored within two separate relational tables in an SQLite [94] database, named *blockchains_for_dataset* and *attributes_for_dataset*, that are connected by an attribute called *blockchain_id*. SQLite is an open-source SQL database engine, which does not require a separate server to operate. It is an extremely light-weighted database and saves the database in a single file. Additionally, SQLite is built-in as a Python library and provides a simple and intuitive API. The models were primarily developed with Python and Jupyter Notebook, relying on Scikit-learn [85] for the ML components. Scikit-learn is a popular Python framework for Data Science and ML. It provides a variety of ML algorithms covering the most essential areas of ML, such as classification, clustering, regression. Moreover, it has an easy-to-use API, and its code design patterns have been widely adopted [81].

Following the ML workflow described in Figure 4.1, the Data Acquisition phase was completed with the data collection, including the identification and generation of the features. This is followed by the Data Processing and Model Engineering stage of the workflow, which will be explained in the following sections. Section 5.1.1 discusses the preparation and the further processing of the dataset. The model training is discussed in Section 5.1.2.

5.1.1 Data Processing

The initial dataset contains exactly one row for each BC, which expresses the properties (features). There is one numeric feature in the data, which is the data size that indicates the amount of data each BC is able to store. The DT classifier, which was trained based on this dataset, showed up some flaws. As the data size is one of the decision nodes, where tests on input variables are performed, this node uses relational operators (e.g., less than or equals) for the splitting criteria. Thus in the case of the less than or equal operator, a suitable BC with a higher data size than specified and desired by the user will be disregarded when the algorithm branches towards the appropriate child node.

To account for the aforementioned issue, the dataset was extended. Each BC's data size has been compared to the data size of all other BCs, and if the size was greater than the size of the compared BC, the row was duplicated, and the data size of the duplicated row was set to the lower value. Stratis, *e.g.,* allows storing an amount of 40 bytes in a transaction. This value is compared against all data sizes from every other BC. In this case, it is only greater than the data size of Stellar and Hyperledger, which allow 28 bytes and 20 bytes, respectively. Accordingly, the row which describes Stratis' properties was duplicated twice, and the data size was set to 28 bytes and 20 bytes, respectively, leaving all other variables unchanged.

Following this procedure, the dataset increased from 16 to 107 rows. As a consequence, the resulting dataset became imbalanced, and BCs with a relatively small supported data size were underrepresented in the dataset. The class distribution is depicted in Figure 5.1. To cope with this imbalance, an approach called *Random Oversampling* [33] was applied, where minority classes are oversampled in order to rebalance the dataset. This approach randomly duplicates examples from the minority class. This was done using Python's package called *Imbalanced learn* [44], which offers a number of re-sampling techniques.

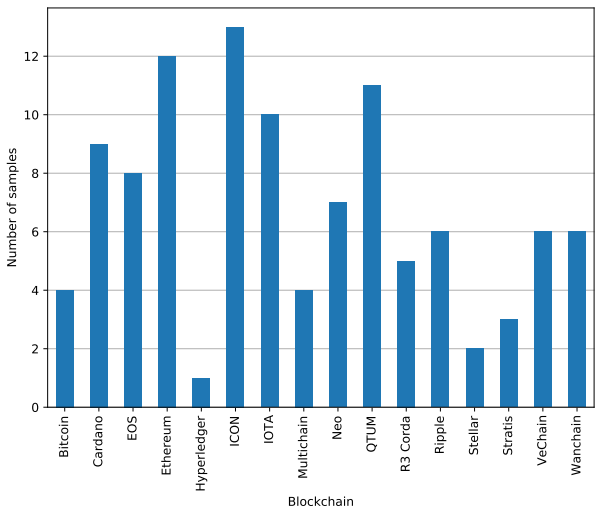


Figure 5.1: Distribution of classes after extension

Moreover, the data was further processed and transformed into a numeric representation, as many ML algorithms require numeric inputs [34] and can not directly operate on

categorical or label data. There are various techniques to encode categorical variables, such as one-hot encoding or Integer/Label Encoding. Integer/Label Encoding replaces categories by assigning numerical labels to each category ranging from 0 to $n-1$, where n is the number of distinct categories of the variable. This was performed for the following categorical features: *Type, Smart Contracts, Turing Completeness, Platform Transaction Speed* and *Popularity*. For this purpose, dictionaries were created, having the category as the key and the corresponding number for the category as the value of the dictionary, as shown in Listing 5.1.

```
1 {"Private": 0, "Public": 1} => Type
2 {"No": 0, "Yes": 1} => Smart Contracts, Turing Completeness
3 {"Low": 1, "Medium": 2, "High": 3} => Platform Transaction Speed, Popularity
```

Listing 5.1: Dictionaries for Integer Encoding

The BC labels were encoded using the `LabelEncoder` provided by Scikit-learn, an efficient tool to encode target labels [68]. Applying the `LabelEncoder` to BC labels led to an assignment of values ranging from 0 to 15 to each label, which can be seen in Figure 5.2.

	blockchain	label_encoded
0	Bitcoin	0
1	Ethereum	3
2	Stellar	12
3	EOS	2
4	IOTA	6
5	Hyperledger	4
6	Multichain	7
7	R3 Corda	10
8	Stratis	13
9	Neo	8
10	Cardano	1
11	Ripple	11
12	QTUM	9
13	ICON	5
14	VeChain	14
15	Wanchain	15

Figure 5.2: Encoded target values

After completing the data preparation, the resulting dataset was used for the next stage of the workflow, the Model Engineering. The first sixteen lines of the final dataset are presented in Figure 5.3.

5.1.2 Model Engineering

Four different ML algorithms were selected for model building: DT, RF, NB and SVM, introduced in Section 2.3. These models have been trained for sixteen different BC implementations, presented in Section 2.1.

	blockchain	type	smart_contract	turing_complete	platform_transaction_speed	popularity	MinArbitraryData
0	Bitcoin	1	0	0	1	3	80
1	Ethereum	1	1	1	2	3	46000
2	Stellar	1	1	0	2	3	28
3	EOS	1	1	1	3	2	256
4	IOTA	1	0	0	1	2	1300
5	Hyperledger	0	1	1	3	3	20
6	Multichain	0	0	0	3	2	80
7	R3 Corda	0	1	1	3	1	100
8	Stratis	0	1	0	3	1	40
9	Neo	1	1	1	2	2	255
10	Cardano	1	0	0	1	2	500
11	Ripple	1	0	0	3	3	150
12	QTUM	1	1	1	1	2	2000
13	ICON	1	1	0	3	2	512000
14	VeChain	1	1	0	2	2	150
15	Wanchain	1	1	1	3	2	150

Figure 5.3: Processed dataset

In a first step, the models were trained by providing them with the relevant features and their corresponding target labels for some observations. Followed by the assessment of the models, where only the features were provided to the models, expecting them to predict the respective labels. Therefore, the data was split into a training and a test set, where the training set was used to train the model and the test set to measure the performance of the model [34].

There are different tools provided by ML libraries, which are dedicated to split the data into training and test sets. For example, Scikit-learn's `train_test_split` method divides the data into two parts according to a specified partitioning ratio. Another widely employed method called *cross-validation* separates the dataset into n splits, where $n-1$ split is utilized for training and the remaining split for testing. In this approach, the model runs n times through the complete dataset, using a different split for testing at each time. Moreover, K-Fold is a popular method to split the data in cross-validation; it divides the data into k groups. K-Fold cross-Validation is generally known to result in a less biased estimate of the model than other approaches such as the train/test split [11]. For model training the `train_test_split` method was used. Whereas a K-Fold cross-validation was used as part of the model evaluation. The detailed comparison of the model performances is discussed in Section 6.1.

Decision Tree Model Training

Building a DT can be achieved with the help of the `DecisionTreeClassifier` algorithm provided by Scikit-learn [87]. After the import, the DT model is instantiated with the default parameters, *i.e.,* criterion parameter set to 'gini' and fitted (Scikit-learn's name for training) on the data attributes and labels. The DT can be visualized as a flow chart, which makes the decision process easy to understand and interpret. Figure 5.4 serves as an illustration of the trained DT classifier. Due to the large size of the DT when it is trained on the entire feature space, the data size variable was omitted for the purpose of this illustration. However, the complete DT can be viewed in Appendix A.

Each node in the DT represents a test case for some attribute/feature, and each edge descending from the node corresponds to the possible answers to the test case. The topmost node, also referred to as the root node, is the starting point, where the DT evaluates the variable that best splits the data. The intermediate nodes also evaluate variables but do not represent the final nodes, so-called leave nodes, where the predictions of the classes are made. The root node in the DT shown in Figure 5.4 starts with all sixteen data points. In this node, the feature which best split the different classes is the *Turing Completeness*. This results in two intermediate nodes, where the *popularity* variable is further examined, which leads to two other children nodes for each node. This partition process continues until no further separation is made, and eventually, a leaf node is reached.

To classify a new example, one can move down the DT, using the features of the example to answer the questions at all decision nodes until a leaf node is reached, where the class will be the prediction.

Random Forest Model Training

Firstly, the RF classifier `RandomForestClassifier` was imported from Scikit-learn [89]. Then, the model is instantiated with the default parameters, *i.e., n_estimators* set to hundred, which defines the number of trees in the forest. Increasing the number of estimators did not further improve the accuracy of the model. Finally, after the model is fitted on the dataset, its performance is further analyzed and evaluated, which is discussed in Chapter 6.

Support Vector Machine Model Training

For the SVM model, the Support Vector Classification, `SVC` from Scikit-learn was imported [88]. The model was built with different kernel functions, *e.g.,* linear, polynomial, radial and sigmoid. Additionally, the type was specified to *one-versus-one*. The penalty term C was set to its default value, *i.e.,* 1, as increasing or decreasing the value did not further increase the performance of the algorithm. The performance measure *accuracy* was used to evaluate the performances of the different kernels.

Table 5.1: Accuracy of different kernel functions

Kernel	**Accuracy**
Linear	0.920635
Polynomial	0.031460
Radial	0.666667
Sigmoid	0.046190

The accuracy measure is depicted in Table 5.1 and revealed that the linear kernel was the best kernel function for this dataset, which is why it has been used for the final SVM model. Further evaluation of the model is discussed in Chapter 6.

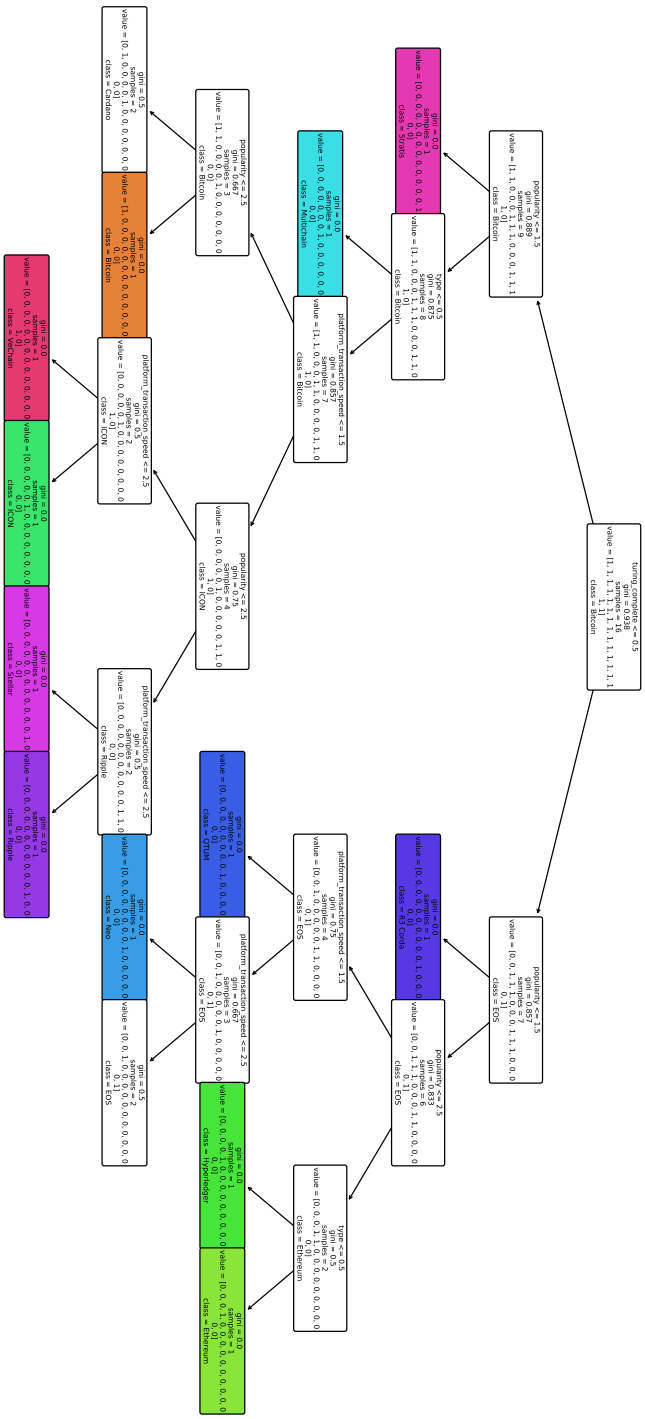


Figure 5.4: Trained decision tree as a flowchart diagram

Naïve Bayes Model Training

For the implementation of the NB classifier, the `MultinomialNB` from Scikit-learn is imported [90], instantiated and fitted to the data. Once the model was trained, further tests and evaluations were performed to measure its applicability to the BC selection process, which is described in Chapter 6.

5.2 ML-based BC Selection Solution Implementation

To utilize the trained models in the BC selection process, it is crucial to offer endpoints that make them available to other services. Such endpoints are supported via an API that adheres to the REST guidelines, as mentioned in Section 4.1. The API provides predictions through the trained ML models. This enables the solution to work as a standalone software component and let it interact with other software components, such as the PleBeuS framework. The main reasoning for this decision is that it allows more flexibility in terms of internal layout or using different programming languages and the opportunity to re-use components and integrate them into other software.

The API is implemented using the Python web framework *Flask* [28] and *Flask RESTful* [12], an extension to Flask that adds support for building REST APIs. Further, *joblib* [46] to save and load the models and label encoder as well as *numpy* [65] to handle input and output data. Listing 5.2 shows the necessary libraries import for this project.

```
1 # Import required libraries
2 import os
3 from flask import Flask, jsonify, request, render_template
4 from flask_restful import Api, Resource, reqparse
5 from model.trainModels import train_models
6 import joblib
7 import numpy as np
```

Listing 5.2: Imports

The first step for building the Flask server was to set up an instance of a Flask app and create an API reference using the flask_restful `Api`. Additionally, the trained models and the label encoder are loaded, as depicted in Listing 5.3.

```
1  # Instatiate flask server
2  app = Flask(__name__)
3  api = Api(app)
4
5  # Decision Tree
6  dt_model = joblib.load('decision-tree.model')
7  # Random Forest
8  rf_model = joblib.load('random-forest.model')
9  # Support Vector Machine
10 svm_model = joblib.load('decision-tree.model')
11 # Naive Bayes
12 nb_model = joblib.load('naive-bayes.model')
13
14 label_encoder = joblib.load('label_encoder.joblib'
```

Listing 5.3: Flask application

In a second step, the route handler had to be defined, which contains the logic and interaction of receiving requests and the response/prediction that is sent back. Therefore, the `MakePrediction` resource class is created, which is responsible for the prediction of the BC. This class is shown in Listing 5.4. It is a subclass of the *Flask-RESTful* class `Resource`, which is used to map HTTP methods to objects. The declared post method within the class enables users to send a body with the API parameters. The body of the request will be parsed, and the arguments are fetched using the *Flask_RESTful's* request parsing interface that provides access to variables. For that, the expected arguments from the JSON input are added with the `add_argument()` method and parsed into a dictionary. Then, the necessary variables, *i.e.,* model, type, smart_contract, turing_complete, transaction_speed, popularity and data_size are taken and stored in the `to_predict_list` variable, which is used as the input for the `blockchain_predictor` method, where the loaded model is applied and the prediction is made. Upon receiving the prediction of the model, a JSON response is generated and returned. Finally, the `add_resource` function registers the route with the framework using the `/api/predict` endpoint where users will be able to access and send requests to.

```
1  class MakePrediction(Resource):
2      @staticmethod
3      def post():
4      # Define parser
5      parser = reqparse.RequestParser()
6      parser.add_argument('model')
7      parser.add_argument('type')
8      parser.add_argument('smart_contract')
9      parser.add_argument('turing_complete')
10     parser.add_argument('transaction_speed')
11     parser.add_argument('popularity')
12     parser.add_argument('data_size')
13     # Creates dictionary
14     args = parser.parse_args()
15     # Convert input to list
16     to_predict_list = [args['model'], args['type'], args['smart_contract'],
17                        args['turing_complete'], args['transaction_speed'],
18                        args['popularity'], args['data_size']]
19     prediction = blockchain_predictor(to_predict_list)
20     return jsonify({'name': prediction})
21
22     api.add_resource(MakePrediction, '/api/predict')
```

