UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

Predictive Modeling for Fair and Efficient Transaction Inclusion in Proof-of-Work Blockchain Systems

Enrico Tedeschi

A dissertation for the degree of Philosophiae Doctor
June, 2023

Abstract

This dissertation investigates the strategic integration of Proof-of-Work (PoW)-based blockchains and ML models to improve transaction inclusion, and consequently molding transaction fees, for clients using cryptocurrencies such as Bitcoin. The research begins with an in-depth exploration of the Bitcoin fee market, focusing on the interdependence between users and miners, and the emergence of a fee market in POW-based blockchains. Our observations are used to formalize a transaction inclusion pattern. To support our research, we developed the Blockchain Analytics System (BAS) to acquire, store, and pre-process a local dataset of the Bitcoin blockchain. BAS employs various methods for data acquisition, including web scraping, web browser APIs, and direct access to the blockchain using Bitcoin Core software. We utilize time-series data analysis as a tool for predicting future trends, and transactions are sampled on a monthly basis with a fixed interval, incorporating a notion of relative time represented by block-creation epochs.

We create a comprehensive model for transaction inclusion in a POW-based blockchain system, with a focus on factors of revenue and fairness. Revenue serves as an incentive for miners to participate in the network and validate transactions, while fairness ensures equal opportunity for all users to have their transactions included upon paying an adequate fee value. The ML architecture used for prediction consists of three critical stages: the ingestion engine, the pre-processing stage, and the ML model. The ingestion engine processes and transforms raw data obtained from the blockchain, while the pre-processing phase transforms the data further into a suitable form for analysis, including feature extraction and additional data processing to generate a complete dataset. Our ML model showcases its effectiveness in predicting transaction inclusion, with an accuracy of more than 90%. Such a model enables users to save at least 10% on transaction fees while maintaining a likelihood of inclusion above 80%. Furthermore, adopting such model based on fairness and revenue, demonstrates that miners' average loss is never higher than 1.3%.

Our research proves the efficacy of a formal transaction inclusion model and

ML prototype in predicting transaction inclusion. The insights gained from our study shed light on the underlying mechanisms governing miners' decisions, improving the overall user experience, and enhancing the trust and reliability of cryptocurrencies. Consequently, this enables Bitcoin users to better select suitable fees and predict transaction inclusion with notable precision, contributing to the continued growth and adoption of cryptocurrencies.

Acknowledgements

To all those who courageously stood by my side and provided unwavering support throughout these years and along this journey, I extend my heartfelt gratitude.

I would like to express my deepest appreciation to my thesis advisor Håvard Dagenborg, and co-advisor Dag Johansen, for their support, invaluable guidance, and insightful suggestions throughout the course of this research. Your expertise and encouragement have been instrumental in the successful completion of this thesis. Thanks to your availability and willingness to assist, I had the privilege of attending several international technical schools, where I received valuable insights and advice to inform my dissertation research. As a foreign student, your guidance provided me with not only technical and academic support, but also personal and cultural insights that proved invaluable throughout my studies. For all these reasons, I consider you to have been my primary point of reference during my academic journey.

I am grateful for the support and camaraderie of the Cyber Security Group (CSG) colleagues at the Arctic University of Norway (UiT) who shared my journey, from the lighthearted moments in the office to the trying times of failed tests and the anxiety of submitting papers for review. I extend a special thanks to my main co-author Tor-Arne Nordmo, whose invaluable knowledge, constant support, and willingness to assist have been instrumental to my success. I would also like to express my gratitude towards my other colleagues for their valuable contributions and for making my time at CSG a memorable experience. Thomas Bye Nilsen, deserves a special mention for the enjoyable table tennis matches we had during our coffee breaks and for his insightful discussions about Norwegian culture and society. Aakash Sharma has been an excellent office mate and a source of laughter with his unique sense of humor, including his joke about his "spiced" Indian heritage and his perspective on being a foreigner in our department, like myself. Aril Bernhard Ovesen, with whom I shared an office and taught a course, has been an excellent collaborator and a supportive colleague. Lars Brenna, with whom I traveled to Los Angeles to present my first publication. His invaluable help and advice were always welcome and precious to me, and our interactions over the course of the trip and beyond have made him a valuable friend. Finally, Bjørn Aslak Juliussen's legal expertise has been an invaluable asset to the CSG, and in the face of the tragedy of the commons, he quickly became my go-to legal consultant. I thank all of my other colleagues, and especially Magnus, Anders, and Andreas, for making my time at CSG a memorable and enriching experience, and I look forward to continuing our collaboration and friendship in the future.

I would like to express my deep appreciation to the Cryptology and Data Security team at the University of Bern for their invaluable contribution to my Ph.D. journey. Specifically, I am grateful for the expert guidance of Dr. Prof. Christian Cachin, whose mentorship helped me expand my knowledge in the fields of blockchain technology, algorithm for distributed consensus, and cyber security. In addition to his exceptional research skills, Dr. Cachin's grounded personality and willingness to be available for his students make him an extraordinary person. His guidance and support have been instrumental to my success, and I am honored to have had the opportunity to work with him. I am grateful for the support and friendship of all my colleagues in this fabulous group, but I would like to give special thanks to Ignacio Amores Sesar. His expertise was fundamental to achieving a valuable publication, and our collaboration has led to a wonderful friendship. Despite his aversion to skiing, we have discovered many shared interests and experiences, such as hikes, board game nights, and trips together, which have made our friendship even more rewarding. I would also like to extend my appreciation to Orestis Alpos for being an excellent skiing buddy in the beautiful Bernese Oberland, and to Jovana Micic and Luca Zanolini for their guidance and tips on exploring Bern. Their support and kindness have made my time here even more enjoyable and memorable.

Thanks to the strong bonds formed within the well-established community of international students and the amiable locals in Tromsø, I have remained in this city far beyond what I initially anticipated. I want to express my heartfelt gratitude to my adventure buddies who have been instrumental in my experiences here. Gigi, you have been an exceptional friend and a constant companion for hiking, skiing, tennis, and partying from the very beginning. Paolo, our delightful conversations, tea sessions, and memorable photography excursions together have meant the world to me. Pietro, your unique personality has brought endless joy and laughter to our adventures. Helge, your friendship has been invaluable, even though you relocated to Milan now, I forgive you for that. Lorenzo, you have always been there as a supportive friend. Mara and Jenny, your joy, kindness, and friendship throughout these years have been essential in keeping me focused and motivated. William, you truly are my favorite drama queen, bringing excitement to every situation. Behrooz, you are are a wonderful person and an incredible flatmate. I must express a hint of sympathy towards you for enduring my challenging presence for such an extended period of time. Ranieri, Greta, and Ilaria, you have been irreplaceable travel companions who have become cherished friends over time.

I would like to take a moment to express my deep appreciation for the exceptional friendships I have cultivated over the years. I firmly believe that friendships between individuals of different genders are not only possible but also incredibly meaningful and invaluable in our lives. First and foremost, I want to extend my heartfelt thanks to Benedetta. Our friendship has encompassed some of the most remarkable years of my life, and it continues to grow stronger every day. You are an incredible person and a dear friend, and I cannot adequately express my gratitude for the positive impact you have had on me. To Elena, our friendship has been invaluable from the very beginning. Your knowledge, unwavering support, and constant presence have played a significant role in shaping the person I am today. I truly consider you one of the most intelligent individuals I have ever had the pleasure of knowing. Your friendship has always been a cherished gift, and I will forever be grateful to you. Please know that you can always count on my unshakable devotion and friendship. I would also like to extend my heartfelt appreciation to my dear friend Julia, whom I had the pleasure of meeting during my first year in Tromsø. Despite the passage of time, our bond remains strong, and your support has always meant the world to me. Francesca, you hold a special place in my heart. The semester we spent together was truly remarkable, and every time we reunite, it feels as though time has not passed at all. Your kindness, honesty, and captivating smile make you someone truly delightful to be around. Being in your presence brings immense joy, and I feel incredibly fortunate to have you as a friend. Michela, your joy and enthusiasm for Tromsø greatly contributed to my enjoyment during those initial months. Our friendship has endured, and you hold a special place among my most treasured friends. Lastly, I want to express my deepest gratitude to my dear friend Sofia. Your invaluable advice, contagious joy, and enthusiasm provided immense support during the final stages of my PhD journey. Your presence in my life has been truly transformative, and I am genuinely grateful for everything you have contributed. To Benedetta, Elena, Julia, Francesca, Michela, and Sofia – each of you has made an immeasurable impact on my journey. I want to express my heartfelt appreciation for being part of my life and for the profound influence you have had on me. Thank you for your friendship and the incredible memories we have shared.

During my time in northern Norway, I had the incredible opportunity to indulge in my passion for photography as a northern lights enthusiast. Embracing the cold nights and venturing into the icy wilderness provided an occasional escape from my studies and work. I am deeply grateful to my initial colleagues, Alessandro and Truls, who not only shared their expertise but also became cherished friends. Gratitude also extends to the entire guiding community and the great individuals I had the pleasure of meeting, including Nicola, Cecilia, Hannah, Jeff, Per, and Raquel. Their warm presence and unwavering support enriched my experience beyond measure.

My journey would have been impossible without the support of my friends in Milan. Despite the distance, their enduring presence was indispensable. I am grateful to those I grew up with and my university friends: Susan, Debora, Francesca, Veronica, Matteo, and Stefano. Their constant support fueled my strength and determination. I also would like to express my heartfelt gratitude to the individuals in the "non belligeranti" group, my dear friends and closest allies, who have been by my side since our days in primary school. Tommaso, your unwavering friendship and frankness have always been a constant source of support, and I truly appreciate your presence. Francesco, you embody the essence of being "non belligerante" within our group, and your friendship has been invaluable throughout the years, and being a physiotherapist, I know I can always count on you if I ever need a massage. Andrea, as our professional flaker, your rational nature and friendship have undeniably shaped my journey. I must express my gratitude for your consistently unpredictable presence and the valuable advice you've provided, which has surprisingly contributed to my success. Simone, my dear friend and cousin, thank you for beating us in every video game we've played together, this has humbled us all and it has been a valuable learning experience that has kept us on our toes and pushed us to improve. Your presence throughout my entire life has been of utmost importance, providing unwavering support and companionship. Lastly, I want to express my deepest gratitude to Bonny, my best friend since we were three years old. Your kind-hearted nature and caring attitude make you the epitome of an ideal friend. Although you may have recently stepped back from the gaming realm due to Andrea's dominance, your friendship remains unmatched. Thank you for being the best friend one could ever imagine.

Last but certainly not least, I want to convey my deepest and warmest appreciation to my family, my father Franco and my mother Antonella. Their constant support, both emotionally and financially, during my studies has been invaluable. They have not only provided for me but also instilled in me a strong set of values including respect, kindness, fairness, integrity, open-mindedness, generosity, and love. Their kindness and love have touched my heart profoundly, they have never opposed my decisions but instead wholeheartedly supported

me in realizing my dreams, and for this, I am eternally grateful. I also want to extend my heartfelt thanks to my brother, whose soundtracks have accompanied me on this incredible journey. Your music has brought joy and motivation to every step of the way.

Finally, I want to express my gratitude to myself for persevering, staying resilient, and never giving up on my dreams. Throughout this journey, I have faced numerous challenges and obstacles, but I have remained steadfast in my belief in my abilities and the path I have chosen. I am grateful to myself for the courage to take risks, the determination to overcome setbacks, and the belief in my own potential. Through moments of self-doubt and uncertainty, I have found the strength within me to keep pushing forward and striving for excellence.

Contents

List of Figures

List of Tables

List of definitions

List of Abbreviations

aBFT asynchronous Byzantine Fault-Tolerant
ACL Access Control List
ACM Association for Computing Machinery
AI Artificial Intelligence
ANN Artificial Neural Network
API Application Programming Interface
ASIC Application Specific Integrated Circuits
AUC Area Under Curve
AUC-ROC Area Under Curve of Receiver Operator Characteristic
BA Byzantine Agreement
BAS Blockchain Analytics System
BBChain Efficient Trustworthy Computing with Blockchains and Biometrics
bct block creation time
BFT Byzantine Fault Tolerant
BNE Bayesian Nash equilibrium
BTC Bitcoin Currency
CA Certificate Authority
CBDC Central Bank Digital Currency
CDF Cumulative Distribution Function
CSG Cyber Security Group
CSV Comma-Separated Values
DAG Directed Acyclic Graph
DDoS Distributed Denial of Service
DLT Distributed Ledger Technology
DNN Deep Neural Network
DoS Denial of Service
EBI Epoch Before Inclusion
EM Expectation–Maximization
ETH Ether
EU European Union
EULA End-User License Agreement
EVM Ethereum virtual machine
FBFT Federated Byzantine Fault Tolerance

FNR False Negative Rate

FPGA Field Programmable Gate Array

FPR False Positive Rate

FPSBA First-Price Sealed-Bid Auction

GCS Group Communication System

GRASP General Responsibility Assignment Software Patterns

GSP Generalized Second Price

HMAC Hash-based Message Authentication Code

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IFC Information Flow Control

JSON JavaScript Object Notation

kNN k-Nearest-Neighbors

LF Latency Function

ls linearly solvable

LSM Least Squares Methods

MAE Mean Absolute Error

ML Machine Learning

MPE Miner Power Efficiency

MPF Miner Profit Function

MTS Multivariate Time Series

nls non linearly solvable

NN Neural Network

NP nondeterministic polynomial time

OPT Ordered Pending Transaction

P2P Peer-to-Peer

PBFT Practical Byzantine Fault Tolerance

PKI Public Key Infrastructure

PoA Proof-of-Activity

PoAu Proof-of-Authority

PoB Proof-of-Burn

PoET Proof-of-Elapsed-Time

PoR Proof-of-Replication

PoS Proof-of-Stake

PoSp Proof-of-Space

PoSt Proof-of-Storage

PoSTi Proof-of-Storage-Time

PoW Proof-of-Work

ReLU Rectified Linear Unit

ResNet Residual Neural Network

RF Random Forest

RLP Recursive Length Prefix
ROC Receiver Operator Characteristic
RPOW Reusable Proof-of-Work
RPT Relapsed Pending Transaction
SPA Second-Price Auction
SPV Simplified Payment Verification
SSL Secure Socet Layer
SVM Support Vector Machines
TAT Temporarily Approved Transaction
TCB Trusted Computing Base
TNR True Negative Rate
TPR True Positive Rate
UiS University of Stavanger
UiT the Arctic University of Norway
UPA Uniform-Price Auction
URI Universal Resource Identifier
UTC Universal Time Coordinate
UTM Universal Transverse Mercator
UTXO Unspent Transaction Outputs
VDF Verifiable Delay Functions
VM Virtual Machine

“The relative success of the Bitcoin proves that money first and foremost depends on trust. Neither gold nor bonds are needed to back up a currency.”

Arnon Grunberg, Writer

1 Introduction

Digital money and cryptocurrencies have revolutionized payment systems, making monetary transactions cheaper and faster. These digital currencies, rooted in public-key cryptography and digital signatures, offer robust authentication and secure algorithms for confidential and non-repudiable digital communication. They can be *centralized*, where institutions or banks control the supply as Central Bank Digital Currencys (CBDCs), or *decentralized*, regulated by a network of users through consensus. Decentralized digital currencies encompass cryptocurrencies and online tokens issued by anyone, operating independently of specific financial security mechanisms.

The underlying data structure of most cryptocurrencies is called a *blockchain*. Blockchains are distributed databases that do not require a central trusted party to operate. They are append only, immutable, and experts believe that they will disrupt many industries, from finance [1, 2] and law [3] to healthcare [4] and education [5, 6].

Cryptocurrencies, such as Bitcoin and Ethereum, have emerged as the main assets in the digital currency landscape. Bitcoin and Ethereum secure around 80% of the total cryptocurrency market cap, as of mid-2020 [7]. Bitcoin, in particular, was designed to provide users with a low-cost payment scheme, with transaction fees initially close to zero [8]. However, the increasing popularity of cryptocurrencies like Bitcoin has revealed scalability challenges, with the underlying POW scheme posing limitations on transaction throughput [9, 10].

These challenges highlight the importance of exploring transaction fees, fee markets, and scalability issues within cryptocurrencies like Bitcoin. By addressing these issues, we aim to enhance the efficiency and cost-effectiveness of transactions within the Bitcoin network, improving the overall user experience and facilitating wider adoption of digital currencies.

In this thesis, we address three main aspects related to blockchain technology: (1) fee mechanisms and overpaying; (2) the incorporation of ML models into POW-based blockchains at scale; (3) optimization of user experiences and trust of such systems. Our contributions stem from the analysis of the largest POW-based blockchain implementation, Bitcoin,[1] by examining fee trends, mining-related monetary costs, and the subsequent fee markets. We introduce a novel formal model, implemented using ML, to investigate patterns and comprehend fee complexity in the context of large-scale blockchain systems. The practical application of this scheme holds the potential to render blockchain technology more cost-effective and enhance user trust in the system by making it cheaper to use.

1.1 Blockchain Consensus Types

The pioneering electronic payment system developed in the 1990s by David Chaum, Digicash [11], provided payments without the need for a trusted third party. While such invention was innovative and ahead of its time, Digicash faced challenges in gaining widespread adoption and ultimately filed for bankruptcy in 1998. The onset of decentralized digital currencies began when the still anonymous character of Satoshi Nakamoto released in 2008 the Bitcoin paper [8]. Bitcoin's underlying technology challenged the efficiency of traditional financial systems. Digital signatures enable trust among participants, while to prevent double spending Bitcoin uses a Peer-to-Peer (P2P) network that timestamps transactions into an ever-growing chain of hash-based blocks. Each block must carry an evidence, or *proof*, of the *work* that has been carried out while creating it, such that the consensus is reached by the largest pool of CPU power. Each block is immutable and similarly to hashcash [12], the cost-function used in the Bitcoin consensus computes a token which can be used as a proof of work. This function is efficiently verifiable, but expensive to compute alongside

1. Throughout this study, when we talk about Bitcoin blockchain we refer specifically to BTC

customizable parameters, in order to prevent double spending and Distributed Denial of Service (DDOS) attacks. The Bitcoin blockchain is then secured with a consensus mechanism called Proof-of-Work (PoW).

POW is a decentralized consensus mechanism that requires members of a network to solve mathematical puzzles before agreeing on any decision, so to deter malicious use of computing power, e.g., sending spam emails or launching DDOS attacks. This idea was firstly conceived in 2004 by the developer Han Finney, when he implemented Reusable Proof-of-Work (RPOW), using hashcash and RSA-signed tokens [8, 13]. In 2009 Bitcoin became the first widely adopted implementation of a POW scheme, and Finney was proven to be the recipient of the very first Bitcoin transaction [14]. Systems based on POW proved to be secure, and decentralized in terms of consensus and governance. However, they also demonstrated to be inefficient in terms of throughput (tps), and they struggle to provide cheap fees. Furthermore, the equipment needed to secure the network, used by the so called *miners*, is powerful and expensive. This increase system's overall energy consumption, miners want to maximize their revenue, and this impacts *how* transactions are included in the next mined block, and eventually, in the blockchain.

In response to ameliorate POW drawbacks, different classes of consensus mechanism for blockchain have been designed, and the deep-seated monetary value for each information exchange, made the cryptocurrency domain well suited for the advance of new blockchain systems. Ouroboros [15], Casper [16], Tendermint [17, 18] and Ppcoin [19], want to reduce POW computational costs, and implement a Proof-of-Stake (POS) consensus mechanism, which selects validators in proportion to their quantity of holdings. Algorand [20], Ripple [21, 22], and Stellar [23] implement a Byzantine Agreement (BA)-based blockchain which aim is to improve POW low latency, by using a consensus mechanism that varies from the traditional Practical Byzantine Fault Tolerance (PBFT). Avalanche [24] does not provide total order of transactions, and it builds instead a Directed Acyclic Graph (DAG)-chain which keeps strong probabilistic safety guarantee in the presence of Byzantine adversaries, while its Byzantine Fault Tolerant (BFT) nature enables it to achieve high throughput and scalability. Filecoin [25, 26] implements a Proof-of-Storage (POSt) consensus mechanism. The system turns cloud storage into an algorithmic market, and the network is secured by miners. They make profits by providing storage to clients.

More blockchain protocols have been proposed and studied, such as Proof-of-Elapsed-Time (POET) [27], Proof-of-Burn (POB) [28, 29], Proof-of-Activity (POA) [30], Proof-of-Space (POSp) [31] using Verifiable Delay Functions (VDF) [32, 33], Proof-of-Authority (POAu) [34, 35], Proof-of-Storage-Time (POSTi) [36], Proof-of-Replication (POR) [37] whose sets the origin to Filecoin's POSt, and many more. Analysis on such systems is made easier by the public and accessible nature of blockchains, and the amount of data available is directly proportioned to their popularity.

1.2 Pattern Recognition and Big Data

The inherent append-only property of blockchains leads to accumulation of data. Information stored in blockchains is publicly accessible and easy to retrieve. This sets up a disposition towards big data analysis, and it opens up for technological progress derived by the study of *patterns* and ML modeling [38]. ML is a branch of Artificial Intelligence (AI) that enables programs and algorithms to *self-learn* and *detect certain patterns in large datasets*. Pattern recognition is the discipline whose goal is to classify data or objects into certain classes or categories [39], it has applications in statistical data analysis [40, 41], signal processing [42], image analysis [43, 44], information retrieval [45], and due to the increased availability of large datasets, ML. We will use this automated mechanism of decision making during our research and throughout the whole thesis.

The opportunities and challenges of ML and big data are subject of study [46, 47, 48], and prediction models are used to foresee future events when enough data are available [49, 50]. Furthermore, ML applications span from healthcare [51, 52] and solar energy [53], to financial market predictions [54, 55], and yet not much attention has been given to the blockchain and cryptocurrency domain. This thesis analyzes different blockchain technologies, such as POW-based like Bitcoin, or POS and DAG-based like Avalanche. We conduct an extensive and longitudinal study on Bitcoin, where a consistent amount of data and information is retrieved, in order to detect main POW drawbacks. Furthermore, we evaluate separately, a security analysis on Avalanche. Our main contribution is the study and formalization of the patterns that govern miners' decisions in POW-based blockchains, particularly Bitcoin. Finally, we use this pattern formalization to build a ML model that can predict transaction inclusion in Bitcoin, while optimizing fees and overpaying.

1.3 Thesis Statement

The high cost of mining has led to an increased usage of transaction fees as a means for miners to make a profit. This led to serious economic implications for users, fee *overpaying* became the norm, and many fee estimators started to adopt higher fees than necessary [56].

We conjecture that by combining public data of Bitcoin with ML models, we can establish a *transaction inclusion pattern* for POW-based blockchains. This pattern allows users to predict transaction inclusion, optimize fees, and improve efficiency. By studying miners' decisions we aim to enhance trust, reliability, and utility in cryptocurrencies. It follows the thesis of this dissertation:

Our thesis is that Bitcoin transaction fees can accurately be modeled and predicted using ML methods, improving utility and efficiency for clients using such cryptocurrencies, while maintaining a fair compensation for miners.

To elaborate our thesis we initially conduct a longitudinal study on Bitcoin, so to evaluate a large amount of data, and how the system behaves in the event of network saturation, price fluctuations, and transaction fee variations. After building knowledge on POW systems at scale, we need to formally define our view on *transaction inclusion*, in order to use such insights to exploit a ML model at best.

To measure our scope's success, a thorough analysis and evaluation of the ML model needs to be carried out. For that, we will use statistic and probabilistic metrics, and we will define the evaluation set up used in this dissertation, so to make results easily replicable. Our thesis recognizes the central role of POW-based blockchains in matter of security, distributed governance, and cheap, instant transactions all over the globe, therefore our evaluations and results will consolidate the use of such systems, in cryptocurrency domain and elsewhere.

Furthermore, considering the inherent nature of POW-based systems, our inclusion pattern study will be easily adaptable to other POW-based blockchains, facilitating their use at scale, and helping users at not getting overburdened of fees.

1.4 Scope and Limitations

To focus on our thesis statement, we provide a clear and concise description of what the study aims to accomplish and outline the boundaries and constraints of the research. Following, we define research objectives, scope, time constraints, limitations, generalization, and ethical considerations.

1.4.1 Research Objectives and Scope

Research objectives of this study include:

- Investigate the strategic integration of POW-based blockchains and ML models to develop a transaction inclusion pattern for improving utility and cost-efficiency in cryptocurrencies like Bitcoin.
- Explore the Bitcoin fee market and understand the interdependence between users and miners.
- Develop the BAS for acquiring, storing, and pre-processing a local dataset of the Bitcoin blockchain, utilizing various data acquisition methods and time-series data analysis.
- Create a comprehensive model for transaction inclusion in POW-based blockchain systems, emphasizing factors of revenue and fairness. Revenue serves as an incentive for miners, while fairness ensures equal opportunity for users upon paying an adequate fee.
- Construct an ML architecture consisting of an ingestion engine, pre-processing, and ML model to predict transaction inclusion, with a focus on feature extraction and data processing to generate a complete dataset.
- Evaluate the effectiveness of the ML model in predicting transaction inclusion, including accuracy, cost savings on transaction fees, and likelihood of inclusion.
- Enable Bitcoin users to make informed decisions by selecting suitable fees and predicting transaction inclusion with precision, contributing to the growth and adoption of cryptocurrencies.

The scope of this dissertation focuses on the model for transaction inclusion within POW-based blockchains, with a specific implementation and testing conducted solely on the Bitcoin network, and specifically, BTC. While the ML model has the potential for use on a global scale, it is important to recognize that its full adoption may bring changes in the way miners include transactions. However, its direct application and testing on a global scale are beyond the scope of this study. The scope includes an exploration of the Bitcoin fee market, which also holds relevance for other POW-based solutions. By investigating transaction inclusion dynamics, the study aims to enhance user experience, contribute to the trust and reliability of cryptocurrencies, and ensure miners continue to receive transaction fee rewards.

1.4.2 Time Constraint and Limitations

The data collection period for the primary findings of this dissertation, involving data processing and refinement, was conducted from January 2021 to May 2021. Monthly information on Bitcoin price, transaction fees, and miner consumption was collected monthly to capture the historical trend of the key factors influencing the study. The research project, including literature review, hypothesis formulation, data collection, and system building, spanned from 2018 to 2021. Following, our focus shifted towards presenting our findings [57, 58], and subsequently compiling this dissertation.

Limitations

- In the free and decentralized market of Bitcoin we observe a transaction fee price uptrend, caused by the cost of mining and the fear of "51% attack". In such scenario, we assume that users ought to chose a desired fee, based on their transaction's total amount and network congestion, without fearing of being left out from miners. Therefore, we adopt a solution where users can select their own fee, while monitoring their transaction confidence of being accepted in the next mined block.

- A rational miner should prioritize transactions based mainly on its revenue. Considering that the reward for mining new blocks is halved periodically, we assume that miners will keep on enforcing this behavior in the long run. We therefore consider miners to be *rational agents* in a POW-based ecosystem, as they need to establish a *Nash equilibrium* [59] among themselves in order to approach complex decisions. Our solution works in the presence of such rational miners, which we conjecture to be present in every POW-based systems.

- A Nash equilibrium should be kept also between users and miners. In order to understand how users and miners interact to approach complex solutions, we assume that despite miners ought to be greedy as the network scales and mining costs rise, users should be deterred in leaving the system. We design for this purpose a model where transactions' waiting time is taken into account, and miner's choices cannot depend by greediness alone.

- The public nature of blockchains facilitates data retrieval and analysis. For our purpose, while adopting ML models to focus on our thesis statement, we adopt a *supervised learning* approach, and therefore, we assume blockchain data to be always available and retrievable. In this way, the patterns we define will establish the guidelines for training ML models, while newer data will shape the model's outcome during time.

1.4.3 Generalization

The findings and conclusions of this research study hold implications for generalization. While the specific focus of the study is on the strategic integration of POW-based blockchains and ML models for transaction inclusion in Bitcoin, the insights gained from this research may have broader applicability. Although the study is conducted within the specific context of Bitcoin, the underlying principles and methodologies explored in this study can potentially be generalized to other POW-based blockchain systems. The examination of the Bitcoin fee market and the development of the transaction inclusion model contribute to a deeper understanding of the dynamics and factors affecting transaction inclusion in POW-based blockchains more broadly.

Each blockchain network have unique characteristics, and the specific implementations and considerations within different systems may vary. However, the insights gained from this research can serve as a foundation for further investigations and can inform the development of transaction inclusion strategies in other POW-based blockchain networks. The generalizability of the research findings beyond the specific case of Bitcoin will depend on various factors, including the similarities in the underlying blockchain architecture, consensus mechanisms, and transaction dynamics across different blockchain systems. Further studies and empirical validations in diverse blockchain environments are necessary to assess the extent to which the findings and models developed in this research can be generalized.

1.4.4 Ethical Concerns

Although we acknowledge the ethical concerns associated with POW and its potential environmental impact, our dissertation does not aim to question the applicability or sustainability of POW. Instead, our study focuses on a widespread phenomenon related to blockchain technology, recognizing that alternative consensus mechanisms may exist that offer solutions to the ethical concerns associated with POW.

1.5 Methodology

The empirical approach of acquiring knowledge about a certain phenomenon, physical or ethereal, is the *scientific method*. This approach is characterized by careful observations and systematic study of the subject, followed by hypothesis formulation via *inductive reasoning*, experimental and measurement-based testing of *deductions*, and finally, refinement of the hypothesis based on the experimental findings. Conclusions reported at the end of the process are fundamental for hypothesis formulation in a new research cycle.

Natural sciences follow the *hypothetico-deductive model* [60], where the scientific inquiry is carried out by formulating hypothesis that can be logically contradicted by an empirical test. Debates on whether computing can be considered a science or not still animates many [61], and only recently was considered to be a natural science, and not only a subject of the artificial [62]. In 1989, the final report of the ACM Task Force on the Core of Computer Science [63] presented a new taxonomy for classifying computing as a science. They root the field's research in three main paradigms, *theory*, *abstraction*, and *design*.

Theory roots its fundamentals in *mathematical* sciences, and it describes objects or events whose properties can be defined using logical reasoning. It is characterized by four steps:

1. *Definition* of the studied objects.
2. Make a *theorem* on their possible relations.
3. *Proof* relations validity.
4. Interpret results.

A glaring example is the study of algorithms, where given an ample set of descriptions, hypothesis can be proven using logical reasoning.

Abstraction roots its fundamentals in *natural sciences*, where the notion of scientific progress is achieved by systematically verify and validate certain hypothesis, about a specific phenomena. It is characterized by four main steps:

1. Form a *hypothesis*.

2. Construct a *model*, and make *predictions*.
3. Collect data.
4. Analyze results.

The studied object can span from biological or chemical processes, to physical phenomena, down to the holistic behavior of a complex computer system. The abstraction paradigm goal is to construct accurate models of the rules and laws that govern the behavior of observable phenomena. The model is evaluated by comparing its predictions to experimentally collected data, and it can be used to predict certain behavior in circumstances that have not been observed yet.