Listing 5.4: Prediction endpoint

The `blockchain_predictor` method is shown in Listing 5.5. It takes the list with the variables as input and converts it into *numpy* array to be fit into the model. The first argument of the list defines what model was selected to be used for the prediction. Dependent on what this argument is, the respective model is used to predict the BC. During training, the target values have been transformed with the Label Encoder, which assigned an integer to each label. Consequently, the returned prediction will be an integer as well. Therefore, the saved *label_encoder* is used to reconvert the categorical variables from numeric back into the respective categories, *i.e.,* the BC name.

```
def blockchain_predictor(to_predict_list):
    #Convert input to array
    features = [np.array(to_predict_list[1:])]
    #Extract chosen model
    chosen_model = to_predict_list[0]
    #Get prediction from chosen model
    if chosen_model == 'decision_tree':
        prediction = dt_model.predict(features)
    elif chosen_model == 'random_forest':
        prediction = rf_model.predict(features)
    elif tchosen_model == 'support_vector_machine':
        prediction = svm_model.predict(features)
    else:
        prediction = nb_model.predict(features)
    #Inverse transform to get the original dependent value
    prediction_decoded = label_encoder.inverse_transform(prediction)
    return: prediction_decoded[0]
```

Listing 5.5: Blockchain predictor

When sending a post request to the prediction endpoint, it is worth noting that the argument values that are required for the prediction need to be numeric. This is due to the models being trained on a numerical representation of the categorical variables in accordance with Listing 5.1. E.g., the request body shown in Listing 5.6 would correspond to input variables shown in Listing 5.7.

```
{
    "model": "decision_tree",
    "type": 1,
    "smart_contract": 1,
    "turing_complete": 0,
    "transaction_speed": 2,
    "popularity": 3,
    "data_size": 28
}
```

Listing 5.6: Example request body

```
{
    "model": "decision_tree",
    "type": "Public",
    "smart_contract": "Yes",
    "turing_complete": "No",
    "transaction_speed": "Medium",
    "popularity": "High",
    "data_size": 28
}
```

Listing 5.7: Corresponding labels

In addition to the prediction endpoint, the `/api/blockchains` endpoint was added. It returns all available BCs along with their corresponding characteristics. Upon receiving a GET request, the flask server connects to the database and executes a SQL query, which retrieves the necessary attributes from each BC and returns them as a JSON response.

A Graphical User Interface was also built on top of the REST API, which can be used to access the ML models. It provides a form where users can input the data for prediction, which is illustrated in Figure 5.5. The different use cases of the Flask Server can be seen in Figure 5.6. These show two high-level usages; a user can either make use of the ML-based selection algorithms by utilizing the provided GUI (1) or by using the PleBeuS framework (2).

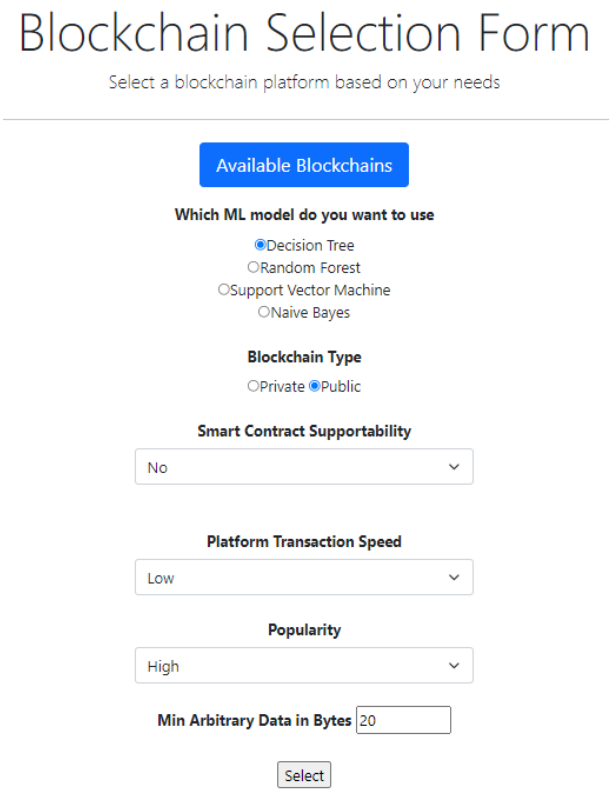


Figure 5.5: Graphical User Interface

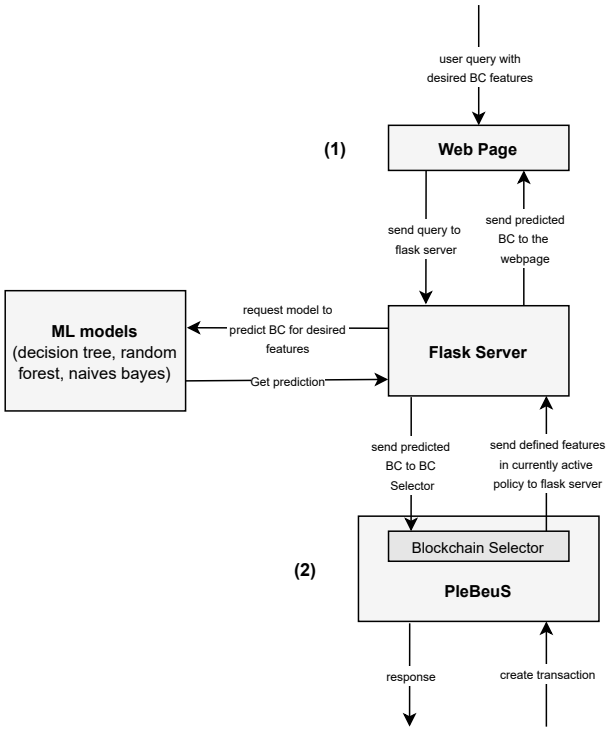


Figure 5.6: Prototype usage

5.3 Integration into PleBeuS

This section describes the two components of the PleBeuS framework in combination with the ML-based BC selection solution and the interaction between the different components. First, PMT is detailed, followed by a description of the Transaction Component.

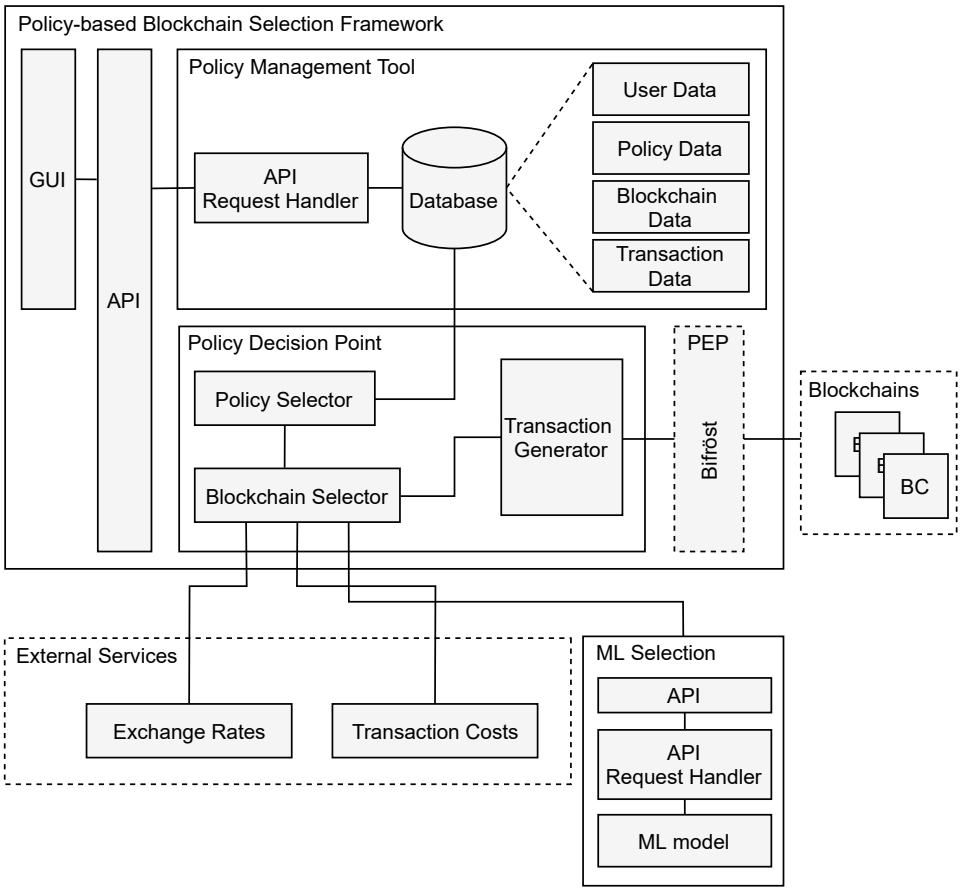


Figure 5.7: Extended PleBeus architecture

The PleBeuS framework can either be used in conjunction with the Blockchain Interoperability API, called Bifröst [83], or as a separate application. The API endpoint, which is used to send data to the framework, also returns a response payload that can be used to extract information about the transaction (shown in Listing 5.8). When the framework is used independently, users can use the response as a suggestion of what BC to use. In general, the framework provides a user-friendly Web interface and allows users to interact with two separate components. It provides a GUI for the management of policies and an API endpoint to execute transactions [84].

```
[
    {
        "username": "TestUser",
        "blockchain": "Ethereum",
        "dataHash": "hashed data using SHA-256"
        "data": "the plain data",
        "cost": 0.01,
        "policyId": "5ce16ca8fb2adb4b443ae2b9 ",
        "costProfile": "performance",
        "interval": "daily"
    }
]
```

Listing 5.8: Example of a transaction payload from PleBeuS [84]

For the integration of the ML-based BC selection solution into PleBeuS, the two core components of the framework had to be adjusted. Firstly, the PMT had to be extended to allow the additional features/parameters to be added to the policies that were not yet supported by the framework. Secondly, the Transaction Component (Policy Decision Point) had to be extended to ultimately allow the framework to use both the regular and the ML-based selection algorithm. The architecture of PleBeuS after the integration is depicted in Figure 5.7.

5.3.1 Policy Management Tool

The PMT is the starting point of the PleBeuS framework. It provides a GUI, where the user can create users and manage policies. This section focuses on the interaction between the user, the GUI and the PMT in conjunction with the ML-based BC selection solution. For that purpose, the different steps involved in the data workflow are described and illustrated with a sequence diagram. The diagram is divided in four sequential steps for illustrative purposes and the sake of clarity. However, the complete sequence diagram is presented in Appendix C. First, the user is prompted to create a user. Upon submission of a username, the PMT checks whether the username already exists; if not, the user is redirected to the policy configuration view, where the default policy can be defined. Otherwise, the user will get an error message (see Figure 5.8).

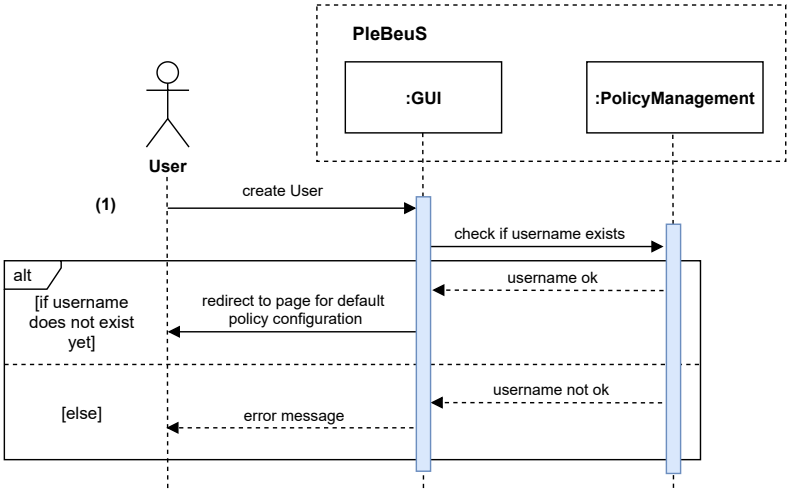


Figure 5.8: Step 1 of PMT workflow

Upon submitting the configured default policy, the input is validated and stored in the database along with the user when a valid input has been passed. Then, the user is redirected to the main view, where additional policies can be created, or existing policies can be edited or deleted. The main view also provides statistics about executed transaction and also indicates which policy is currently active (see Figure 5.9).

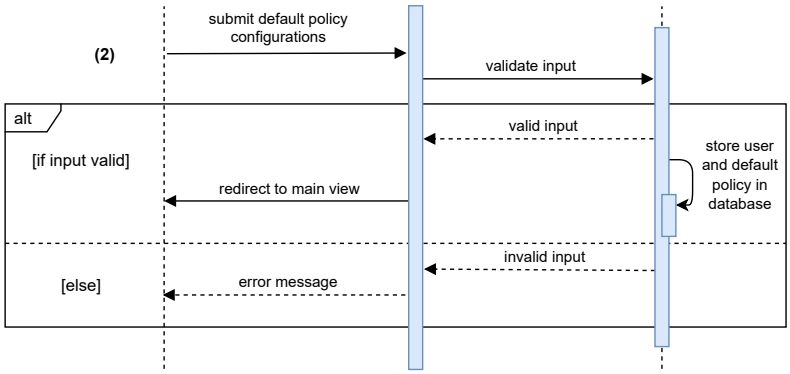


Figure 5.9: Step 2 of PMT workflow

When the user decides to create an additional policy, the user is redirected to the configuration view. A form will be provided, where the user can configure the policy (see Figure 5.10).

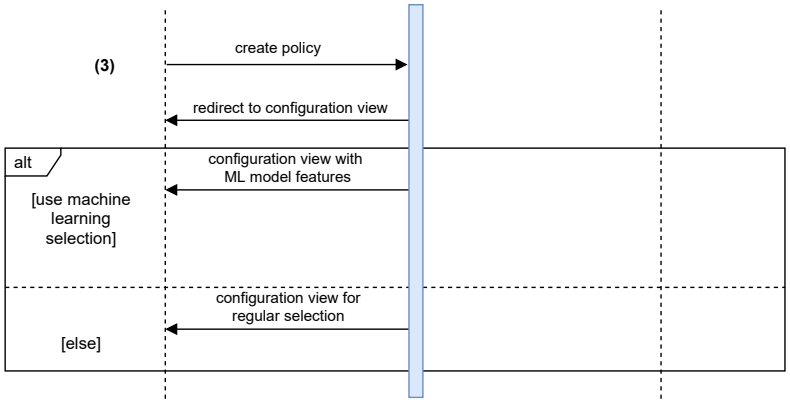


Figure 5.10: Step 3 of PMT workflow

The default form allows the configuration of the policy parameters, described in Section 2.2, which can also be seen in Figure 5.11. To provide the user with the option to choose between the regular and the ML-based selection algorithm, a button *"Use ML selection algorithm"* has been added to the form. Suppose the user wishes to use the ML-based selection algorithm. In that case, the button can be selected, and the form will be extended with the parameters which are required for the model prediction and which are not yet part of the form, namely the preferred *ML model*, *Smart Contracts*, *Platform Transaction Speed* and *Popularity*. Additionally, the form will exclude parameters that are not necessary for the ML-based selection algorithm.

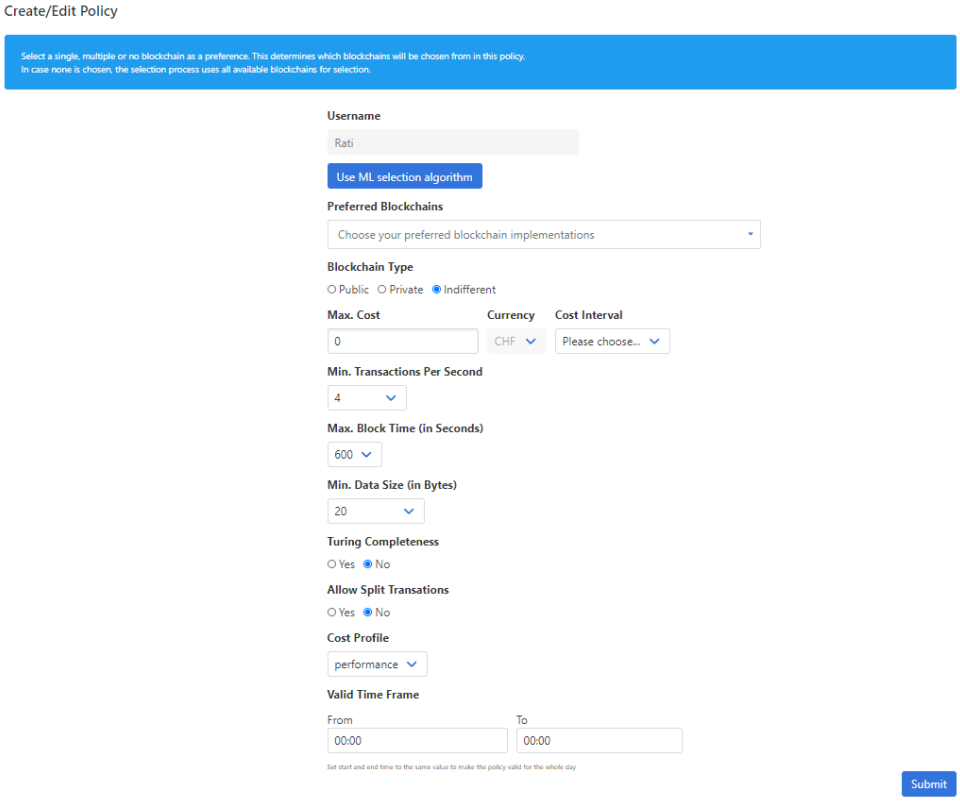


Figure 5.11: Policy configuration view

The excluded parameters are *Preferred Blockchains*, *Min. Transaction Per Second*, *Max. Block Time*, *Allow Split Transaction* and the *Cost Profile*. The rationale behind this exclusion is the following reasons:

(a) The ***preferred BC*** parameter allows the user to select a single or multiple BCs that should be considered for the selection. As the ML models are trained for all BC implementations, it is not possible to narrow down the range of BCs for consideration without training a model for every possible variation of BC set. This would imply an increased overhead, which is avoided by using the existing selection algorithm instead of the ML selection algorithm if the user wishes to make use of this parameter.

(b) The ***Min. Transaction Per Second*** and ***Max. Block Time (in Seconds)*** parameters determine the performance of a BC. There are various factors that influence the performance of a BC [16]. One factor is the transaction throughput, representing the number of transactions a BC network can successfully process per second. [4]. Other factors are the block time, which determines the time/interval that is needed to append a new block on the BC and the block size, which defines how many transactions a block can store. The transaction throughput is a function of these other two factors. The throughput is the result of dividing the block size by the block time. Due to the dependency of the parameters, these parameters are replaced by a single parameter called **"Platform transaction Speed"**, which has been introduced in Section 4.2.

(c) The ***Allow Split Transaction*** defines whether the data should be stored across all valid BCs. Since the ML model will predict exactly one BC for a given input, it is not possible to split the transaction over multiple BCs within one policy. Therefore, this parameter was excluded as well.

(d) The ***Cost Profile*** parameter is used whenever multiple BC implementations fulfill the criteria defined in the policy. There are two options to choose from, a performance or an economic-based cost profile. The former would always select the most performant BC, and the latter would select the BC with the lowest transaction costs. As there is only one BC selected in case the ML selection algorithm is performed, this parameter becomes obsolete, so it is also excluded.

The other policy parameters, *Blockchain Type*, *Max. Cost*, *Min. Data size*, *Turing Completeness* and the *Valid Time Frame* remain selectable in the new provided form. In addition, the new configurable features, *ML model*, *Smart Contracts*, *Platform Transaction Speed* and *Popularity*, will become apparent. The updated view of the policy configuration is illustrated in Figure 5.12.

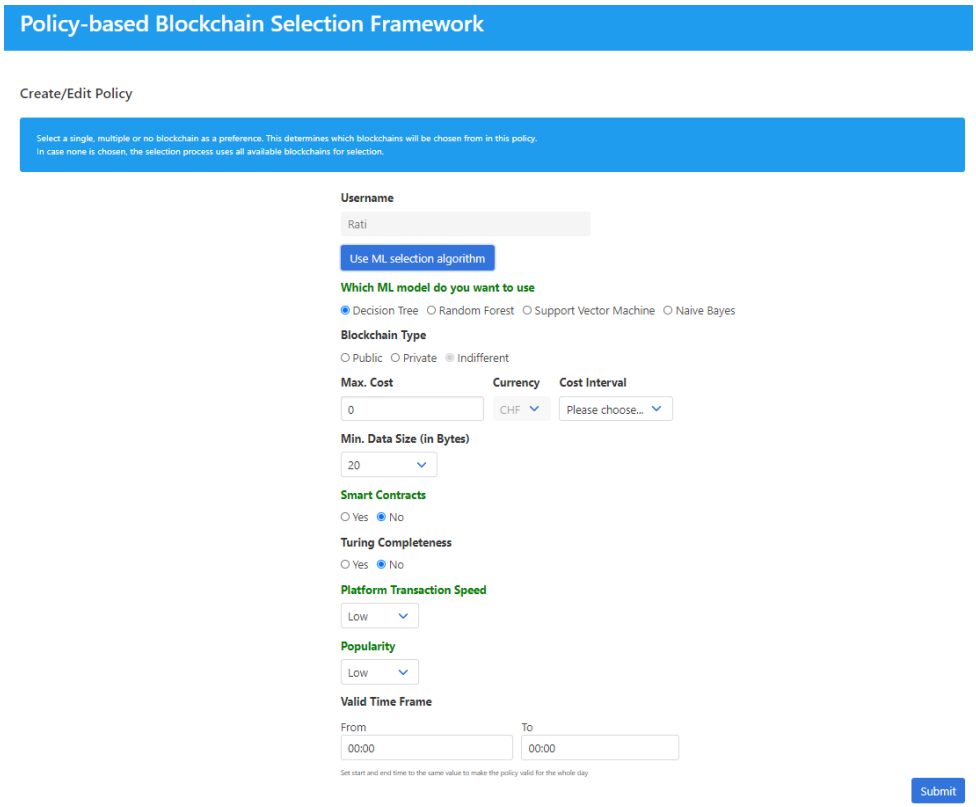


Figure 5.12: ML features configuration view

As soon as the policy parameters have been configured and submitted, all information along with the username is sent to the PMT, which checks if any policy conflicts exist. If there are no conflicts, the user is redirected to the main view again. Otherwise, the user receives an error message stating what the reason for the conflict was (see Figure 5.13).

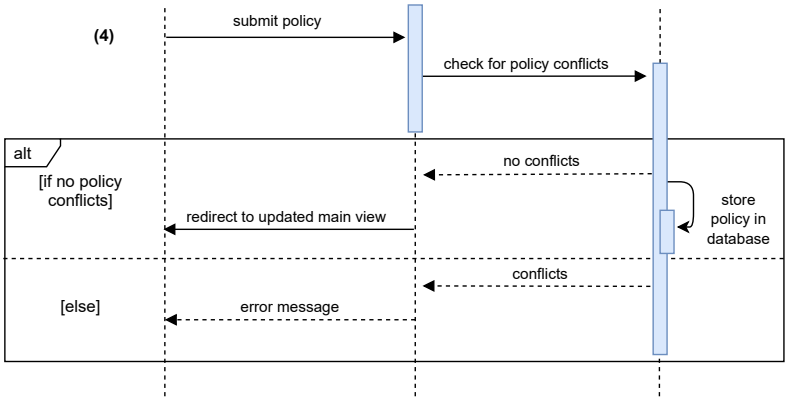


Figure 5.13: Step 4 of PMT workflow

The following events can trigger a policy conflict:

- No BCs meet the criteria defined in the policy in case the regular selection algorithm is selected [52].

- The *Cost Threshold* and *Cost Interval* which belong to the externally driven policy parameters, allow users to set a maximum cost amount they are willing to spend in a particular interval. As soon as the accumulated transaction costs exceed the defined threshold, the currently active policy switches to the next policy with a higher threshold. Therefore, a policy with a higher cost interval must have a higher cost threshold, otherwise a conflict arises [52].

- The *Time Frame* parameter allows users to specify when a policy should be active. An overlap of time frames within the same cost interval is not allowed and will result in a conflict [52].

- When using the ML Service, it is required to choose a preferred type *i.e.,* public or private. Otherwise, when no BC type has been selected, this will also result in a conflict.

From the main view, the user can now create additional policies or edit and delete previously defined ones. When editing policies, the PMT pulls the policy of interest from the database and presents it to the user in the configuration view, with the existing data already filled into the form. Deleting a policy is always possible, except in the case of the default policy, ensuring that the user always has an active policy. Due to the partially different parameters for the ML-based selection algorithm, the main view was updated, as shown in Figure 5.14.

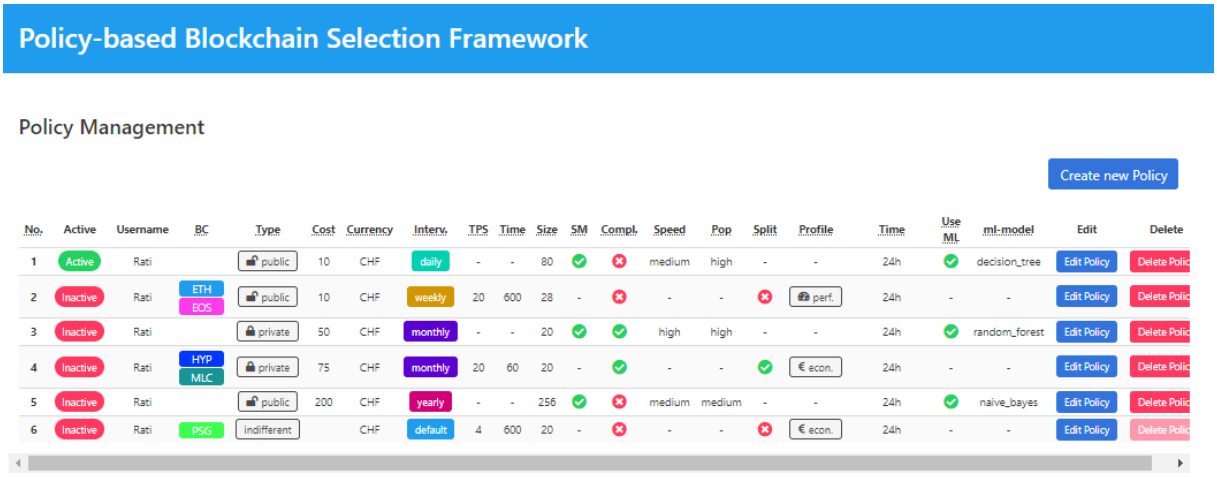


Figure 5.14: Updated main view of PMT

In particular, the updated view includes information on whether or not the ML-based algorithm was chosen for the policy, indicated by a checkmark in the column named *Use ML*. If that is the case, it provides an overview of the selected policy parameters required for the ML selection, *i.e.,* type, cost, currency, interval, data size, smart contracts, turing completeness, transaction speed, popularity, time frame and the ML model. Policy parameters that are not required for the ML selection are left blank (BC, tps, block time, split and the cost profile). The same applies when the regular selection algorithm is chosen, the policy parameters that are used for the ML algorithm are simply left empty in the main view. Additionally, the policy schema was extended with the new parameters in order for the information to reach the Transaction Component workflow.

5.3.2 Transaction Component

The main integration of the ML-based selection functionality occurred in the Transaction Component of the PleBeuS framework. In specific, the BC selection module within the Transaction Component was extended. Rather than using the database to retrieve the BC that conform to the policy, a POST request to the API endpoint `/api/predict` of the Flask application is executed, with the request body containing all information about the policy parameters. The API, in turn, will use the ML model to predict a BC and return it to the BC Selector, which is then used for further processing. Figure 5.15 shows the extended Transaction Component. The interaction between the user, the Transaction Component and the Flask application is explained in the following paragraphs, which is also illustrated in the sequence diagram in Figure 5.16.

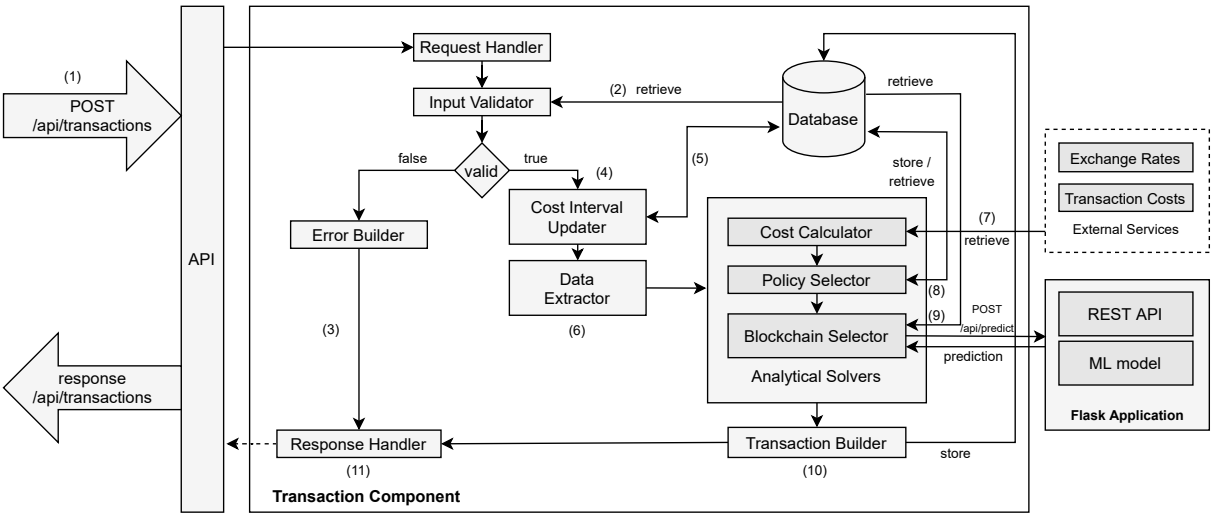


Figure 5.15: Extended Transaction Component workflow

Once the policies have been defined, a transaction can be sent to the `/api/transactions` endpoint. Since PleBeuS focuses on the use case of cold chain monitoring, an Excel file containing sheets with temperature data and the minimum and maximum temperature thresholds needs to be included in the transaction. Alternatively, a string instead of an Excel file and the temperature thresholds can be passed as well [52]. By sending a POST request with this information along with the username in the body of the request, the transaction workflow gets triggered. Upon receiving the request, the Transaction Component checks whether all parameters are valid and if the user is available and already defined at least one policy. If the parameters are valid, the Transaction Component further processes the data. In case the Excel file has been sent, the sheets from the file are parsed, and all the violating data, temperature below the minimum and above the maximum threshold, are extracted from the sheets. Each sheet corresponds to a single transaction and represents a temperature measurement of a delivery [52].

As soon as the transaction costs are calculated, each sheet's violation data is checked step by step. In an initial step, the policy is selected based on the cost threshold. The policy with the lowest time interval within the time frame that has not reached the max costs yet is selected. For a detailed explanation of these steps, refer to [52]. This policy is further examined; if the user had opted for the ML-based selection algorithm, a POST request is sent to the `/api/predict` endpoint, containing the parameters of the policy. The endpoint parses the incoming request and based on which ML model the user has chosen in the policy, the corresponding ML model is used to predict the BC. The API response is then returned to the Transaction Component, where the predicted BC is used for the transaction. If the user opted for the regular selection algorithm in the currently active policy, a BC pool is determined, with all BCs that conform to the policy. Depending on the cost profile of the policy, either the most performant or the cheapest is selected from this pool [52]. Either way, all cost intervals of the user are updated with the calculated transaction costs before the next sheet will go through the same process. If the user passes a string, the process is almost identical, with the exception that the data is not looped through and only one transaction is executed. Finally, for each transaction, the selected BCs are gathered and returned to the user. If the user passes a string for the transaction

instead of an Excel file, the process is essentially the same. The only difference is, that the data is not looped through and therefore, only one transaction is executed [52].