Design roots its fundamentals in *engineering*, and it uses a systematic approach to construct a system that solves a given problem. It is characterized by three main steps:

1. Define *requirements* and *specifications* based on a series of observations.
2. Design and implement a *prototype system* based on such specifications.
3. Build a set of *experiments* to evaluate such system, by following the stated requirements.

Engineers expect to iterate these steps, and they share the notion that progress is achieved by designing a system that solves a posed problem.

In computing these three paradigms are often intertwined [63], and researchers generally resort to all three paradigms to varying degrees. The work presented in this dissertation initiates with a theoretical nature. The studied subject needs a formal definition before any abstraction or design fundamentals are applied. We use theory to define mathematically the Bitcoin market, and a possible miners inclusion pattern. Definitions must be rigorous and specific, before any hypothesis is expressed. We rely on well-established theory regarding economical principles and market in Bitcoin, including POW properties and limitations.

The rational and systematic approach used in this dissertation involves a *top-down* scheme where, once the broader theory is defined, we use abstraction to form hypotheses for narrowing down our subject of study. We make assumptions and construct a model that will be evaluated and analyzed after collecting and storing information. Big data analysis and longitudinal studies are key parts of this monograph, allowing us to focus our attention in a narrower perspective for building a ML-based system prototype, and establishing a *proof-of-concept* [64] of it.

Theory and abstraction applied in earlier stages of this study set the backbone for our system prototype implementation. Based on collected data and the theoretical knowledge acquired, we apply design principles to define requirements of such system. We refine the proof-of-concept in order to build our ML-based prototype. Afterwards we construct a set of experiments to evaluate such system, including different metrics to verify its *performance* and yielding evidence for *usability* in a real world scenario, although its proof lay outside the scope of this dissertation. Such performance metrics include theory of pattern recognition, information retrieval, and classification in ML.

This dissertation focuses on deriving principles that govern POW-based markets and miners decisions, when such systems scale. Within our research, we use a top-down approach that focalize in building a prediction model for transaction inclusion. Results are subject of a process of continuous refinement, and the model we build is not final nor static. Empirical measurements and practical experiences might challenge initial assumptions when new network statuses or events occur. Ultimately, we aim to set the baselines for studying such systems at scale, by formalizing market fundamentals and a plausible inclusion model.

1.6 Research Context

This dissertation has been carried out in the context of CSG[2] at UiT. The project has been founded by the Research Council of Norway, together with BBChain[3] project, and Nofima.[4] Studies on blockchain security have been conducted in the context of Cryptology and Data Security Research Group at the University of Bern.[5] The dissertation here presented relates with previous scientific studies done in the CSG [65, 66], and to place this dissertation in the right context, a brief overview of previous works is surveyed.

The CSG is investigating fundamental system problems rooted in practical application domains. The group undertakes high-impact interdisciplinary research and innovation at the intersection of computer science, law, sports science,

2. Cyber Security Group at UiT, Norway `https://site.uit.no/arcsecc/`
3. BBChain at University of Stavanger (UiS) `https://bbchain.no`
4. The Norwegian food research institute `https://nofima.com`
5. Cryptology and Data Security Research at University of Bern, Switzerland `https://crypto.unibe.ch`

psychology, statistics, and medicine, and it is now also focused on blockchains and big data analysis. Early works of the CSG include creating streaming infrastructure for football [67], devising a mobile agent middleware architecture for supporting distributed applications in a wide-area network [68], optimizing adaptive streaming and video composition over HTTP [69], and proposing a mechanism for expressing and enforcing security policies for shared data [70]. The cutting-edge and innovative nature of CSG, allowed to explore P2P and early-blockchain domain since 2006 with Fireflies [71], a leaderless BFT consensus protocol. The group research focuses also on AI solutions. One example is monitoring and surveillance in privacy-sensitive and unstable offshore environments. For that purpose, systems for executing distributed AI applications on the edge has been built [72, 73]. Further works using AI have been deployed, Dorvu [74] is a digital platform for real-time privacy-preserving sustainability management in the domain of commercial fishery surveillance operations, and it implements distributed artificial intelligence algorithms on mobile. The group expertise in ML solutions is fundamental for the purpose of this research.

The BBChain project aims to combine blockchain and biometrics to build a privacy-preserving and fault-tolerant public database of digitally authenticated documents. This to enable academic degree certificates to be issued and verified from any place in the world with a high degree of trust [75]. They also use a permissioned blockchain to form verifiable contracts between clients and storage providers [76]. Their knowledge in the blockchain domain is the key for a valuable background education and backbone knowledge for this dissertation.

Nofima is a leading food research institute that explores and develops for the aquaculture, fishing, and food industries. Their projects include using blockchain technology for traceability in the food industry [77, 78, 79], and studying blockchain applicability for enterprises dealing with aquaculture, fisheries, and agriculture, providing a farm-to-consumer holistic view. Their competence in blockchain technology applied in real world scenarios exhibits an indispensable resource of this dissertation applicability.

During the exchange in Bern, at the Cryptology and Data Security Research Group, we surveyed the Avalanche protocol and analyzed its security [80]. The group research is notorious for addressing cryptographic protocols [81], distributed consistency, consensus [82, 83, 84], and cloud-computing security [85], with applications to blockchains [86], distributed ledger technology, cryptocurrencies [87, 88], and their economics. The group's theoretical knowledge in the area of blockchain consensus mechanisms and cryptography, is fundamental to deepen security constraints of such technology, and to understand protocol limitations even under non-optimal circumstances.

1.7 Impact

This research has far-reaching implications for the rapidly evolving field of cryptocurrencies and the broader financial landscape. By providing a detailed understanding of the transaction fee mechanism in Bitcoin and developing a ML model that accurately predicts transaction inclusion, our work contributes to enhancing the trust, efficiency, and utility of cryptocurrencies for end-users. The insights gained through this research have the potential to improve user experiences, enabling them to make more informed decisions about fee selection and increasing their confidence in the likelihood of transaction inclusion in the next mined block.

Furthermore, our findings may guide future research, policy-making, and the development of practical applications, ultimately fostering the growth and adoption of cryptocurrencies in various sectors of the global economy. This study, therefore, represents a significant step towards bridging the gap between the theoretical understanding and practical application of cryptocurrencies, paving the way for a more accessible and efficient financial ecosystem.

1.8 Summary of Contributions

The major contributions of this dissertation are based on the publications listed in Appendix B. From a longitudinal study on Bitcoin [65, 66], we made assumptions and conjectures about *correlation* between transaction fees and latency. We refined our linear regression approach to a more accurate ML-based one [57]. After that we formalized and defined a pattern for transaction inclusion [58], and gathered information in our dataset with a mechanism based on block-epochs [89], useful for increasing prediction accuracy. Following, is a brief summary of each contribution:

Longitudinal Study and Initial Analysis

We identified three main issues of POW-based blockchains as (1) *scalability*, (2) *performance*, and (3) *costs*. We conducted a longitudinal analysis on Bitcoin to study relations between transaction *fees* and *latency*. We showed how scalability affects performance, then how costs and fees are dependent on them both. Using polynomial interpolation we defined two functions for characterizing fee/latency relation. We stated that applications can improve messaging latency by paying transaction fees, although overpaying does not always improve transaction latency. In our future works we investigated the reasons for that, thus opening our research to a careful study on Bitcoin market ecosystem, and a more thorough approach on prediction models.

Machine Learning Approach

Using data from previous longitudinal studies, we explored the Bitcoin market ecosystem and made some conjectures on what could cause the latency to drastically increase. We analyzed the *first-price* auction market in Bitcoin, and presented a novel ML model solving a binary classification problem, that can predict transaction fee volatility in the Bitcoin network so that users can optimize their fees expenses and the approval time for their transactions. A feature that we included provides information on how many bytes were already occupied by other transactions in the mempool, assuming they are ordered by fee density in each mining pool. With such information, the ML model could predict transaction inclusion with an accuracy of 86%.

Model Formalization

In order to be more accurate in predicting transaction latency, we formally defined a novel inclusion model that describes the mechanisms and patterns governing miners decisions to include individual transactions in the Bitcoin system. We abstracted and defined concepts like *fairness* and *revenue*, adding new definitions and approaches for increasing prediction accuracy. We also defined a new method for storing local dataset of the Bitcoin blockchain. The approach we followed is based on a block-epoch collection of transactions. Each transaction has time-based information according on when it was observed. And the time metric used is the block creation time (or block epoch). Using this model and data collected, we devised a novel ML approach to predict transaction inclusion, with an overall accuracy of 91%.

Dataset Definition

The dataset we built stores part of the Bitcoin blockchain. Blocks are sampled every month and information about transactions and blocks were separated to save disk space and avoid redundancies. This dataset was used to generate a ML model that predicts transaction inclusion. Information is stored to generate all the necessary features for the ML model, and one transaction can be represented as a multivariate time series, based on the block-epoch approach we have previously described.

Security Analysis

Our study on blockchain technologies adjourns with an orthogonal project whose aim is to study security of a newer protocol. We inspect the DAG-based chain of Avalanche [24]. Theoretical studies outlined how Avalanche lacks a complete abstract specification and a matching formal analysis. To address this drawback, we presented a detailed formulation of Avalanche through pseudo-code. Furthermore, we performed an analysis of the formal properties fulfilled by Avalanche in the sense of a generic broadcast protocol that only orders related transactions. And finally, the security analysis revealed a vulnerability that affects protocol liveness. Despite the considerable investment of time and effort dedicated to researching the security of Avalanche, culminating in a published paper [80], the content of this research is not incorporated into the main body of this dissertation. Instead, the paper and its findings can be found in Appendix B, accompanied by a concise introduction to DAG-based blockchains in Section 2.2.2.

1.9 Outline

In this dissertation, we first provide a background on cryptocurrencies, blockchains, and ML in Chapter 2, which lays the foundation for the subsequent chapters. Following this, Chapter 3 discusses the principles and rules governing the Bitcoin ecosystem at scale, specifically focusing on the interdependence between users and miners, as well as the emergence of a fee market in POW-based blockchains. This chapter establishes the importance of understanding the fee market for formalizing a transaction inclusion pattern.

In Chapter 4, we present the Blockchain Analytics System (BAS), which we developed for acquiring and storing a local dataset of the Bitcoin blockchain. We explain the data acquisition methods and techniques, as well as the structuring and pre-processing of the data for further analysis. In Chapter 5, we discuss our approach using time-series data analysis as a tool for predicting future trends, specifically focusing on the factors of revenue and fairness in a comprehensive model for transaction inclusion in POW-based blockchain systems.

Chapter 6 presents the ML architecture used in our study, covering the ingestion engine, pre-processing stage, and the ML model itself. This chapter details how raw data is transformed and processed to be suitable for analysis, including feature extraction and training set generation. In Chapter 7, we evaluate the ML model and discuss the datasets used for training and testing, the evaluation metrics employed, and the results of the analyses. The findings of this study demonstrate the efficiency of our ML model in predicting transaction inclusion and its potential as a powerful tool for end-users, offering significant savings in transaction fees.

Chapter 8 delves into a discussion of our findings, exploring the implications and potential applications of our study. We reflect on the limitations of our work and suggest areas for future research to further refine and expand upon our model. In the final chapter, Chapter 9, we present our concluding remarks, summarizing the key takeaways from the research and emphasizing the significance of our ML model for predicting transaction inclusion in the Bitcoin blockchain. Our study not only contributes to the understanding of the fee market and transaction dynamics but also showcases the potential benefits for end-users in terms of transaction fee savings.

2 Background

This chapter provides an overview of blockchain technologies and their applications in cryptocurrencies. The history of blockchain technology will be discussed, including the transition from centralization to decentralization and distribution. The various types of Distributed Ledger Technology (DLT) will be compared and the advantages and disadvantages of using blockchains in different business sectors will be examined. The consensus protocol of Bitcoin will be discussed, along with its limitations at scale and alternative consensus mechanisms used in cryptocurrencies. Additionally, the role of ML in this research will be explained, including the specific ML models that have been adopted.

2.1 Blockchains

It is important to note that the terms blockchain and cryptocurrency are often used interchangeably, but they refer to distinct concepts. A distinction between the two can be made as follows:

A *blockchain* is a DLT that allows multiple nodes to maintain and share a consistently replicated and verifiable record of data, without the need for central administration or governance.

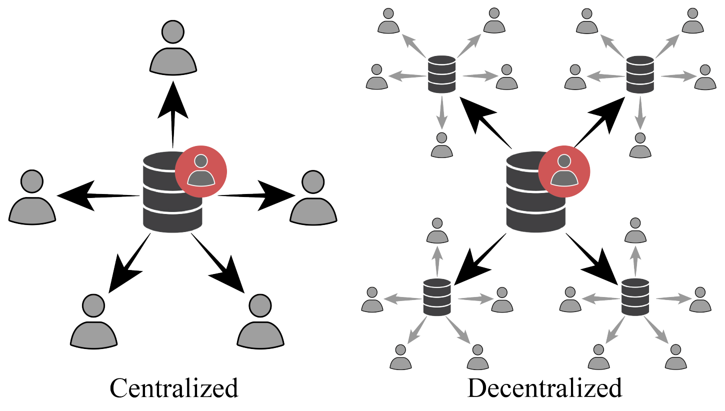


Figure 2.1: In a centralized database system, data is stored on a single point and all workload is placed on a single node. The authority to manage the database is held by a single entity (red user). On the other hand, in a decentralized database system, a consistent and replicated view of the data is maintained across multiple nodes, allowing for users to be geographically distributed. However, the governance of the decentralized database is still centralized.

A *cryptocurrency* is a decentralized digital currency that is used as a medium of monetary exchange through a network of computers. It is not controlled by any central authority.

Blockchain technology enables the creation of systems for storing distributed data without relying on a central trusted authority. Cryptocurrencies utilize this technology to facilitate the exchange of information based on the principle of distributed trust inherent in blockchains.

2.1.1 Centralization and Decentralization

Before discussing DLTs and blockchains, it is useful to provide an overview of centralized and decentralized systems (showed in Figure 2.1), particularly databases, as the main function of blockchains is to securely and permanently store data while maintaining eventual consistency.

Centralized A centralized database is stored and maintained on a single location, often used by an organization or institution that also holds governance over it. Data can be added, modified, or deleted by the central authority. Users must connect to the sole available node to access the database, which has the advantages of less duplication, data integrity, and ease of organization.

Decentralized A decentralized database is a system in which data is stored and replicated across multiple physical locations. This architecture provides increased fault tolerance by eliminating a single point of failure. The central authority maintains governance rights in a manner similar to that of a centralized database. This type of system has the potential to accommodate a larger number of users, even those who are geographically dispersed, as depicted in Figure 2.1.

The governance of distributed systems refers to the process of establishing and maintaining the legitimacy of decision-making within the system [16]. In traditional centralized and decentralized systems, this process is typically carried out by a trusted authority or small group of reliable parties, with hierarchical structures and control maintained through legal systems. In a distributed environment, governance is distributed equally among participants, following rules established by the consensus algorithm.

2.1.2 Distribution and DLTs

The primary characteristic of DLTs is distribution. This refers not only to the Peer-to-Peer (P2P) gossip-based substrate messaging protocol, but also to the governance and decision-making process for preserving data consistency and integrity. An illustration of the distribution of governance in a DLT network is shown in Figure 2.2.

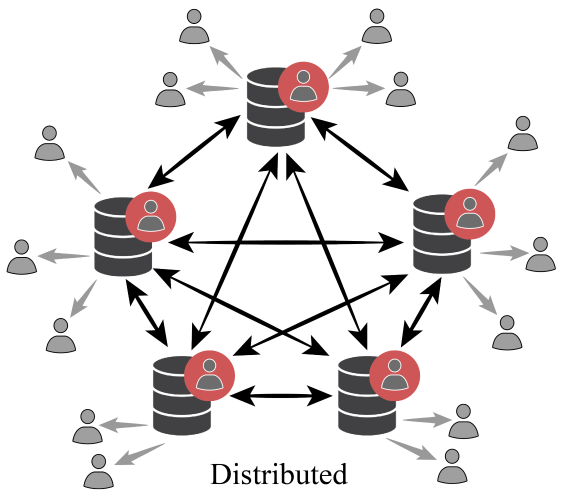


Figure 2.2: Distribution in DLTs. The governance is distributed, and the consistency is maintained by a consensus algorithm that enables each party to observe the same order of events.

Distributed A distributed ledger, or DLT, is a type of digital database that, like a decentralized one, is replicated, shared, and synchronized among multiple geographic sites, countries, or institutions. Its consistency and integrity are maintained through a consensus mechanism that enables parties to trust each other despite operating in a potentially untrustworthy environment. Unlike centralized and decentralized databases, there is no central administrator or trusted authority in a distributed ledger.

In a DLT, each replica maintains a copy of the ledger and independently updates itself. When an update occurs, every node constructs an updated view of the ledger, and each participant in the consensus process votes on which copy is correct. Once a consensus has been reached, all other nodes update themselves with the new, accurate copy of the ledger [90]. Finally, security is ensured through the use of cryptographic keys and signatures.

DLTs can be classified based on their data structure and consensus mechanism, each of which have their own advantages and disadvantages. Figure 2.3 presents three common implementations of DLTs: blockchains, DAGs, and hashgraphs.

Blockchains are a linear linked list data structure in which each block is totally ordered. In the event of conflicts, the longest chain prevails, as

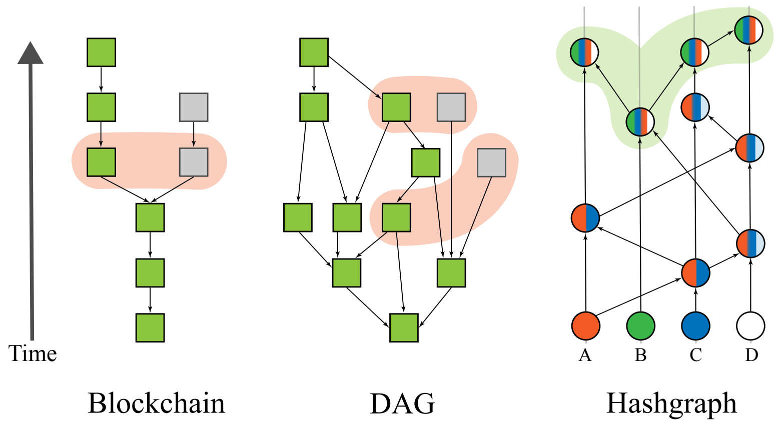


Figure 2.3: A comparison of the data structures of different DLTs. Blockchains use linked lists to connect blocks with cryptography, while DAGs and Hashgraphs use a Directed Acyclic Graph. In the first two diagrams, the green blocks represent accepted blocks, while the red sets represent conflicts or instances of double spending. The third diagram illustrates that all parties should eventually agree on the same order of events (shown as the green set).

depicted in Figure 2.3. The first widely implemented blockchain technologies, Bitcoin [8] and Ethereum [91], were introduced in 2008 and 2014, respectively. Bitcoin uses a POW consensus mechanism, while Ethereum recently switched to Proof-of-Stake (PoS), with validators referred to as miners and minters, respectively. These solutions (especially POW) offer high reliability and security, but have low throughput and high energy consumption at scale.

DAGS are tree-like data structures in which the total ordering of transactions is not necessarily important, unless conflicts arise. In Figure 2.3, one of the two conflicting blocks (red set in DAG) is discarded, and therefore it does not have any child blocks. Systems that implement a DAG-based chain emerged in 2018 with IOTA [92] and in 2019 with Avalanche [24]. While implementing a DAG is more complex than creating a blockchain, it is more efficient in terms of scalability, although its reliability and security are not necessarily proven to be as strong as those of POW systems.

Hashgraphs are a type of DLT that utilize a DAG as their core data structure. Transactions in hashgraphs are stored in parallel among all validators and the order of these transactions is not based on timestamps but rather on the occurrence of events. In hashgraphs, events are generated through a process called *gossip about gossip*, [93], in which validators reach agreement on the order of events. Each event contains one or more transactions, a timestamp, a digital signature, and cryptographic hashes of two earlier events. The hashgraph technology was first introduced in 2016 [93], and a cryptocurrency implementation called Hedera [94] was released in 2019. Hedera utilizes an asynchronous Byzantine Fault-Tolerant (aBFT) consensus algorithm. Hashgraphs are known for their scalability and speed compared to other DAG-based technologies, but the correctness of the entire protocol has faced criticism [95] and their use typically requires a private chain or a closed consortium of voters.

This dissertation focuses on a specific type of DLTs, known as blockchains. In particular, we examine the most widely used one of Bitcoin, which utilizes a POW-based consensus mechanism. Additionally, in our recent publication [80], we provide a brief security analysis of Avalanche, a DAG-based DLT solution.

2.1.3 Elements of Blockchains

Blockchains are distributed ledgers characterized by linked lists of blocks that contain information about transactions between parties. These blocks are cryptographically linked to create an immutable ledger with an append-only structure. The *access policy* of a blockchain determines who can read the information, leading to a classification as either public or private. The *control policy* determines who can participate in the advancement of the blockchain and how new blocks can be appended, resulting in classification as either permissioned or permissionless. The *consensus policy* regulates the progression of the protocol.

In 1979, Merkle introduced the concept of using a cryptographic hash to link information in an immutable chain, a structure now known as a *Merkle hash tree* [96]. Each data node in the tree is hashed (*H* function in Figure 2.4) and the resulting Merkle leaves are paired and hashed together to form branches, eventually leading to a root hash that includes the information from every other node in the tree. This allows for the authentication of a set of messages stored in the data nodes using a unique signature (the Merkle root) without disclosing the other information. The verifier only needs to fetch a small portion of the tree (green nodes in Figure 2.4) to reconstruct the hashes up to the Merkle root, which can then be compared with the root from a trusted source to verify

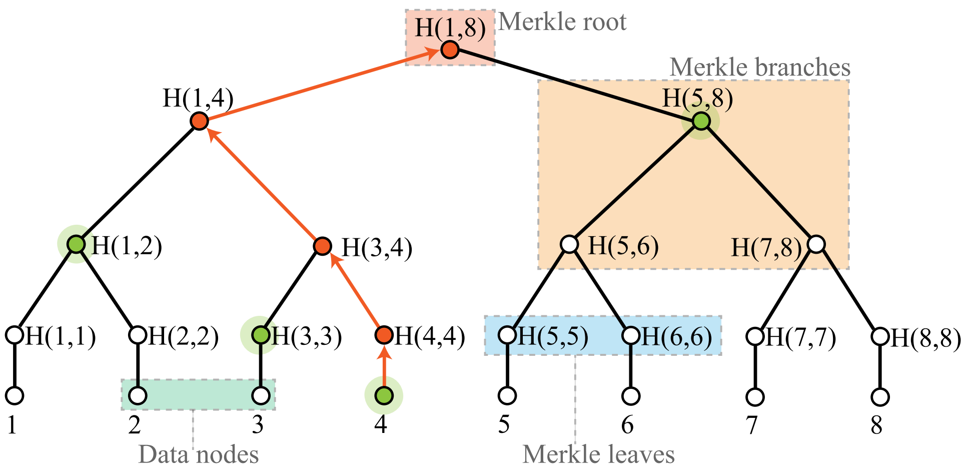


Figure 2.4: Illustration of a Merkle tree with 8 data nodes. The data nodes are hashed to create the merkle leaves, and the merkle leaves are then hashed together in pairs to form higher nodes in the tree. To verify the authenticity of the data contained in node 4, a verifier must retrieve the nodes in green and use them to reconstruct the tree from the data to the root. The rebuilt nodes are marked in red, and the root is used for verification.

the authenticity of the data. Similarly, in blockchains, the latest records or blocks contain the history of the entire chain. Such data structure is adopted by Dynamo [97], the efficient key-value storage system of Amazon. In fact, it minimizes the amount of data that needs to be transferred for synchronization and it reduces the number of disk reads performed during the anti-entropy process.

Access policies refer to the rules that determine which parties have the ability to access information stored on a distributed ledger. *Public* blockchains are accessible to anyone with an Internet connection, while *private* blockchains are restricted to certain organizations or consortiums and can only be accessed by parties who have been granted access. There is a range of control policies that can be implemented in blockchain protocols [98], which determine the ability to write information on the ledger and define the blockchain as either *permissioned* or *permissionless*. These control policies have an impact on the governance of the system. Permissionless blockchains allow anyone with an Internet connection to become a validator of the network and participate in the consensus process that drives the progression of the blockchain. In contrast, permissioned blockchains place restrictions on the ability to append data to the ledger.

Blockchains are defined by the structure of an append-only linked list of records that are secured using cryptography. However, the choice of control and access policies gives rise to different consensus mechanisms. These mechanisms describe the procedure by which network validators reach agreement on a single data value among distributed processes. A consensus mechanism should be fault-tolerant and provide security guarantees of termination, finality, and consistency for any decision under stated assumptions. Verification of validators may be required to increase trust, but this also reduces anonymity for nodes. The trade-off between security and speed, as well as the decision to require node verification or not, results in a variety of combinations of public and private, permissioned and permissionless blockchains, as depicted in Figure 2.5.

As mentioned above, various control and access policies determine their inherent consensus mechanisms:

Public / Permissionless Public and permissionless blockchains allow any participant to join the consensus mechanism without verifying their identity. This type of blockchain can provide a high level of anonymity, but it also requires a higher level of trust and may have slower overall performance and higher cost due to the use of POW consensus mechanisms like Bitcoin. Public and permissioned blockchains that require node verification are faster than POW, but do not offer anonymity and their security assumptions are weaker compared to POW systems. These blockchains often use consensus protocols such as PBFT or Federated Byzantine Fault Tolerance (FBFT), as implemented by cryptocurrencies such as Ripple [21] and Stellar [23].

Public / Permissioned Permissioned blockchains require that nodes satisfy certain conditions to participate in the consensus process. These conditions may include verification requirements or the need to put money at stake as an incentive to act honestly. These systems are typically faster than POW systems, but there is a higher risk of having a small number of wealthy validators control the network. Ethereum, which was originally a POW system, has transitioned to a POS system [16]. Another example of a permissioned blockchain is Avalanche [24], which uses a POS protocol.

Private / Permissionless Hybrid blockchains are private systems that allow access to only a select group of restricted members, but also implement permissionless features and are controlled by a single organization or consortium. These systems provide the level of oversight performed by

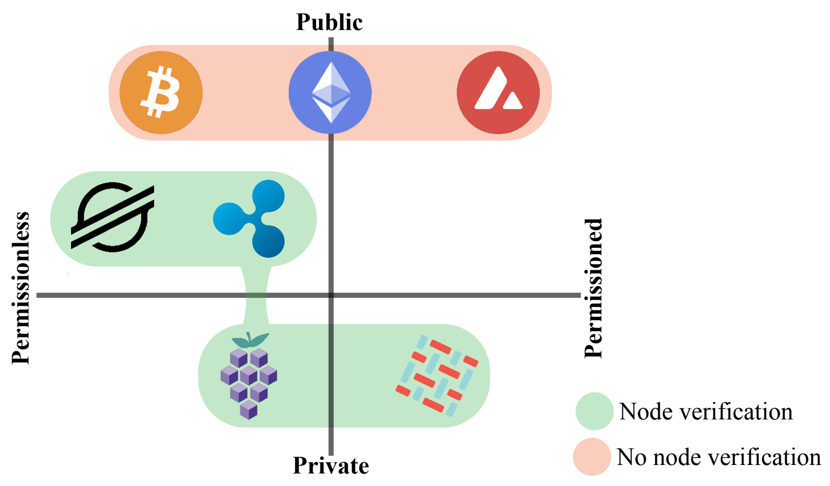


Figure 2.5: Public and permissionless blockchains allow anyone to participate in the consensus mechanism. Some examples of this type of blockchain include Stellar and Ripple, which may require node verification, and Bitcoin, which does not. Permissioned but public blockchains impose some restrictions on participation in the consensus process, with examples including Avalanche and Ethereum. Private and permissioned blockchains are controlled by a single organization, such as Hyperledger Fabric. Private and permissionless blockchains, such as IBM Food Trust, are hybrid solutions that are controlled by a single entity but implement permissionless functionality.

public blockchains for transaction validation, while also preserving a level of privacy. An example of a hybrid blockchain is IBM Food Trust [99], which uses the private and permissioned blockchain of Hyperledger Fabric [100] as its core.

Private / Permissioned Private and permissioned blockchains are controlled by a single entity, with the central authority determining which individuals or entities can act as validators. While such systems may be efficient in terms of throughput, they are only partially decentralized and do not fully embody the decentralized nature of blockchains. An example of this type of blockchain is Hyperledger Fabric.

In our opinion, the most significant impact of blockchain technology lies in its ability to create a fault-tolerant, tamper-proof, immutable, and verifiable system with decentralized governance. Therefore, the focus of our dissertation is on permissionless and public blockchains, specifically Bitcoin. In the following sections, we will discuss the history of blockchains, the POW consensus mechanism of Bitcoin, and the POS scheme of Avalanche. In the next chapter, we will examine the fee market that has emerged as a result of the scalability limitations of POW systems.

2.1.4 History of Blockchains

A proposal for a blockchain system was first presented in 1982 by Chaum [101, 11], in which he described the design of a distributed computer system that could be established, maintained, and trusted by groups that do not necessarily trust each other. This system implemented a public record-keeping solution with group membership consistency, utilizing cryptographic primitives such as symmetric and asymmetric encryption, cryptographic hash functions, and digital signatures. It is worth noting that Chaum's system predates the concept of permissioned and permissionless blockchains, and therefore does not clearly fit into either of these categories [102].

The problem of reaching consensus among unreliable or fallible processes, known as the Byzantine Generals problem, has garnered significant attention from researchers and academics. In 1982, Lamport et al. [103] introduced this problem, and in 1984 Schneider [104] proposed a solution that laid the foundation for consensus mechanisms in permissioned blockchains. Protocols such as PBFT as presented by Castro and Liskov [105] in 2002 and Paxos introduced by Lamport [106] in 1998 serve as the basis for achieving Byzantine agreement in open networks with node verification, such as Ripple and Stellar.

In 1993, Dwork and Naor [107] introduced a computational method for addressing junk mail in their publication. Their approach involves requiring users to perform computationally difficult, but not impossible, functions in order to gain access to resources, thus preventing frivolous use. In 1997, Back proposed the concept of *Hashcash* [12], a mechanism designed to address the systematic abuse of unmetered Internet resources such as email and anonymous remailers, without being aware of Dwork and Naor's earlier work on the subject.