The transaction of the extended framework returns a slightly different response when the ML-based selection is performed, which is shown in Listing 5.9. The ML-based selection process does not require the user to define a Cost Profile, as discussed in the previous section, which is why it has been omitted from the transaction payload. Instead, it includes information about the selected ML model.

```
[
    {
        "username": "TestUser",
        "blockchain": "Stellar",
        "dataHash": "hashed data using SHA-256"
        "data": "the plain data",
        "cost": 0.01,
        "policyId": "5ce16ca8fb2adb4b443ae2b9 ",
        "interval": "daily",
        "mlModel": "naive_bayes"
    }
]
```

Listing 5.9: Example of a transaction payload from the PleBeuS framework when using ML for the selection

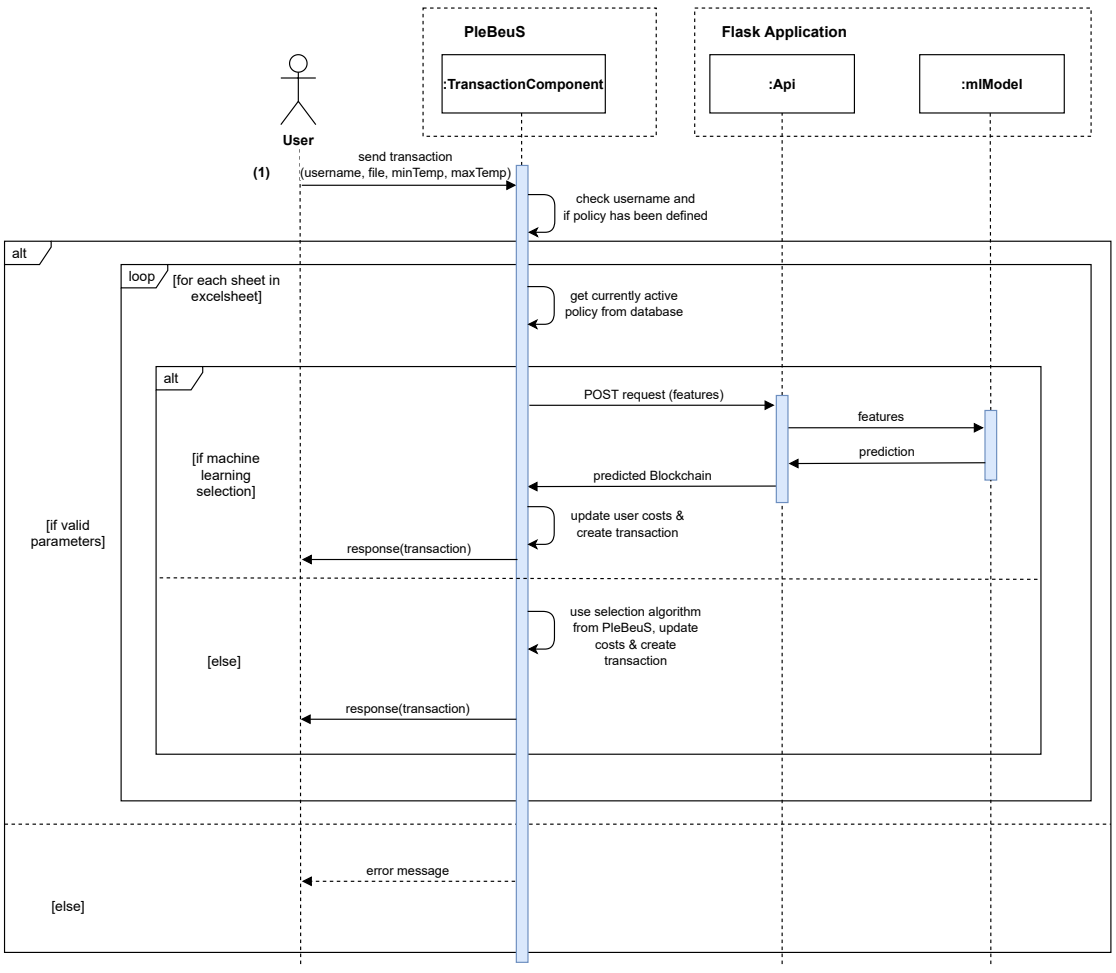


Figure 5.16: Transaction flow

Chapter 6

Evaluation and Discussion

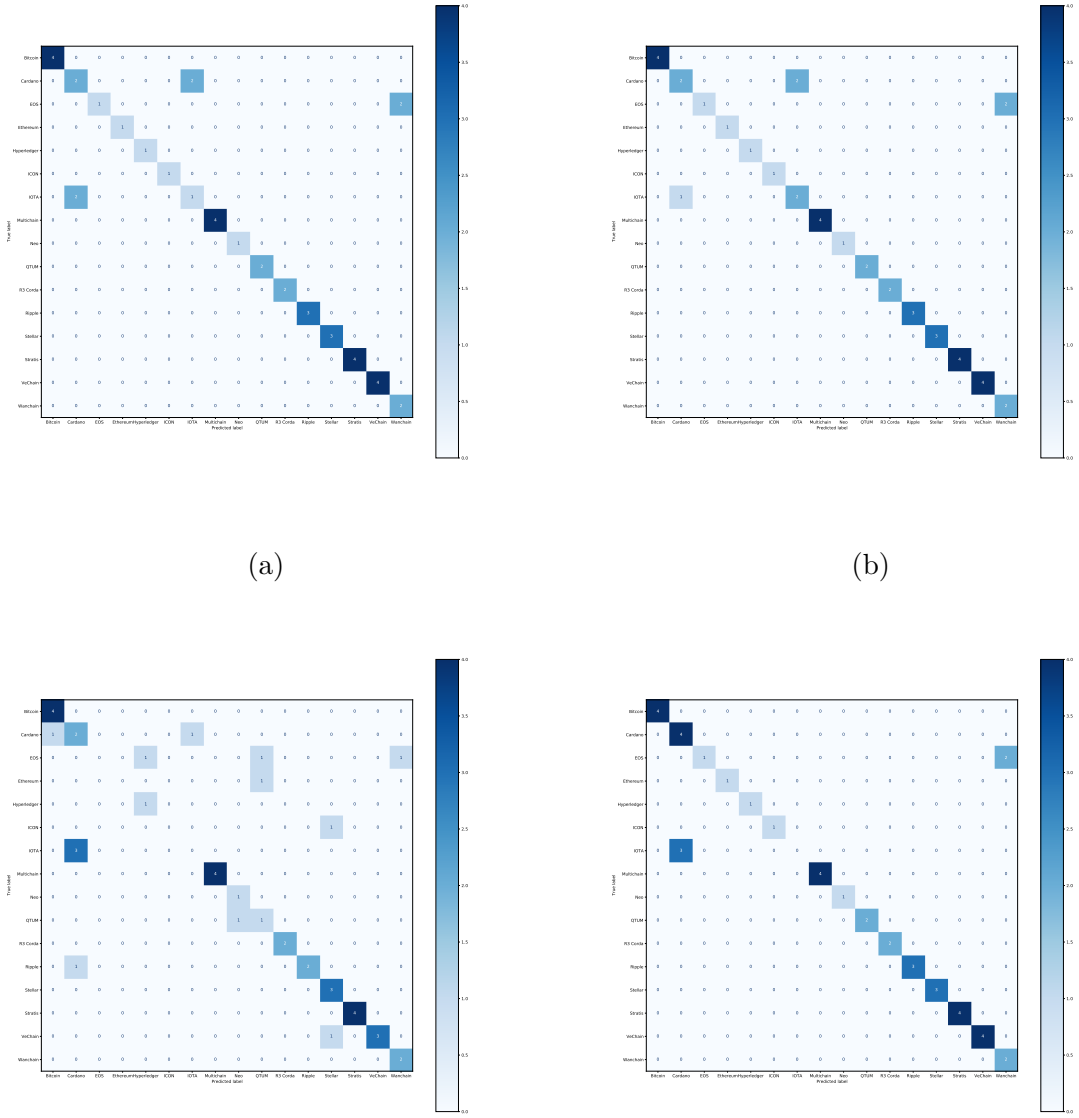
This chapter presents the results of the performance evaluation of the ML algorithms as well as use case scenarios describing and evaluating the practical application of the ML models in regards to the BC selection process. Moreover, it provides the result of the performance testing of the prototype (*i.e.,* the integration of the ML solution into PleBeuS).

All the benchmarks were conducted on the same machine. The system uses an i7-6600U CPU @ 2.60GHz and has 16 GB of RAM. The operating system is Windows 10 Pro. PleBeuS was deployed in a docker container while the ML-based solution was running locally without relying on virtualization.

6.1 Comparison of ML Algorithms

Different ML algorithms have been trained for this project, as described in Section 5.1.2. For evaluation and comparison of these algorithms, several performance metrics were considered. E.g., visualization methods such as confusion matrices were used. Confusion matrices visualize the performance of the models in a tabular way. Each entry indicates the number of predictions that were made and whether or not the model predicted the correct classes. In order to calculate a confusion matrix, the dataset was split into two sets: the *training set* and the *test set*. As the names suggest, the models were trained using the training set and then tested against the test set. While testing, the models are used to predict the class labels (predicted labels), which in turn are compared to the actual labels (true labels). The confusion matrices of the models are illustrated in Figure 6.1. Due to the large size of the matrices, they were down-scaled for the purpose of this illustration. However, the original full-sized matrices can be viewed in Appendix B. The classes are listed in the same order in the rows as in the columns, such that the correctly classified elements are on the main diagonal from top left to bottom right.

In Figure 6.1a, 6.1b and 6.1d, it is observable that the DT, RF and SVM correctly predicted the classes in most cases. In all these cases, some samples belonging to IOTA have



(a) (b)

(c) (d)

Figure 6.1: Confusion matrices. (a) DT (b) RF (c) NB (d) SVM

been predicted with Cardano or vice versa. Similarly, some EOS samples have been classified as Wanchain and vice versa. This is due to these BCs having the same characteristics. IOTA and Cardano, as well as EOS and Wanchain, share the same properties, besides the data size variable, which is apparent on the dataset depicted in Table 4.5 in Section 4.3. Therefore, these classification anomalies are not surprising and ultimately not incorrect. Overall, these models performed with a high accuracy score, which is presented in Table 6.1. The accuracy scores of DT, RF and SVM algorithms are very close to each other, whereby the RF and the SVM models slightly outperformed the DT model by roughly 2%.

The NB model, in contrast, has provided the worst performance. Figure 6.1c shows the respective confusion matrix. The model misclassified the test samples in more than 30% of the cases.

Table 6.1: Performance

ML model	Accuracy Score	Time Performance (Seconds)
Decision Tree	86%	0.074009
Random Forest	88%	4.767519
Naive Bayes	69%	0.079040
Support Vector Machine	88%	48.710479

A possible explanation for the poor performance of the NB model is its sensitiveness to correlated features, as it assumes independence of all attributes, which does not hold in this case. This is evident in the correlation matrix in Figure 6.2, which was created with the help of seaborn's heatmap functionality [113]. There is a high correlation between Turing Completeness and Smart Contract Support and between the popularity of the BC and its type. Furthermore, there is a notable negative correlation between the Transaction Speed of a BC and its type and a positive correlation between the Transaction Speed and the SC Support of a BC.

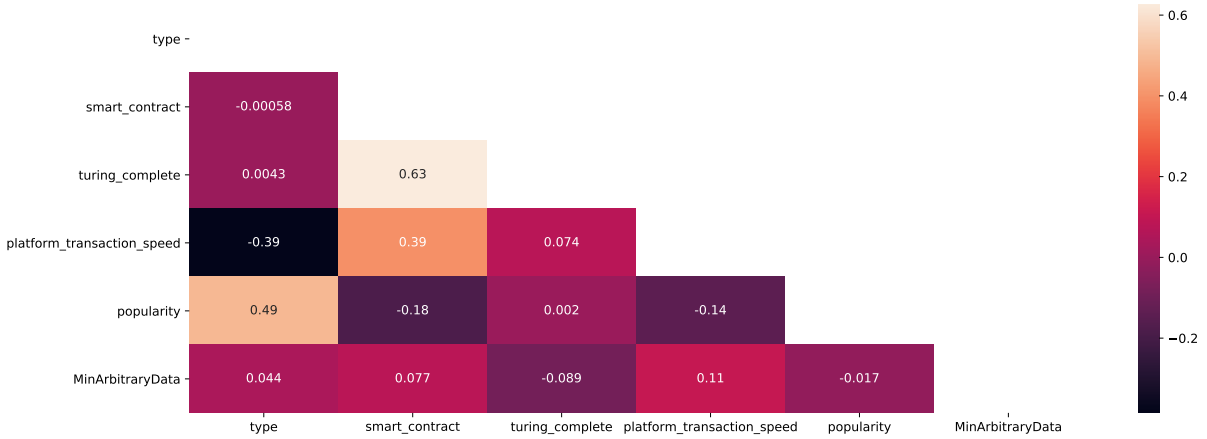


Figure 6.2: Correlation matrix using a heatmap for the variables

Furthermore, the average accuracy and the distribution of model accuracies have been computed. The algorithms have been evaluated on a consistent test harness. Each algorithm was evaluated with the 10-fold cross-validation procedure, which was configured with the same random seed to ensure that the same splits are performed on the data, and each algorithm is being evaluated in the exact same way. The box plot in Figure 6.3 shows the spread of the accuracy scores across each cross-validation fold for each algorithm. This comparison demonstrates yet again, that the DT, RF and SVM models are almost equally accurate on average. It also shows the poor performance of the NB model on this dataset, with a mean accuracy of only 22%.

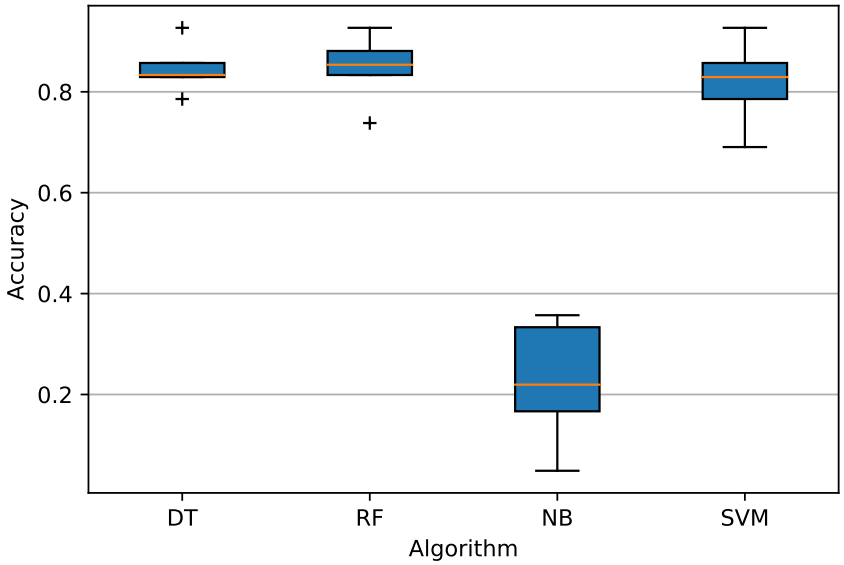


Figure 6.3: Comparison of different ML algorithms

The performance in terms of training time was also evaluated. For this purpose, the models have been trained thirty times on the dataset, and the respective time (in seconds) has been measured for each algorithm. The evaluation results are depicted in Table 6.1. It is noticeable that the DT and the NB classifiers had the fastest training time of around 80 milliseconds. It is obvious that the RF algorithm has a higher training time than a single decision tree, as in each run, 100 decision trees are built for the RF classifier. The SVM classifier had by far the slowest training time, which was around 48 seconds. This is probably due to its high computational complexity. Traditional SVM requires $O(n^2)$ time complexity, which is why it is impractical for large datasets [111]. Nonetheless, considering that the models only have to be trained once, the training time seems to be reasonable for all the applied algorithms.

6.2 Use Case Scenarios

For the evaluation of the ML-based BC selection solution as a separate application, two scenarios were considered. In the first scenario, each model is provided with BC features that exactly match an underlying BC, expecting them to return the correct BC. In the second scenario, BC features that do not precisely correspond to a specific BC are taken as input for the models, expecting them to predict the most appropriate BC given these parameters. These predictions can finally be used as a suggestion and further evaluated by the user as not all desired properties are covered by one of the supported BCs.

6.2.1 Scenario #1

The results of the first scenario are presented in Table 6.2. This table illustrates the prediction made by each of the models and whether or not the prediction of the BC of interest was correct, indicated by a green checkmark (✓), or if the prediction was wrong, indicated by a red cross (✗). It is apparent that the DT and the RF correctly predicted the BCs in all cases, while the SVM model rightly classified the BCs for the most part, except in the cases of Cardano and Wanchain, where it predicted IOTA and EOS, respectively. However, as discussed in the previous section, this classification is eventually not wrong because these BC share the same defined properties and fit the input parameters.

In contrast to the good generalization capabilities of the DT, RF and SVM model, the NB model achieved bad results once more. It incorrectly classified the instances in half of the cases, which was highly anticipated, considering the low accuracy score of the NB model.

Table 6.2: Predicted BCs given Scenario 1

Blockchain	**ML Models**			
	DT	**RF**	**NB**	**SVM**
Bitcoin	✓	✓	✗	✓
Ethereum	✓	✓	✗	✓
Stellar	✓	✓	✓	✓
EOS	✓	✓	✗	✓
IOTA	✓	✓	✗	✓
Hyperledger	✓	✓	✓	✓
Multichain	✓	✓	✗	✓
R3 Corda	✓	✓	✗	✓
Stratis	✓	✓	✓	✓
Neo	✓	✓	✗	✓
Cardano	✓	✓	IOTA	IOTA
Ripple	✓	✓	✓	✓
QTUM	✓	✓	✗	✓
ICON	✓	✓	✓	✓
VeChain	✓	✓	✓	✓
Wanchain	✓	✓	EOS	EOS

6.2.2 Scenario #2

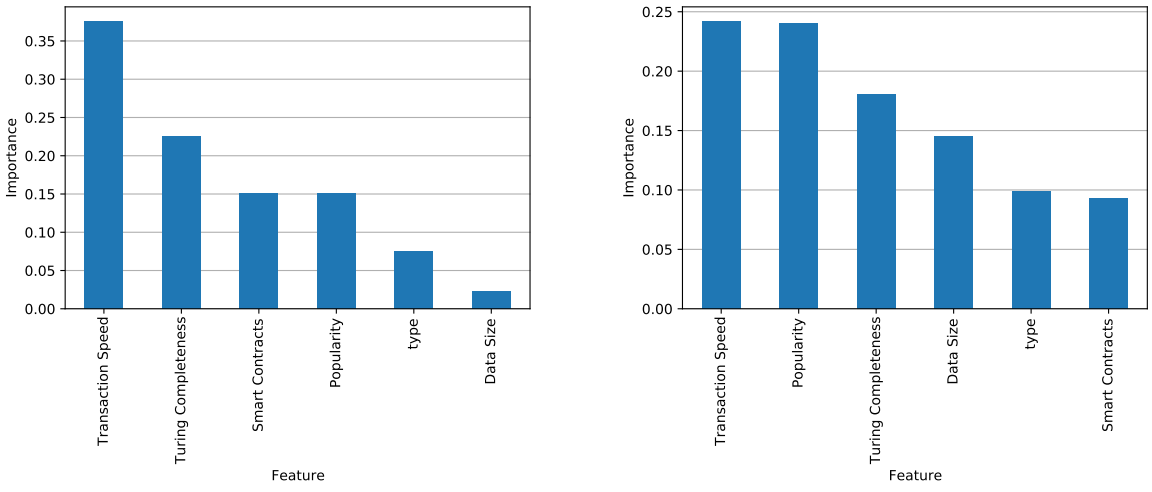
For the second scenario, BC characteristics were chosen that do not fully meet the properties of one of the supported BCs. This was done to evaluate if the models' predictions are reasonable and if they finally could serve as user recommendation. It is worth mentioning that the regular selection algorithm in PleBeuS does not offer such a flexible recommendation mechanism, as the user would be prompted with an error message, stating that no BC with provided parameters is available. For this scenario, the NB model was omitted

due to its inadequacy and the poor quality of its predictions. Overall, five cases were defined, which are then passed as input to the models, which in turn make predictions based on these values. These test cases, along with the corresponding prediction of each model, are illustrated in Table 6.3.

Table 6.3: Test cases for scenario 2

No.	Type	Smart Contracts	Turing Completeness	Platform Transaction Speed	Popularity	Data Size	Prediction		
							DT	RF	SVM
1	Public	Yes	Yes	High	High	250	EOS	EOS	EOS
2	Public	Yes	No	High	High	20	ICON	Stellar	ICON
3	Public	No	No	Low	Medium	2000	IOTA	IOTA	IOTA
4	Private	Yes	Yes	High	Medium	100	R3 Corda	R3 Corda	Wanchain
5	Private	No	No	Medium	Low	20	Multichain	Multichain	Stratis

In case No. 1, four different BC implementations (Ethereum, EOS, Neo and QTUM) would be eligible if only the type, SC, turing completeness, and supported data size variables are considered. However, none of these BCs has both a high transaction speed and high popularity. The predicted BC was the same across all models, namely EOS, which covers all variables except for having a medium popularity score. Neither of the models chose Neo or QTUM because neither of them has a high transaction speed nor high popularity. The other option would have been Ethereum, which has high popularity but medium transaction speed. To comprehend why the models favoured EOS over Ethereum, the feature importance of the DT and the RF model was computed, which indicates the relative importance of each feature towards the output variable. The DT and RF model both provide a built-in *feature_importances_* property that can be accessed to retrieve the relative importance scores of the features, which are illustrated in Figure 6.4. It is evident that the transaction speed is the top feature contributing to the prediction in both models, thus favouring BCs with a matching transaction speed.



(a) DT (b) RF

Figure 6.4: Feature Importance

In case No. 2, three BCs qualify for selection (Stellar, ICON and VeChain) when considering all parameters except the transaction speed and the popularity score. The RF

model predicted Stellar, thus, this time weighting the popularity more than the transaction speed, as Stellar has a high popularity but medium transaction speed. This could be because the importance of transaction speed and popularity in the case of the RF is almost equally high. On the other hand, the DT and the SVM models predicted ICON, which has a high transaction speed but medium popularity. None of the models predicted VeChain because it neither has a high transaction speed nor high popularity.

In case No. 3, the input variables are chosen in such a way that two BCs, IOTA and Cardano, would qualify for selection except that they both don't support 2000 bytes in one transaction. All models returned IOTA, which is presumably due to IOTA's higher data size support of 1300 bytes, compared to Cardano's 500 bytes. Generally, if the defined input corresponds to one specific BC apart from the data size, that BC is selected nevertheless, which might be a disadvantage in some cases, *e.g.,* when the specified data size is inevitable for the user.

In case No. 4 and 5, predictions of private BCs were the main focus of the analysis. The only private BC that supports a data size of 100 bytes, as defined in case No. 4, is Corda. However, it does have low popularity in opposition to the medium popularity specified in the input. In this context, the DT and RF predicted Corda as the most suitable BC, whereas the SVM returned Wanchain, which conforms with all input variables besides the BC type. In consequence, it is possible that a model will output a public instead of a desired private BC, which could potentially be a significant drawback, especially when the sensitivity of the data stored on the BC is crucial to the user.

In case No. 5, all models correctly predicted a private BC. The DT and RF returned Multichain as the most fitting BC, while the SVM forecasted Stratis. Multichain satisfies the SC and turing completeness constraint but provides a higher transaction speed and popularity than defined in the input values. Whereas Stratis does not fulfill the smart contract constraint but meets the low popularity score and also offers a higher transaction speed.

6.3 Performance Testing

For the evaluation of BC selection of the ML models compared to the regular selection algorithm, the average response time has been measured for each ML algorithm and the regular selection algorithm. Therefore, 1000 requests have been sent to the Transaction Component via POST request to the /api/transactions endpoint. Each request containing an Excel file that consists of 10 sheets, resulting in 10000 randomly generated transactions. The policies have been configured in a way that both algorithms return the same BC for all transactions. The specific configuration is outlined in Table 6.4, which both result in the selection of Stellar.

Table 6.4: Policy configuration

Algorithm	Policy Configuration							
	Type	**TPS**	**Block Time**	**Transaction Speed**	**Data Size**	**SC**	**TC**	**Popularity**
ML models	Public	-	-	Medium	20	Yes	No	High
Regular	Public	1000	5	-	20	-	No	-

For testing, Postman [70] was used, which is an interactive API testing tool. It provides a user-friendly GUI for constructing requests and reading responses. Following the policy configuration, the POST request with the required parameters was defined and executed 1000 times. The specific body of the POST request in Postman is shown in Figure 6.5.

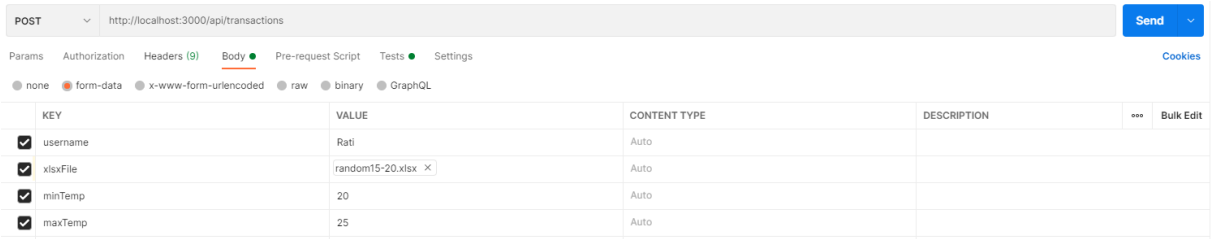


Figure 6.5: Transaction body of POST request in Postman

Finally, the average response time was calculated for each algorithm, which is depicted in Figure 6.6. The results show that all the ML models had a similar average response time of approximately 1200 ms per request and 120 ms per selection. It further demonstrates that the integrated ML solution does not produce a significant overhead in terms of response time compared to the regular algorithm used in PleBeuS.

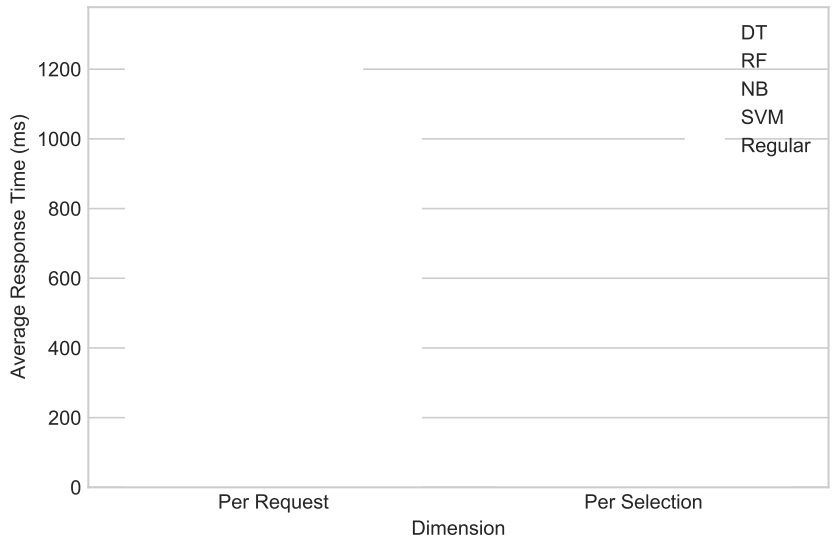


Figure 6.6: Average response time for 1000 iterations

6.4 Discussion

This section attempts to answer RQ 1 as described in Chapter 1 and discusses the main findings of the evaluation.
Can ML be useful in the BC selection? And which algorithms are most suitable?

The comparison of the ML models revealed that the models can be highly accurate and used for a multiclass classification task such as the BC selection process. It showed that the NB algorithm is not suitable for this dataset due to its low accuracy score and bad performance on the use case scenario in Section 6.2.1. The three other models, *i.e.,* DT, RF and SVM, performed significantly better than the NB models across all tests being conducted for this evaluation, except for the evaluation of the training time, where the NB model was the second-fastest learning algorithm. Between these three models, there was no essential or significant difference in terms of accuracy. Their corresponding confusion matrices, as well as their accuracy scores, demonstrated that most of the instances of the test cases were correctly classified. In particular, if the input variables completely match with an underlying BC, that specific BC is predicted by the models.

In cases where two different BC share the same properties, *e.g.,* EOS and Wanchain, and both meet the criteria defined in the parameters, only one of them is outputted by the model. As seen in the model evaluation, this led to misclassification in some cases, despite the models predicting one of the two possible BCs, which was also the reason why the models did not yield a higher accuracy score. This could possibly be addressed by including a multi-label classification algorithm, which is used to predict properties of samples that are not mutually exclusive, *i.e.,* each instance can be assigned simultaneously into multiple categories instead of only one like in the case of multiclass classification. Such an approach could also be applied to cases where the input variables do not fully match an underlying BC implementation (discussed in Section 6.2.2), providing users with a recommendation of multiple BCs instead of only one based on their requirements. Scikit-multilearn is a dedicated library for multi-label classification problems, which is built on top of Scikit-learn ecosystem [99]. Unfortunately, this approach was not further elaborated due to time constraints and may be the subject of future work.

Section 6.2.2 has proved that the models provide good predictions when no BC entirely covers the desired properties. However, it might give rise to certain disadvantages. As demonstrated, there are cases where users wish to have a particular data size or BC type, but since not all properties are covered by a single BC, it can happen that a model predicts a BC which does not support the desired data size or does not match the type specified. Obviously, the same can happen to all other features. To cope with this issue, a feature importance metric could be introduced, which would allow users to define the relative importance of their preferences that should be considered when making a prediction.

Finally, the performance testing of the ML solution in combination with the PleBeuS framework showed that there is no significant difference in terms of response time between using the ML models and the regular selection algorithm for selection within the framework.

6.5 ML vs Rule-Based Systems

This section aims to answer RQ 2 as described in Chapter 1.
How does a ML-based selection approach compare to a rule-based system?

ML models learn inherent patterns from data to make a prediction of an output for new data points by using the learned knowledge. Hence, there is no need for explicit hand-coded rules or numerous if-then conditions that are usually involved in creating a decision making/support system.

In this context, a rule-based approach would determine a suitable BC based on the required features by using a set of decision rules, such as it is the case in the selection algorithm used for the PleBeuS framework. These rules instruct the system to use relevant features of a policy that has been configured from the user to identify the most appropriate BC. Although a rule-based system is human-comprehensible and might be improved over time, it comes with several disadvantages. On the one hand, it requires in-depth domain knowledge and can become a time-consuming process, especially when the system is complex. On the other hand, it can result in poor scalability, as adding new rules can impact existing rules. In contrast, a supervised ML approach would learn to make these classifications based on a dataset, *e.g.,* the dataset outlined in Section 4.3, where no person has to explicitly define the rules. New instances or features can easily be added to the dataset, such that new tasks can be learned from the new data without having to code new rules explicitly.

However, the data acquisition and preparation process is a crucial part of ML and usually involves multiple steps and essential decisions that will affect the overall model quality and performance. This process can also become a time-consuming and tedious task that can involve many challenges, such as lack of necessary data or unbalanced data. Moreover, the performance of the models need to be closely evaluated to validate the models' effectiveness and whether or not the outcomes are acceptable and could be applied in a real-case scenario. As shown in the case of the NB model, some models can have poor applicability and might not be suitable for a specific use case.

In a multiclass classification task such as the BC selection process, an ML-based solution can provide valuable recommendations to users when the input variables do not fully match an underlying class. A rule-based algorithm, in contrast, would not be able to make a classification (selection) in such a case. *E.g.,* use case scenario 2 in Section 6.2.2 showed that the ML-based selection algorithm is able to make appropriate predictions when no applicable BC implementation fits all the chosen parameters. In opposition, the existing rule-based algorithm of PleBeuS would inform the user that there is no matching BC implementation.

As can be seen, the question of whether ML is beneficial and suitable for a particular use case depends on several factors that would have to be considered. But generally, ML is better suited for cases with a large number of features and data since it is hard to manually develop rules or identify patterns in the data. Contrarily, rule-based systems might be more suitable for cases with a lower volume of data and for which the rules are fairly simple to determine. Nevertheless, both approaches aim to automate the decision process with a high degree of accuracy.

6.6 Feature Importance

This section attempts to address RQ 3 as described in Chapter 1.
Which features are important for the selection process?

The features which are used for model training and input variables were introduced in Section 4.2. These features are very important as they are the building blocks of the dataset and therefore directly influence the models and their results. Given new input features, the models should forecast the output as accurately as possible.

The most common explanation technique for classification models is the feature importance score which provides insights into model behavior [6]. Feature importance describes how important a feature was for the classification of a model. A specific feature might be more important for one classification model than for another; this is evident when comparing the feature importance scores of the trained DT classifier and the RF classifier depicted in Figure 6.4. Not all the features are equally important to predict the BC, *i.e.,* the target value. While the Transaction Speed feature has a relatively high importance in both models, the other features vary in importance depending on the considered model. However, none of the features has a zero importance on the target variable, and thus, all features contribute to the prediction of the models.