The main idea behind Hashcash is to use a pricing function to create strings that, when processed through the SHA-1 hash algorithm, result in a string with the first N bits equal to zero, where N is typically around 20-30. An example of this process is illustrated in Figure 2.6, which shows an Hashcash token with a 28-bit collision ($N = 28$). The hexadecimal output has 7 leading zeros, corresponding to 28 bits in binary.

This concept of a cost function is similar to the one used by Finney [13] in 2004 to develop the first POW scheme, called Reusable Proof-of-Work. In this system, Hashcash is used as a POW token, and in exchange, RSA-signed tokens, or RPOW-tokens, are created. These tokens can be transferred from one person to another and are as rare and valuable as the Hashcash used to create them, but they are reusable, unlike Hashcash. These primitives are computationally expensive, as they require a proof of computation, but they offer high security and resistance to Sibyl attacks [108]. This foundation for establishing consensus in permissionless and public blockchains that do not require user verification is known as the POW protocol and is used in Bitcoin and previously in Ethereum. It allows for consensus to be reached in a distributed and untrusted environment, and it ensures a high level of security based on computational power rather than the number of participants.

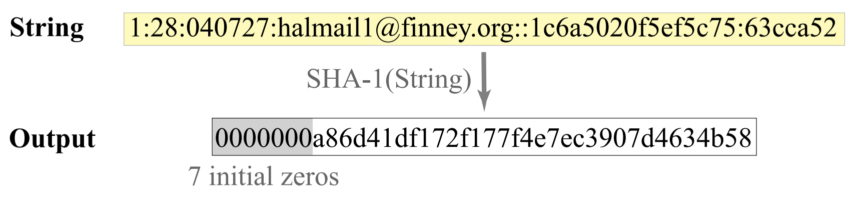

String
1:28:040727:halmail1@finney.org::1c6a5020f5ef5c75:63cca52
SHA-1(String)
Output
0000000a86d41df172f177f4e7ec3907d4634b58
7 initial zeros

Figure 2.6: The Hashcash cost function maps a string using the SHA-1 hashing algorithm, ensuring that the first 28 bits of the output string are equal to zero as per the specified rule.

2.2 Cryptocurrencies

Cryptocurrencies, are generally not considered to be currencies in the traditional sense, and are instead treated as a separate asset class in practice [109, 110]. Cryptocurrencies do not have any inherent or legislated value and their worth is determined by the supply and demand in the market. Despite having an initial value of less than half a cent in 2009 (when Laszlo Hanyecz used 10,000 Bitcoins to purchase two pizzas, an event now known as Bitcoin Pizza Day on May 22 [111]), cryptocurrencies have gained popularity for their decentralization, anonymity, and cost-effectiveness, leading to an increase in the overall market capitalization from one billion dollars in 2013 to almost three trillion dollars in 2022.[1]

Following the release of Bitcoin, many other cryptocurrencies have been developed and, at present, there are approximately 23,000 different cryptocurrencies that can be traded on more than 250 exchanges.[2]

2.2.1 Bitcoin and Proof-of-Work

In 2008, a person or group using the pseudonym Satoshi Nakamoto published a paper on Bitcoin [8] and released the open-source software for the cryptocurrency. Bitcoin was the first successful application of blockchain technology that used POW as a consensus mechanism. It timestamps transactions by hashing them into a chain of hash-based POW, making it difficult to alter the recorded information without redoing the POW and rebuilding the chain from scratch. Since the longest chain is generally considered to be the correct one, as long as the majority of the computing power is controlled by honest nodes, they will be able to generate the longest chain and outpace any attackers.

Transactions

According to Nakamoto [8], an electronic coin can be represented as a *chain of digital signatures*. In Figure 2.7, we observe that each owner transfers a coin to the next by signing the hash of the previous transaction with their private key and including the next owner's public key. The recipient of the coin can verify the signature (signed by the previous owner) using the previous owner's public key. While this system allows for the transfer of ownership of a single

1. According to Coinmarketcap `https://coinmarketcap.com`
2. Coinmarketcap `https://coinmarketcap.com/rankings/exchanges/`

coin, it does not prevent double spending. In fact, the recipient cannot know if the previous owner has signed any earlier transactions. To address this issue and prove the authenticity of transactions without relying on a central trusted authority, transactions must be publicly announced and participants must reach consensus on the order of transactions.

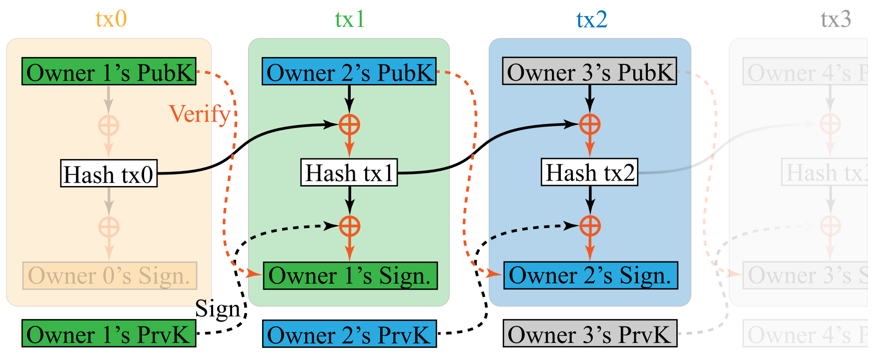


Figure 2.7: Bitcoin Transactions. In the context of Bitcoin, the concept of ownership of a particular coin is established through a sequence of digital signatures. An example of coin transfer is depicted in green as Transaction 1 (tx1), is made from Owner 1 (green) to Owner 2 (blue). This transfer is facilitated through the creation of a hash that incorporates Owner 2's public key (PubK), and the previous transaction hash (Hash tx0). The validity of the transfer can be confirmed through the utilization of Owner 1's private key (PrvK) to sign Hash tx1, which can then be verified by Owner 2 using Owner 1's public key. Subsequent transfers of the coin, such as the transfer to Owner 3 (gray), result in the formation of a chain of digital signatures.

Blocks

To address the double spending problem, Nakamoto's initial solution was to implement a *timestamp server* that creates a hash of a block of transactions that need to be timestamped and publishes this hash as proof of the transactions' existence at that time. For efficiency and security purposes, each transaction in the block is represented by a leaf in a Merkle tree, and the root of this tree is used to generate the timestamp (block) hash. As shown in Figure 2.8, a single block hash contains information about the previous hash, time, and Merkle root. This allows every transaction to be publicly announced, and if any of the transactions are tampered with, the entire block hash will be changed. Furthermore, each block includes the previous block's hash in its own hash, forming a chain where each additional timestamp strengthens the ones before it and makes manipulations visible at any point in the chain, unless the entire chain is rebuilt. Once transactions have been proven to be publicly announced, a distributed timestamp server is needed to reach consensus among validators (called miners in Bitcoin) on a single record history. This is achieved through the use of a POW implementation similar to that proposed by Hashcash [12].

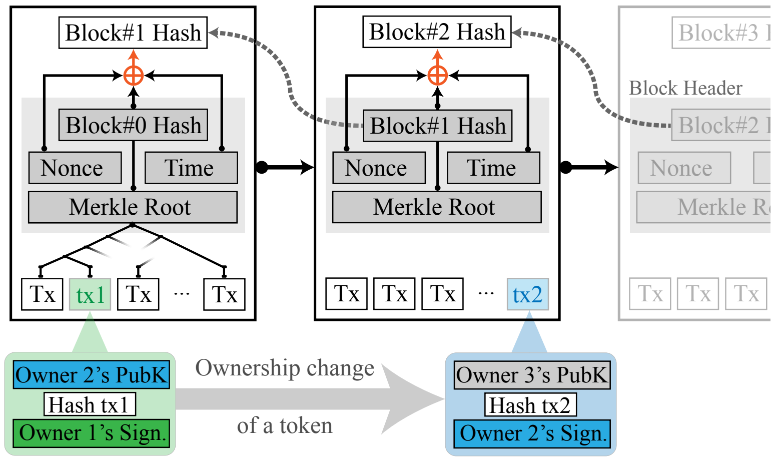


Figure 2.8: Blocks are linked together through a chain of hashes. Each transaction in a block is represented by a leaf in the generated Merkle tree, and the root of this tree is included in the block header. The block header also includes the time, nonce, and hash of the previous block. These values are used to generate the hash of the new block, which is then linked to the previous block through its hash. The chain of digital signatures depicted in Figure 2.7 can be contextualized in terms of blocks, where tx1 and tx2 represent a change in ownership or a particular token.

Proof-of-Work

In Bitcoin, validators reach consensus on a single record history by verifying that newly created blocks are valid. When a block is deemed valid, it indicates that a peer has successfully solved the POW puzzle associated with it, demonstrating that a certain amount of work has been done. This peer becomes the leader for this round and proposes a new block. To solve the POW puzzle, a miner must find a block hash with a specified number of leading zeros that is below a certain target value. This target value is determined by the network difficulty, as described in Section 2.1.4.

The SHA-256 algorithm is used to hash the block header twice, and the nonce (a number used only once) is incremented by one for each failed attempt. The probability of finding the target value (shown as the yellow box in Figure 2.9) through this process is low, requiring a significant amount of trial and error. The solution can be verified quickly by re-hashing the block header and comparing it to the posted block hash.

In a POW-based system, the individuals responsible for validating blocks are called *miners*, and the process of solving the POW puzzle is called *mining*. Once a block has been mined, it is immutable and cannot be altered without building a longer chain starting from the tampered block. Therefore, the more hashing power the network has, the harder it is for an individual or group to overtake the original chain and confirm the tampered block. POW was originally developed to combat spam emails and DDOS attacks, but Bitcoin was the first system to use it for both mining (where validators are compensated when they find a new block) and achieving consensus [102]. In summary, POW involves finding the nonce that, when hashed twice with the rest of the block header using the SHA-256 algorithm, produces a block hash with a specified number of leading zeros. The main drawbacks of using POW are the high energy consumption required for mining and the low transaction throughput due to block size and time constraints for producing new blocks [58].

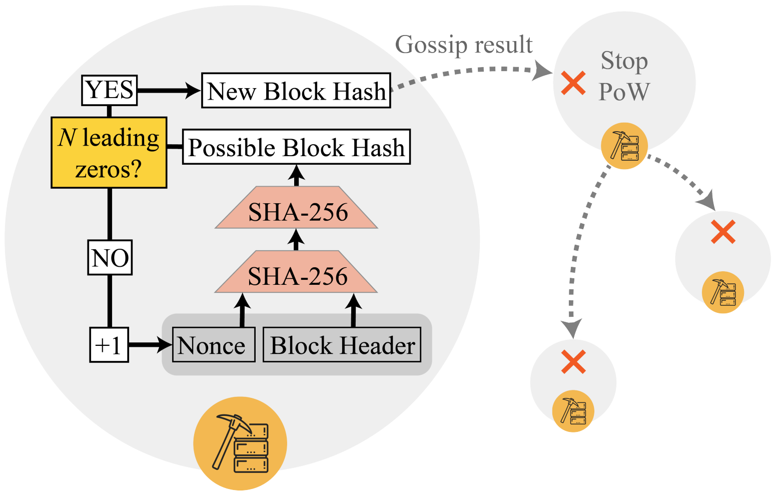


Figure 2.9: For each attempt, the nonce is incremented and the block header is hashed twice using the SHA-256 algorithm. The first miner to solve the puzzle broadcasts the solution to other peers. If the proposed hash is valid, each peer aborts their current POW and begins a new one with a new set of transactions and a new Merkle root to hash.

Difficulty

In relation to POW, *difficulty* refers to the measure on how hard is to find a hash below a specified target. This difficulty can be increased or decreased by modifying the target value according to the scheme in Figure 2.10. Difficulty serves to maintain a consistent block creation frequency, despite variations in Bitcoin's overall hashing power over time. The desired block creation time in the Bitcoin network is set at 600 seconds, which is determined by the core algorithm and design of the network. The difficulty level of mining is adjusted approximately every 2,016 blocks, equivalent to around 14 days. This adjustment ensures that the network maintains a consistent block creation rate. The calculation of the difficulty adjustment involves normalizing the mean creation time of the past 2,016 blocks. This mean creation time is denoted as $\mathcal{T}'$ and can be calculated using Equation 2.1:

$$\mathcal{T}' = \frac{\sum_{i=1}^{2016} \mathcal{T}_i}{2016} \tag{2.1}$$

In this equation, $\mathcal{T}'$ represents the average block creation time, which is obtained by summing the creation times of the past 2,016 blocks and dividing the sum by 2,016. This normalization process helps adjust the mining difficulty to maintain the desired block creation time. The difficulty value at a specific block height,[3] denoted as x (where $x \bmod 2016 = 0$), is represented

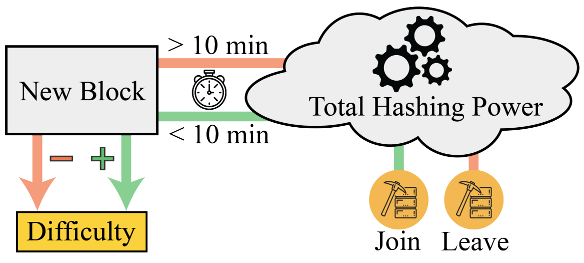


Figure 2.10: In the Bitcoin network, the difficulty of creating new blocks adjusts in response to changes in the overall hashing power of the network. When a miner joins, the overall hashing power increases, leading to the likelihood of faster block generation and an increase in difficulty to maintain a stable block creation time. Conversely, when a miner leaves, the difficulty decreases to compensate for the decrease in hashing power. While the illustration shown depicts the creation of a single new block, in the Bitcoin network, the difficulty is actually adjusted every 2,016 blocks.

3. The height represents the number of blocks that have been added to the blockchain from its inception. The condition $x \bmod 2016 = 0$ indicates that the height of the block is a multiple of 2,016, which typically corresponds to the point at which the network adjusts the mining difficulty.

by Equation 2.2.

$$d_x = \begin{cases} 1 & \text{if } x = 0 \\ d_{x-1}\frac{\mathcal{T}}{\mathcal{T}'} & \text{if } x > 0 \end{cases} \tag{2.2}$$

At every block creation, the miner or pool of miners who successfully completes POW with a given difficulty will be compensated with a *coinbase* transaction. This transaction is included as the first transaction in every block and consists of the sum of transaction fees plus a block reward of newly mined Bitcoins, which is periodically halved every 210,000 blocks.

2.2.2 Avalanche and Proof-of-Stake

In 2018, a group of scientists, initially referred to as Team Rocket, published a novel metastable consensus protocol family for cryptocurrencies [24]. These protocols provide a high level of probabilistic safety in the presence of Byzantine adversaries and are claimed to be both fast and environmentally friendly, as they do not rely on POW-based blockchains. The protocol family includes a naïve implementation known as *Slush* and a more sophisticated consensus algorithm called *Snowball*, which serves as the backbone of the Avalanche protocol.

Slush Slush introduces the concept of *metastability* and serves as the foundation for this protocol family. It allows for the eventual reaching of consensus by choosing between two conflicting colors (blue and red in Figure 2.11) in m rounds, where m is a sufficiently large value. This algorithm is almost memoryless meaning that a node retains no state between rounds other than its current color and does not maintain a history of interactions with other peers. Each round involves randomly sampling a small, constant-sized group of k nodes from the network, and once the querying node collects k responses, it checks the color of the fraction $\geq \alpha k$, where $\alpha > 0.5$. The querying node adopts the winning color as its own and re-issues the query with a different set K. Even if peers are divided in a 50/50 split, the network will eventually reach a decision within m rounds. However, Slush is not tolerant to Byzantine faults, as an adversary could attempt to flip nodes to the opposite color decision in an effort to maintain balance within the network. Figure 2.11 (1–2) shows the first of m cycles for a blue-queried node (1), while Figure 2.11 (3–4) represents the second cycle for $m = 2$.

Snowflake Snowflake extends the Slush protocol by adding a *counter* that tracks a node's conviction about a particular color choice. This counter stores the number of consecutive samples of the network that have all yielded the same color. Snowflake introduces BFT through the use of the parameter β, which indicates the counter threshold required to reach consensus. At every color change, the node resets β. Whenever a successful query ($\geq \alpha k$) occurs, the node increments the counter. Consensus is reached when the threshold β is reached. The protocol preserves liveness, but it can be significantly delayed, as there is no finite counter m present and β can be reset every time there is a disagreement, leading to an ephemeral notion of state in Snowflake [24].

Snowball Snowball augments Snowflake by incorporating *confidence counters*. Each validator stores in memory its confidence level for each color. Mul-

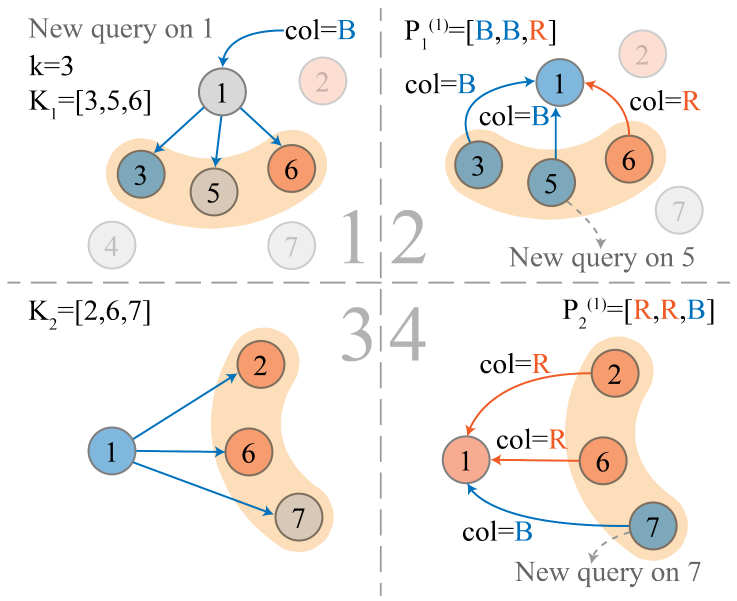


Figure 2.11: In the Slush protocol, during the first frame, node (1) is queried with the color blue. Node (1) then selects k random nodes and sends its decision to choose blue. In the second frame, node (3) has already decided on blue, node (6) has been confirmed as red, and node (5) has not yet made a decision. Node (3) and node (6) reply with their preferred colors, while node (5) becomes blue and issues a new query with blue as the preferred color. The pool for node (1) at step $m = 1$, $P_1^{(1)}$, contains two blue and one red answer, so node (1) becomes blue. In the third frame, the $m = 2$ cycle is represented, and a new set K_2 is selected. Node (1) sends its blue view to the k chosen nodes. In the final frame, node (1) turns red as its pool for node (1) at $m = 2$, $P_2^{(1)}$, has a majority of reds, while node (7) turns blue and issues a new blue query. The algorithm continues until the m^{th} cycle.

tiple counters track the number of queries that have yielded a result of $\geq \alpha k$ for their corresponding color. Whenever a successful query occurs, the node increments its confidence counter for that color. A node will switch colors when the confidence level for its current color becomes lower than that of the other color. When a counter reaches the threshold of β, the current color is accepted. Snowball is more resistant to attack than Snowflake and serves as the backbone of the Avalanche protocol.

Avalanche Avalanche generalizes Snowball by implementing a dynamic, append-only DAG of all known transactions. Each DAG vertex contains a collection of items starting from the *genesis vertex*. The DAG structure offers two main benefits: (1) increased efficiency, as a single vote on a vertex implicitly means voting for all transactions on the path to the genesis vertex; and (2) improved security, as the DAG intertwines the history of transactions, making it difficult for an attacker to reverse a decision without the approval of the correct nodes, similar to the Bitcoin blockchain.

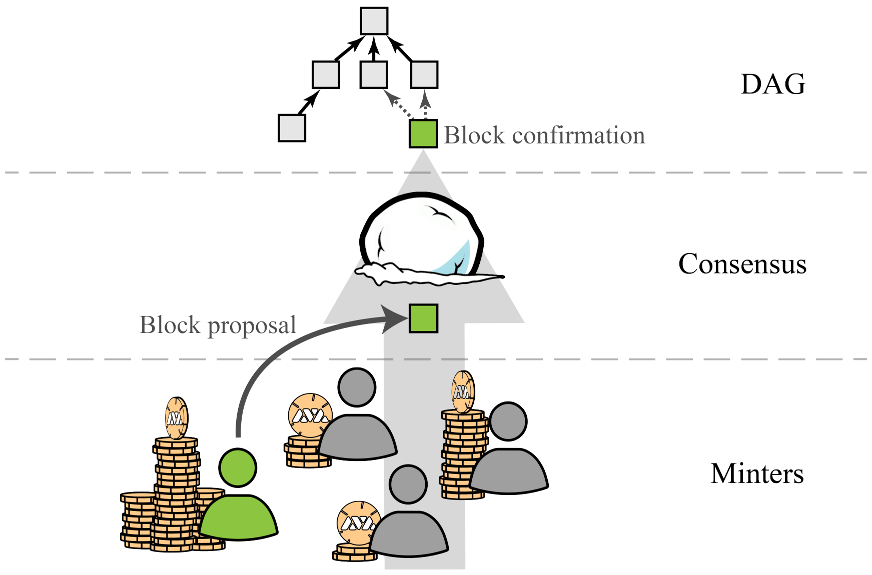


Figure 2.12: In Avalanche, Proof-of-Stake is used to determine the selection of a minter to propose a new block. The minter is chosen based on the amount of funds they have staked. Once the block is proposed, the minters must reach consensus on the solution using the Snowball consensus process. Upon successful consensus, the new block is added to the DAG.

Proof-of-Stake

Proof-of-Stake is a consensus mechanism that reduces the computational cost of POW by requiring validators to put their funds at risk while participating in the consensus process. Miners in POS systems are referred to as minters, and the verification of nodes is not necessary due to the amount of stake held by the minters. In POS, the honest majority of computational power is replaced by the honest majority of stake values. To maintain the integrity of the single history of records in POS systems, minters must honestly verify new blocks to avoid having their stake slashed or destroyed. A pool of minters is selected at each round to review newly proposed blocks, with the validators chosen based on the amount of their stake. If a validator submits fraudulent transactions, their stake will be destroyed and they will no longer be able to participate in the system. On the other hand, honest minters are rewarded with additional coins.

Avalanche consensus does not solely rely on POS, but it also incorporates the substrate BFT-based-based consensus protocol earlier described, Snowball. POS is utilized in Avalanche to prevent Sybil attacks and preserve the anonymity of validators, while the stake of each minter plays a role in determining who will lead the BFT consensus at each round. Typically, validators with larger stake are more likely to be selected for block proposal, making it difficult for dishonest actors to execute a Sybil attack without risk of losing their funds. One major concern with POS is the potential for centralization, particularly when stake is based on financial resources. This can lead to a situation where the wealthy become even wealthier, making it financially prohibitive for many individuals to become validators and undermining the decentralization of governance. In Avalanche, the minimum required stake for a validator is 2,000 AVAX,[4] which is equivalent to $ 30,000 in May 2023.

Directed Acyclic Graph

In contrast to a blockchain, DAGs do not impose a total ordering on transactions, instead providing a partial ordering of decisions. DAGs are constructed using a similar chain of cryptographic links logic as used in blockchains. For example, in Avalanche, each vertex in the DAG contains information about transactions, the chain ID, a list of parent IDs, the epoch, and the version. This information is hashed using SHA-256 to create a new vertex ID. The use of DAGs significantly

4. From Avalanche documentation at `https://docs.avax.network/nodes/validate/staking`

increases the frequency of block creation, as non-conflicting transactions can be included simultaneously in two separate vertices that have a common parent, as illustrated by the vertices *b* and *c* in Figure 2.13. At each vertex creation, the consensus mechanism establishes a shared view of the updated DAG.

Consensus Comparison

To summarize, Table 2.1 presents a comparison of the characteristics of various consensus mechanisms. Ethereum was initially developed as a POW system. It then adopted a hybrid approach, using both POW and POS, where POW was used for more critical operations and POS was employed for the remainder. At the time of writing, Ethereum has fully transitioned to using POS. Avalanche implements both POS and BFT solutions. Finality in POW systems takes longer to achieve compared to Avalanche POS systems. In Bitcoin, for example, a transaction is considered finalized once it has been included in six consecutive blocks in the blockchain. However, the time it takes to create these blocks is relatively slow in POW systems. POS and BFT are generally faster and more environmentally friendly solutions, while POW offers a more robust system through the use of hash rate-based leader selection.

2.3 Machine Learning

ML algorithms are able to analyze large datasets and outperform humans in classifying new data based on what they have previously learned. In contrast to

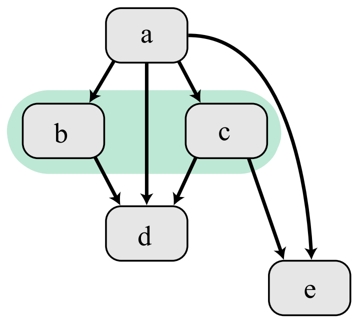


Figure 2.13: When DAGs are used as a blockchain, we can establish a partial ordering of the vertices. For example, we can determine that vertex *a* comes before vertex *c*, vertex *c* comes before vertex *e*, and therefore vertex *a* comes before vertex *e*. However, we have no information about the relative ordering of vertices *b* and *c*.

Consensus mechanism comparison			
	POW	**POS**	**BFT**
Leader selection	Hash rate	Stake amount	Trust
Energy consumption	Significant	Negligible	Negligible
Speed (txs/s)	Poor	Good	Good
Finality	Poor	Good	Immediate
Applications	Bitcoin, Ethereum.	Ethereum, Cardano, Algorand, Avalanche.	Avalanche, Hyperledger Fabric, Ripple, Stellar.

Table 2.1: A summary of various consensus mechanisms.

traditional programming, which uses predetermined patterns to process data and generate output, ML aims to create patterns by combining input data with its corresponding output. In this section, we will explore the key elements of ML and various types of ML models, with a focus on identifying the model that potentially is best suited for our classification task. We will also consider how large amounts of data available through public blockchains can be used to formally define patterns for inclusion, given that miners follow certain criteria for selecting transactions.

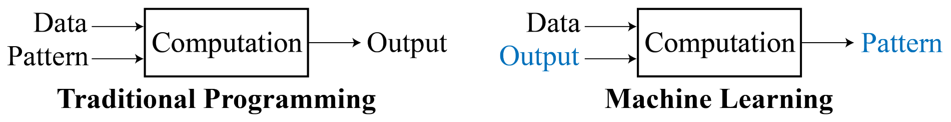


Figure 2.14: Traditional programming and ML.

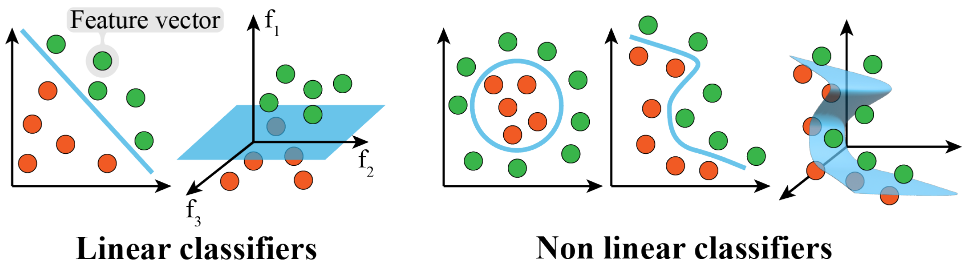


Figure 2.15: This figure illustrates linear and non-linear classifiers, with measurements for features depicted on different axes (f_1, f_2, f_3). Each point, depicted as either green or red, represents a labeled feature vector.

2.3.1 Elements of Machine Learning

ML is a discipline focused on developing and understanding algorithms that are able to learn from and improve their performance on a given task through exposure to data [112]. ML has a wide range of applications, including data clustering [113, 114], data transformation, and transfer learning [115]. In this thesis, we utilize ML as an automated classification machine. Given a dataset, our ML model must be able to label data that it has not seen before. In order to achieve optimal classification performance, it is important to select the most appropriate input features, which can be represented as a feature vector that uniquely identifies a single object. A set of feature vectors constitutes a training set, which is derived from the initial dataset and used to train the ML model for future predictions on new data. In order to solve a classification task, we can envision a classifier as a decision boundary that separates different classes, as depicted in Figure 2.15. In some cases, linear classifiers such as the perceptron, linear Support Vector Machines (SVM), or Least Squares Methods (LSM) can be used when the classes are linearly separable in two or more dimensions [39]. However, not all problems are solvable through linear methods, so other techniques such as k-means, Random Forest (RF) [116], kernel SVM, or Neural Network (NN) [117] may be employed.

For the purpose of this study, features such as transaction fee and size may be important, and a specific Bitcoin transaction can be represented as a feature vector belonging to a specific class. Based on the complexity and variety of features derived from the non-deterministic interaction of miners and users in the Bitcoin ecosystem, a solution that can address a wide range of tasks is required. Literature suggests that using a non-linear classifier is a safer option in terms of solvability for non-linear problems, as it is generally injective (i.e., if a problem is linear, it can also be solved as a non-linear problem, but not vice versa). Therefore, our approach follows this principle.

2.3.2 Types

The availability or scarcity of data plays a significant role in determining the methodology employed by a ML model for classifying data. When a set of labeled training data is available, the classifier is designed to utilize this prior knowledge to train itself in a pattern recognition task called supervised learning. When such information is not available, the given non-labeled training set is used to identify underlying similarities and cluster similar feature vectors together. This type of classification is known as unsupervised learning, and it is applicable to a variety of fields, including social sciences and engineering, such as remote sensing, image segmentation, and image and speech coding [39]. Different algorithms can be used to implement each type of pattern recognition task, with variations in complexity, speed, or efficiency. Careful selection of the appropriate model algorithm can significantly impact the outcome of the clustering task.

Supervised learning The ML task of learning a function that maps an input to an output based on example input-output pairs is known as supervised learning [118]. In this approach, a function is inferred from labeled training data in order to map new, unlabeled examples. This is an optimal solution when the training set is representative of the overall distribution. Supervised learning models include SVM, k-Nearest-Neighbors (kNN), perceptrons, and Artificial Neural Network (ANN).

Unsupervised learning The ML task of learning patterns from unlabeled data is known as unsupervised learning. Unsupervised methods have the ability to self-organize and capture patterns as probability densities [119]. This approach is particularly useful when the training data is limited. Unsupervised learning models include clustering, k-means, and the Expectation–Maximization (EM) algorithm. An example of an unsupervised clustering task is shown in Figure 2.16.