Figure 6.2 illustrates the correlation between the features. As can be seen, there is a considerable correlation between Turing Completeness of a BC and its SC Compatibility. This is logical, as Turing Completeness is only available when the BC supports SC. The Platform Transaction Speed is negatively correlated to the BC type, suggesting that private BCs do generally have a higher transaction speed than public BCs. Its relative high correlation to the SC feature indicates that BCs which support SCs tend to have a higher transaction speed. Moreover, it seems that the type of a BC has an impact on the popularity score, as they are positively correlated. But since there is no strong or perfect correlation between any of the features (correlation score ≥ 0.9), there are no variables that would convey redundant information and could be removed without losing valuable information.

Chapter 7

Conclusion and Future Work

As presented in this thesis, choosing a suitable BC for a particular use case is not a trivial task and requires domain-specific knowledge. Recently, a policy-based management approach has been applied to automate the process of the BC selection, which recommends BC implementations based on transaction information and pre-defined policies [84]. This process is governed by a simple selection algorithm that applies straightforward filtering. The ML-based BC selection solution introduced in this thesis extends the existing solution and shows that novel approaches, such as ML, could be applied to the selection process and automatically select an appropriate BC given user-defined parameters.

In specific, four ML algorithms namely, DT, RF, SVM and NB, were trained and evaluated on their applicability in the BC selection process. The prototypical design of the solution involved a full integration into the policy-based selection framework, PleBeuS. For this purpose, a dedicated dataset was generated and refined, which enabled the introduction of new policy parameters that allow users to configure additional preferences within a policy. It includes the definition of the ML model, the SC support, the platform transaction speed and the popularity of the BC.

The ML-based BC selection solution was deployed as a REST API and can be used either through a GUI, which is also a product of this thesis or in conjunction with PleBeuS. By using the framework, users define policies via the PMT, where they have the additional option to choose among the ML and the regular selection algorithm. When a transaction is passed through the Transaction Component of the framework, the policies are used to make a decision on a BC implementation.

The evaluation of the solution has uncovered several findings. The first finding is that not all ML algorithms chosen for the solution design were equally suitable. While the DT, RF and SVM achieved good results, the NB model turned out to be less efficient. The performance evaluation showed that the framework works well in combination with ML and does not add significant overhead to the existing solution. However, the evaluation showed up flaws when making predictions with the ML models. The models generally performed extremely well when the underlying input variables were an exact match for a specific BC. If this was not the case, the models are still able to predict an appropriate BC as opposed to the existing rule-based algorithm, where this would lead to a policy

conflict. In some cases, this might pose a problem, *e.g.,* when users wish to have a certain supported data size, but the models suggest a BC that does not satisfy this requirement. Nevertheless, these predictions can be used as a recommender system, aiding users in choosing the most fitting BC.

7.1 Future Work

The ML-based BC selection solution presented in this thesis focuses on providing a means to automatically select a suitable BC implementation based on desired user requirements. The solution can be used with the PleBeuS framework, which acts as PMT and PDP. However, the models implemented in the solution are only capable of predicting one BC for each policy, even if there might be more than one fitting BC. As such, the functionality of splitting transaction into multiple BCs is not possible with the ML-based selection. Other algorithms need to be researched and examined on their applicability to enable such a functionality. *E.g.*, deep learning neural networks natively support multi-label classification problems that support predicting multiple mutually non-exclusive classes.

There are additional BC implementations in the ML-based solution that are not yet supported by PleBeuS. In order to make these BCs available for the regular selection algorithm, the framework has to be extended; otherwise, they will only be available through the ML-based selection. Such an extension would also allow the accompanied GUI to show statistics about transaction information of the user involving these BCs. Furthermore, not all parameters of the solution proposed in this work are supported by the original PleBeuS implementation, which could also be subject to an extension. Specifically, these parameters are SC support, the platform transaction speed and the popularity of a BC.

Another subject of discussion is the BC Costs Monitor of the framework, which computes transaction fees that are used to make decisions regarding the user-defined Cost Threshold. External services are being used to determine the generated transaction costs for public BCs, which involve various calculations to convert the costs into costs per byte. These costs are used to update the cost intervals for the current user and enabling them to set their preferences towards their preferred Cost Profile. In the ML-based BC selection solution, such a functionality has not been implemented as it was not the main focus of the thesis. Instead, an arbitrary cost of 0.01 per transaction was added to each BC implementation that is newly supported by the ML-based solution. In this context, the framework has to be extended to support cost calculations and the true reflection of the actual costs of transactions involving these BCs.

Bibliography

[1] MAS working with industry to apply Distributed Ledger Technology in securities settlement and cross border payments. https://www.mas.gov.sg/news/media-releases/2017/mas-working-with-industry-to-apply-distributed-ledger-technology (2017)

[2] Abdo, J., Zeadally, S.: Neural network-based blockchain decision scheme. Information Security Journal: A Global Perspective (2020). https://doi.org/10.1080/19393555.2020.1831658, https://doi.org/10.1080/19393555.2020.1831658

[3] Aly, M.: Survey on multiclass classification methods. Neural networks. **19**, pp. 1–9 (2005)

[4] Baliga, A., Subhod, I., Kamat, P., Chatterjee, S.: Performance Evaluation of the Quorum Blockchain Platform (2018)

[5] Belotti, M., Božić, N., Pujolle, G., Secci, S.: A Vademecum on Blockchain Technologies: When, Which, and How. IEEE Communications Surveys Tutorials **21**(4), 3796–3838 (2019)

[6] Bhatt, U., Xiang, A., Sharma, S., Weller, A., Taly, A., Jia, Y., Ghosh, J., Puri, R., Moura, J.M.F., Eckersley, P.: Explainable machine learning in deployment. In: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency. p. 648â657. FAT* '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3351095.3375624, https://doi.org/10.1145/3351095.3375624

[7] BitInfoCharts: Cryptocurrency statistics, https://bitinfocharts.com/

[8] Blockchainhub: What is a Smart Contract? Auto enforceable Code - Blockchain, https://blockchainhub.net/smart-contracts/, (Accessed: 12/27/2020)

[9] Blockchair: Explore Blockchains, https://blockchair.com/

[10] Breiman, L.: Random Forests. Machine Learning **45**, 5–32 (Jan 2001). https://doi.org/10.1023/A:1010933404324

[11] Brownlee, J.: A Gentle Introduction to k-fold Cross Validation, https://machinelearningmastery.com/k-fold-cross-validation/, (Accessed: 01/16/2021)

[12] Burke, K., Conroy, K., Horn, R., Stratton, F., Binet, G.: Flask-RESTful, https://flask-restful.readthedocs.io/en/latest/, (Accessed: 04/10/2021)

[13] Buterin, V.: A Next Generation Smart Contract and Decentralized Application Platform, https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf, (Accessed: 12/29/2020)

[14] Cardano: Producing New Blocks, https://docs.cardano.org/en/latest/explore-cardano/how-are-new-blocks-produced.html/, (Accessed: 12/31/2020)

[15] Cardano: What is Cardano's block size limit?, https://forum.cardano.org/t/what-is-cardanos-block-size-limit/25510, (Accessed: 03/24/2021)

[16] Chen, S., Zhang, J., Shi, R., Yan, J., Ke, Q.: A Comparative Testing on Performance of Blockchain and Relational Database: Foundation for Applying Smart Technology into Current Business Systems. In: Streitz, N., Konomi, S. (eds.) Distributed, Ambient and Pervasive Interactions: Understanding Humans. pp. 21–34. Springer International Publishing, Cham (2018)

[17] Chris, A.: Top 10 Search Engines In The World (2021 Update), https://www.reliablesoft.net/top-10-search-engines-in-the-world/, (Accessed: 01/27/2021)

[18] Coindesk: A (Short) Guide to Blockchain Consensus Protocols (March 2017), https://www.coindesk.com/short-guide-blockchain-consensus-protocols, [Online; posted 04-March-2017]

[19] CoinMarketCap: CoinMarketCap Market Capitalizations. https://coinmarketcap.com (2020)

[20] Corda: Transactions Per Second (TPS), https://www.corda.net/blog/transactions-per-second-tps/, (Accessed: 12/28/2020)

[21] Corporporation, I.: Introduction - Sawtooth Documentation, https://sawtooth.hyperledger.org/docs/core/releases/latest/introduction.html, (Accessed: 12/28/2020)

[22] Corporporation, I.: PoET 1.0 Specification, https://sawtooth.hyperledger.org/docs/core/nightly/1-2/architecture/poet.html, (Accessed: 12/28/2020)

[23] CryptoLions: EOS Network Monitor, https://eosnetworkmonitor.io/, (Accessed: 12/27/2020)

[24] Dubovitskaya, A., Zhigang, X., Ryu, S., Schumacher, M., Wang, F.: Secure and trustable electronic medical records sharing using blockchain. Proceedings of the AMIA annual symposium 2017 (CONFERENCE), 10 p. (2017), http://hesso.tind.io/record/2896

[25] EOS: EOS.IO Technical White Paper v2, https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md, (Accessed: 12/27/2020)

[26] Ethereum: Consensus mechanism, https://ethereum.org/en/developers/docs/consensus-mechanisms/, (Accessed: 12/27/2020)

[27] Farshidi, S., Jansen, S., España, S., Verkleij, J.: Decision Support for Blockchain Platform Selection: Three Industry Case Studies. IEEE Transactions on Engineering Management pp. 1–20 (2020)

[28] Flask: Flask Homepage, https://flask.palletsprojects.com/en/1.1.x/, (Accessed: 04/10/2021)

[29] Frankenfield, J.: VeChain, https://www.investopedia.com/terms/v/vechain.asp, (Accessed: 03/24/2021)

[30] Frauenthaler, P., Borkowski, M., Schulte, S.: A Framework for Assessing and Selecting Blockchains at Runtime. In: 2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). pp. 106–113 (2020). https://doi.org/10.1109/DAPPS49028.2020.00013

[31] Frauenthaler, P., Borkowski, M., Schulte, S.: A Framework for Blockchain Interoperability and Runtime Selection (2019)

[32] Gal., A.: The Tangle: an Illustrated Introduction, https://blog.iota.org/the-tangle-an-illustrated-introduction-4d5eae6fe8d4/, (Accessed: 12/27/2020)

[33] Ganganwar, V.: An overview of classification algorithms for imbalanced datasets. International Journal of Emerging Technology and Advanced Engineering **2**(4), 42–47 (2012)

[34] Géron, A.: Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems. O'Reilly, Sebastopol, CA, first edition edn. (2017)

[35] Greenspan, G.: MultiChain Private Blockchain - White Paper, https://www.multichain.com/download/MultiChain-White-Paper.pdf, (Accessed: 12/28/2020)

[36] Hucker, M.: Multiclass Classification with Support Vector Machines (SVM), Dual Problem and Kernel Functions, https://bit.ly/2SmEzy7, (Accessed: 03/31/2021)

[37] ICON: Devportal - ICON JSON-RPC API v3 Specification, https://www.icondev.io/docs/icon-json-rpc-v3#icx_sendtransaction, (Accessed: 03/24/2021)

[38] ICON Foundation: ICON - Hyperconnect the world, https://icon.foundation/, (Accessed: 01/19/2021)

[39] ICON Foundation: ICON Blockchain Explorer, https://tracker.icon.foundation/, (Accessed: 03/24/2021)

[40] ICON Foundation: ICON Tracker, https://tracker.icon.foundation/

[41] ICON Foundation: ICON White Paper, https://icon.foundation/resources/whitepaper/ICON_Whitepaper_EN.pdf, (Accessed: 01/19/2021)

[42] ICX_Station: ICON, deconstructed, https://medium.com/helloiconworld/icon-deconstructed-5eb7f99eeeb1, (Accessed: 03/24/2021)

[43] IEEE: IEEE Xplore Digital Library, https://ieeexplore.ieee.org/Xplorehelp/overview-of-ieee-xplore/about-ieee-xplore, (Accessed: 02/09/2021)

[44] Imbalanced-learn Developers: Imbalanced Learn Documentation, https://imbalanced-learn.org/stable/, (Accessed: 04/10/2021)

[45] IOTA Foundation: The Next Generation of Distributed Ledger Technology | IOTA, https://www.iota.org/, (Accessed: 03/24/2021)

[46] Joblib Developers: Joblib Homepage, https://joblib.readthedocs.io/en/latest/, (Accessed: 04/12/2021)

[47] Kenton, W.: Block Time, https://www.investopedia.com/terms/b/block-time-cryptocurrency.asp, (Accessed: 02/25/2021)

[48] Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017. pp. 357–388. Springer International Publishing, Cham (2017)

[49] Killer, C., Rodrigues, B., Scheid, E.J., Franco, M., Eck, M., Zaugg, N., Scheitlin, A., Stiller, B.: Provotum: A Blockchain-based and End-to-end Verifiable Remote Electronic Voting System. In: IEEE 45th Conference on Local Computer Networks (LCN 2020). pp. 172–183. Sydney, Australia (November 2020)

[50] Klein, S., Prinz, W.: A Use Case Identification Framework and Use Case Canvas for identifying and exploring relevant Blockchain opportunities. In: Proceedings of 1st ERCIM Blockchain Workshop 2018. European Society for Socially Embedded Technologies (EUSSET) (2018)

[51] Kotsiantis, S.: Supervised Machine Learning: A Review of Classification Techniques. Informatica (Ljubljana) **31** (10 2007)

[52] Lakic, D.: Design and Implementation of a Policy-based Blockchain Selection Framework. Master's thesis, Zürich, Switzerland (June 2019), https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/theses/ma-daniel.pdf

[53] Lo, S.K., Xu, X., Chiam, Y.K., Lu, Q.: Evaluating Suitability of Applying Blockchain. In: 2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS). pp. 158–161 (2017). https://doi.org/10.1109/ICECCS.2017.26

[54] Mazières, D.: The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus, `https://www.stellar.org/papers/stellar-consensus-protocol`, (Accessed: 12/27/2020)

[55] Mike Hearn, R.G.B.: Corda: A distributed ledger, `https://www.r3.com/wp-content/uploads/2019/08/corda-technical-whitepaper-August-29-2019.pdf`, (Accessed: 12/28/2020)

[56] Mohammed, M., Khan, M., Bashier, E.: Machine Learning: Algorithms and Applications. CRC Press (07 2016). https://doi.org/10.1201/9781315371658

[57] Monika, Bhatia, R.: Interoperability Solutions for Blockchain. In: 2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE). pp. 381–385 (2020). https://doi.org/10.1109/ICSTCEE49637.2020.9277054

[58] Moore, B., Ellesson, E., Strassner, J., Westerinen, A.: RFC 3060 - Policy Core Information Model. `https://tools.ietf.org/html/rfc3060` (2001)

[59] Müller, A.C., Guido, S.: Introduction to Machine Learning with Python. O'Reilly Media Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472 (2016)

[60] MultiChain: Customizing blockchain parameters, `https://www.multichain.com/developers/blockchain-parameters/`, (Accessed: 12/28/2020)

[61] Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2009)

[62] NEO: Neo White Paper, `https://docs.neo.org/docs/en-us/basic/whitepaper.html`, (Accessed: 01/05/2021)

[63] NEOscan: NEOSCAN - Neo related blockchain information, `https://neoscan.io/`

[64] Norta, A., Dai, P., Mahi, N., Earls, J.: A Public, Blockchain-Based Distributed Smart-Contract Platform Enabling Mobile Lite Wallets Using a Proof-of-Stake Consensus Algorithm. In: Abramowicz, W., Paschke, A. (eds.) Business Information Systems Workshops. pp. 368–380. Springer International Publishing, Cham (2019)

[65] NumPy: NumPy Homepage, `https://numpy.org/`, (Accessed: 04/12/2021)

[66] Orcutt, M.: No, Ripple Isn't the Next Bitcoin, `https://www.technologyreview.com/2018/01/11/146252/no-ripple-isnt-the-next-bitcoin/`, (Accessed: 01/03/2021)

[67] Pahl, C., El Ioini, N., Helmer, S.: A decision framework for blockchain platforms for IoT and edge computing. International Conference on Internet of Things, Big Data and Security (2018)

[68] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[69] Pongnumkul, S., Siripanpornchana, C., Thajchayapong, S.: Performance Analysis of Private Blockchain Platforms in Varying Workloads. In: 2017 26th International Conference on Computer Communication and Networks (ICCCN). pp. 1–6 (2017). https://doi.org/10.1109/ICCCN.2017.8038517