Blockchains are public ledgers of data that can be accessed and read by anyone, resulting in a large amount of data that is always available and retrievable. Therefore, it is assumed that labeled data is always available and that the training set is representative of the overall distribution, with no missing data. Based on these assumptions, the classification task at hand requires a non-linear classifier and a supervised approach. As a result, ANNs, specifically Deep Neural Networks (DNNs), were chosen as the model of choice, as ML theory suggests that they are the most suitable solution for this problem.

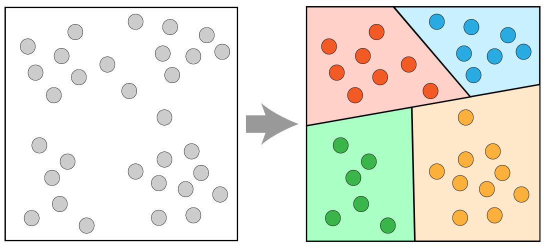


Figure 2.16: This figure illustrates a clustering task, in which unlabeled data is grouped into clusters based on a pattern that minimizes errors for feature vectors belonging to the same guessed cluster, using methods such as euclidean distance, probability density, or nearest neighbors.

2.3.3 Artificial Neural Networks

Artificial Neural Networks (ANN), or Neural Networks (NN) for short, are models inspired by the structure of the brain, including biological neurons and synapses. NNs are defined by Hopfield [120] as networks or circuits of biological neurons, or, in a technological context, as being composed of artificial neurons (or nodes). Each artificial neuron is a computational structure with multiple inputs and a single output. The building block of an NN model is the artificial neuron, and the simplest NN implementation consists of a single artificial neuron with a set of inputs and a single output.

Artificial Neuron

An artificial neuron, also known as a perceptron, is an algorithm used to learn a binary classifier, or *linear discriminant function*, that takes in a feature vector as input and produces a value $\psi(a)$ as output, where a is the result of the weighted features plus the bias, and ψ is an activation function that activates the neuron and forwards the final output. Figure 2.17 illustrates the feature vectors $\mathbf{f}$ and weights $\mathbf{w}$. The input to the activation function ψ is then the dot product $\mathbf{f} \cdot \mathbf{w}$ plus the bias w_0, which is formalized in Equation 2.3. The bias term w_0 is a randomly initialized value that shifts the discriminant and increases the neuron's ability to classify inputs. If the number of features is n, then a is given by:

$$a = \sum_{i=0}^{n} \mathrm{f}_i w_i + w_0 \tag{2.3}$$

The activation function is a non-linear function that is used to introduce non-

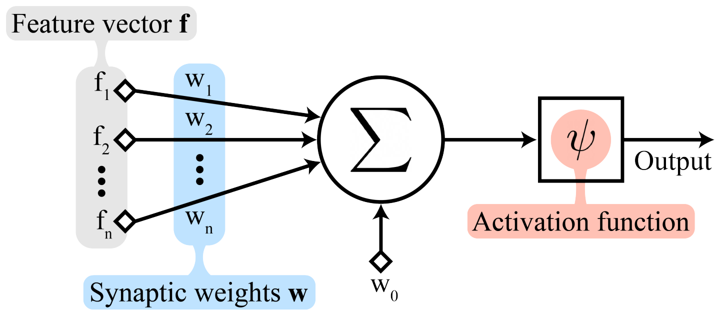


Figure 2.17: An artificial neuron, also known as a perceptron, receives a feature vector $\mathbf{f}$ as input and applies a set of weights $\mathbf{w}$ and a bias term w_0. The resulting output is then passed through an activation function ψ.

linearity into NNs. Without an activation function, a NN with N layers can be reduced to a single linear layer, which is the linear combination of all the other layers. The activation function determines how information is propagated through the network. Some commonly used activation functions include:

Unit step function It sets the value to either 0 or 1, to elements that comes before, or after a certain threshold. This function works well with binary classifiers, but it is less suitable for problems concerning more than two classes. Formally $(a < 0) \longrightarrow \psi(a) = 0$, and $(a > 0) \longrightarrow \psi(a) = 1$. The neuron can either be active or non active, as the function can fire or inhibit the neuron.

Sigmoid function One of the most commonly used activation function. It is easy to analyze and to compute, and it provides a *soft* transition between 0 and 1, with a threshold of 0.5. The most used sigmoid function type is the *logistic function*: $\psi(a) = {}^{1}/_{1+e^{-a}}$. The resulting curve is more steep towards 0, meaning that $a \longrightarrow 0$ values are mapped in a significantly distant space from each others. On the other hand, values for $a \longrightarrow \pm\infty$ are mapped not far enough for the neuron to change its activation, and consequently, it will learn slowly or not learn at all. This is know as the *vanishing gradient problem*. The output is a continuous value.

Rectified Linear Unit The Rectified Linear Unit (ReLU) activation function is defined as the positive part of its argument: $\psi(a) = a^{+} = max(0, a)$. It is widely used because it is computationally efficient, as only comparison, addition, and multiplication are involved. It offers a better gradient propagation, with fewer vanishing gradient problems compared to sigmoidal activation, as ReLU only activates after a certain threshold, and not in both directions [121], it is scale-invariant, as $max(0, ba) = b\,max(0, a)$ for $b \geq 0$.

Softmax function The softmax function is often referred to as normalized exponential function, and it is a generalization of the logistic function to multiple dimensions. It is also commonly used as the *last* activation function of a NN. The softmax function is used for multi-class classification in ANN, and it normalizes the network output to a probability distribution for every predicted class.

The implementation discussed in this dissertation utilizes a DNN with softmax and ReLU activation functions.

Deep Neural Networks

The NN architecture consists of multiple artificial neurons that are connected together. Each neuron utilizes algorithms to process mathematical equations, such as multi-dimensional polynomials (as described in Equation 2.3). Figure 2.18 illustrates a multi-layer NN with one hidden layer, input features (f_1 and f_2), and two potential outputs (green and red). In this model, the neurons are organized into layers, with each layer fully connected to the preceding one. The output of each artificial neuron in the previous layer serves as an input for each neuron in the subsequent layer, as depicted by the arrows in Figure 2.18. This type of ANN is referred to as a fully connected NN.

A NN model learns through the process of training. The resulting trained NN model is representative of the specific dataset it was trained on, so it is important that the training set accurately reflects the desired pattern for the classification task. Each input in the NN has a weighted parameter, allowing for the classification impact of each input to be controlled and adjusted as necessary. Every layer in the NN handles dependencies by calculating the values of adjacent inputs, creating both weak and strong paths through the network based on the activation functions. It is challenging to represent the information contained within a NN model, and it is common for two NN models trained on the same dataset to have different content.

The perceptron employs a discriminant function to linearly separate classes.

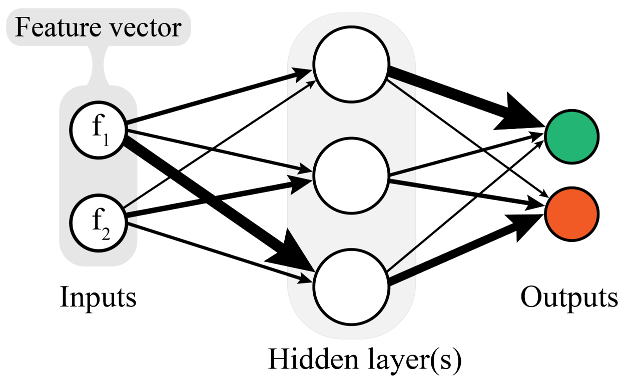


Figure 2.18: A fully connected Artificial Neural Network is depicted, with inputs, one hidden layer, and outputs represented. Each input constitutes a feature vector, and the hidden layer(s) can consist of one or multiple layers. The connections between the various components of the ANN are weighted according to the strength of the relationship between a particular feature and its corresponding outcome.

In cases where this is not possible, and the classes are not linearly separable, an arbitrary number of neurons (and therefore discriminant functions) can be added to separate the classes. An example of this can be seen in Figure 2.19, which depicts the OR and XOR problems plotted on a two-dimensional plane. While the OR task can be solved with a single neuron (shown on the left plane in Figure 2.19), due to its linear separability, the XOR problem requires a nonlinear classifier. The right plane of Figure 2.19 illustrates that two discriminant functions can be used to efficiently classify the elements, meaning that two perceptrons arranged in parallel (forming a layer of neurons) can be utilized to solve this classification task.

Neural Networks are considered deep if they consist of more than one hidden layer. Each layer trains a distinct set of features, which are determined by the output of the previous layer. The hidden layers can be understood as an ensemble of perceptrons that are arranged in parallel in order to solve nonlinear classification tasks, and in series in order to combine previous information with a new classification task. The use of weight calculations and dot products limits the input data to be either floating point or integer values. In cases where the points in the multidimensional space are widely scattered, it is often necessary to employ *normalization* techniques.

Backpropagation is a key component of NNs. It calculates weight gradients, allowing the network to be trained by minimizing the loss function. This is achieved through backward passes, updating the weights to bring the output closer to the target [122]. The frequency of backpropagation depends on the batch size of the training set. Negative gradients are propagated and weights are adjusted to reduce errors within each batch. Training occurs over multiple passes (epochs) of the entire dataset. The required training time is indeterminate, as it depends on the distance the weights must move to

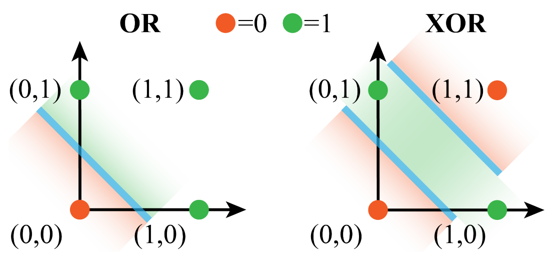


Figure 2.19: The OR and XOR problems demonstrate that a single perceptron is unable to effectively distinguish between ones and zeros when plotted on a two-dimensional plane using the XOR operation. However, the OR problem can be solved through linear means.

reach a solution. The minimum number of neurons, layers, and weights is also indeterminable.

Residual Neural Networks

Residual Neural Network (ResNet) implements skip connections in NN. The construction of DNN using this technique is provided in a way to bypass certain layers of the network. In a standard DNN, each layer is connected to the subsequent layer such that the input to a given layer is the output of the preceding layer. In contrast, skip connections allow the input to be passed directly to a layer that is further down the network, effectively creating a shortcut. This way, the network can learn not only from the output of the preceding layer but also from the input itself, resulting in better training and more accurate predictions.

Skip connections were first introduced in the ResNet architecture in 2015 [123]. Since then, they have become a popular technique in neural network design, particularly for DNNs with many layers. By providing a way for information to flow directly through the network, skip connections can help prevent the vanishing gradient problem that can occur in very deep networks. This can lead to improved performance and faster convergence during training, despite their effectiveness may depend on the specific problem being addressed. In our study, it has been determined that the implementation of the ResNet architecture is more effective and useful than standard DNN models.

Summary

This chapter provided a comprehensive overview of blockchain technologies and their applications in cryptocurrencies. It covered the historical development of blockchain, and the advantages and disadvantages of using blockchains in various business sectors are explored. The chapter focused on explaining POW and POS consensus mechanisms applied to cryptocurrencies such as Bitcoin and Avalanche. The consensus protocol of Bitcoin is discussed, highlighting its limitations at scale and alternative consensus mechanisms employed in cryptocurrencies. Furthermore, a brief introduction to ML is provided, and its role in this research is explained, including the specific ML models that have been adopted.

3 The Fee Market in Bitcoin

In this chapter we unravel principles and rules governing the Bitcoin ecosystem at scale, focusing on user-miner equilibria and showing how these parties depend on one another. We explain how miners make profits, what are their resulting costs, and how such factors are crucial for transactions inclusion. We discuss how a fee market emerges in PoW-based blockchains, describing the auction schemes that miners could adopt, and how this can lead to fee dynamism in Bitcoin. Notions of cryptocurrencies and PoW explained in Section 2.2 are useful for understanding mechanisms and reasons behind main issues in Bitcoin, including expensive fees, low throughput, and high energy consumption. In this dissertation, we focus on high fees and overpaying, we study how miner's criteria for transactions inclusion changes over time, and how these regime shifts are dictated by the mass adoption of Bitcoin and its inner throughput limitations, favoring a fee market to emerge. The study of such market is fundamental for our purpose of formalizing a transactions inclusion pattern.

3.1 Profits in Proof-of-Work

In this section, we investigate mining revenue and costs. We formalize the profit equation and calculate the cost of mining with and without fees. For that we take into account various parameters, such as electricity and Bitcoin prices and individual and total hash rates. As we discuss in Section 2.2.1, the security of the Bitcoin network increases as the number of miners grows. However, we also show that the individual cost of mining rises in tandem with the total hash rate, prompting rational miners to develop strategies for optimizing their profit. This makes their critical role in securing the network more expensive than originally intended, and therefore, it is essential to explore the sources of profit for the sustainability of the system.

3.1.1 Formalization

In Section 2.2.1, we discussed coinbase transactions and explained that the concept of *creatio ex nihilo* does not apply to digital tokens, as a certain amount of computational work is required to secure the network. The coinbase transaction enables miners to generate revenue from two sources: (1) transaction fees, denoted as M; and (2) block reward, denoted as R. However, explaining miners' profit solely in terms of coinbase transactions is overly simplistic. Rizun [9] formalizes miners' profit, denoted as $\langle \Pi \rangle$, as the difference between revenues $\langle V \rangle$, and costs $\langle C \rangle$, as shown in Equation 3.1.

$$\langle \Pi \rangle = \langle V \rangle - \langle C \rangle \tag{3.1}$$

The expected cost for a single miner, denoted as $\langle C \rangle$, is formally defined in Equation 3.2 as the product of its hardware's price per hash, denoted as η, its hash rate, h, and the time required to mine a block, denoted as $\mathcal{T}$.

$$\langle C \rangle = \eta h \mathcal{T} \tag{3.2}$$

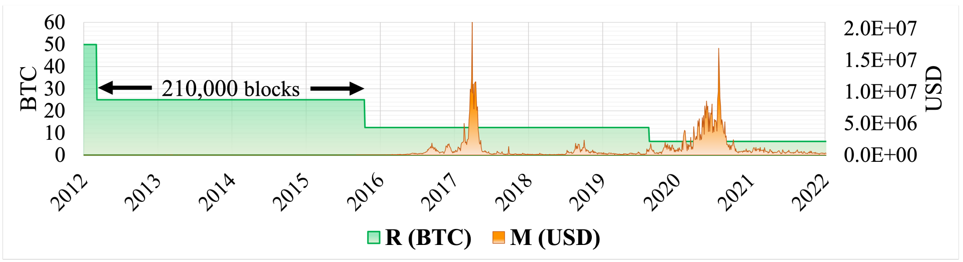


Figure 3.1: Block reward R (BTC) and transaction fees M (USD) are represented respectively on the left y-axis, and on the right y-axis. Public data of Bitcoin fetched from blockchain.com at `https://www.blockchain.com/explorer/charts`

The expected revenue from mining is given by the earnings from the coinbase transaction, denoted as $M+R$, multiplied by the probability of orphaning,[1] based on the individual miner's hashing power relative to the total hash rate of the Bitcoin network, denoted as H. The expected revenue, denoted as $\langle V \rangle$, is formalized in Equation 3.3.

$$\langle V \rangle = (R+M)\frac{h}{H}(1-\mathbb{P}_{orphan}) \tag{3.3}$$

Replacing equations 3.2, 3.3, and A.1, with Equation 3.1, the *miner's profit equation* is defined as:

$$\langle \Pi \rangle = (R+M)\frac{h}{H}e^{-\frac{\tau}{T}} - \eta h \mathcal{T} \tag{3.4}$$

A rational miner's goal is to maximize $\langle \Pi \rangle$, which is inversely proportional to the total hashing power of the Bitcoin network, but directly related to three main factors: (1) the reward and transaction fees $(R+M)$, (2) the individual hashing power (h), and (3) the probability of *not* orphaning the block just mined $(1-\mathbb{P}_{orphan})$. The individual hashing power's margin for increasing revenue is minimal, as it is normalized with the total hash rate of Bitcoin. Additionally, reducing the probability of orphaning a block would require reducing the orphaning rate of the network, which a miner cannot directly control. Considering that factors (2) and (3) have a diminishing effect on a miner's potential earnings, it can be concluded that a rational miner can earn additional revenue solely from factor (1). Figure 3.1 illustrates the proportion of miners' earnings over time. As the mining reward is periodically halved every 210,000 blocks, *transaction fees become the main source of revenue for miners in the long run.*

3.1.2 Calculations

To support our previous statement, we analyzed real-world miners' profit using the revenue and cost equations above. We observed that with a zero-fee policy, miners struggle to make any profit and instead lose money unless the electricity price is near zero. To conduct this analysis, we assume that miners are using Antminer S19 Pro, which has a hashing rate of 110 TH/s and a Miner Power Efficiency (MPE) of 29.55 J/TH. We then examine electricity prices (e_p) for countries with low (Qatar with 0.032 \$/kWh), medium (U.S. with 0.162 \$/kWh), or high (Denmark with 0.469 \$/kWh) costs. Finally, we adapt Rizun's equations

1. Detached or orphaned blocks are valid blocks that are not part of the main chain. They can occur when two miners produce blocks at the same time, and one block gets discarded because of higher propagation delay. See Appendix A.1

to calculate the daily profit assuming a block creation time of ten minutes, a Bitcoin price of $ 15,000, and minimum and maximum fees of $ 1 and $ 70, respectively. We ignore the orphaning rate as it was recorded to be 0.31 % of blocks mined per day from 2014 to 2017[2], and Shahsavari et al. [124] measured it to be even lower at 0.09 %, resulting in a negligible decrease in daily revenue if the block size is kept at 1.1 MB. We calculate the daily profit $\langle \Pi \rangle_{\text{day}}$ as:

$$\langle \Pi \rangle_{\text{day}} = \langle V \rangle_{\text{day}} - \langle C \rangle_{\text{day}} \tag{3.5}$$

if s = 86,400 seconds in one day, we have:

$$\langle V \rangle_{\text{day}} = g_B \frac{h}{H} B_{\text{day}} \qquad g_B = (R \times \text{BTCP}) + M \qquad B_{\text{day}} = \frac{s}{\mathcal{T}}$$

$$\langle C \rangle_{\text{day}} = e_p \eta_{\text{day}} \qquad \eta_{\text{day}} = s\eta h \qquad \eta = \frac{\text{MPE}}{3.6 \times 10^{12}}$$

Since 1 W = 1 J/s, the cost per hash in kWh/TH can be calculated by dividing the MPE by 60 seconds/minute × 60 minutes/hour and then by 1,000 W/kW. Since η is represented in kWh/hash, it is further divided by 10^{12}.

Figure 3.2 shows profit calculations for miners using Antminer S19 Pro in different countries. It is evident that, with the selected Bitcoin price of $ 15,000 and a reward of ₿ 6.25, it is not feasible to expect low fees. In fact, with a uniform fee of $ 1, mining is only profitable for individuals where the electricity price is near zero. With the current Bitcoin hash rate, it is not possible to profit with a uniform fee of $ 1 in the U.S., and a rational miner should not try to recover from this loss by increasing their individual hash rate, as this will only increase their mining costs. Looking at the first U.S. plot in Figure 3.2 and fixing the x-axis value at the current Bitcoin hash rate while moving along the y-axis, we can observe a downward profit trend as the individual hash rate increases.

As shown in Table 3.1, the price of Bitcoin also plays an important role in determining profit. However, this price cannot be controlled by miners, and the uncertainty associated with it means that adopting a uniform fee of $ 1 will lead to a loss of profit in many different scenarios, such as changes in the Bitcoin price, increases in the total hash rate, or shifts in electricity prices. Table 3.1 also considers the upcoming scenario of halving the reward. It shows that even with a uniform high fee of $ 70 per transaction, miners will have little profit unless the Bitcoin price increases again above $ 30,000.

2. According to data stored in blockchain.info at `https://www.blockchain.com/explorer/charts/n-orphaned-blocks`

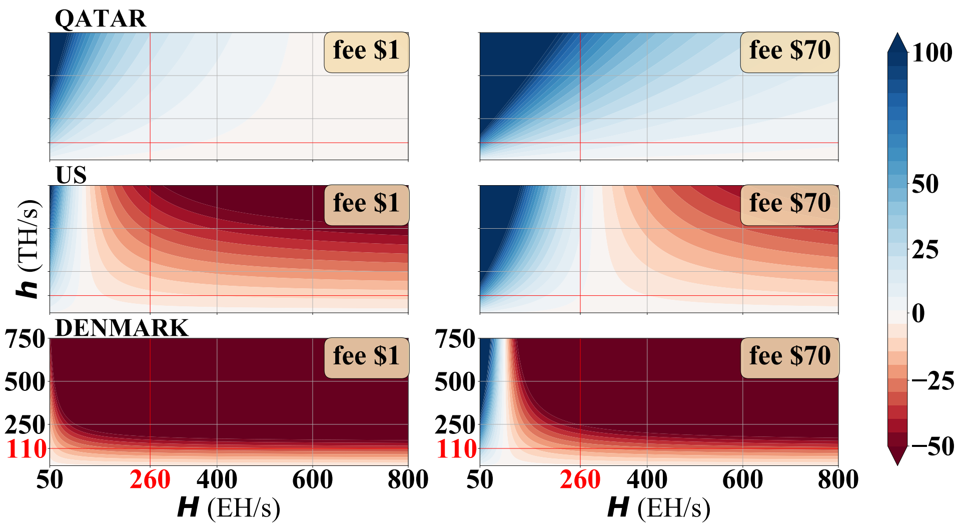


Figure 3.2: The profitability of Bitcoin mining operations is influenced by several factors, including the miner's individual hashing power (e.g., 110 TH/s per second for the Antminer S19 Pro) and the overall hash rate of the Bitcoin network (currently 260 EH/s). In this analysis, the cost of electricity is considered for three countries: Qatar, the United States, and Denmark, with average fees ranging from $1 to $70. The daily profit for these miners is estimated to fluctuate between a loss of $50 and a gain of $100, depending on the specific market conditions.

e_p ($/kWh)	Daily profit in U.S. dollars with $1 fee					
Bitcoin price ($)	1000	10000	15000	30000	60000	100000
Qatar : 0.032	-1.99	1.43	3.33	9.04	20.47	35.7
US : 0.162	-12.13	-8.7	-6.8	-1.09	10.33	25.56
DK : 0.469	-36.08	-32.65	-30.75	-25.04	-13.61	1.61
	Daily profit in U.S. dollars with $70 fee					
Qatar	6.41	9.84	11.74	17.45	28.87	44.1
US	-3.72	-0.3	1.6	7.31	18.73	33.96
DK	-27.67	-24.25	-22.34	-16.63	-5.21	10.01
	Daily profit with $70 fee and halved R					
Qatar	6.22	7.93	8.88	11.74	17.45	25.07
US	-3.91	-2.2	-1.25	1.06	7.31	14.92
DK	-27.86	-26.15	-25.2	-22.34	-16.63	-9.01

Table 3.1: The profitability of the Antminer S19 Pro mining operation is influenced by fluctuations in both the market price of Bitcoin and the cost of electricity. At present, the Bitcoin network's hash rate is measured at 260 EH/s.

Why do fee-markets emerge?

As Bitcoin's popularity grew, the number of miners and total hash rate increased, resulting in a significant loss in miners' revenue. This popularity also translated into more transactions being submitted per second, leading to increased competition for inclusion due to the inherent throughput limitations of POW (1 block every 10 minutes). As discussed in Chapter 3.1.2, rational miners cannot rely on the price of Bitcoin for revenue, and they should not attempt to compensate for the loss of profit due to the rise of the total hash rate (H) by increasing their individual hashing power (h). Their mining power would be outperformed by H regardless of any individual boost, and the total costs would increase. The only rational way for miners to continue profiting is to change their behavior towards transaction inclusion in a profit-oriented manner, focusing on *fees* and *transaction size*.

3.2 Auction Market Types

This section presents different auction schemes that can be adopted by miners when fee markets emerge in POW-based blockchains. In an auction market, buyers and sellers enter competitive bids simultaneously, and the good trades when the highest bidding price matches the lowest selling price. This competitive bidding process helps determine the equilibrium price at which the market clears and the trade is executed. For POW-based cryptocurrencies, miners act as auctioneers while users are bidders. The users are *buying space in the next mined block*, while the miners are *selling their block space availability* as competition increases. In the largest blockchain implementations of Bitcoin and Ethereum, miners have adopted different inclusion schemes over time, including the well-known First-Price Sealed-Bid Auction (FPSBA), Uniform-Price Auction (UPA), and Second-Price Auction (SPA).

3.2.1 First-Price Sealed-Bid Auction

A FPSBA is a common type of auction in which all bidders simultaneously submit sealed bids, unknown to other participants. The highest bidder pays the submitted or truthful price. For POW-based blockchains governed by miners who use FPSBAs, users propose their transaction fees without knowing the bids of other users. If their transactions are included in a block, the submitted bids will be paid. While Bayesian Nash equilibrium (BNE) (see Appendix A.3) is efficient for FPSBAs with identical items and symmetric bidders, these equilibria are unlikely to occur in practice. The result is a strategic bidding scheme in which transactions initially bid low and then increase their fee if approval is taking too long. This scheme leads to fee instability in Bitcoin, particularly when the number of transactions to be processed increases and bidders fear being left out of the next mined block. Figure 3.3 shows different outcomes for two bidders. The first bidder, who submits tx1, overpays for their space in the block due to incorrect assumptions about other sealed bids, while the second bidder, with tx2, is able to be included with a minimal fee increment.

3.2.2 Uniform-Price Auction

The UPA scheme, as depicted in Figure 3.4, charges each bidder with the price paid by the lowest included bid. The concept behind this approach is that any bidder can offer as much as they believe their transaction is worth, regardless of the size of their bid. This means that a bidder's bid may be high, but they will not necessarily have to pay that amount unless every other offer is equally high. This scheme allows bids to affect only their inclusion in the next block, and not the price paid (non-truthfulness concept).

If a bidder offers a price of x for a transaction t_x and that transaction is included in a block, the bidder will pay a fee that is less than or equal to x. If the minimum price required for inclusion in the block is higher than x, the

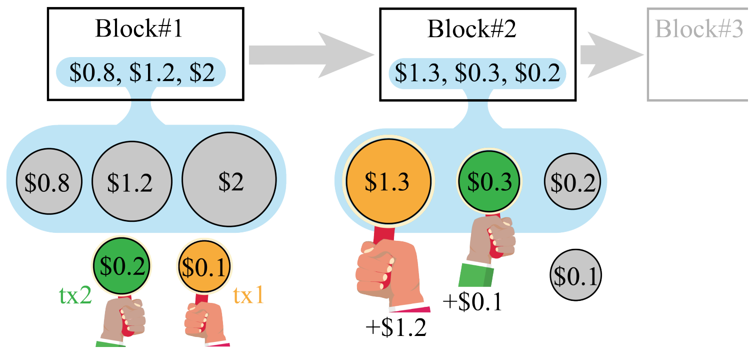


Figure 3.3: In this scenario, Users 1 and 2 submit transactions tx1 and tx2, respectively, but the fees they have paid are insufficient to be included in Block 1. User 1 then proposes a new bid using the average of all fees in Block 1. However, the other bids in Block 2 are considerably lower, resulting in User 1 overpaying for their transaction. User 2 instead decides to slightly increase their fee without taking into account the fees of previous transactions, and in this particular case, this strategy is successful. When miners adopt the FPSBA scheme, fee unpredictability is common.

transaction will not be included since the bidder is not willing to pay more than x.

Despite this solution generally preventing overpayment, in a fee market governed by rational and profit-oriented miners, it can result in two major weaknesses: (1) a block proposer can include their own transactions in a block in order to increase the clearing accepted price (as illustrated by the red $ 2 transaction in Figure 3.4); (2) a block proposer could collude with some portion of bidders, asking them to submit higher offers and then refunding them through a separate channel. These attacks are possible because a bidder can substantially boost a miner's revenue by making only a slight increase in their bid. In contrast, the FPSBA does not exhibit this vulnerability.

3.2.3 Second-Price Auction

Even though the FPSBA method discourages auctioneers from including their own transactions, bidding the true valuation can at times hinder the achievement of Nash equilibrium. An alternative solution, known as the SPA, or Generalized Second Price (GSP), is a non-truthful bidding method for multiple items. In this scheme, each bidder places a bid and, if there are more bids than available slots ($N > K$), the slots are assigned from the highest to the lowest bid. Unlike the FPSBA, the highest bidder pays the second-highest bid, the

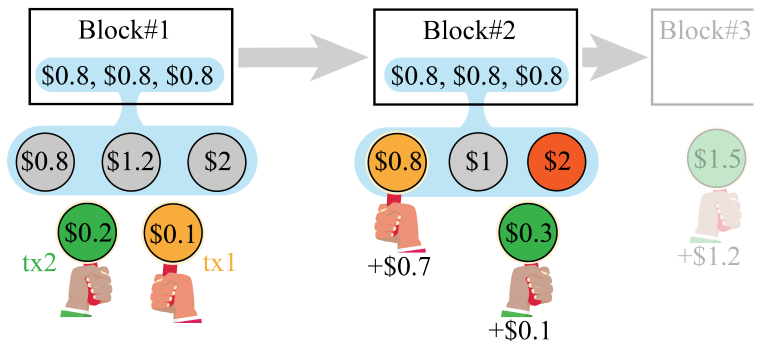


Figure 3.4: In this scenario, Users 1 and 2 submit transactions tx1 and tx2, respectively, but the fees they have paid are insufficient to be included in Block 1. User 1 then offers the previously accepted fee of $ 0.8, while User 2 slightly increases their fee. Although User 1 does not overpay, a rational and profit-oriented miner may still choose to exclude User 2's transaction and include one of their own transactions instead (e.g., the red $ 2 one) in order to raise the clearing price to $ 0.8 rather than receiving a uniform compensation of $ 0.3, as no other bidder was offering more.

second-highest bidder pays the third-highest bid, and so on. This is illustrated in Figure 3.5, where each bidder, if their bid is among the top K bids (top 3 in Figure 3.5), pays the highest bid before their own. In this example, tx1 is among the top 3 bids for Block 2 with $k = 2$, and therefore it pays the $k = 3$ bid.

Designing a mechanism to improve the FPSBA in Bitcoin is challenging, as an SPA scheme can easily be manipulated by miners who can submit fake transactions after observing the fees in the mempool. Miners can use any criteria for including transactions and can manipulate the results of the auction after learning the proposed fees, which can lead to overpayment as shown in Figure 3.6.

3.3 Evolution of Transaction Fees

The policies that miners follow for the inclusion of transactions in blocks are not publicly known, making it difficult to determine the adopted auction scheme and the preferred patterns used by validators. However, it is known that transaction fees in the Bitcoin network have evolved from a mining-based structure to a market-based ecology [125]. The distributed nature of Bitcoin and the absence of a central authority contribute to a dynamic fee structure that can be unpredictable due to a range of endogenous and exogenous factors.

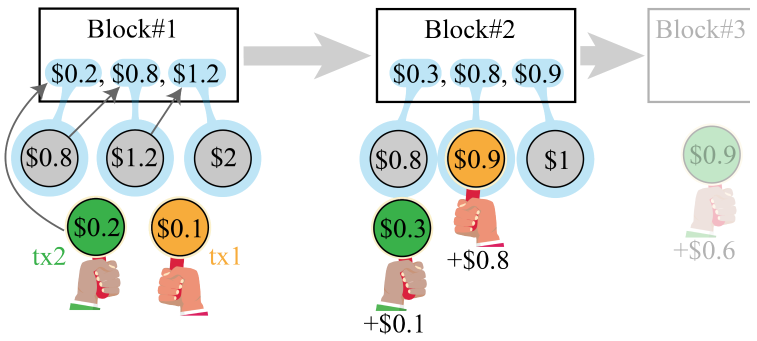


Figure 3.5: In this scenario, Users 1 and 2 submit transactions tx1 and tx2, respectively, but the fees they have paid are insufficient to be included in Block 1. Despite this, tx2's fee is used to pay for the last bid included in Block 1. User 1 then offers a higher fee of \$ 0.9, while User 2 only slightly increases their fee without being aware of the auction scheme adopted by miners. As a result, User 1's transaction is included in Block 2 at position $k = 2$, with a fee pool of [1, .9, .8, .3]. This means that User 1 pays the bid of the user in position $k = 3$, which is \$ 0.8.

This section discusses the implications of these changes and provides an overview of previous research on high fees, POW limitations in terms of scalability and throughput, and the factors that influence miner behavior. Early works in 2014 [126, 127] anticipated the high fee issue before it occurred, and later research in 2017 [65, 57] examined the relationship between transaction fees and latency. Understanding the evolution of transaction fees in Bitcoin is important as it can provide insight into the potential evolution of fee markets in other POW-based systems at scale.

3.3.1 Donations for Miners

The Nakamoto [8] paper states that Bitcoin is designed as a low-cost payment scheme, with occasional fees serving as an incentive for miners:

> "*The incentive* ***can also*** *be funded with transaction fees.*"
>
> Nakamoto [8]

The quote suggests that miners can often ignore transaction fees. Furthermore, in the official online documentation of Bitcoin before 2014 [126], we read:

> "*At the moment, many transactions are typically processed in a way where no fee is expected at all, but for transactions which draw coins from many bitcoin addresses and therefore have a large data size, a small transaction fee is usually expected.*"

While such statements may be true in relatively small environments, they do not take into account global Bitcoin adoption. Already in 2014, the study of Kaskaloglu [126] discussed the issue of high fees. The author argues that an increase in transaction fees in Bitcoin is inevitable, transforming the so-called *donations* into real fees. Donations are financially unsustainable in the long term for two reasons: (1) the cost of mining, and (2) mitigating the 51 %attack. The cost of mining depends on electricity prices and the variation between individual mining power and the total hash rate. To mitigate the 51 % attack, a majority of hashing power must remain honest, leading security in Bitcoin to follow the principle of *the more miners the merrier*, which in turn increases mining costs.

Bitcoin is also highly energy inefficient by design, and to prevent Sybil attacks, the work required to secure blocks must be difficult for miners but easy to verify for any verifier, as a form of nondeterministic polynomial time (NP) decision problem. The predetermined parameter $\mathcal{T}$ makes it impossible to speed up the mining process, and as the total network hash rate increases and costs rise, miners must find new ways to generate profits. Other exogenous factors such as difficulty, Bitcoin price, and cost of mining can change in a dynamic ecosystem of miners, investors, and users, resulting in an unpredictable scenario for determining fees and prices.

Distributed governance enables Bitcoin to transition from its current operational mode, where transaction fees are mostly a voluntary tip to miners, to a situation where a *fee market* effectively regulates all traffic. With such a fee market, low-fee transactions may potentially remain pending for hours, days, or even weeks while waiting for approval and inclusion by a miner. As the energy demands of the Bitcoin network increase and mining becomes more costly, the transition to a fee market becomes more evident. Despite being designed as a low-cost payment scheme, the system has become expensive for all parties involved.

3.3.2 Fee is Mandatory

Later studies began to examine miners' behavior with respect to transaction fees. In 2015, Möser and Böhme [127] acknowledged the role of fees as a critical factor in the stability of the system. The study provided empirical evidence of agents' behavior regarding payment of transaction fees, along with several regime shifts caused by changes in the default client software. The research demonstrated the trend of a long-established POW-based blockchain moving towards a fee-oriented market. Two years later, Bitcoin experienced a significant scalability issue with regard to transactions. Blocks became saturated and miners started to reject zero-fee transactions, in what appeared to be a fixed price auction scheme.

In 2017, we conducted a study [65] that investigated the relationship between fees and waiting times. Our findings suggest that latency and fees are inversely related, although spending more than 300 sat/byte was found to be ineffective.

In 2019, Easley et al. [125] examined endogenous and exogenous features of Bitcoin. Endogenous properties, which can be changed internally by users or miners, include transaction fees and the voluntary inclusion of transactions by miners. In contrast, exogenous properties, which are imposed by the Bitcoin protocol, include factors such as block size, network difficulty, and block reward. Although waiting time is influenced by both endogenous and exogenous factors, not all external factors affect transaction inclusion. For example, block reward

(R) does not impact waiting time. If delays are reasonable, a FPSBA scheme may be effective, as transactions carrying a minimum fee will eventually be included. However, as the number of transactions per second increases, FPSBA has been shown to be a costly solution for users.

3.3.3 Overpaying

Figure 3.6 illustrates the average transaction fee per transaction in Bitcoin from 2016 to 2022. From this data, we can observe that the scheme adopted by miners led to excessive fee payments, particularly when blocks were saturated in 2017 and 2022. The FPSBA scheme has failed to provide users with stable prices for their services, and historical analysis shows that Bitcoin users could have saved $ 272,528,000 in transaction fees, while miners could have reduced the variance of fee income by an average factor of 7.4 times [128]. Clearly, an FPSBA market is unsuitable for large-scale POW blockchains. The market does not provide stable coin prices, resulting in unpredictable transaction fees and enormous variance. Additionally, capacity and demand do not always align, forcing users to overpay for space in the block.

In 2018, Ethereum co-founder Vitalik Buterin proposed improving how miners in Ethereum are paid by adopting the UPA scheme [129], which benefit both miners and users. As discussed in Section 3.2.2, the proposed solution mitigates overpayment as no transaction affects the fee paid by other transactions. However, it also has major drawbacks, such as the potential for malicious miners to increase the fee paid by every bidder.

The study of Messias et al. [130] reveals that miners in practice deviate from

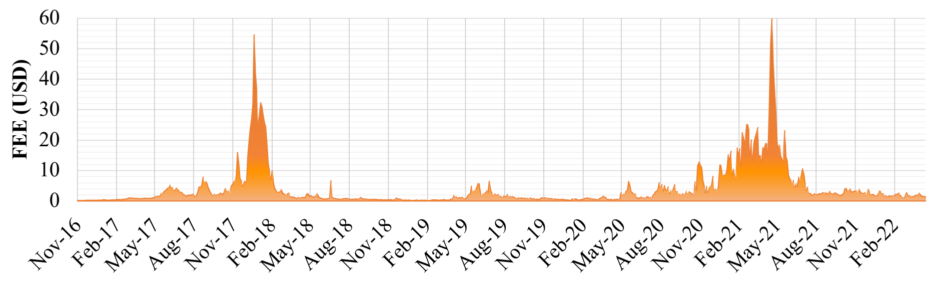


Figure 3.6: The chart shows the daily average of Bitcoin transaction fees in USD from 2016 to 2022, with significant unexpected spikes that result in excessive fee payments for users. Data source: blockchain.com at `https://www.blockchain.com/explorer/charts/fees-usd-per-transaction`.

the FPSBA scheme and instead adopt alternative strategies. A commonly used approach by many fee estimators is to adopt the *fee-per-byte* dequeuing policy, where transactions are ordered by their feerate (fee per byte) rather than by their fees. This has significant economic implications for users, as overpayment is the norm. Messias et al. [130] demonstrate that in June 2019, more than 30% of transactions offered a feerate that was two orders of magnitude higher than the minimum recommended.[3]

Basu et al. [128] introduce a mechanism inspired by the GSP, where each transaction is assigned a value. Every bidder knows their assigned value and the number of bidders, but not the values assigned to other bidders. Prices are paid using a UPA, where all bidders pay the $(K+1)^{th}$ bid. This could be a potential solution for Bitcoin if miners could commit to an auction form and the protocol could use bids to resolve payments. Another variant of the GSP auction model is presented by Li et al. [131, 132], where they employ a novel *rank-by-cost* rule to order transactions. The cost is calculated using the user-submitted fee and the waiting time. With this approach, they show that the daily saved fees for users can reach an average of ₿24.5985. Our model for transaction inclusion does not rely on a single auction scheme. Instead, we group the possible K-slots available at every block epoch and use a novel ranking system based on *feerate*, *waiting time*, and *current space available in the block* to generate a scheme that is likely followed by miners.

3.3.4 Miners and Users Equilibria

Figure 3.7 compares the evolution of transaction fees with exogenous factors such as the total hashing rate in Bitcoin and the number of new incoming transactions in the mempool. The first plot shows that the hashing rate, which affects difficulty, does not appear to have a significant impact on transaction fees. Drops in the total hashing power are caused by miners leaving the network (e.g., May-June 2021), as they no longer have incentives to continue mining due to low fees or high mining costs. The overall hashing rate has followed a monotonically increasing trend, indicating that it has always been profitable for miners to mine. The second plot in Figure 3.7 shows that the mempool count and transaction fees follow a similar trend. Although we acknowledge that correlation does not imply causation, an understanding of POW ecosystems can help us make educated conjectures about miner behavior. We observe that when the rate of incoming transactions is low, fees tend to be lower. However, as the mempool starts to fill, the equilibrium shifts, and only high-fee transactions

3. 10^{-5} BTC/kB $\equiv 1,000$ sat/kB $\equiv 1$ sat/byte, according to Bitcoin Core

are included by miners. In May 2017, the influx of new incoming transactions saturated the mempool, resulting in high fees (in BTC) but low relative USD value due to the favorable BTC-USD conversion rate for USD fees.

The transaction fee equilibria between miners and users must take into consideration a range of factors, such as the price of Bitcoin, the cost of electricity, and network difficulty, which exhibit strategic complementarity. Often, such equilibria are Pareto-ranked,[4] as transaction fees can lead to user non-participation, and conversely, low fees can cause miners to exit the market. The unpredictability of POW-based blockchain systems at scale, coupled with the insecurity stemming from the uncertain non-inclusion of transactions, can result in overpayment. Increasing transaction fees may attract more miners, but it also raises the difficulty level, thereby increasing the costs of mining. This demonstrates that higher revenue does not necessarily equate to higher profits. Figure 3.8 presents the calculation of the revenue over time for a single miner using Antminer S7 from 2015 to 2017, Antminer S9 from 2018 to 2020, and Antminer S19 Pro from 2020 under the assumption that no fees are paid by users. It should be noted that the profit is also influenced by changes in electricity prices, Bitcoin hash rate, Bitcoin price, and block rewards over time. Our analysis shows that revenue was particularly high when the Bitcoin price

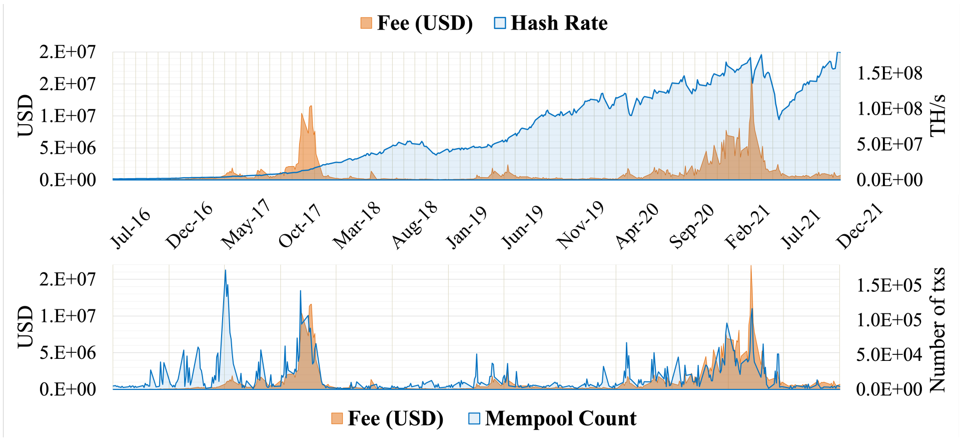


Figure 3.7: Factors that may contribute to high fees in Bitcoin are represented by the sum of transaction fees (in USD) paid daily from 2016 to 2021. This value is compared first with the total hashing rate in Bitcoin and then with the daily mempool count, which is the number of new incoming transactions submitted to Bitcoin each day. Open and available data of Bitcoin are fetched of blockchain.com.

4. No improvements can be made to at least one participant's well-being without reducing any other participant's well-being

increased in late 2017. However, despite the fact that the Bitcoin price was even higher in 2020, revenues were lower due to an increase in the total hash rate. We also observe that revenues began to increase again after January 2021, possibly due to a decrease in the number of miners on the network (as indicated by the drop in hash rate in Figure 3.7). Determining a pattern for transaction inclusion poses a dynamic challenge for the evolving Bitcoin blockchain, where transaction fees cannot solely determine inclusion. Our approach offers an alternative by considering fees as a significant factor in transaction inclusion, but not the only one.

Approach

Our objective is to address two key challenges arising from the fee market in Bitcoin: (1) unpredictability of fees and (2) instances of users overpaying. For this, we propose to construct a model that employs ML techniques, such as a multi-layer NN, to define a pattern for the inclusion of transactions in the Bitcoin network. The model considers both the block size and the mempool size, which sets it apart from traditional FPSBA predictors. Additionally, we incorporate features that are not fee-based so that our model is not solely dependent on a SPA scheme for its predictions.

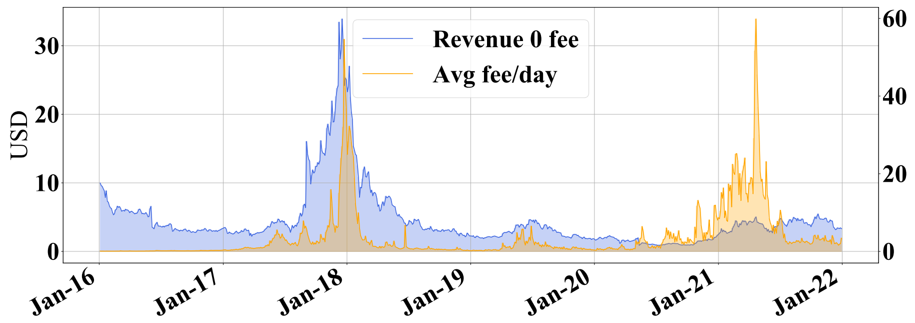


Figure 3.8: The calculation of revenues without fees takes into account the total hash rate of Bitcoin, its price, and historical electricity prices in the United States from 2016 to 2022 for each point. The block reward decreases from a starting value of ₿ 50 to ₿ 6.25, and it is assumed that a miner will switch from using the Antminer S7 (released in 2015), Antminer S9 (released in 2017), to the Antminer S19 Pro as mining hardware advances. We derived the revenue using open data of Bitcoin about total hash rate, transaction fees, and Bitcoin price, fetched on blockchain.com at `https://www.blockchain.com/explorer/charts/fees-usd-per-transaction`.

The utilization of ML models for the identification and prediction of patterns within large datasets has been widely adopted across various fields. For example, in the field of hydrology studies such the one by Diez-Sierra and del Jesus [133] have employed large amounts of data to recognize patterns and trends. Similarly, within the domain of network security, research such as Nanda et al. [134] have leveraged these models to detect and prevent cyber-attacks. Additionally, the work of Yazdinejad et al. [135] has demonstrated the efficacy of ML techniques in identifying cryptocurrency malware threats.

When analyzing patterns related to transaction inclusion, a significant volume of heterogeneous data is often encountered, which necessitates organization and structured analysis. An ML-based approach, in this case, allows for predictions and decisions to be made using a subset of the data (i.e., training data) and has been widely adopted across multiple industries, including education [136, 137], business and marketing [138, 139], healthcare [140, 141], financial services [142, 143], and transportation [144, 145]. As was demonstrated in our previous study [57], the utilization of ML techniques to analyze and identify patterns in large datasets is a viable approach, and the transparent and readily available Bitcoin blockchain serves as an excellent source for this purpose.

Summary

This chapter delved into the principles and rules that govern the Bitcoin ecosystem at scale, with a specific focus on the interdependence between users and miners. We explored how miners generate profits, the associated costs they incur, and the crucial role these factors play in determining transaction inclusion. We also presented mining profit calculations for corner case countries with an high, medium, and low electricity price, and show how mining can be profitable or not. The emergence of a fee market in POW-based blockchains, including miners' auction schemes, is discussed, emphasizing its impact on fee dynamics in Bitcoin. Understanding this market is crucial for formalizing a transaction inclusion pattern.

In the upcoming chapter, we outline our data acquisition and organization methodology. We discuss essential elements for proper model functioning and explore unnecessary elements. We cover different data acquisition methods, such as using the Bitcoin core client software and external APIs. The effectiveness of our approach relies on optimizing and organizing the dataset to meet our objectives.

4 Blockchain Analytics System

In this chapter, we present the methods and techniques used for data acquisition from the Bitcoin blockchain, as well as the structure of our dataset. Our study employs a combination of web scraping, APIs, and direct access to the blockchain using dedicated Bitcoin Core software, to acquire relevant data. The acquired data is structured and pre-processed to be suitable for the analysis in the following chapters. The system we designed and implemented to automate these tasks is referred to as the Blockchain Analytics System (BAS). This chapter provides an overview of the data acquisition process and the structure of the dataset, including any cleaning or pre-processing steps taken, which will be used in the subsequent analysis.

4.1 Data Sources

The Bitcoin blockchain is a rich source of information that can be used to gain valuable insights into the workings of the Bitcoin network and its underlying technology. Despite the availability of this data, the process of acquiring and storing it in a manner that is suitable for analysis requires a comprehensive understanding of the underlying technology as well as the various methods and tools available for data collection. The sheer volume of data generated by the Bitcoin network can exacerbate the difficulty of this task. Effectively collecting and storing data from the Bitcoin blockchain necessitates a combination of technical expertise and careful planning to ensure that the data is acquired in a manner that is both efficient and rigorous. In this study, three distinct methodologies are employed to acquire blockchain data: utilization of web sockets and APIs offered by blockchain.com, utilization of the Bitcoin Core client software, and implementation of web scraping techniques.

4.1.1 Web Sockets and APIs

The `blockchain.com` website offers web socket APIs as a service.[1] While we have not extensively utilized this service, we found it useful for real-time monitoring of the Bitcoin mempool. Specifically, it allows for the observation of the number of pending transactions awaiting confirmation. To monitor pending transactions, we subscribed to a designated endpoint and implemented a process to periodically update a local file at regular intervals. This file stores the hash of each retrieved unconfirmed transaction. Through this method, we were able to compare instances of unconfirmed transactions to the contents of the file and subsequently identify those that have been confirmed, allowing for their removal from the list.

Pseudo-code in Algorithm 1 demonstrates how to use web socket APIs to fetch unconfirmed transactions in the Bitcoin network. We assume that the `WebSocketAPI` class has implemented methods for subscribing, fetching data, and unsubscribing. Our `processData()` function is available to process the fetched data, and to manage the storage of unconfirmed transactions on a local level. This includes periodically updating the unconfirmed transactions at a specified `timer` interval and removing transactions that have been approved upon the creation of new blocks.

BAS primarily obtains stored information from the blockchain.com[2] platform by utilizing their RESTful APIs. BAS includes a function dedicated to the retrieval of block information, `FetchBlocks()` in Algorithm 2, utilizing RESTful API endpoints (e.g., the RAW BLOCK endpoint listed in Appendix C.1) to fetch and store said information within a locally maintained dataset.[3] The `FetchBlocks` function in Algorithm 2 takes three parameters: the hash of the block where

1. connection URL: `wss://ws.blockchain.info/inv`
2. blockchain.com at `https://www.blockchain.com/explorer/api`
3. Raw block endpoint at height 600000: `https://blockchain.info/rawblock/600000`

Algorithm 1 Retrieve unconfirmed transactions using web sockets.

```
1: procedure FetchUnconfirmedTxs(timer, sub, unsub)
2:     ws ← WebSocketAPI(sub, unsub)
3:     ws.subscribe()          ▷ subscribe at {"op":"unconfirmed_sub"}
4:     while True do                ▷ time listening the socket is arbitrary
5:         sleep(timer)
6:         unconfirmed_txs = ws.fetchData()
7:         processData(unconfirmed_txs) ▷ manipulate data, store it locally
8:     ws.unsubscribe() ▷ unsubscribe at {"op":"unconfirmed_unsub"}
```

the retrieval should start, if it is not provided, it will try to read from a previous stored file; the number of blocks to be fetched; a boolean read flag, that if set to `True`, it will read from a previous stored file. The function uses a `getJson()` procedure to fetch the JSON data of the first block (starting from the provided hash or height) and converts it into a Block and Transaction objects. On each iteration, the JSON data of the current block is obtained, parsed into a Block object, and appended to the previously initialized block dataset (ds_b in Algorithm 2).

Additionally, a transaction dataset (ds_t in Algorithm 2) is initialized and populated with transaction objects extracted from the current block. The iteration proceeds by updating the hash to the next block, until the specified number of blocks have been downloaded. The function ultimately returns both the block and transaction datasets.

4.1.2 Bitcoin Core

Bitcoin Core is the reference open source implementation of the Bitcoin protocol.[4] That provides functionality for full node participation were the entire history of the Bitcoin blockchain is downloaded and maintained on the user's device. Bitcoin Core is considered the most secure implementation of the Bitcoin protocol but also requires a significant amount of storage and memory to run. In our case, we hosted a full node on Azure, which is currently maintaining a storage capacity of over 400 GB for the purpose of holding blockchain data. The process of retrieving data from the Bitcoin Core platform can be challenging in terms of efficiency and time. One specific example is the calculation of transaction fees, which requires the examination of spent outputs across multiple transactions, as a result of the Unspent Transaction Outputs (UTXO)

4. `https://bitcoin.org/en/bitcoin-core/`

Algorithm 2 Retrieve block information using RESTful API.

```
1: procedure FETCHBLOCKS(hash, n, read)
2:     $ds_b$ ← Pandas.DataFrame()                          ▷ initialize block dataset
3:     if read is True then
4:         $ds_b$, $ds_t$ ← readDS()                  ▷ read last stored dataset instance
5:     else
6:         for 0 to n do
7:             data ← getJson(RAW BLOCK + hash)      ▷ raw block endpoint
8:             b ← Block(data)                        ▷ create a Block instance
9:             $ds_t$ ← Pandas.DataFrame()        ▷ initialize transaction dataset
10:            for all tx in b do
11:                t ← Transaction(tx, b)        ▷ create Transaction instance
12:                append(t, $ds_t$)                      ▷ append row to $ds_t$
13:            append(b, $ds_b$)                          ▷ append row to $ds_b$
14:            hash ← b.next_block
15:        return $ds_b$, $ds_t$
```

model implemented by Bitcoin.

During our experiments we have observed that Algorithm 3 results in an average retrieval time of 30 seconds for fee information within a single block. In contrast, utilizing APIs for data retrieval can provide the same amount of fee information across 10 blocks within the same time frame, illustrating a significant improvement in efficiency.

Algorithm 3 Obtaining information about one transaction fee in Bitcoin Core.

```
1: procedure GETTXFEE(t)
2:     $s_{in}$, $s_{ou}$ ← 0                      ▷ sums of transaction inputs and outputs
3:     for inp in t.vin do                                   ▷ for each input in t
4:         index ← inp.vout                  ▷ index of the spent output in inp
5:         $tx_{in}$ ← getRawTransaction(inp.txid) ▷ local call using bitcoin-cli
6:         $v_{in}$ ← $tx_{in}$.vout[index].value                 ▷ current input value
7:         $s_{in}$ ← $s_{in}$ + $v_{in}$                        ▷ update transaction input
8:     for out in t.vout do                                 ▷ for each output in t
9:         $v_{ou}$ ← out.value                                ▷ current output value
10:        $s_{ou}$ ← $s_{ou}$ + $v_{ou}$                       ▷ update transaction output
11:    return $s_{in}$ - $s_{ou}$                           ▷ return transaction fee
```

The function detailed in Algorithm 3 is utilized to calculate the transaction fee of a given transaction in JSON format. The algorithm iterates through the transaction inputs, `t.vin`, of the transaction and retrieves corresponding information by utilizing the `getRawTransaction` command provided by the Bitcoin-cli (Appendix A.1). The sum of all inputs, s_{in}, is then computed. Subsequently, the algorithm iterates through the transaction outputs, `t.vout`, and computes the sum of all outputs. The transaction fee is calculated as the difference between the sum of inputs and the sum of outputs. The function then returns this value as the transaction fee.

4.1.3 Web Crawling

The use of web crawling for data acquisition, presents a significant challenge due to the need for a comprehensive analysis of each individual HTML page. This process cannot be easily fully automated, and thus requires manual examination of each page. In our study, we utilize web crawling only in specific instances where alternative methods of data retrieval are not feasible. One example is the identification of miners utilizing their names rather than IP addresses, as this information may not be readily available through alternative means. The class representation of the web crawler, as depicted in Figure 4.1, illustrates the inheritance relationship between the classes `TransactionPage`, `BlockPage` and the parent class `Page`. The former two classes implement specific methods, namely `get_latency()`, `get_fee()`, and `get_miner()`, which extract and parse relevant information such as the latency of a transaction in seconds, the transaction fee in satoshi,[5] and miner information, from the content of an HTML web page of a certain transaction, or block, on the Bitcoin blockchain. Algorithm 4 is an example of how the crawler methods work. The helper function `findHTML()` takes in a page content string, a starting substring, and an ending substring as input. It finds the first occurrence of the starting

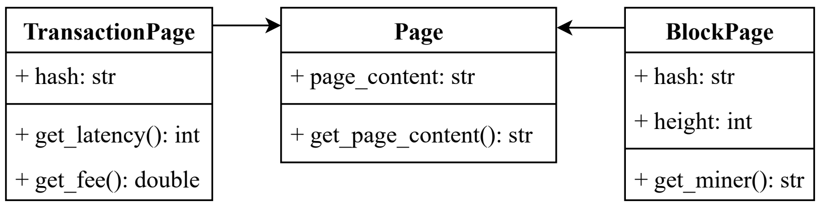


Figure 4.1: A class diagram representing the entity-relationship model for web crawler classes, with the `TransactionPage` and `BlockPage` classes inheriting from the `Page` class, and implementing methods for obtaining transaction latency, fees, and miner from the web page content.

5. satoshi, or sat, is the unit of Bitcoin, 1satoshi = 0.00000001 BTC

substring in the page content and finds the first occurrence of the ending substring after it. It then returns the substring of the page content between these indexes, or an empty string if either substring is not found.

Algorithm 4 Retrieving transaction fee from an HTML page.

```
1: procedure GET_FEE
2:     p ← find 'Fees' in page content and returns a portion
3:     p ← findHTML(p, '">', 'BTC</')                ▷ start and end char
4:     fees ← re.findall("\d+\.\d+", p)               ▷ find float
5:     fees ← float(fees[0]) $\times 10^8$            ▷ from BTC to satoshi
6:     return fees
```

4.2 Data Structure

Once the data is retrieved, it is stored locally in the following structure:

```
dataset
└─ Mmm-yy : samples for a specific month and year
    ├─ blocks
    │   ├─ info.txt : metadata for block files
    │   └─ $\mathcal{D}_B$ : block dataset, partitioned every 30 MB
    └─ transaction
        ├─ info.txt : metadata for transaction files
        └─ $\mathcal{D}_T$ : transaction dataset, partitioned every 30 MB
```

Our approach to data storage involves separating blocks and transactions, and storing their respective information in separate datasets. This design mitigates redundancies and conserve storage space, as certain information pertaining to blocks is deeply ingrained within every transaction, and would otherwise be repeated numerous times within a single block. Hence, we categorize the datasets containing raw transactions and raw blocks, referred to as $\mathcal{D}_T$ and $\mathcal{D}_B$, respectively, as those in which information is stored in its original format as it was fetched, prior to any processing or feature engineering. The data is organized by month, for facilitating the construction of ML models and the evaluation process in the future. To mitigate the potential issues associated with handling large files, the transaction dataset is partitioned into smaller files with a maximum size of 30 MB. The file structure of the dataset is depicted above and a portion of the dataset have been made publicly available on the Dataverse platform as referenced in [89].

4.2.1 Blocks

For each month the `block` folder contains the metadata file, `info.txt` (Appendix C.2), and the data. The former serves as a means to store key information about the dataset, such as the start and end date, start and end hash, and the number of blocks retrieved. The latter comprises information about the blocks that occurred during that month. The Block class in Figure 4.3, is a representation of a single block in the blockchain, and is instantiated with a block in JSON format. It comprises various attributes, both raw data, which are saved as they are retrieved, such as the block hash, height, hashes of previous and subsequent blocks, or block size. Processed data derives from various attributes or is acquired from external sources when those attributes are not available within the JSON object, for instance, the block creation time.

The Algorithm 5 defines two methods for calculating processed data. The `get_miner()` procedure uses the web crawler defined in Section 4.1.3 to retrieve information about a miner associated with a block. The `get_bct()` procedure calculates the block creation time by fetching the previous block from the analyzed block's epoch.

Algorithm 5 Block's methods.

```
1: procedure GET_MINER                          ▹ using web crawler to get miner info
2:     return BlockPage(self.hash).get_miner()
3: procedure GET_BCT                            ▹ block creation time
4:     prev_b ← getJson(RB + prev_b)            ▹ fetch previous block
5:     return self.epoch − prev_b.epoch
```

4.2.2 Transactions

Similarly to that of the block files, the `transaction` folder comprises the metadata file, referred to as `info.txt`, as well as the corresponding data. The representation of the Transaction class is illustrated in Figure 4.3, and, similar to the block files, it encompasses both raw and processed data. Algorithm 6 outlines the methodology utilized to calculate the processed features for each retrieved transaction. The calculation of the transaction fee is accomplished by utilizing information regarding the inputs and outputs of the transaction, as described in Appendix C.3. As this information, present in the JSON response, is not germane to our research objectives, it is not retained in our local dataset $\mathcal{D}_T$.