[70] Postman: Postman Homepage, `https://www.postman.com/api-platform/api-testing/`, (Accessed: 04/10/2021)

[71] Precht, H., Wunderlich, S., Gómez, J.: Applying Software Quality Criteria to Blockchain Applications: A Criteria Catalog (01 2020). https://doi.org/10.24251/HICSS.2020.769

[72] Puthal, D., Malik, N., Mohanty, S.P., Kougianos, E., Das, G.: Everything You Wanted to Know About the Blockchain: Its Promise, Components, Processes, and Problems. IEEE Consumer Electronics Magazine **7**(4), 6–14 (2018). https://doi.org/10.1109/MCE.2018.2816299

[73] Qtum: Official blockchain explorer of Qtum, `https://qtum.info/`

[74] Qtum: Qtum Exchange Usage Guide and Info, `https://docs.qtum.site/en/Qtum-Exchange-Usage-Guide-and-Info.html`, (Accessed: 03/24/2021)

[75] Qtum: SCAR: Scalable Consensus Algorithm, `https://docs.qtum.site/en/SCAR-Consensus/`, (Accessed: 01/05/2021)

[76] Raileanu, L.E., Stoffel, K.: Theoretical Comparison between the Gini Index and Information Gain Criteria. Annals of Mathematics and Artificial Intelligence **41**(1), 77–93 (2004)

[77] Ripple: The XRP Ledger Protocol - Consensus and Validation, `https://xrpl.org/consensus.html`, (Accessed: 01/03/2021)

[78] Ripple: XRP: The Best Digital Asset for Global Payments, `https://ripple.com/xrp/`, (Accessed: 03/24/2021)

[79] Roman, V.: Unsupervised Machine Learning: Clustering Analysis (March 2019), `https://towardsdatascience.com/unsupervised-machine-learning-clustering-analysis-d40f2b34ae7e`, [Online; posted 6-March-2019]

[80] Roman, V.: Unsupervised Machine Learning: Dimensionality Reduction (April 2019), `https://towardsdatascience.com/unsupervised-learning-dimensionality-reduction-ddb4d55e0757`, [Online; posted 17-April-2019]

[81] Sarkar, D., Bali, R., Sharma, T.: Practical Machine Learning with Python: A Problem-Solver's Guide to Building Real-World Intelligent Systems. Apress, USA, 1st edn. (2017)

[82] Scheid, E., Rodrigues, B., Stiller, B.: Toward a Policy-based Blockchain Agnostic Framework. In: IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019). pp. 609–613. Arlington, VA, USA (April 2019)

[83] Scheid, E.J., Hegnauer, T., Rodrigues, B., Stiller, B.: Bifröst: a Modular Blockchain Interoperability API. In: 2019 IEEE 44th Conference on Local Computer Networks (LCN). pp. 332–339 (2019). https://doi.org/10.1109/LCN44214.2019.8990860

[84] Scheid, E.J., Lakic, D., Rodrigues, B.B., Stiller, B.: PleBeuS: a Policy-based Blockchain Selection Framework. In: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium. pp. 1–8 (2020)

[85] Scikit-Learn: Scikit-Learn Homepage, `https://scikit-learn.org/stable/`, (Accessed: 04/10/2021)

[86] Scikit-Learn Developers: Naive Bayes, `https://scikit-learn.org/stable/modules/naive_bayes.html`, (Accessed: 04/03/2021)

[87] Scikit-Learn Developers: Scikit-Learn Decision Trees, `https://scikit-learn.org/stable/modules/tree.html`, (Accessed: 04/14/2021)

[88] Scikit-Learn Developers: Scikit-Learn Naive Bayes, `https://scikit-learn.org/stable/modules/naive_bayes.html`, (Accessed: 04/14/2021)

[89] Scikit-Learn Developers: Scikit-Learn Random Forest Classifier, `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`, (Accessed: 04/14/2021)

[90] Scikit-Learn Developers: Scikit-Learn Support Vector Classification, `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`, (Accessed: 04/14/2021)

[91] Singh, A., Thakur, N., Sharma, A.: A review of supervised machine learning algorithms. In: 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom). pp. 1310–1315 (2016)

[92] Smith, K.: 60 Incredible and Interesting Twitter Stats and Statistics, `https://www.brandwatch.com/blog/twitter-stats-and-statistics/`, (Accessed: 01/26/2021)

[93] Soulo, T.: Keyword Search Volume: Things you didn't know you don't know, `https://ahrefs.com/blog/keyword-search-volume/`, (Accessed: 01/27/2021)

[94] SQLite Consortium: SQLite Homepage, `https://www.sqlite.org/index.html`, (Accessed: 04/10/2021)

[95] Stratis: FAQ: What is Stratis?, `https://stratisfaq.com/what-is-stratis`, (Accessed: 03/26/2021)

[96] Stratis: Saving arbitrary data on the Stratis blockchain, https://stratisplatform.atlassian.net/wiki/spaces/code/pages/983321/Saving+arbitrary+data+on+the+Stratis+blockchain, (Accessed: 03/24/2021)

[97] Stratis Developers: Stratis Homepage, https://www.stratisplatform.com/, (Accessed: 12/28/2020)

[98] Sultan, K., Ruhi, U., Lakhani, R.: Conceptualizing Blockchains: Characteristics & Applications. CoRR **abs/1806.03693** (2018), http://arxiv.org/abs/1806.03693

[99] Szymański, P., Kajdanowicz, T.: A scikit-based Python environment for performing multi-label classification. ArXiv e-prints (Feb 2017)

[100] T, S.: Entropy: How Decision Trees Make Decisions, https://bit.ly/3xiBbUM, (Accessed: 01/13/2021)

[101] Takashima, I.: Ripple: The Ultimate Guide to the World of Ripple XRP, Ripple Investing, Ripple Coin, Ripple Cryptocurrency, Cryptocurrency. CreateSpace Independent Publishing Platform, North Charleston, SC, USA (2018)

[102] Tokenview: Wanchain Explorer, https://wan.tokenview.com/

[103] Tuwiner, J.: Introduction to NEO - An Open Network For Smart Economy, https://cryptoslate.com/introduction-to-neo-an-open-network-for-smart-economy/, (Accessed: 01/05/2021)

[104] Twitter Developers: Twitter API Documentation, https://developer.twitter.com/en/docs/twitter-api

[105] Uddin, S., Khan, A., Hossain, M.E., Moni, M.A.: Comparing different supervised machine learning algorithms for disease prediction. BMC medical informatics and decision making **19**(1), 281 (December 2019). https://doi.org/10.1186/s12911-019-1004-8, https://europepmc.org/articles/PMC6925840

[106] Underwood, S.: Blockchain beyond Bitcoin. Commun. ACM **59**(11), 15â17 (Oct 2016). https://doi.org/10.1145/2994581, https://doi.org/10.1145/2994581

[107] VeChain Foundation: VeChain and University of Oxford Jointly Propose An Evaluation Framework for Blockchain Consensus Protocols, https://bit.ly/3dQ7aD2, (Accessed: 03/24/2021)

[108] VeChain Foundation: VeChain Explorer, https://explore.vechain.org/

[109] Verma, D.C.: Simplifying network administration using policy-based management. IEEE Network **16**(2), 20–26 (2002). https://doi.org/10.1109/65.993219

[110] Wanchain: What is Wanchain?, https://www.explorewanchain.org/#/, (Accessed: 03/24/2021)

[111] Wang, S., Li, Z., Liu, C., Zhang, X., Zhang, H.: Training data reduction to speed up SVM training. Applied intelligence **41**(2), 405–420 (2014)

[112] Wang, W., Hoang, D.T., Hu, P., Xiong, Z., Niyato, D., Wang, P., Wen, Y., Kim, D.I.: A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks. IEEE Access **7**, 22328–22370 (2019). https://doi.org/10.1109/ACCESS.2019.2896108

[113] Waskom, M.: Seaborn Homepage, `https://seaborn.pydata.org/index.html`, (Accessed: 04/10/2021)

[114] Wayne, J.: VeChain's Mainnet Just Set New Records With 165 Transactions Per Second, `https://thedailychain.com/vechains-mainnet-just-set-new-records-with-165-transactions-per-second/`, (Accessed: 03/24/2021)

[115] Website, G.: Gini Impurity and Entropy in Decision Tree - ML, `https://www.geeksforgeeks.org/gini-impurity-and-entropy-in-decision-tree-ml/`, (Accessed: 01/13/2021)

[116] Wüst, K., Gervais, A.: Do you Need a Blockchain? In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). pp. 45–54 (2018)

Abbreviations

RQ	Research Question
BC	Blockchain
BTC	Bitcoin
TPS	Transaction Per Second
SC	Smart Contracts
PoW	Proof of Work
PoS	Proof-of-Stake
dPoS	delegated Proof-of-Stake
PoET	Proof-of-Elapsed-Time
EVM	Ethereum Virtual Machine
UTXO	Unpsent Transaction Output
ML	Machine Learning
DT	Decision Tree
IG	Information Gain
RF	Random Forest
NB	Naïve Bayes
SVM	Support Vector Machines
IoT	Internet of Things
API	Application Programming Interface
GUI	Graphical User Interface
PMT	Policy Management Tool
PDP	Policy Decision Point