Algorithm 6 Transaction's methods.

```
1: procedure GET_TL
2:     return self.b_epoch − self.epoch
3: procedure GET_DELTA                              ▷ transaction waiting time
4:     prev ← self.b_epoch − self.bct                 ▷ previous block epoch
5:     return prev − self.epoch
```

4.2.3 Data Processing Pipeline

Figure 4.2 illustrates the data processing pipeline in the BAS system, starting with the retrieval of data from various sources, as described in Section 4.1. Most of data exchanged between the sources and the ingestion engine is through JSON messages. Processed data is stored locally using the Pandas library [146]. The ingestion engine performs pre-processing, selecting, and extracting relevant features, which are saved in the form of NumPy text files [147] for use as training sets for the ML model. The chosen ML model is implemented using a TensorFlow [148] backend with Keras [149] modules. The rationale behind the selection of features is discussed in the next chapter.

The illustration presented in Figure 4.3 depicts the relationship between the datasets designated as $\mathcal{D}_B$ and $\mathcal{D}_T$. This relationship is established through a many-to-one association, utilizing the block hash (called *ha*) as a unique identifier, denoted as $r : \mathcal{D}_T \rightarrow \mathcal{D}_B$. It can be inferred that for every element, ha', within the dataset $\mathcal{D}_T$, there exists a corresponding element, ha'', within the dataset $\mathcal{D}_B$, such that $ha' = ha''$. The resulting dataset, referred to as the

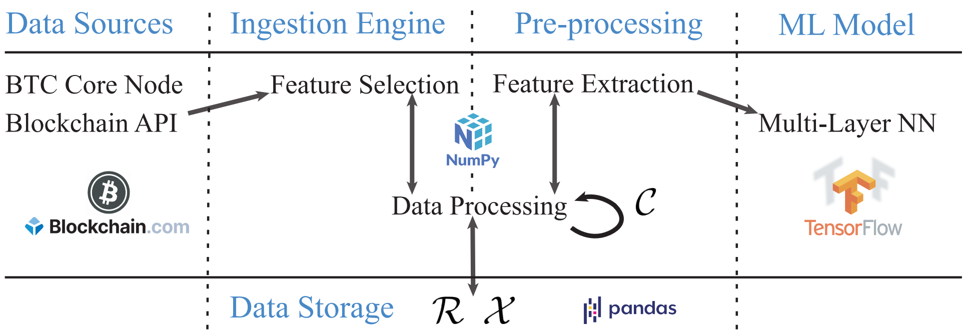


Figure 4.2: In BAS, data flow is facilitated from the blockchain to the ML framework. In this process, various datasets are utilized, denoted as $\mathcal{R}$, $\mathcal{C}$, and $\mathcal{X}$. Specifically, dataset $\mathcal{R}$ represents raw data extracted from the blockchain, dataset $\mathcal{C}$ represents a comprehensive dataset utilized during runtime, and dataset $\mathcal{X}$ encompasses all data utilized for training purposes.

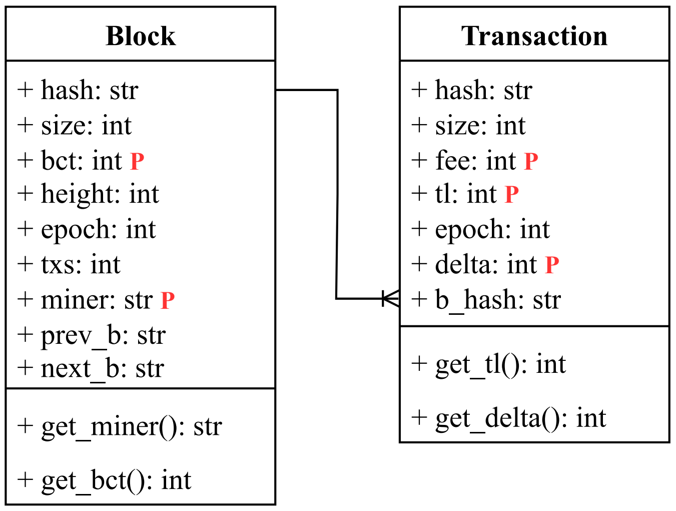


Figure 4.3: Structure that organizes the preservation of information related to both $\mathcal{D}_B$ and $\mathcal{D}_T$. The red P serves as an indicator to signify that the corresponding data has undergone processing as opposed to being obtained directly.

raw dataset and denoted as $\mathcal{R}$.

Summary

The chapter discussed data retrieval and storage methods for analyzing the Bitcoin blockchain. It mentioned three approaches for data acquisition: using web sockets and APIs for real-time and historical monitoring, utilizing the Bitcoin Core client software, and employing web scraping in specific cases. Data storage involved separating blocks and transactions into separate datasets to reduce redundancies. The data processing pipeline involved retrieving data, storing it locally, and performing pre-processing for feature selection. Efficient data handling was crucial for analysis and future modeling.

Chapter 6 will provide a comprehensive explanation of the three datasets, while in the next Chapter we formalize the transaction inclusion model based on our conjectures.

5 Transaction Inclusion Model

Time-series data analysis has been widely recognized as a valuable tool for predicting future trends in various fields, including finance [150], physics [151, 152], and economics [153, 154, 155]. In this study, we adopt a time-series observational approach to analyze transactions in a blockchain system. Our methodology is based on the collection of time-series data, where transactions are sampled on a monthly basis with a fixed interval. We also incorporate a notion of relative time, represented by the block creation epochs.

In this chapter, we present a comprehensive model for the inclusion of transactions in a POW-based blockchain system. In Section 5.1, we explain the rationale behind our choice, and how transaction data is analyzed. The time-series approach allows accurate analysis of the dynamics of transaction inclusion. Building on this foundation, we define and analyze two main factors that can affect the inclusion of transactions in a blockchain system. These factors are referred to as *revenue* and *fairness*. In Section 5.2, we examine the concept of revenue and its impact on transaction inclusion. Revenue is a critical factor as it serves as an incentive for miners to participate in the network and validate transactions. Section 5.3 explores the notion of fairness in the context of transaction inclusion. Fairness implies that all users have an equal opportunity to have their transactions included in the blockchain, regardless of their computational power or resources, upon paying an adequate fee value. Our model takes into account both revenue and fairness as complementary factors for an accurate study and prediction of transaction inclusion. A holistic view of the inclusion process must be sought through the consideration of both revenue and fairness concepts.

5.1 Observational Approach

The present study adopts a time-series-based methodology that considers the sequential arrangement of transactions and their status at the point of each block creation. The conceptual framework is explicated in the subsequent section and formalized in the one that follows.

5.1.1 Block-Epoch-Based Collection

The idea behind our observational approach is that a transaction carries different information throughout the time it is pending in the network. This phenomenon arises from the inherent supply and demand dynamics in the blockchain domain, which continually change every time a new block is appended to the blockchain. The approach uses a block-epoch-based collection, meaning that a transaction can potentially change its worth to miners every time a new block is created. As a result, we define the time interval as the time between two consecutive block creation epochs.

Each transaction is uniquely identified in each time frame by a pair of values ha_t and be_x. Here, ha_t is the hash of transaction t and be_x represents the block epoch at block height x. The transaction at a specific block height is referred to as $t^{(x)}$, representing an instance of the transaction during the time slot between block creation epochs be_x and be_{x+1}.

The goal of this study is to gain insights about network saturation and waiting times at each block epoch by tracking transactions over time. To accomplish this, three concepts are defined: (1) *timeline set*, which represents the set of all block time epochs; (2) *lifespan*, which represents the time interval between a transaction's first and last occurrence in the dataset; (3) *number of occurrences*, which represents the number of times a transaction appears in the dataset. These concepts allow us to describe and analyze the state of the network at specific points in time and how transactions are behaving within it.

5.1.2 Formalization

Definition 5.1.1: Timeline set

The timeline set, identified as $\mathbb{T}$, is a set that contains all the block-time epochs, defined as:

$$\mathbb{T} = \{be_x | 0 \leq x \leq \xi\} \tag{5.1}$$

□

Where be_x is the block time epoch at height x and ξ is the maximum block height. This set is used to delineate the time slots used for the analysis of transactions in our system.

Definition 5.1.2: Lifespan

The lifespan of inclusion of a transaction t, represented as $\mathcal{L}(t)$, is a range in epoch time, that starts when the first node sees the transaction to when it is included in a mined block. More formally, if t is included at height x, a transaction lifespan is defined as:

$$\mathcal{L}_t^x = [ep_t, be_x] \tag{5.2}$$

□

Here, ep_t is the time the first node sees the transaction and be_x is the block-time epoch at which the transaction is included in the blockchain. Understanding lifespan helps unravel its inclusion dynamics and determines how many occurrences can be counted in the complete dataset (denoted with C and explained in Chapter 6.2.3).

Definition 5.1.3: Occurrences of t

The number of occurrences (or cardinality) of a transaction t in a dataset, represented by γ_t, is identified by (ha_t, be) pairs with same ha_t but different be as follows:

$$\gamma_t = |(ha_t, be)|_{\forall be \in \mathbb{T}} \tag{5.3}$$

□

This equation sums the number of times the transaction t occurs in the dataset, where n goes from $\texttt{min}(be \in \mathcal{L}(t))$ to $\texttt{max}(be \in \mathcal{L}(t))$, and a value is summed if $be \in \mathbb{T}$. The occurrence value defines how many times a transaction appears to the miners before it is included in a block, and as such it is a measure of the network saturation and waiting time at each block epoch. A transaction t has as many occurrences in C as the number of blocks appended to the blockchain before its inclusion. In Table 5.1 we represent γ occurrences of the same transaction before its approval. The waiting time differs at each block epoch. As we will demonstrate in subsequent sections, the inclusion of additional features allows for a more comprehensive understanding of how network conditions affect the transaction's waiting time at each block epoch.

	hash	**be**	**w time**
1	49baa25...	...22049	-958
2	49baa25...	...23235	38
...			
γ	49baa25...	...27793	5399

Table 5.1: Representation of one transaction in a block-epoch-based collection.

5.2 Revenue for Miners

As seen in Chapter 2.2.1, Bitcoin's mining difficulty is determined by the total hashing power on the network (H). When the total hashing power grows, of difficulty of mining also increases, making it more expensive for miners to scale their operations.[1] As long as the honest nodes have more computational power than the malicious nodes, the blockchain will be secure. A high total hashing power is therefore preferable for the security of the network, but yields higher operational expenses for the miners to maintain their operations. Hence, transaction fees tend to correlate with hashing power. High security on the blockchain may be desirable for the integrity of the network, but it also comes with a cost that ultimately affects the miner's profit.

1. The adjustment of difficulty levels in a blockchain network is contingent upon the speed at which the collective pool of miners can solve the Proof-of-Work puzzle. An increase in the number of miners results in higher consumption levels, even though the frequency output of blocks remains unchanged.

5.2.1 Revenue Optimization Strategies

To mitigate a loss of revenue as the total hashing power (H) grows, a miner can adopt one of the following four strategies:

1. reduce its mining costs,
2. increase its hashing power h,
3. reduce the chances of block orphaning ($\mathbb{P}_{orphan}$),
4. increase the sum of transaction fees M and reward it receives R.

Reducing costs is challenging as the exogenous properties $\mathcal{T}$ and η cannot be altered, since they are normally distributed with a fixed known mean (600 seconds in Bitcoin). Therefore, reducing hashing power (h) would be in conflict with the second strategy. Furthermore, as previously discussed in Chapter 3.1.2, augmenting the individual hashing power does not always result in a higher revenue, as the associated costs may outweigh the benefits gained from the increased h. The probability of block orphaning depends on the block propagation delay, which is impacted by the block size [156]. In theory, a rational miner could reduce the orphaning rate by making its blocks lighter and faster to disseminate, but this only partially increases revenue as fewer transactions would also lead to less fees and decrease the overall network throughput. Additionally, the occurrence of orphaning has been calculated to happen only 3 times in every 1,000 blocks, and thus has limited impact on miner revenue [156].

Since the block reward R is halved every 210,000 blocks, transaction fees M are the only remaining source of profit. Most miners are assumed to act rationally [157] and implement individualized policies for transaction inclusion that maximize their profits. Because miners can arbitrarily select transactions, their main endogenous source of profit is *transaction fee*, and it is assumed that a transaction's inclusion is highly dependent on its fee and is selected rationally.

5.2.2 Feerate

Miners increase their potential revenue with each transaction they include, but simultaneously reduce their block propagation rate, thus reducing the probability of earning a reward. The time required for the propagation of blocks and achieving consensus among participants depends on the block size [158], which is generally fixed at 1 MB in the case of Bitcoin. Miners may attempt to optimize their revenue by including the maximum number of transactions paying higher fees within the fixed block size by calculating the feerate (ρ), defined as the ratio of transaction fee and transaction size as follows:

$$\rho = {}^{\phi}\!/_{q} \tag{5.4}$$

The dequeueing feerate policy is the commonly accepted norm among miners [130], and it forms the foundation of their revenue. However, many fee estimators that incorporate the feerate still result in overpayment.[2] Analyses indicate that on average, between 50 % and 70 % of transactions offer fee rates that are two orders of magnitude higher than the recommended minimum,[3] and that 88 % of all Bitcoin transaction inputs pay higher fees than necessary [130]. Thus, we contend that revenue is not the only metric to consider, and that a miner must also consider *fairness* by allocating space for transactions that have been waiting for a longer period of time, as long as they are offering fair fees, in order to sustain the overall system.

5.3 Fairness for Users

The concept of fairness in transactions encompasses the idea that a transaction, upon payment of a fee, should not be subjected to unjustified delay as a result of the arrival of newly submitted transactions with higher fees. In other words, the scheduling of transactions should ensure that no transactions are left in a state of starvation. It is essential to incorporate fairness into our model, as transactions should not be left in a state of indefinite waiting, especially after the payment of an *expected fee*. The expected fee is a value that is deemed reasonable by both parties to secure prompt processing of a transaction. The concept of fairness ensures that, once a fee is paid, a transaction should not experience undue delay due to the presence of transactions with higher fees that arrived after it.

To provide a clearer understanding of fairness, we introduce the concepts of Epoch Before Inclusion (EBI) and Relapsed Pending Transactions (RPTs). Additionally, we provide a definition for the expected fee value, which represents the fee that a transaction ought to pay in order to receive prompt processing and inclusion within reasonable time.[4] These concepts serve to form a basis for evaluating the fairness of transactions within the system.

2. E.g. bitcoinfees `https://bitcoinfees.earn.com`
3. Recommended minimum feerate is 10^{-5} BTC/kB $\equiv 1,000$ sat/kB $\equiv 1$ sat/byte, according to Bitcoin Core
4. By reasonable time we refer to a timeframe typically ranging from a few minutes to a few hours

5.3.1 Epoch Before Inclusion

EBI represents the epoch of the most recent block mined prior to the inclusion of a given transaction t. This value can be expressed formally through the Equation 5.5 as β_t, where the latest block epoch is represented by be_ξ, and ξ is the height of the last block mined.

Definition 5.3.1: Epoch before inclusion

For a transaction t at height x, denoted as $t^{(x)}$, if t has yet to be included, the Epoch Before Inclusion (EBI) is defined as $\beta_t{}^{(x)} = be_x$, where $[be_x, be_{x+1}]$ represents the time slot at height x.

$$\beta_t = \begin{cases} be_{x-1} & \text{if } t \text{ is included in block at epoch } be_x \\ be_\xi & \text{if } t \text{ is yet to be included} \end{cases} \tag{5.5}$$

□

5.3.2 Expected Fee

The expected fee value determines the appropriate fee for a given transaction based on its size. The value is calculated by multiplying the feerate (denoted as ρ^* in the equation) by the size of the transaction (denoted as q).

Definition 5.3.2: Expected fee

The feerate must be greater than or equal to 1 sat/byte, which is the minimum feerate required for a transaction to be processed. The expected fee value is expressed as:

$$\mathbb{E}[f] = \rho^* q, \text{ where } \rho^* \geq 1\,\text{sat/byte}. \tag{5.6}$$

□

In this equation, $\mathbb{E}[f]$ represents the expected fee value, and the condition $\rho^* \geq$ 1,sat/byte ensures that the fee rate is sufficient to meet the network's minimum fee requirements.

5.3.3 Relapsed Pending Transaction

Next we introduce the concept of Relapsed Pending Transaction (RPT).

Definition 5.3.3: Relapsed pending transactions

Given a block height y, $\mathcal{P}^{(y)}$ represents the set of RPTs at that height. The set includes transactions that have not been included in any block until height y, have experienced at least one block creation in their lifespan, have not been included up until height y, and have a fee equal to or greater than the expected fee. The conditions are specified in Equation 5.7.

$$\begin{aligned} \mathcal{P}^{(y)} = \{\ t | ep_t - {\beta_t}^{(y)} < 0 & \quad \wedge \\ be_y < be_x,\ \text{if } \mathcal{L}(t) = \mathcal{L}_t^x & \quad \wedge \\ \phi_t \geq \mathbb{E}[f_t]\ \} & \end{aligned} \tag{5.7}$$

□

We say t is a RPT at height x, if $t^{(x)} \in \mathcal{P}^{(x)}$. The more generic definition of $\mathcal{P}$ is obtained by taking the union of all block-epoch-based $\mathcal{P}$ sets as $\bigcup_{i=0}^{\xi} \mathcal{P}^{(i)}$. We conjecture that a transaction appearing multiple times in $\mathcal{P}$ has a higher chance of being included in the next mined block.

5.4 Model Formalization

A successful transaction inclusion framework must consider the pivotal notions of RPTs, EBI, and the lifespan of each transaction. Additionally, it is critical that the framework continuously monitors the current status of the network during both the training and testing phases. To attain this goal, it is crucial to accurately record the moment of inclusion of each analyzed transaction into a newly generated block. In support of this objective, we propose the definition of a set of Temporarily Approved Transactions (TATs) which incorporates the previously mentioned key concepts.

5.4.1 Temporarily Approved Transaction

To understand the transaction inclusion process in a block-epoch-based time division, we must determine each transaction's status in each time frame. The set of TATs offers valuable information regarding the inclusion prospects of each transaction, serving as a valuable resource during both the training and testing phases. This set allows for a comprehensive understanding of the specific moments at which a transaction is slated for inclusion in the blockchain. We formalize the set $\mathcal{A}$ of TATs in Definition 5.4.1.

Definition 5.4.1: Temporarily approved transactions

Let us define the set $\mathcal{A}^{(y)}$ as the collection of Temporarily Approved Transactions at height y. This set comprises all transactions that are selected for inclusion at height y during the time interval $[be_y, be_{y+1}]$ and are eventually included at height $y + 1$. We shall refer to a transaction t as a TAT at height y if $t^{(y)} \in \mathcal{A}^{(y)}$. This set can be mathematically represented as defined in equation (5.8).

$$\mathcal{A}^{(y)} = \{\ t | \mathcal{L}(t) = \mathcal{L}_t^{y+1}\ \} \tag{5.8}$$

□

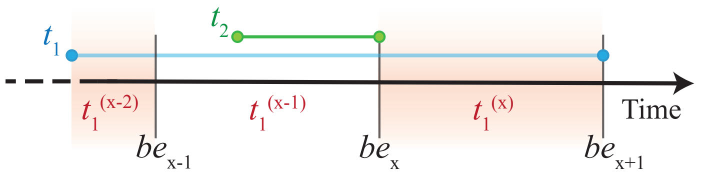


Figure 5.1: Two distinct transactions, denoted as t_1 and t_2, are initiated at a different time, respectively ep_{t_1} and ep_{t_2}. The transaction t_1 is recorded in the block at height $x + 1$, whereas t_2 is immediately recorded in the block at height x following its initiation in the same block-epoch time frame. The number of occurrences for t_1 can then be represented as $\gamma = 3$, and for t_2 as $\gamma = 1$.

Figure 5.1 depicts two instances of block-epoch-based transactions, designated as t_1 and t_2. The information conveyed by t_1 varies based on its location and is therefore named differently, such as $t_1^{(x-2)}$, $t_1^{(x-1)}$, and $t_1^{(x)}$. The membership of t_1 and t_2 in sets $\mathcal{A}$ and $\mathcal{P}$ changes over time, which is formalized in Equations 5.9 and 5.10. Given that the lifespan of t_1 is defined as $\mathcal{L}(t_1) = \mathcal{L}_{t_1}^{x+1}$, it follows that:

$$\begin{aligned} t_1^{(x-2)} &: \notin \mathcal{P}^{(x-2)} \wedge \notin \mathcal{A}^{(x-2)} \\ t_1^{(x-1)} &: \in \mathcal{P}^{(x-1)} \wedge \notin \mathcal{A}^{(x-1)} \\ t_1^{(x)} &: \in \mathcal{P}^{(x)} \quad \wedge \in \mathcal{A}^{(x)} \end{aligned} \tag{5.9}$$

In a similar manner, the lifespan of t_2 is defined as $\mathcal{L}(t_2) = \mathcal{L}^x_{t_2}$, resulting in the following equation:

$$t_2^{(x-1)} : \notin \mathcal{P}^{(x-1)} \wedge \in \mathcal{A}^{(x-1)} \tag{5.10}$$

5.4.2 Ordered Pending Transactions

It follows the definition of the Ordered Pending Transaction (OPT) set.

Definition 5.4.2: Ordered pending transactions

We define S at a given block height x as the set of the OPT, denoted as $S^{(x)}$. The set includes all non-approved transactions whose lifespans end in a block height greater than x and whose inceptions occur before be_{x+1}. The set is ordered in ascending order based on the transactions' feerate, ρ_t, as follows:

$$\begin{aligned} S^{(x)} &= \{\ t | \mathcal{L}(t) = \mathcal{L}^y_t \wedge ep_t < be_{x+1}\ \} && \forall y > x \\ S^{(x)} &= [t_1, t_2, \ldots t_n] && \text{is an ordered set} \\ \text{where}\quad & \rho_{t_1} \leq \rho_{t_2} \leq \cdots \leq \rho_{t_n} \end{aligned} \tag{5.11}$$

□

Miners play a crucial role in verifying and confirming transactions. Their incentive to do so is driven by the reward of block subsidies and transaction fees. As mentioned in Section 5.2, it is widely acknowledged that miners seek to maximize their profit by including transactions with the highest feerate (or ρ) first. This behavior is rational as it leads to the greatest financial reward for the miner. Given this reality, it is imperative that the transaction inclusion model takes into account the miners' priority of selecting transactions with the highest feerate. To this end, we introduce the concept of an ordered set of pending transactions, S, which prioritizes transactions based on their feerate. This concept is formally defined in Definition 5.4.2. By incorporating this principle, our proposed transaction inclusion model is better aligned with the financial incentives of the miners and the widely accepted norm in the blockchain ecosystem.

The proposed model employs information garnered from sets $\mathcal{P}$, $\mathcal{A}$, and S in order to make informed decisions regarding the inclusion of transactions. Understanding the impact of revenue and fairness (as described in Sections 5.2-

5.3) on sets $\mathcal{P}$ and S is crucial in determining the dynamic membership of a transaction within set $\mathcal{A}$. This information is utilized to generate new features through a feature extraction process (as detailed in Section 6.1.1).

Figure 5.2 illustrates the evolution of the $\mathcal{P}$ $\mathcal{A}$ S ecosystem over time, taking into account four transactions with varying feerate values. It can be observed in Figure 5.2 that transaction t_1, initiated within the block-epoch time frame of be_{x-2}, is included in the first block created at epoch be_{x-1}, and thus it is present in $\mathcal{A}^{(x-2)}$ and not in $\mathcal{P}$ at any time. On the other hand, transaction t_2, which was also initiated during the block-epoch time frame of be_{x-2}, has a lower feerate, implying that it could potentially remain in a pending state for a longer duration. As a result, it appears in both $\mathcal{P}^{x-1}$ and $\mathcal{P}^{(x)}$. The same criteria apply to transactions t_3 and t_4, initiated in the block-epoch time frame of be_{x-1}, and ultimately included in the blockchain, with the former being included immediately and the latter after a single block creation. The lifespan of t_4 is shown in the figure and can be represented as $\mathcal{L}_{t_4}^{x+1}$.

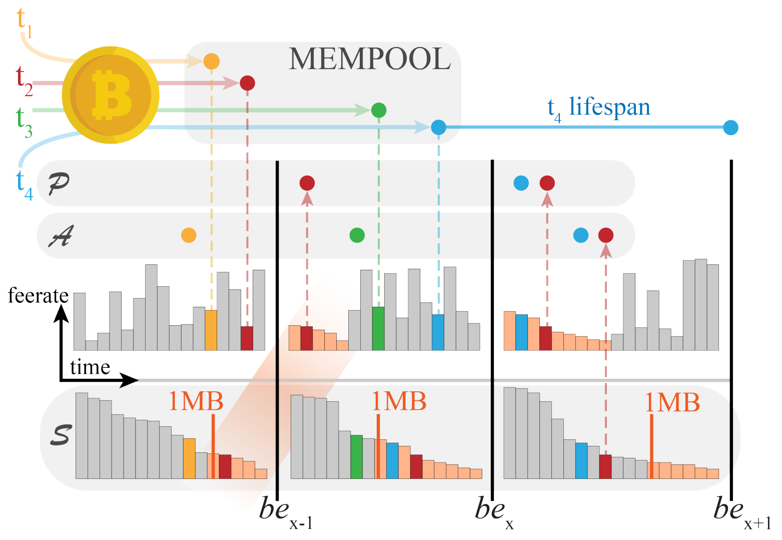


Figure 5.2: We examine a selected group of transactions, denoted as $t_1, \ldots, t_4$, spanning from time be_{x-2} to be_{x+1}, in order to graphically demonstrate their progression towards inclusion and their designation within the $\mathcal{P}$, $\mathcal{A}$, and S sets. Upon creation, transactions are inserted into the mempool and possess information regarding their feerate. Throughout various time epochs, these transactions may or may not belong to the aforementioned sets. The S set orders transactions based on feerate, the $\mathcal{P}$ set provides a temporal view of their waiting period, and finally, transactions are part of the $\mathcal{A}$ set when they are imminent for inclusion.

Summary

In this chapter, we have established a formalized perspective on the transaction inclusion model, emphasizing the notions of fairness for users and revenue for miners. Additionally, we have introduced our observational approach, which is based on the consistent evaluation of pending transactions by miners during each block creation. This emphasized the significance of utilizing a block-epoch-based collection methodology for our research goals. These foundations are crucial as they lay the groundwork for the selection and extraction of relevant features, enabling the generation of the necessary training set.

In the subsequent section, the methodology for data retrieval, processing, and feature engineering will be explained, and the model for transaction inclusion will be developed and evaluated.

6 Machine Learning Architecture

This chapter outlines our methodology for constructing training datasets, focusing on the ingestion engine and pre-processing stages. The ingestion engine utilizes locally stored data, selects relevant features, and performs data processing and storage. The pre-processing stage involves features extraction and data processing to generate a comprehensive dataset with relevant and non-redundant features. Throughout these stages, we ensure the training dataset adheres to principles of fairness and revenue, previously described. This chapter also introduces the ML model, employing the ResNet architecture to forecast future trends in the Bitcoin blockchain market. The model's optimization includes the use of the ReLU activation function, resulting in improved accuracy and faster training times.

6.1 Ingestion Engine

The ingestion engine processes and transforms raw data obtained from the blockchain into a format suitable for further analysis and storage. Data analysis libraries like Pandas and NumPy are used to perform various data processing tasks such as filtering, aggregation, and transformation. For instance, Pandas is used to read raw data from the blockchain into a data frame, perform data cleaning and pre-processing, and then write the transformed data into a storage system such as a database or a file. NumPy is used for numerical computations and analysis, such as calculating various statistics or performing matrix operations on the data. With the ingestion engine, we extract relevant information from the Bitcoin blockchain and prepare this data for further analysis.

6.1.1 Features Selection

The process of feature selection begins by downloading data from the blockchain and storing it locally. Only relevant[1] data is retrieved, allowing us to maintain a partial local copy of the Bitcoin blockchain with the necessary data for fast access. This results in an average disk space savings of 68% comparing to store the full blockchain.[2] The space saving are achieved by eliminating redundancies, preserving only the information needed for prediction, separating blocks and transactions data, and incorporating newly extracted features. The features selected for analysis are initially listed in Section 4.2 and summarized again in Tables 6.1-6.2. To construct the block-epoch-based collection discussed in Section 5.1.1, it is necessary to select features that include information about the creation time epoch of a block (`bepoch`), the inception time epoch of a transaction (`tepoch`), and the block height. Additionally, gathering data on

Block $b^{(x)}$			
Feature	**Symbol**	**Type**	**Processed**
`bhash`	ha	`string`	False
`bsize`	Q	`int`	False
`bct`	$\mathcal{T}^{(x)}$	`int`	True
`height`	block height	`int`	False
`bepoch`	be_x	`int`	False
`n_txs`	t_B	`int`	False
`miner`	block's miner	`string`	True
`prev_block`	previous block's hash	`string`	False
`next_block`	next block's hash	`string`	False

Table 6.1: Features of blocks within $\mathcal{R}$.

1. Data that are useful for the purpose of our research, like transaction data about fee or size, while information related to security and block/transaction verification are omitted
2. We store 1.3 GB of information from the actual Bitcoin blockchain, in only 0.4 GB. We refer to Table 7.1 and 7.2 for more information

transaction fee (`fee`), transaction size (`size`), and transaction latency (`tl`) is important for obtaining information on miners' revenue.

Transaction t			
Feature	**Symbol**	**Type**	**Processed**
`hash`	ha_t	`string`	False
`size`	q	`int`	False
`fee`	ϕ	`int`	True
`tl`	l_t	`int`	True
`tepoch`	ep_t	`int`	False
`delta`	$\Delta\mathcal{P}$	`int`	True
`bhash`	ha	`string`	False

Table 6.2: Features of transactions within $\mathcal{R}$.

The processed column of the tables above indicates whether any manipulation, editing, or other form of processing has been applied to the data prior to storage in our local dataset instance. The value of features, such as the block creation time (bct), is not directly recorded in the Bitcoin blockchain; instead, it must be derived during the data processing phase to define the time frame of each block-epoch-based transaction instance (as described in Section 6.2). Additionally, some newly engineered features (e.g., `delta`) require further aggregation and transformation, as we will explain in Section 6.2.2. Careful feature selection is necessary to construct the sets $\mathcal{P}$, $\mathcal{A}$, and S discussed in Chapter 5.

6.1.2 Data Processing and Storage

The features subjected to data processing prior to storage encompass transaction fee, transaction latency, feerate, and bct. Notably, feerate is computationally straightforward to derive during runtime, and its omission from saving it locally can save space. Nonetheless, we choose to include it within the stored features in our locally stored raw blockchain, $\mathcal{R}$, for ease of analysis.

Transaction Fee and Feerate

Transaction fee and feerate are revenue-based features that we use to identify inclusion patterns. The fee is computed using function f_ϕ, which is listed in Algorithm 3 and is based on the inputs and outputs of transactions. When a transaction t involves n inputs and m outputs, the transaction fee is calculated using Equation 6.1, where ϕ represents the transaction fee, inp_i represents the i-th input of the transaction, and out_j represents the j-th output.

$$\phi = \sum_{i=1}^{n} inp_i - \sum_{j=1}^{m} out_j \tag{6.1}$$

Finally, we define the fee-function f_ϕ as:

$$(inp, out) \mapsto \phi = f_\phi(inp, out)$$

The feerate-function f_ρ is designed to acquire information pertaining to transaction feerate,[3] in keeping with the revenue principle. Feerate is calculated using Equation 5.4, which gives us the function f_ρ as follows:

$$(\phi, q) \mapsto \rho = f_\rho(\phi, q)$$

Transaction Latency

The function for transaction latency computes the duration from the initiation of a transaction t or its first sighting by a miner until its approval. Transaction latency is calculated as showed in Equation 6.2, if be is the epoch of the block that includes t.

$$l_t = be - ep_t \tag{6.2}$$

Then the function for calculating transaction latency can be identified as f_l, where:

$$(be, ep_t) \mapsto l_t = f_l(be, ep_t)$$

Block Creation Time

In our block-epoch-based approach, the unit of measurement is represented by the block creation time, which refers to the duration between the creation of one block and the subsequent block. The formalization of the function that computes bct is presented in Equation 6.3.

$$\mathcal{T}^{(x)} = be_x - be_{x-1} \tag{6.3}$$

The function is denoted as $f_\mathcal{T}$, which is defined as a mapping that takes the block epoch be_x and the previous block epoch be_{x-1} as input parameters

3. as previously described, the feerate represents the ratio between the transaction fee and its size

and outputs the computed value $\mathcal{T}^{(x)}$. The function and its mapping can be expressed as follows:

$$(be_x, be_{x-1}) \mapsto \mathcal{T}^{(x)} = f_{\mathcal{T}}(be_x, be_{x-1})$$

Raw Dataset

In Section 4.2.3, we briefly outlined the procedure for generating the raw dataset $\mathcal{R}$, which involves utilizing the datasets $\mathcal{D}_B$ and $\mathcal{D}_T$ to perform a join operation over the block hash ha, which can be expressed as:

$$\mathcal{R} = \mathcal{D}_B \bowtie_{ha} \mathcal{D}_T$$

Each row in $\mathcal{R}$ represents a transaction t at the time of its approval, while each column corresponds to a specific feature. The dataset $\mathcal{R}$ is used to store information locally, perform real-time data processing, and generate additional datasets, as depicted in Figure 4.2.

Algorithm 7 illustrates the process of generating the dataset $\mathcal{R}$. The algorithm employs the Pandas library routines to parse the datasets $\mathcal{D}_B$ and $\mathcal{D}_T$ and generate the resulting dataset $\mathcal{R}$. The algorithm begins by reading the metadata files corresponding to the transactions and blocks and instantiates runtime objects for $\mathcal{D}_T$ and $\mathcal{D}_B$. Subsequently, for each block and transaction file in their respective directories, the algorithm combines the block and transaction instances. Finally, the two instances are merged based on the block hash, yielding the dataset $\mathcal{R}$ as the output.

Algorithm 7 Reads $\mathcal{D}_B$ and $\mathcal{D}_T$ and produces $\mathcal{R}$.

```
1:  procedure READ_ALL
2:      $info_b, info_t \leftarrow$ read(info files for txs and blocks)
3:      $df_t, df_b \leftarrow$ empty dataframes
4:      for bf in $info_b$.files do                          ▷ bf is block file name
5:          $df_b^* \leftarrow$ read(bf)
6:          $df_b \leftarrow$ concat($df_b$, $df_b^*$)     ▷ concatenates two Pandas datasets
7:          for tf in $info_t$.files do                  ▷ tf is transaction file name
8:              $df_t^* \leftarrow$ read(tf)
9:              $df_t \leftarrow$ concat($df_t$, $df_t^*$)
10:     return merge($df_b$, $df_t$)                          ▷ on block hash
```

6.2 Pre-Processing

The pre-processing phase is a crucial step in preparing the data stored in $\mathcal{R}$ for training the ML model. This involves a series of steps that transform the raw data into a suitable form for the subsequent analysis. The first step involves dividing the transaction information into a block-epoch-based collection, which facilitates the analysis of the current network state based on the time period.

The second step extracts new features that are related to fairness and revenue. This step aims to enhance the discriminatory power of the data and improve the accuracy of the model.

Finally, the complete set C is assembled to serve as the input for the generation of the training set $\mathcal{X}$, and to proceed with the subsequent training phase of the model. The effectiveness of the pre-processing phase is important for the overall performance of the ML model, impacting both the accuracy and reliability of the predictions.

6.2.1 Feature Vector and Complete Transaction

Next, we detail the representation of a transaction and its associated features in a block-epoch-based format, and represent a feature vector in the context of the dataset C. Once the selection of features has been made, we establish their structure in the data set $\mathcal{R}$, so that they can be used in a systematic manner by the ML model. Our ingestion engine and pre-processing phases create a temporary, locally generated version of $\mathcal{R}$ referred to as C. Each row of C constitutes an instance of a feature vector $\mathbf{t}$, which identifies a transaction within a specified time frame, as described in Chapter 5.1.2. Subsequently, the block-epoch-based representation of a transaction $t \in \mathcal{R}$ is defined as $\boldsymbol{T}$, which encompasses all information pertaining to the transaction over its entire lifespan.

Definition 6.2.1: Feature vector t

A feature vector $\mathbf{t}$ is a list of features that identify a block-epoch-based transaction $\boldsymbol{T}$ in a specific time slot, and it is defined as: $\mathbf{t} = [k_1, k_2, \ldots, k_n]$ with $|K| = n$. $\square$

Definition 6.2.2: Complete transaction $\boldsymbol{T}$

A Multivariate Time Series (MTS) $\boldsymbol{T} = [\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_\gamma]$ is an ordered set of feature vectors $\mathbf{t}$, and consists on γ different univariate time-series with $\mathbf{t} \in \mathbb{R}^n, \forall \mathbf{t} \in \boldsymbol{T}$, and $|K| = n$. □

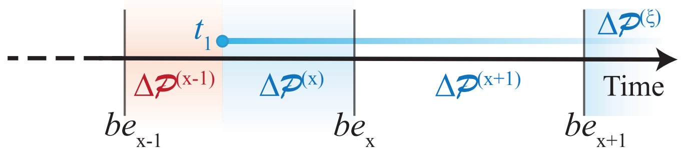
The information contained within $\boldsymbol{T}$ can be utilized to extract novel features (e.g., `delta`) that convey contextual information over distinct temporal intervals, in our case, block epochs. Such information is leveraged by the pre-established set S to create a feature, denoted as *offset*, which is explained further in the subsequent paragraph.

6.2.2 Features Extraction

The procedure of feature extraction entails the creation of engineered features that are used for a supervised classification in a DNN model. The feature we are using are rooted in the notions of fairness and revenue, and computed using the pending transactions function and the offset function, as will be described below.

Pending Transaction Function

This fairness-based function evaluates the extent to which a transaction t belongs to the set of the relapsed pending transactions $\mathcal{P}$, and consequently ascertain the probability of it being included in the set of the temporarily approved transactions $\mathcal{A}$ in the near future. The function $f_{\mathcal{P}}$ generates the feature $\Delta\mathcal{P}$ (or `delta`, presented in Table 5.1), described in Equation 6.4, and depicted in Figure 6.1.



t_1
$\Delta\mathcal{P}^{(\xi)}$
$\Delta\mathcal{P}^{(x-1)}$
$\Delta\mathcal{P}^{(x)}$
$\Delta\mathcal{P}^{(x+1)}$
Time
be_{x-1}
be_{x}
be_{x+1}

Figure 6.1: A transaction t_1 submitted at time ep_{t_1} and yet-to-be included, such that $\mathcal{L}(t_1) = \mathcal{L}^{\xi}_{t_1}$, takes different $\Delta\mathcal{P}$ values over time. For instance, at height $x+1$ we have that $\Delta\mathcal{P}^{(x+1)} = be_{x+1} - ep_{t_1}$. Blue $\Delta\mathcal{P}$ represents a positive number, while the red $\Delta\mathcal{P}^{(x-1)}$ indicates a negative value.

Definition 6.2.3: Pending transaction function

$f_{\mathcal{P}}$ is the duration a transaction $t^{(x)}$ has been waiting inclusion into $\mathcal{A}$, starting from its inception until the closest block epoch be_x.

$$\begin{aligned} \Delta\mathcal{P}_t^{(x)} &= \beta_t{}^{(x)} - ep_t \\ (\beta, ep) \mapsto \Delta\mathcal{P} &= f_{\mathcal{P}}(\beta, ep) \end{aligned} \tag{6.4}$$

□

Note that if $t^{(x)}$ is not part of $\mathcal{P}^{(x)}$, then $\Delta\mathcal{P}^{(x)}$ assumes a negative value, as is evident from the case of $\Delta\mathcal{P}^{(x-1)}$ in Figure 6.1. The feature $\Delta\mathcal{P}$ enables us to consider the maximum amount of time a transaction can wait for approval if it has paid a fair fee. Moreover, since γ represents the number of instances in $\boldsymbol{T}$, we can compute the time elapsed in relation to the number of blocks seen by $\boldsymbol{T}$ before being approved, and accordingly determine time with respect to both, an absolute view in seconds and a relative view in γ-occurrences. Consequently, a block-epoch-based transaction will have γ different instances of the feature $\Delta\mathcal{P}$, as described in Section 5.1.2.

Offset Function

The offset function f_δ is a revenue-oriented metric that orders pending transactions based on their feerate while considering the restrictions on block space. The output of this function is the engineered feature *offset* which is denoted by δ.

Definition 6.2.4: Offset function

For each transaction **t** in a block-epoch, the offset value at height x indicates the quantity of bytes that have already been taken up in the block space by unconfirmed transactions with a higher feerate. This value is represented as $\delta_t^{(x)}$.

$$\begin{aligned} \delta_i^{(x)} &= \sum_i^n q_i, \\ (S^{(x)}, q) \mapsto \delta &= f_\delta(S^{(x)}, q). \end{aligned} \tag{6.5}$$

□

The offset assigns a position to each transaction in the forthcoming block. A higher offset indicates a lower likelihood that t is included in the set $\mathcal{A}^{(x)}$. Assuming that the set S at height x includes n transactions ordered by feerate, the offset value for any index $0 \leq i \leq n$ can be defined as in Equation 6.5.

Like the scenario with $\Delta\mathcal{P}$, every block-epoch-based transaction possesses γ distinct offset values, as the offset for each transaction is recalculated with every block creation. Algorithm 8 presents the procedure for calculating the offset for transactions in $S^{(x)}$. The algorithm is composed of two procedures: DEFINES and OFFSET. These procedures define the set S and the offset δ for each transaction in S. Specifically, the DEFINES procedure takes as input $\mathcal{R}$ (with the $\Delta\mathcal{P}$ feature), a block height x, and it generates a set $S^{(x)}$. S' contains all transactions that occurred between two block epochs. The set S is then formed by adding any pending transactions from $\mathcal{P}$. The transactions in $S^{(x)}$ are ordered by feerate, after which the offset is calculated for each transaction.

The OFFSET procedure takes as input a transaction t and the set S. It initializes δ and, for each transaction at a position i after $t \in S$, it adds its size, q_i, to δ.

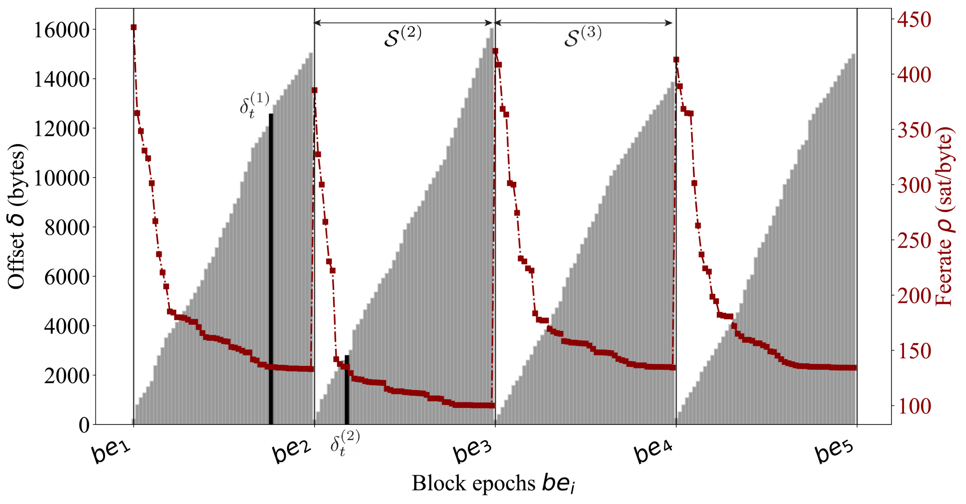


Figure 6.2: Data sampled for five consecutive blocks, and within each block-epoch time slot, transactions are arranged by feerate, followed by the calculation of their corresponding offset value. As observed in this example, the offset value for transaction $t^{(x)}$ fluctuates according to the block-epoch. Specifically, the offset value for $t^{(1)}$ at height $x = 1$ is larger than its offset value at height $x = 2$ for $t^{(2)}$. Thus, we can infer that $\delta_t^{(1)} \gg \delta_t^{(2)}$.

Algorithm 8 Defining S and δ.

```
1:  procedure DEFINES($\mathcal{R}_{\Delta\mathcal{P}}, x$)                ▷ $\mathcal{R}$ including $\Delta\mathcal{P}$ feature
2:      $S' = \{t | t \in \mathcal{R}_{\Delta\mathcal{P}}, be_x \leq ep_t < be_{x+1}\}$    ▷ txs between $be_x$, and $be_{x+1}$
3:      $S \leftarrow \varnothing$
4:      for all $t \in S' \cup \mathcal{P}^{(x)}$ do                        ▷ waiting txs in $\mathcal{P}^{(x)}$
5:          $\rho_t \leftarrow f_\rho(\phi_t, q_t)$                        ▷ calculate feerate of $t$
6:          $S \cup \{t\}$                                    ▷ add $t$ to the set $S^{(x)}$
7:      sort($S, \rho$)                       ▷ ascending order of txs in $S$ by feerate
8:      for all $t \in S$ do
9:          $\delta_t \leftarrow$ OFFSET($t, S$)        ▷ set the offset for every transaction in $S$
10:     return $S$
11:
12: procedure OFFSET($t, S$)                  ▷ index $i$ is the place of $t$ in the set $S$
13:     $\delta_t \leftarrow 0$
14:     $i \leftarrow 1$
15:     for all $t \in S$ after position $i$ do             ▷ for txs in $S$ that come after $t$
16:         $\delta_t \leftarrow \delta_t + q_i$
17:         $i++$
18:     return $\delta_t$
```

Compared to other features, the offset requires a significantly higher computational overhead. The procedure DEFINES in Algorithm 8 has a time complexity of $O(n^2)$, where n is the number of transactions in $S^{(x)}$. This is due to the fact that the execution time of OFFSET is upper-bounded by $O(n)$, and since the latter procedure is executed n times, the total number of operations can be approximated by $\sum_{i=0,j=0}^{n} ij \sim O(n^2)$. Figure 6.2 depicts the trend of the calculated offset over time for five consecutive blocks. Within each block-epoch time frame (be_1–be_5), transactions waiting for confirmation are organized in the mempool according to their feerate and assessed for their relative offset value. For instance, transaction $t^{(x)}$ appears in the pool at height $x = 1$ and $x = 2$ with different offset values. As a result, the same transactions can provide distinct information based on their block-epoch snapshot, providing valuable insights into the network status at each block-epoch time slot.

6.2.3 Complete Set

After data collection and feature engineering, the runtime dataset C is generated to include information about every transaction in a block-epoch-based manner. The creation of C is described in Algorithm 9, where all the block epochs present in the input dataset $\mathcal{R}$ are iterated over, and for each epoch, the algorithm adds the $\Delta\mathcal{P}$ feature, defines the set $\mathcal{P}$ and the set S using the DEFINEP (Algorithm 10) and the DEFINES (Algorithm 8) functions, respectively. Subsequently, the algorithm appends the rows of S for each block height x to the complete set C.

Algorithm 9 Creation of C.

```
1: procedure COMPLETE_SET($\mathcal{R}$)
2:     $be_{\text{all}} \leftarrow$ list($\mathcal{R}$.bepoch)    ▷ list of all block epochs with no duplicates
3:     for all $be \in be_{\text{all}}$ do
4:         $x \leftarrow$ height($be$)    ▷ gets height of a block from the epoch
5:         $\mathcal{R}_{\Delta\mathcal{P}} \leftarrow$ DELTA($\mathcal{R}, x$)    ▷ add $\Delta\mathcal{P}$ feature
6:         $\mathcal{P}^{(x)} \leftarrow$ DEFINEP($\mathcal{R}_{\Delta\mathcal{P}}, x, be$)    ▷ make $\mathcal{P}$ set, used in DEFINES
7:         $S^{(x)} \leftarrow$ DEFINES($\mathcal{R}_{\Delta\mathcal{P}}, x$)    ▷ make $S$ set
8:         $C \leftarrow$ append rows in $S^{(x)}$
9:     return $C$
```

The C dataset is utilized to generate the training and test datasets $\mathcal{X}$, where the cardinality of C is greater than $\mathcal{X}$, which in turn is greater than the input dataset $\mathcal{R}$. Additionally, the features that are used in $\mathcal{X}$ is a subset of the features used in C, which themselves are a subset of the features used in $\mathcal{R}$. To derive new features, as explained in Sections 6.1.2-6.2.2, several functions are applied to the datasets to generate new features with unique names. The pre-processing layer and ingestion engine exchange and update information, enabling the inclusion of these new features in the dataset $\mathcal{R}$. Subsequently, these new features can be incorporated into the training and test datasets, thereby enhancing model performance and improving prediction accuracy.

Algorithm 10 Creation of $\mathcal{P}$.

```
1: procedure DEFINEP($\mathcal{R}, x, be$)
2:     $be_{x-1}$ = epoch($x - 1$)    ▷ get epoch from height
3:     $\mathcal{P}^{(x)} \leftarrow \{t | t \in \mathcal{R}, \phi_t \geq \mathbb{E}[f_t]\}$    ▷ remove cheap txs
4:     $\mathcal{P}^{(x)} \leftarrow \{t | t \in \mathcal{P}^{(x)}, ep_t \leq be_{x-1}\}$    ▷ remove new txs
5:     return $\mathcal{P}^{(x)}$
```

6.3 Machine Learning Model

We have chosen a supervised learning method for our model because the vast amount of available data on Bitcoin blockchain allows us to train and test our algorithm with precision. When there is an abundance of labeled data available, ResNet is considered the most appropriate models for generating accurate predictions [159, 160]. In line with this theory, we have employed the ResNet architecture, as presented in Section 2.3.3, to develop our prediction model for forecasting future trends in the Bitcoin blockchain market. The main purpose of the prediction model is to define whether or not a transaction t at height x is likely to be included in $\mathcal{A}^{(x)}$.

To optimize our ResNet model, we use the ReLU activation function. ReLU offers several advantages over other activation functions, such as being computationally efficient, easier to optimize, and able to prevent vanishing gradients, which can be a common issue in deep neural networks [121]. With ReLU, our model achieves better accuracy and faster training times, leading to more accurate predictions. Overall, our DNN prediction model using ReLU is a powerful tool for predicting future trends in the Bitcoin blockchain market, providing insights for both investors and researchers.

6.3.1 Training and Test Sets

Our ResNet model is trained and tested using two datasets: $\boldsymbol{X}$ for training and $\boldsymbol{Xte}$ for testing. These sets are constructed from the larger dataset $\mathcal{X}$ based on a percentage criterion. For example, if 5% of $\mathcal{X}$ is allocated for testing, the remaining 95% is used for training in $\boldsymbol{X}$. In accordance with the definitions of feature vector and complete transaction, represented by **t** and $\boldsymbol{T}$ respectively (as per definitions 6.2.1-6.2.2), the training and test datasets are characterized as an M-dimensional multivariate time series collection. Each pair $(\boldsymbol{T}_i, \boldsymbol{Y}_i)$ within this collection constitutes an instance of a block-epoch-based transaction i with relative labeled information for the validation. In this notation, $\boldsymbol{Y}_i$ identifies information about the inclusion of a specific transaction, while $\boldsymbol{T}_i$ represents some of its corresponding features. More specifically, $\mathbf{Y}_i$ represents the corresponding one-hot label vector[4] of transaction $\boldsymbol{T}_i$. In particular, $\mathcal{X}$ is expressed as $(\boldsymbol{T}_1, \boldsymbol{Y}_1), (\boldsymbol{T}_2, \boldsymbol{Y}_2), \ldots, (\boldsymbol{T}_M, \boldsymbol{Y}_M)$. The one-hot label vector $\mathbf{Y}_i$ has a length of γ_i, and each of its elements, $j \in [1, \gamma_i]$, is equal to 1 if $j = \gamma_i$, which is the corresponding time slot of $\boldsymbol{T}$'s inclusion, and 0 otherwise. Table 6.3

4. A one-hot label vector is a representation of a categorical variable in ML, where each category is assigned a unique binary value.

$\mathcal{X}$		$\boldsymbol{T}_i$				$\boldsymbol{Y}_i$
i		**hash**	**bepoch**	**w time**	**offset**	**incl.**
$i=1$	**t** $_1$	49baa25...	...22049	-958	1,630,459	0
	t $_2$	49baa25...	...23235	38	12,312,700	0
	...					
	t $_{\gamma_1}$	49baa25...	...27793	5399	897,480	1
. . .						
$i=M$	**t** $_1$	860b21a...	...21154	-732	1,882,965	0
	t $_2$	860b21a...	...21759	128	2,592,452	0
	...					
	t $_{\gamma_M}$	860b21a...	...23528	4892	978,382	1

Table 6.3: Representation of an M-dimension multivariate time series collection.

illustrates an instance of a multivariate time-series pair, $(\boldsymbol{T}_i, \boldsymbol{Y}_i)$. Specifically, the hash feature within the transaction remains constant throughout its lifespan, while the remaining features, identified as be (or block epoch), $\Delta\mathcal{P}$, and δ_t, are dependent on the contextual block-epoch.

Algorithm 11 Creation of $\mathcal{X}$.

```
 1: procedure TRAINING_SET($C$, k=[$\delta_t$, $\Delta\mathcal{P}$, $\phi$, . . . ])   ▷ k is vector of keys
 2:     for all key in k do                                  ▷ for each feature name in k
 3:         $\mathcal{X}$ ← add_feature($C$, key)         ▷ add "key" column from $C$ to $\mathcal{X}$
 4:     $\mathcal{X}$ ← INCLUSION($\mathcal{X}$)                                      ▷ make Y
 5:     $\mathcal{X}$ ← $\mathcal{X}$.values        ▷ NumPy representation of Pandas dataframe
 6:     $\mathcal{X}$ ← NORMALIZATION($\mathcal{X}$)                    ▷ z-score normalization
 7:     return $\mathcal{X}$
 8:
 9: procedure INCLUSION(d)                                         ▷ inputs a dataset
10:     d[incl] ← np.where(d[$be$] == d[$\beta_t$], 1, 0)    ▷ 1 if $be$ =$\beta_t$, 0 otherwise
11:     return d              ▷ same dataset with new "inclusion" feature, or Y
```

Algorithm 11 outlines the procedure to generate the set $\mathcal{X}$. This involves utilizing the normalization techniques illustrated in Algorithm 12 in combination with the NumPy libraries to construct the set $\mathcal{X} = (\boldsymbol{T}, \boldsymbol{Y})$. Algorithm 11 computes the training set $\mathcal{X}$ for a given set of feature names k and a the complete set C. The procedure begins by iterating through each feature in k and adding the corresponding column from C to $\mathcal{X}$. Then, the INCLUSION function is applied to $\mathcal{X}$ to create the binary **Y** feature, where 1 indicates the inclusion of a transaction and 0 otherwise. If the block-epoch value (be) for a block-epoch-based transaction matches the block epoch when the transaction is included

(β_t), then the corresponding element in the new column is set to 1. Next, the NumPy representation of $\mathcal{X}$ is obtained and passed through the normalization process. Finally, the normalized NumPy array is returned as the final training set $\mathcal{X}$.

Normalization

The selection of an appropriate normalization method is influenced by the properties of the data and the objectives of the specific task. Each normalization technique offers distinct advantages and disadvantages, and selecting an appropriate method can have a profound effect on the quality and effectiveness of the model or analysis. For example, the decimal scaling normalization method is advantageous for preserving the order of magnitude of the data, while min-max normalization is useful for comparing variables that have varying units and scales.

Algorithm 12 Normalization of $\mathcal{X}$.

```
1: procedure NORMALIZATION($\mathcal{X}$)
2:     $\mu \leftarrow$ np.mean($\mathcal{X}$)          ▷ expected value (using NumPy as np)
3:     $\sigma \leftarrow$ np.std($\mathcal{X}$)           ▷ standard deviation
4:     $\mathcal{X}_{\text{norm}} \leftarrow (\mathcal{X}-\mu)/\sigma$      ▷ set the normalize training set
5:     return $\mathcal{X}_{\text{norm}}$
```

The normalization technique employed in Algorithm 12 is a commonly used method known as z-score normalization, or standardization. This method involves transforming a distribution of data points to have a mean of 0 and a standard deviation of 1 by subtracting the mean of each data point and dividing it by the standard deviation [161]. The z-score normalization technique is valuable for comparing variables with different units and scales and for identifying outliers. It is particularly advantageous when working with normally distributed data and it can help to ensure that input features have similar ranges, preventing the dominance of one feature over the others. Furthermore, it can aid in improving the convergence of training algorithms by preventing them from getting trapped in local optima.

6.3.2 Prediction Model

To ensure accuracy and maintain up-to-date knowledge of the network status, our model needs to be updated after some time, typically on a monthly basis (weekly for a better outcome). For this study, we have considered to deploy two different models: a standard DNN and a ResNet model that utilizes skip connections.

Considering the superior performance demonstrated by the ResNet model compared to the DNN model [57], we adopt the former as the model of focus in this dissertation. The ResNet architecture consists of several densely connected layers, with each layer fully connected to the previous one. The activation function used for each hidden layer is ReLU, which facilitates the learning of non-linear relationships between the input and output variables. To initialize the weights in the neural network and improve the speed and stability of training, the He normalization technique[5] is used as the kernel initializer. Additionally, the ResNet model incorporates several skip connections, which enable information to bypass certain layers and be directly fed to subsequent layers, thus improving the flow of information throughout the network. The output layer of the ResNet model has a softmax activation function, which generates a probability distribution over the classes. The present study does not include the outcomes of the DNN model, as it has been surpassed in performance by the ResNet model. The results and discussions regarding the DNN model can be found in our previous work [57].

The trained ResNet model takes a transaction t as an input and generates a vector $\boldsymbol{\theta}_t$, with the confidence level of inclusion or exclusion in the next mined block. Since we have a binary classification problem, the output vector $\boldsymbol{\theta}_t = [P_t(v_0), P_t(v_1)]$ assigns the class v_1 to inclusion and v_0 to exclusion. As such $\boldsymbol{\theta}_t$ represents the probability $P(v_i)$ of transaction t falling within the class v_i, where $i \in \{0, 1\}$. We adopt a supervised classification approach whereby the known outcomes of transactions in the training set, denoted as $\boldsymbol{X}$, are used to assess the accuracy of the model during the training phase, which is accomplished through the use of the labels $\mathbf{Y}$. A subset $\boldsymbol{Xte} \subseteq \mathcal{X}$ of the data is reserved for testing.

During validation we employ a dynamic approach that alters the number of hidden layers in the neural network, with each node in the network being characterized by the ReLU, except for the output layer where the Normalized Exponential Function (softmax) is used. The weights are initialized using the He normalization technique, which accounts for ReLU, thereby facilitating convergence in deep models [162]. During training and testing phases, as

5. is a technique used to initialize the weights of artificial neural networks. By using He normalization, the weights are initialized in a way that prevents them from becoming too large or too small. This technique helps the network converge faster and improves its ability to learn complex patterns and representations from the data

outlined in the previous section, $\mathcal{X}$ is normalized with the z-score method, which was useful as the features exhibit varying orders of magnitude.

The hyperparameters of the ResNet that cannot be estimated from data are determined manually through a trial-and-error. This include the number of hidden layers, the number of skip connections, the batch size, and the number of epochs. The batch size governs the granularity or precision of gradient descent, thereby optimizing the internal parameters of the model for every batch size of tuples. On the other hand, the number of epochs determines the number of times that the learning algorithm will iterate through the entire training dataset, ideally getting closer to the optimal solution with each iteration. The configuration of the model's hyperparameters is critical to achieve optimal model performance and accuracy.

Summary

In this chapter, we provided an in-depth overview of the architecture of our ML model. We outlined the entire data processing and transformation pipeline, beginning with the ingestion engine. The ingestion engine was responsible for processing and transforming the data, selecting relevant features, and storing them locally.

We then delved into the pre-processing phase, which consisted of three key steps. First, we divided transactions into block-epoch-based tuples, enabling efficient analysis. Next, we performed feature extraction to derive meaningful characteristics from the data. Lastly, we created a complete dataset by integrating all relevant features.

Additionally, we introduced the ResNet model that we developed. We discussed the composition of the training set, the normalization of data, and the output of the model.

In the following section, we present experiments and evaluations of the ML model described thus far. A detailed account of the hyperparameters can be found in Appendix A.

7 Evaluation

In this chapter, we present the evaluation of our ML model. In Section 7.1, we give a detailed description of the datasets used for both training and testing our model, including how we selected relevant features. In Section 7.2, we outline the evaluation metrics we use and elaborate on our definition of model accuracy, as our accuracy assessment transcends the mere measurement of the proportion of correctly classified instances out of the total instances. We also account for other metrics such as the completeness of the data used for training and testing the model, as well as the degree to which the dataset is updated. By incorporating these additional metrics in our assessments, we obtain a more comprehensive evaluation of our model's performance, while also accounting for potential sources of bias. In Section 7.3, we report the results of our analyses in terms of overall accuracy, feature importance, and model cyclicity. We investigate how the model's performance is affected by the selected features and how often the model's predictions are influenced by its current network status. Finally, in Section 7.4 we quantify how much our solution can benefit end-users.