List of Figures

List of Tables

Appendix A

Full Decision Tree Representation

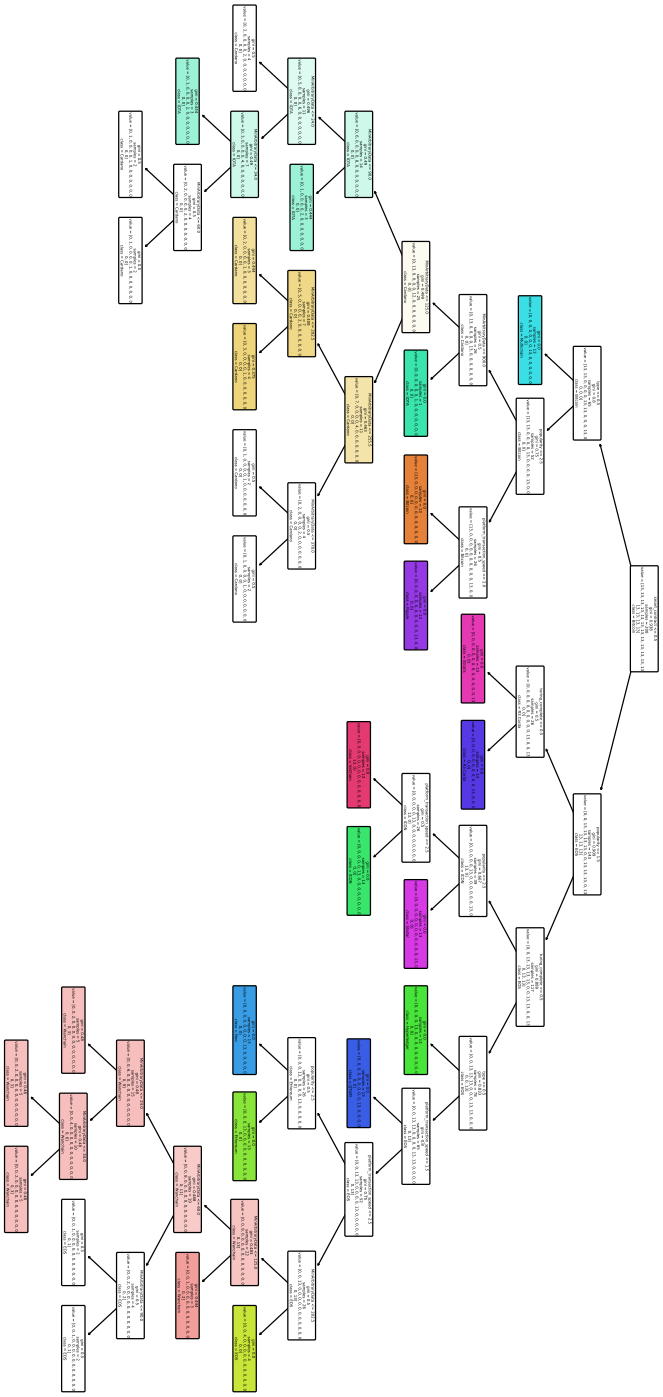


Figure A.1: Entire decision tree as a flowchart diagram

Appendix B

Algorithms Confusion Matrices

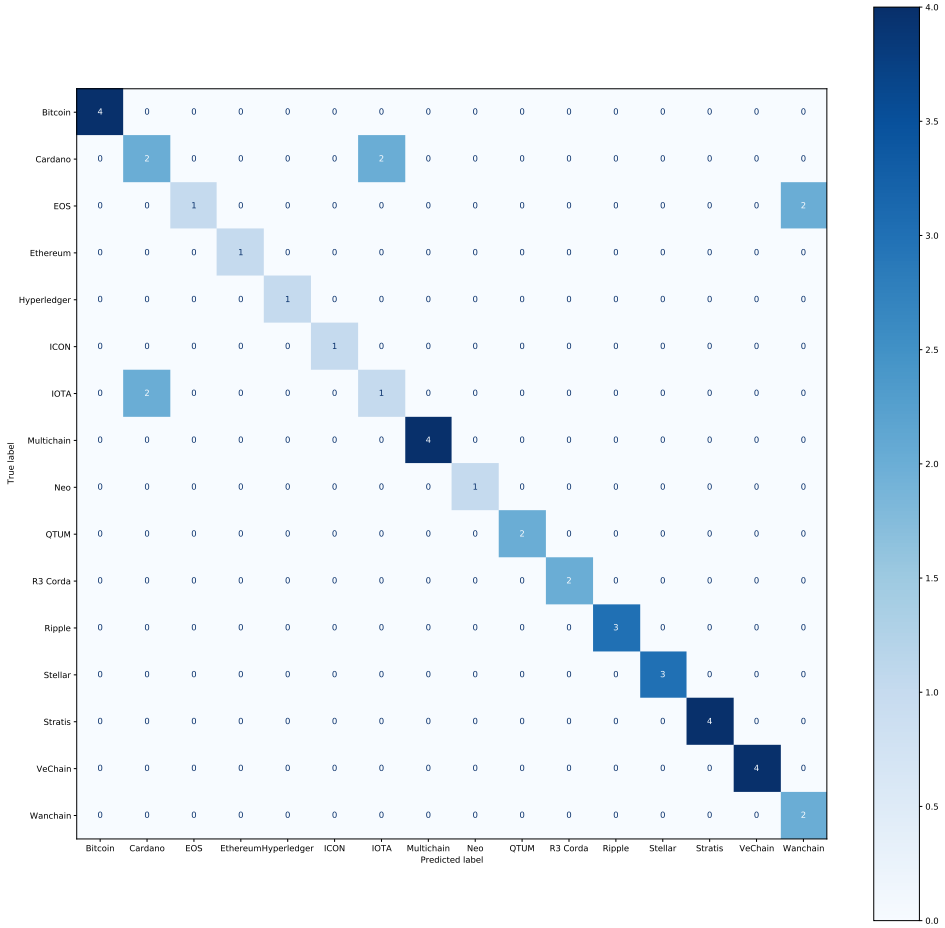


Figure B.1: Decision Tree

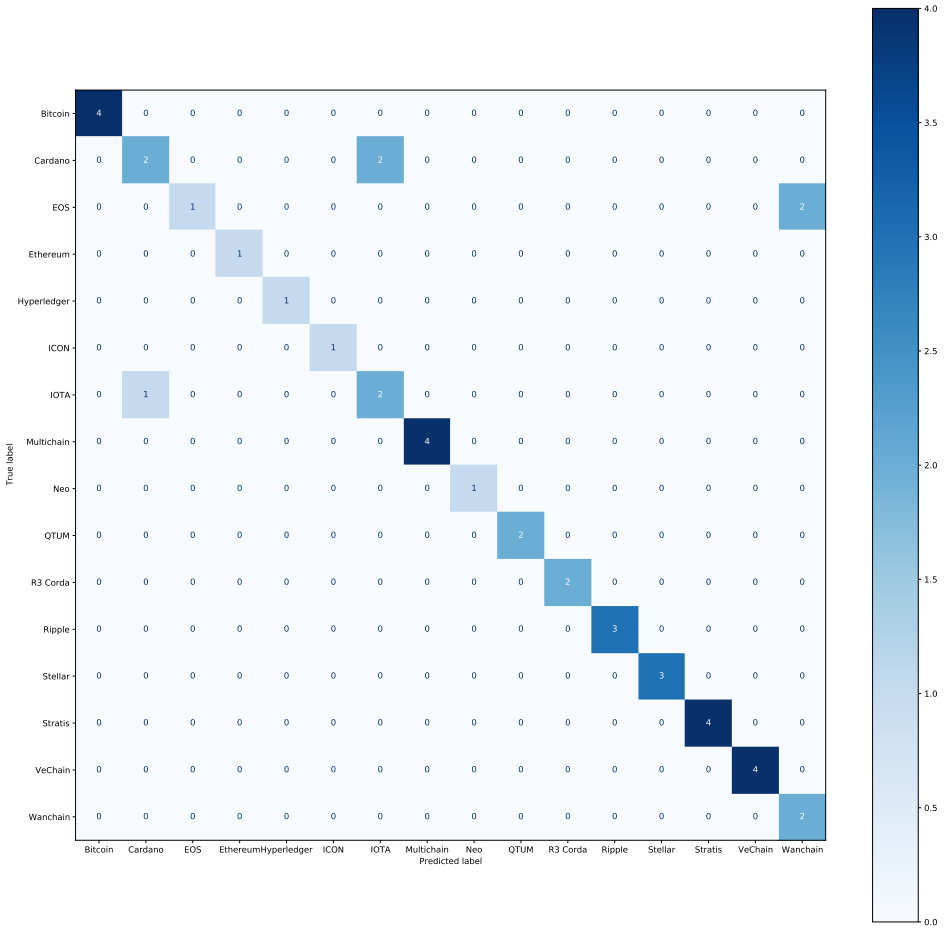


Figure B.2: Random Forest

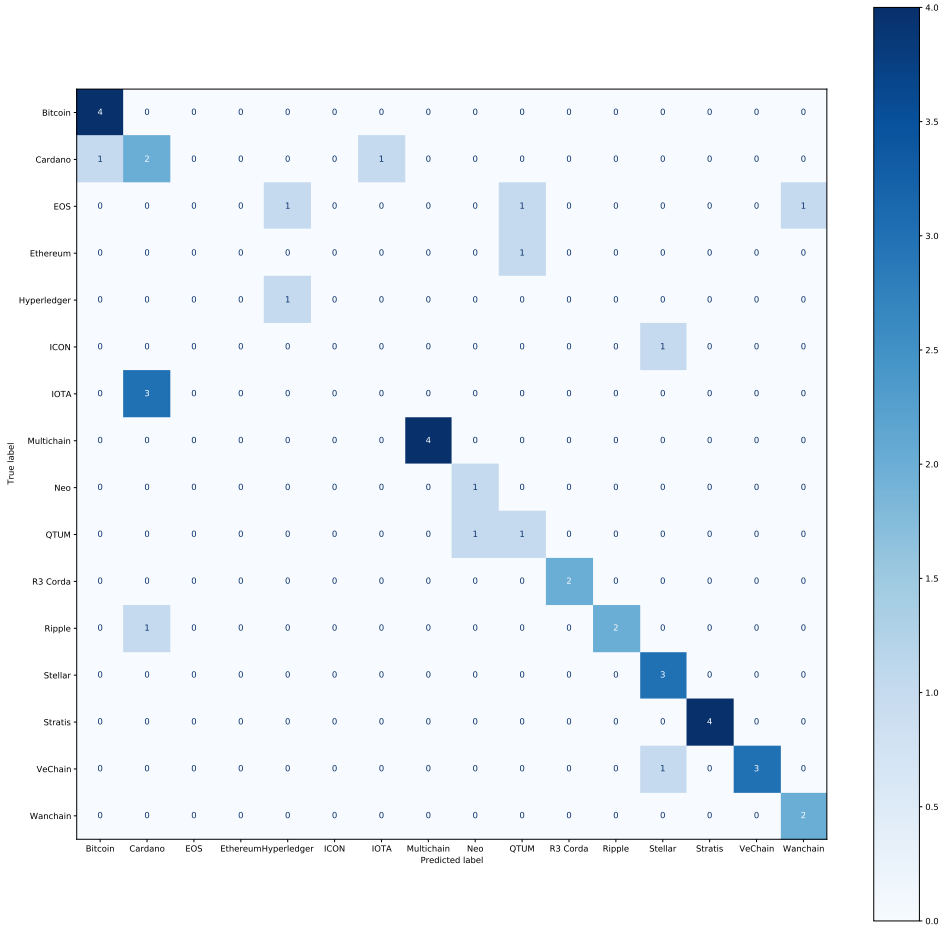


Figure B.3: Naive Bayes

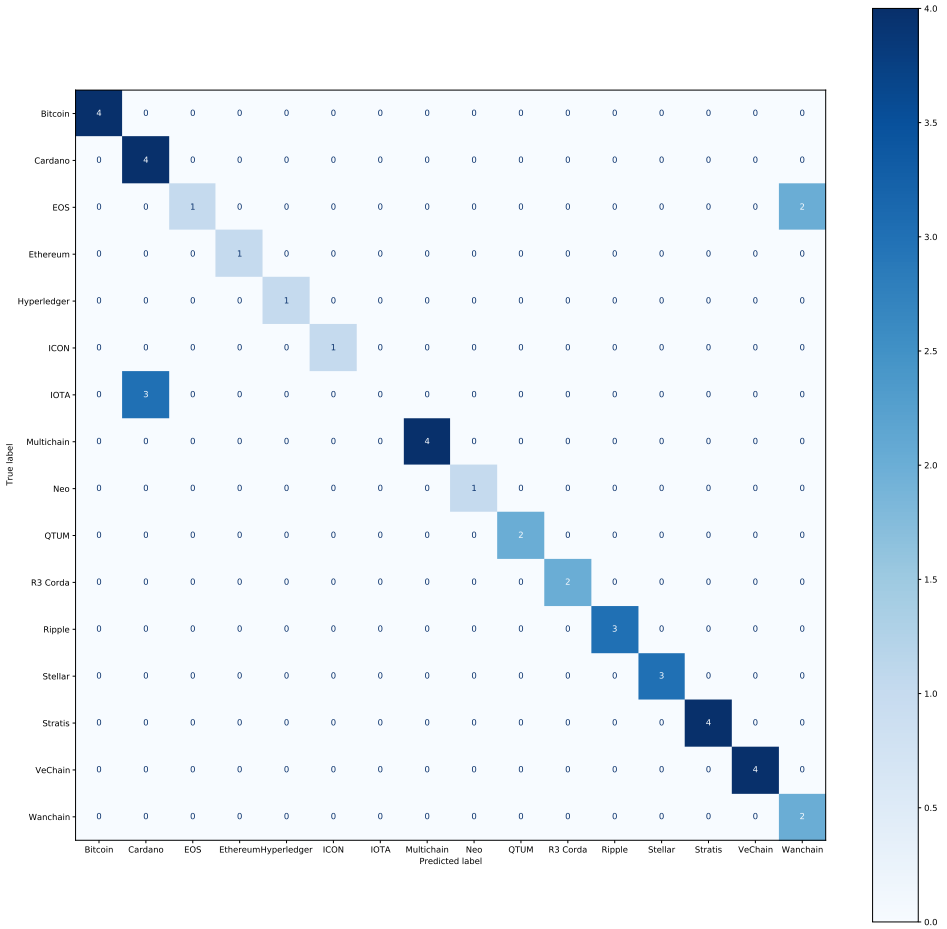


Figure B.4: Support Vector Machine

Appendix C

Sequence Diagrams

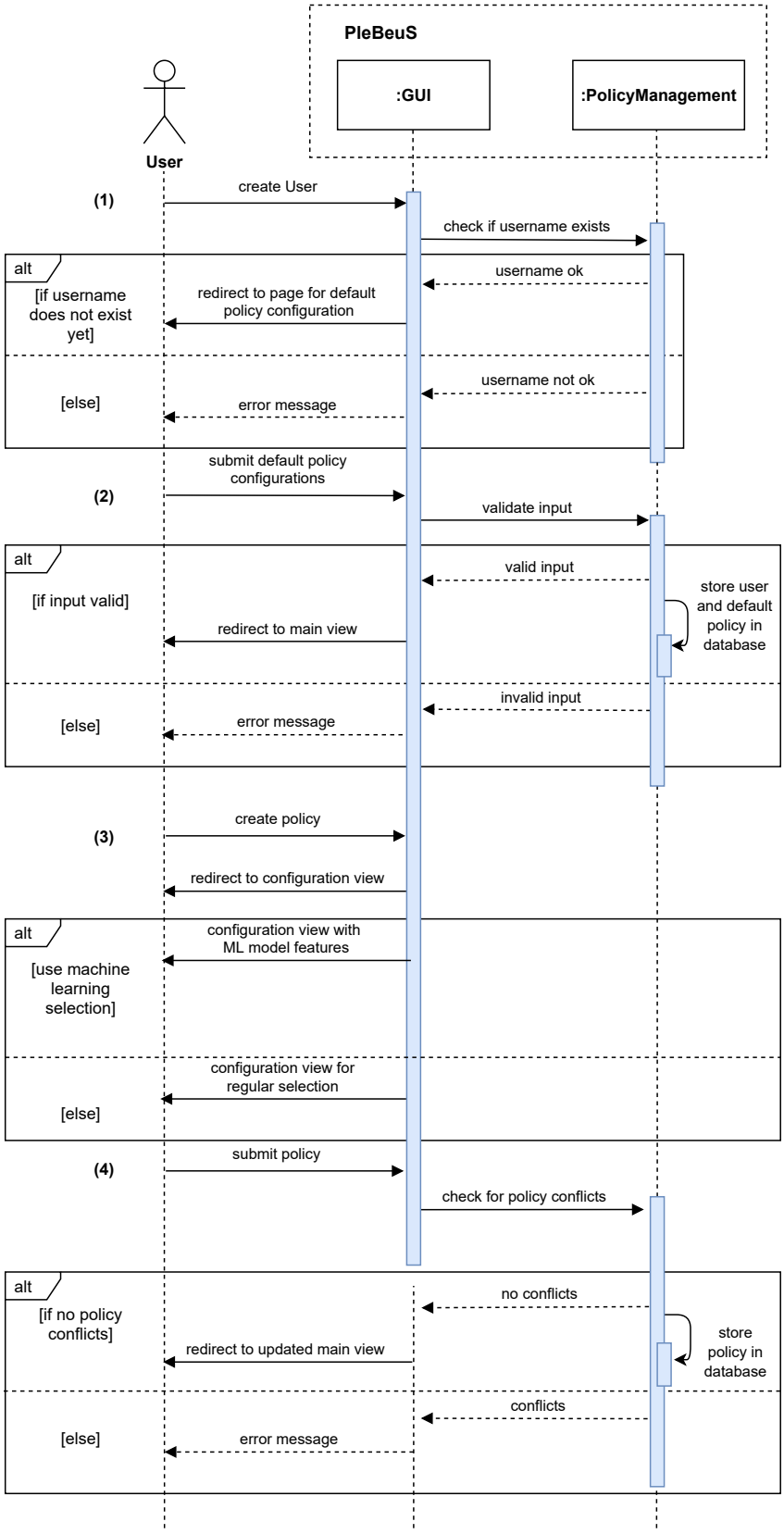


Figure C.1: Policy Management

Appendix D

Installation Guidelines

The source code for the ML-based BC selection solution implemented in this thesis can be found at `https://github.com/Raybook90/ml_blockchain_selection`.

The ML models are implemented and deployed in Python. To install the application, Python version 3.6 or later is required. If Python is not yet installed on your system, consult the official Python installation guide at `https://docs.python.org/3/using/`. Then, follow the steps shown in Section D.1 to set up the Flask Application that offers an API endpoint and a GUI to access the ML models. The GUI is accessible through `http://localhost:5000`.

To make use of the ML-based BC solution in conjunction with the policy-based BC selection framework (PleBeuS), the framework needs to be installed as well. The source code of the extended version of the framework with the integrated ML functionality can be found at `https://github.com/Raybook90/PleBeuS-Integration`. The GUI of the framework is accessible through `http://localhost:3000`.
It can either be set up and developed with Docker or locally. Prerequisites for Docker are a working Docker installation and Docker Compose [52]. To develop with Docker refer to Section D.2.1. For the local installation see Section D.2.2.

D.1 ML-based BC Selection (Flask App) Installation

Clone the repository and enter the project directory:

```
$ git clone https://github.com/Raybook90/ml_blockchain_selection.git
$ cd ml_blockchain_selection
```

Create a virtual environment:

```
$ python -m venv venv
```

Activate the virtual environment:

```
$ venv\Scripts\activate (Windows)
or
$ source venv/bin/activate (Linux, macOS)
```

Install the dependencies:

```
(venv) $ pip install -r requirements.txt
```

Run the application:

```
(venv) $ python app.py
```

Deactivate the virtual environment (when you want to leave virtual environment):

```
(venv) $ deactivate
```

D.2 PleBeuS Installation

This installation guideline follows the same steps as in [52]. An additional step ensures the successful connection to the API endpoint of the ML-based BC selection solution (refer to Section D.2.3).

D.2.1 Docker

A working Docker installation and Docker Compose are prerequisites for a successful Docker set up. Execute one of the following commands. based on what you need:

```
1 #set up docker container
2 docker-compose up
3 #set up docker container and rebuild image
4 docker-compose up --build
5 #set up docker container and start in detached mode
6 docker-compose up -d
```

Listing D.1: Docker Compose commands

D.2.2 Local Installation (Alternative to Docker)

For local installation, Node.js version > 10.x.x (LTS) and MongoDB need to be installed. Additionally, the project supports hot-reloading by using Nodemon. Nodemon is a utility that will monitor any changes in the source code and automatically restart the server. For hot-reloading, Nodemon needs to be installed globally. The project includes a dump of the static BC data needed for this project in foler `/db-dump`. Executing the commands listed below, finalizes the installation process.

```
#install nodemon globally
npm install -g nodemon
#install all dependencies
npm install
#load BC data into the database
mongorestore --nsInclude policy-framework.*
```

Listing D.2: Setup commands for the framework

D.2.3 Configuration

Before the framework can be used, two parameters have to be set as environment variables. One is used for authentication towards the CoinMarketCap API and the other is the URL for the MongoDB database. In case the provided Docker configuration is used, the database URL corresponds to: `mongodb://mongo:27017/policy-framework`.
The use of the ML-based BC Selection solution requires the specification of an additional parameter, which is used to connect to the API endpoint of the Flask application. Fill in the IP address of your machine in the local network for a successful connection.

```
COINMARKETCAP_API_KEY= ""
DB_URL= ""
IP_address= ""
```

Listing D.3: Contents of the .env file

For local or docker usage, the provided `.env.example` file can be copied and renamed to `.env`. Then, the corresponding values can be set. As long as the application is started in development mode, the application is going to search for values set in this file. For production usage the environment variables have to be in the environment.

D.2.4 Run PleBeuS Server

```
#start application in debug mode with nodemon
#running on port:  3000, remote debug port:  3001
npm run devstart
#start application in production mode
npm start --production
```

Listing D.4: Commands for running the application

D.2.5 Usage

In order to use the framework with the ML-based Selection solution, make sure the ML application is running (refer to Section D.1).

PleBeuS consists of two different components. The GUI of the framework allows the creation and management of user policies.

As soon as policies have been defined, data can be passed to the `http://localhost:3000/api/transactions` endpoint via HTTP POST request with the parameters presented in Listing D.5 or D.6.

```
Header:

"Content-Type": multipart/form-data

Body:
{
    "username": String,
    "xlsxFile": File,
    "minTemp": Integer,
    "maxTemp": Integer

}
```

Listing D.5: Commands for POST request with temperature file

```
Header:

"Content-Type": multipart/form-data

Body:
{
    "username": String,
    "data": String
}
```

Listing D.6: Commands for POST request with data as a string

Appendix E

Contents of the CD

This work comes with a CD containing following items:

Code Contains the source code of the ML-based BC Selection prototype. Equivalent to the GitHub repository found at `https://github.com/Raybook90/ml_blockchain_selection`. It also includes the source code of the extended version of the PleBeuS framework with the integrated ML functionality, which can be found at `https://github.com/Raybook90/PleBeuS-Integration`.

Data Contains the jupyter notebooks to generate the figures included in the data acquisition part of the thesis (Twitter data, Google searces, Conference papers, Platform Transaction Speed). Also contains the script used for model training and evaluation, as well as the corresponding database file.

Presentation Contains the slides of the midterm presentation

Thesis Contains the written part of the thesis as PDF file. Also contains the latex source code including the figures and diagrams used in this thesis.