7.1 Experimental Setup

In this section, we describe a comparative analysis of various datasets in relation to the disk space consumption associated with the Bitcoin blockchain. We introduce the distinct metrics used to evaluate the accuracy of the model and ultimately examine and enumerate the chosen features.

7.1.1 Datasets Analysis

We first investigate the data derived from the public Bitcoin blockchain. Our analysis includes over 30 million transactions across 15,000 blocks from January 2021 to May 2021. Table 7.1 demonstrates the requisite disk space for these three datasets when accommodating 100,000 to 5,000,000 transactions. As shown, the complete dataset $\mathcal{C}$ exhibits a superlinear growth pattern. This observation aligns well with our expectations, considering that $\mathcal{C}$ not only includes all transactions in $\mathcal{R}$, but also captures the block-epoch based dependent features delineated in Chapter 6, and possesses knowledge of $\mathcal{P}$ and S sets. The comparison in Table 7.1 highlights the disk space efficiency of the developed datasets in relation to the actual Bitcoin blockchain.

Disk storage (bytes)				
N of txs	**100k**	**500k**	**1M**	**5M**
Dataset				
$\mathcal{R}$	2.9×10^7	1.45×10^8	2.9×10^8	1.2×10^9
$\mathcal{C}$	4×10^7	7.3×10^8	1.6×10^9	1.37×10^{10}
$\mathcal{X}$	7×10^6	1.3×10^8	2.88×10^8	2.4×10^9
₿	6×10^7	3×10^8	6×10^8	3×10^9

Table 7.1: This table displays the amount of disk space utilized by various sets of instances containing information on 100,000, 500,000, 1,000,000, and 5,000,000 transactions, respectively. The final row presents the corresponding space occupied by an equivalent number of transactions in the real Bitcoin blockchain.

7.1.2 Metrics

We obtain and store various instances of $\mathcal{R}$, one for each month of evaluation (see Table 7.2). For each evaluation period, 3,010 blocks[1] are fetched. A predictive model is constructed from each dataset $\mathcal{R}^i$, using the inclusion

1. Which is the equivalent of the number of blocks mined on 20 days on average

pattern delineated in Section 5. Also, a C^i dataset is produced, from which novel information and features are derived. Afterwards, data is selected from the C^i dataset to form the training and testing datasets ($\mathcal{X}^i$). For each period i, a model is trained using the initial 50% of transactions in $\mathcal{R}^i$. For testing such model, the remaining 50% of transactions is used. Tests are performed and evaluated with parameters denoting the *completeness* and *freshness* of the testing set. These parameters are identified as α and ψ, respectively, as defined below.

Completeness

Definition 7.1.1: Completeness

The completeness parameter α of a testing set identifies the proportion of transactions used for testing relative to the total number of points used for training.

$$\alpha = \frac{|\boldsymbol{Xte}|}{|\boldsymbol{X}|} \tag{7.1}$$

□

From Equation 7.1, we can see that when $\alpha = 0.5$, the testing volume of transactions is equal to half of the training volume. The completeness value is crucial for precise prediction in the absence of live (or real-time) information.[2]

Raw dataset for each period				
i	Date	$\lvert\mathcal{R}^i\rvert$	$\mathcal{R}^i$ size	₿ price
1	Jan	6.5M	1.09 GB	29k to 35k $
2	Feb	6.7M	1.12 GB	33k to 56k $
3	Mar	6.2M	1.05 GB	49k to 58k $
4	Apr	6.2M	1.04 GB	58k to 56k $
5	May	5.2M	877 MB	58k to 36k $

Table 7.2: Specification of the raw time-series datasets used for evaluation. For each month, 3,010 blocks are analyzed. $|\mathcal{R}^i|$ is the number of transactions analyzed in each set, while $\mathcal{R}^i$ size identifies the set's storage requirements. We also include the Bitcoin price at time of evaluation to discuss any correlation with model prediction and coin price at that time.

2. Since the offset value is determined by the number of transactions evaluated, having a complete set of transactions is crucial to compute the transaction space in the next mined

Due to the supervised nature of our testing methodology and the simultaneous analysis of millions of transactions, we were precluded from relying on unlabeled real-time data, and therefore we needed to distinguish a complete testing set from a non complete one. As $\alpha \longrightarrow 1$, the set becomes *complete*, and the constructed offset value for testing closely approximates the one employed for training, thus reducing false-positive points when $\alpha \ll 1$ or false negatives when $\alpha \gg 1$. A complete set offers an accurate representation of the mempool size over time.

Freshness

Assuming `mo` represents the difference between the index i in Table 7.2 of training and testing datasets, as:

$$\texttt{mo} = i^{X_{te}} - i^{X}$$

then the freshness is normalized through a sigmoid function to yield a bounded range of $[0, 1]$.

Definition 7.1.2: Freshness

The freshness parameter of a testing set, ψ, identifies the temporal distance between the test and training sets, normalized through a sigmoid function.

$$\psi = \text{sigmoid}(\texttt{mo}) \tag{7.2}$$

□

When training and test data are derived from the same month, where ψ is equal to $\text{sigmoid}(0) = 0.5$, designating the model as new, or *fresh*. If testing is conducted with an older model, $\texttt{mo} > 0$ and $\psi \longrightarrow 1$. Conversely, if the model is trained a-posteriori and applied to prior data, $\texttt{mo} < 0$ and $\psi \longrightarrow 0$. The ψ parameter reflects the degree of freshness of the tested transactions relative to the trained ones, thereby measuring the model's currency with respect to the existing network status. Additionally, information on ψ proves valuable during subsequent analysis of model cyclicity.

block. Evaluating and testing live data results in having a complete set.

7.1.3 Features

The selection of features to be used for training our model, is a combination of parameters that are extracted from the blockchain, and data that are computed according to the guidelines presented in Section 6.1.1. The training set $\mathcal{X}$ is expected to contain relevant information about fairness and revenue, as detailed in Sections 5.3 and 5.2. Our model is trained using a set of features denoted by:

$$K_{\mathcal{X}} = [\phi, q, \rho, \Delta\mathcal{P}, \delta, \Delta\mathcal{P}_N, \delta_N]$$

where $\Delta\mathcal{P}_N$ and δ_N represent normalized values of $\Delta\mathcal{P}$ and δ, respectively. The feature $\Delta\mathcal{P}_N$ is determined as the number of blocks a particular transaction $\mathbf{t}_\gamma$ has encountered since its inception (as indicated by Equation 7.3).

$$\begin{aligned} \Delta\mathcal{P}_N^{(\gamma)} &= \sum_{i=0}^{\gamma} \omega^{(i)} \quad \text{where} \\ \omega^{(i)} &= \begin{cases} 0, & \text{if } \mathbf{t}_i \notin \mathcal{P}^{(i)} \\ 1, & \text{if } \mathbf{t}_i \in \mathcal{P}^{(i)} \end{cases} \end{aligned} \tag{7.3}$$

The second feature, δ_N, corresponds to the normalized offset in relation to the maximum block space of approximately 1.1 MB. This normalization results in δ_N being represented as a percentage, which provides information about what portion of the mempool is already occupied by richer transactions (as showed by Equation 7.4).

$$\delta_N^{(x)} = \frac{\delta^{(x)} \times 100}{Q} \tag{7.4}$$

7.2 Evaluation Metrics

While classification accuracy is our primary evaluation metric, it may not always be sufficient to fully assess performance. To address this, we also evaluate performance using the confusion matrix and the area under the curve metric. The confusion matrix provides information about the number of true and false positive and negative classifications made by our model, which help identify areas of strength and weakness. The area under the curve metric measures the performance of our model across a range of classification thresholds, providing a more nuanced assessment of its ability to correctly classify instances. The following sections provide a brief description of the evaluation metrics we have used, while Section 7.3 presents our evaluation results.

7.2.1 Classification Accuracy

To evaluate our model, we initially employ the *classification accuracy* metric, which measures the ratio of the number of correct predictions to the total number of input samples using the formula:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Classification accuracy is a straightforward measure that we use as a general tool to compare accuracy between different models. However, it may not always provide an accurate evaluation of a model's performance, particularly when the class distribution is heterogeneous. In our study, we observed an unbalanced class distribution and chose not to reduce the number of sampled data. As a result, we incorporated additional metrics to evaluate our model.

7.2.2 Confusion Matrix

The confusion matrix is useful for analyzing the accuracy of our model. The confusion matrix **CM** is defined in Equation 7.5, and formalized in Table 7.3. The matrix **CM** is obtained by dividing the raw count of correctly and incorrectly classified instances by the total number of instances, and then scaling the resulting values by a factor $\mathbf{Fr}^{-1}$. The matrix $\mathbf{Fr}_{2,2}$ represents the total number of actual classifications for each class.

$$\begin{aligned} \mathbf{CM} &= \mathbf{Fr}^{-1} \cdot \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \\ \text{where} \quad \mathbf{Fr}_{2,2} &= [b_{ii}], \quad b_{ii} = \sum_j a_{ij} \end{aligned} \tag{7.5}$$

The values a_{ij} in the confusion matrix **CM** represent the number of elements that truly belongs to class i but were classified as belonging to class j. We use two metrics derived from **CM**, namely *recall* and *precision*. The recall R_i for class v_i represents the proportion of instances in v_i that were correctly classified, while the precision P_i for class v_i represents the proportion of instances that were classified as belonging to v_i that actually belong to v_i. These metrics are calculated using Equation 7.6.

$$R_i = \frac{a_{ii}}{\sum_{j=0}^{1} a_{ji}} \qquad P_i = \frac{a_{ii}}{\sum_{j=0}^{1} a_{ij}} \tag{7.6}$$

7.2.3 Area Under Curve

The Area Under Curve (AUC) measures the probability that a model will rank a randomly selected sample in class v_1 higher than another randomly selected sample in v_0. AUC is commonly used in binary classification problems. The range of values for the AUC is zero to one, with higher values indicating better classifier performance. An AUC value of one indicates that the classifier is able to perfectly distinguish between the two classes, while a value of 0.5 indicates that the model cannot differentiate between points in the v_0 and v_1 classes. A value of zero indicates that the classifier would predict all points in v_0 as v_1 and vice versa.

Evaluation of binary classification models benefits from careful consideration of several performance metrics, including the Area Under Curve of Receiver Operator Characteristic (AUC-ROC). To calculate the AUC-ROC, we follow the scheme presented in Table 7.3 and use the formulas:

$$\begin{aligned} TPR &= \frac{TP}{TP+FN} \qquad & FNR &= \frac{FN}{TP+FN} \\ TNR &= \frac{TN}{TN+FP} \qquad & FPR &= \frac{FP}{TN+FP} \end{aligned} \tag{7.7}$$

where *TP* represents true positives, *TN* represents true negatives, *FP* represents false positives, and *FN* represents false negatives. The True Positive Rate (TPR) identifies R_1, or *sensitivity*, while the True Negative Rate (TNR) represents the *specificity*, or the proportion of negative class v_0 samples that are correctly classified. The False Positive Rate (FPR) is equal to $1-$specificity and represents the proportion of v_0 samples that are incorrectly classified. Finally, the False Negative Rate (FNR) indicates the proportion of positive class v_1 samples that are incorrectly classified.

The Receiver Operator Characteristic (ROC) curve is a graphical representation of the TPR against the FPR and is useful for evaluating the performance of binary classifiers. The AUC is equal to the area under the ROC curve.

7.3 Performance

In Section 7.3.1, we provide an overview of the classification accuracy of our classifier based on the evaluation metrics defined in Section 7.2. Specifically, we report the classification accuracy for each month of the evaluation period (January 2021 to May 2021) under optimal conditions of $\alpha = 1$ and $\psi = 0.5$. Additionally, a confusion matrix representing the entire evaluation period

		Actual values	
		0	1
Predicted values	0	True Negative Rate	False Negative Rate
	1	False Positive Rate	True Positive Rate

Table 7.3: The confusion matrix model is utilized as the foundation for computing the AUC-ROC.

is presented. Section 7.3.2 discusses the significance of selecting appropriate features, examines how the inclusion or exclusion of certain information affects accuracy, and highlights the potential impact of incorrect assumptions. In these experiments, we set hyperparameters to an optimal case scenario. Finally, in Section 7.3.3, we emphasize the importance of updating the model regularly. We evaluate different model parameters and report the AUC-ROC score for each of them, which are subsequently compared.

7.3.1 Overall Accuracy

Next, we report the overall classification accuracy of each month from January to May 2021 using the optimal parameters of $\alpha = 1$ and $\psi = 0.5$, summarized in Table 7.4. The average accuracy for the entire period of analysis is also included in the table. Table 7.5 displays the computed confusion matrix for all the points analyzed over the course of the evaluation, which includes more than 30 million transactions.

As indicated in Table 7.4, the overall accuracy of the model is above 90% for three out of five months during the entire training/test period, while it struggles between April and May due to the coin price plunge (discussed in Section 8). Nonetheless, even in our worst-case scenarios, the model remains relatively robust. The confusion matrix in Table 7.5 demonstrates that the model correctly classified 91% of the transactions in the υ_0 class and 88% in the υ_1 class.

Classification accuracy 2021 (%)							
α	ψ	Jan	Feb	Mar	Apr	May	**Overall**
1	0.5	90.07	90.9	91.08	85.52	88.29	89.17

Table 7.4: The presented results include the classification accuracy for each month of the evaluation period, which spans from January to May 2021. The overall accuracy is calculated as the average classification accuracy across the entire evaluation period. The optimal parameters of α and ψ are employed in order to ensure that the model is complete and up-to-date with respect to the tested data.

Overall **CM** score		
	v_0	v_1
v_0	0.91	0.09
v_1	0.12	0.88

Table 7.5: The presented confusion matrix represents the overall score for a test conducted between January 2021 and May 2021, with parameters $\alpha = 1$ and $\psi = 0.5$. The matrix depicts how over 30 million transactions were classified using five distinct models, each of which corresponded to a different month, thereby providing a complete and up-to-date view. The false negative rate is determined to be 9% for the negative class, while the false positive rate is observed to be 12% for the positive class.

Classification accuracy 2021 (%)						
Feature set	Jan	Feb	Mar	Apr	May	**Overall**
1 : **Primitive**	75.13	78.54	77.77	62.52	69.97	72.78
2 : **Fairness**	84.57	86.24	84.63	83.64	82.11	84.23
3 : **Revenue**	88.24	87.73	89.4	80	86.21	86.31
4 : **Complete**	89.51	90.36	90.04	85.35	88.23	88.69

Table 7.6: The accuracy of classification across various sets was assessed. Each set denotes a distinct feature property. The primitive set includes only transaction fee and size, while a revenue-based solution include features that are associated with miners' revenue, such as fee, offset, and feerate.

7.3.2 Features Performance

To validate our assumptions on the importance of certain features in predicting transaction inclusion, we conducted tests and verified the relevance of four specific feature subsets. The classification accuracy and confusion matrix for models trained using each of the four feature sets are presented in the following results.

The first set, denoted as *primitive information*, includes only fetched features such as transaction size and fee, commonly used by fee predictors but with poor results in terms of fee overpaying.

The second set, denoted as *fairness assumptions*, excludes features based on the revenue principle, assuming that a miner needs only to be fair to include transactions in the next block.

The third set, denoted as *revenue assumptions*, excludes features based on the fairness principle to monitor the impact of revenue on the transaction inclusion pattern.

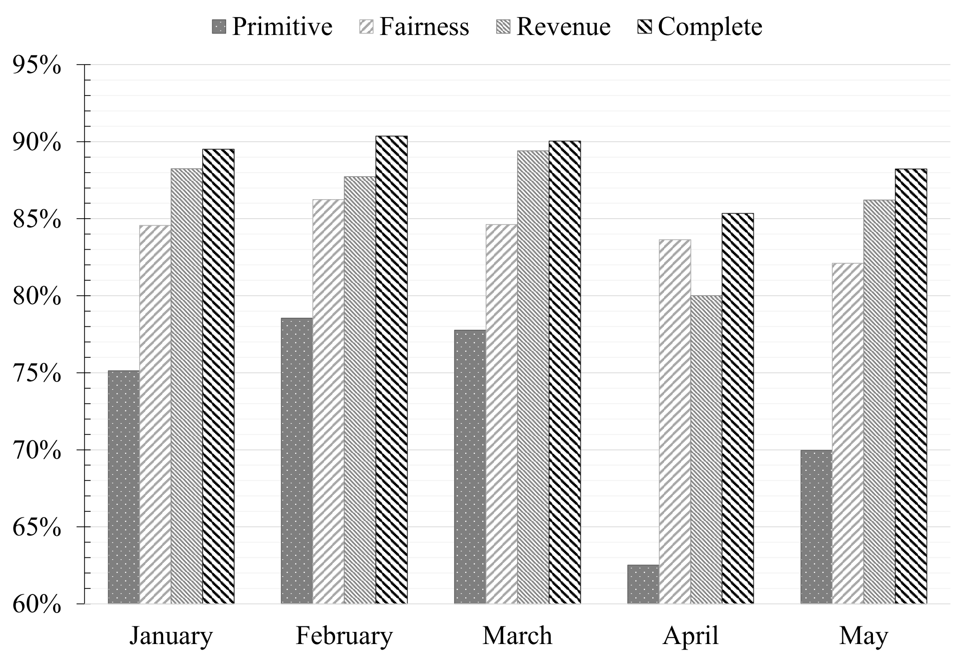


Figure 7.1: Classification accuracy outcomes for each set of features. The findings demonstrate a substantial enhancement in model accuracy when utilizing a complete feature set, as opposed to relying solely on transaction size and transaction fee as features.

Finally, the fourth set, denoted as *complete information*, includes fetched and extracted features, aiming to verify the reliability of our initial assumptions, including both fairness and revenue concepts. The specific feature sets for each evaluation are defined as follows:

1. Primitive: $[\phi, q]$
2. Fairness: $[\phi, q, \Delta\mathcal{P}, \Delta\mathcal{P}_N]$
3. Revenue: $[\phi, q, \rho, \delta, \delta_N]$
4. Complete: $[\rho, \Delta\mathcal{P}, \Delta\mathcal{P}_N, \delta, \delta_N]$

The experiments conducted in this study were designed with a completeness of $\alpha = 1$ and freshness of $\psi = 0.5$. Observed classification accuracy are reported in Table 7.6 and the corresponding plot is presented in Figure 7.1. We observed that the ResNet classifier struggles when Bitcoin prices experience a sudden and significant drop, such as in April 2021. Specifically, the accuracy dropped by 15% when the primitive set of features was used, while the accuracy drop was limited to 5% when complete information was used. This finding highlights the relevance of assumptions made in predicting transaction inclusion. Notably, when fairness and revenue principles were applied separately to the data, the model achieved an average improvement in accuracy ranging from 12% to 14%, and up to 23% when these principles were combined.

The variance of the fairness sub-feature set classification accuracy appeared to be smaller than that of revenue sub-feature set, underscoring the importance of the fairness concept in determining transaction inclusion. In April 2021, an inversion of the Bitcoin price uptrend was observed and miner revenue reached an all-time high. During this period, data indicates that miners appeared to be prioritizing revenue over fairness.

7.3.3 Model Cyclicity

To demonstrate the potential improvement in model classification accuracy when trained on well-formed data, this paragraph illustrates a deviation from the optimal parameter settings. This updated model is cyclically trained over time with complete ($\alpha = 1$) and fresh ($\psi = 0.5$) information. By varying the value of ψ, we can test the model's behavior with both older and newer transactions. Testing is conducted for each month with various combinations of values for α and ψ. Although some of these scenarios may be unrealistic, they

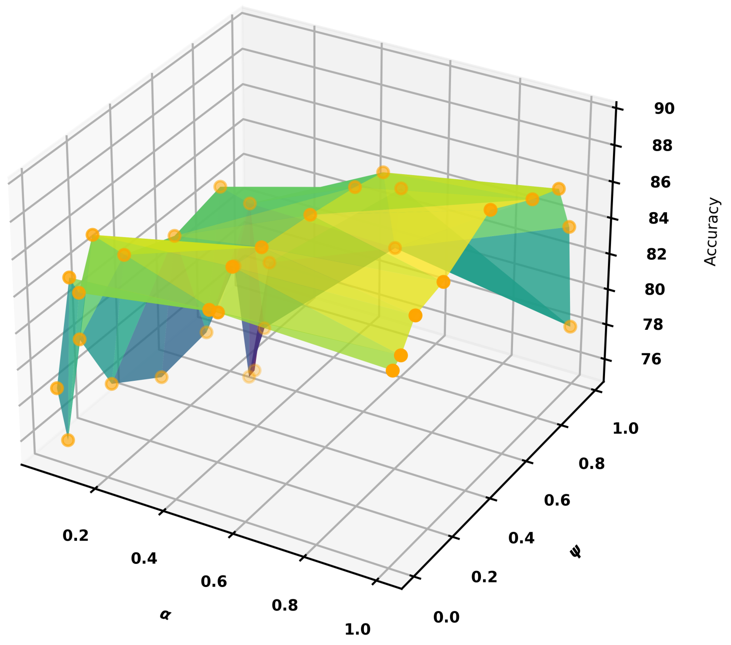


Figure 7.2: Classification accuracy values (z-axis) for multiple models, tested with diverse settings. The y-axis, represented by the ψ parameter, indicates the extent to which a particular model is updated, while the x-axis identifies the model's completeness, as determined by the α parameter. To improve classification accuracy, it is important to highlight the importance of the model's cyclicity.

Evaluations on model cyclicity				
type	α	ψ	Accuracy (%)	AUC-ROC
Worst-	0.04	0.02	78.63	0.82
Average-	0.2	0.12	84.71	0.92
Optimal	1	0.5	91.08	0.97
Average+	0.2	0.88	81.25	0.89
Worst+	0.04	0.98	70.25	0.8

Table 7.7: AUC-ROC results obtained for different parameter values. The optimal evaluation is achieved when the model is complete and the testing is performed within the same month as the training. The average evaluation is obtained using a model that is 20% complete and with a two-month deviation between training and testing. The worst evaluation case corresponds to a model that is only 4% complete and tested four months after training.

are useful for monitoring the model's performance under different conditions. For each month, we perform tests on various combination of values for α, which include 0.05, 0.1, 0.5, and 1, as well as ψ, which includes 0.01, 0.05, 0.12, 0.26, 0.5, 0.73, 0.88, 0.95, and 0.98.

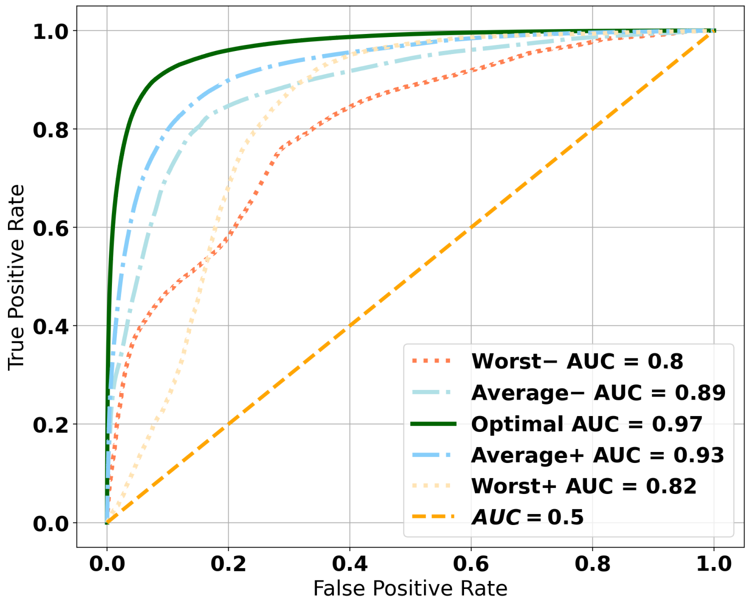


Figure 7.3: Five tests conducted to measure AUC-ROC in optimal, average, and worst-case scenarios. The optimal test was performed using $\alpha = 1$ and $\psi = 0.5$, and is indicated by the green continuous line. The two average tests were conducted with $\alpha = 0.2$ and $\psi = [0.12, 0.88]$, and are represented by the dot-dashed light blue lines. The two worst-case tests were conducted with $\alpha = 0.04$ and $\psi = [0.02, 0.98]$, and are represented by dotted yellow and red lines, respectively. When the AUC is equal to 0.5, the classifier can no longer distinguish between positive and negative class points.

The results are presented in Figure 7.2. Each data point corresponds to the average classification accuracy over five months, for a specific combination of parameter values (α on the x-axis and ψ on the y-axis). The plot demonstrates that the accuracy is highest (indicated by yellow color) when ψ is close to 0.5 and α is close to 1. When the precision of the offset decreases (smaller α values), the impact of model cyclicity (ψ) becomes less significant. These findings highlight the importance of selecting appropriate values for both parameters. Specifically, ψ is relevant when it incorporates accurate information about the mempool size, which is provided by an appropriate choice of α. By contextualizing the model over time through the offset, accurate predictions can be made. Without an accurate calculation of the offset, the model would rely solely on the fairness concept, which appears to have less accuracy variance over time (as demonstrated in Figure 7.1, fairness bar).

Figure 7.3 displays the AUC-ROC curves obtained from five tests that were conducted using different parameter values. These values were carefully selected to represent optimal, average, and worst-case scenarios, as described in Table 7.7. We distinguish between two average cases and two worst cases, depending on whether testing was performed before (–), or after (+) training. In the optimal case, the model achieves a solid classification result, with an area under the curve of 0.97, indicating its ability to distinguish between classes. Even when allowing for a FPR of 10%, the FNR does not exceed 10%. Furthermore, if a FPR of 20% is accepted, the FNR remains below 5%.

7.4 Benefits for End-Users

Analyzing fee reduction for end-users is essential as it provides valuable insights into the extent to which our solution can benefit users. In this context, our study focused on testing the impact of fee reductions across all months of the evaluation period, covering all transactions in our local dataset $\mathcal{R}$. We specifically focused on transactions belonging to v_1, namely, those that were selected by miners in the next round of mining.

Purpose We aim to determine the extent to which users can potentially decrease their transaction fees while still maintaining a favorable probability of inclusion in the next mined block.

Methodology We analyze the effect of fee reductions, ranging from 0% to 80% of the original fee, on our ML model for each block-epoch-based transaction in our complete set C. We also take into account the overall adoption of our model, as this directly affects parameters like feerate and offset values at each block-epoch. To explore this effect, we test various levels of overall adoption, ranging from 1% to 100%. In the lowest adoption scenario, only a random 1% sample of transactions will have their fees reduced, while in the highest adoption scenario, fees of all transactions will be reduced.

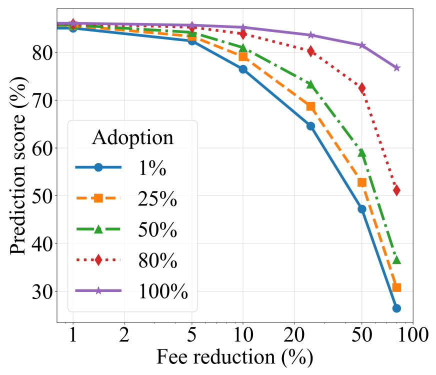
Experimental setup The tests are conducted as simulations of reduced fees rather than real-world transactions executed on the Bitcoin network.

In these simulations, we utilize transactions stored in our local dataset, lower their transaction fees, and input them into our ML model to assess the variance in accuracy score. We have deliberately chosen to focus on labeled transactions in v_1, as they have already been selected for inclusion. Consequently, it is essential to examine the extent to which a fee reduction will affect their likelihood of being included.

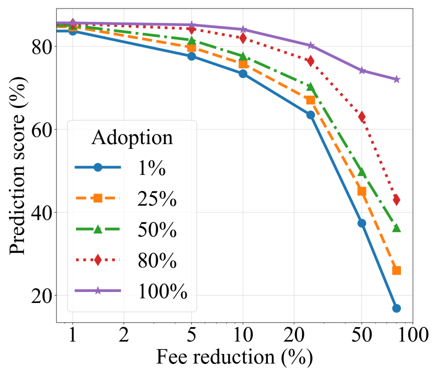
Analysis In order to evaluate the significance of our results, we analyze the variance in the accuracy score of the identical ML model when applied with different fee and adoption parameters. This analysis helps us understand the impact of varying these parameters on the accuracy of our ML model.

Results Figure 7.4 indicates that on average, a hypothetical adoption rate of 80% can result in a reduction of transaction fees up to 30%, without compromising the 80% threshold of inclusion prediction score. Furthermore, tests showed that with complete adoption, transaction fees could be reduced by as much as 50%, without resulting in a predicted likelihood of transaction inclusion falling below 70% in any of the months evaluated. The remarkable finding was that our model demonstrated a predicted likelihood of transaction inclusion of over 80% in most months of the evaluation period, even when fees were reduced by as much as 10%. Hence, our model enables users to reduce their fee expenses with great confidence while still maintaining a high probability that their transactions will be included in the next mined block. Our analysis can help guide end-users in optimizing their transaction fees, resulting in significant cost savings. These findings contribute to our understanding of the practical applicability of our model and its ability to assist users in optimizing transaction costs.

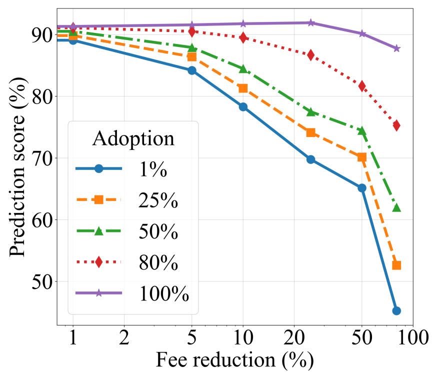
Limitations By conducting these tests, we aim to determine the extent to which users can potentially decrease their transaction fees while still maintaining a favorable probability of inclusion in the next mined block. However, it is important to note that these findings are based on simulations. In practice, miners may decide to alter their policy regarding transaction inclusion if they observe changes in trends across the entire user base. Nonetheless, our model should possess the ability to accommodate such modifications if they are founded on comparable concepts as those expounded in our study. Analyzing fee reduction for end-users can provide valuable insights into the benefits that our solution can offer and help to guide decision-making processes related to transaction fees.



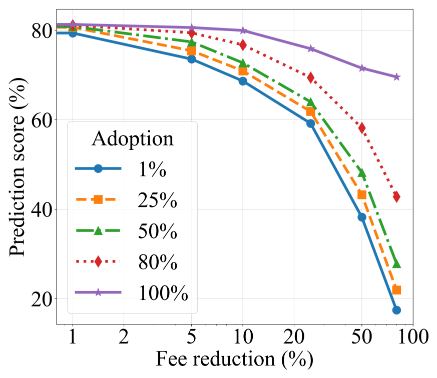
(a) Jan-21



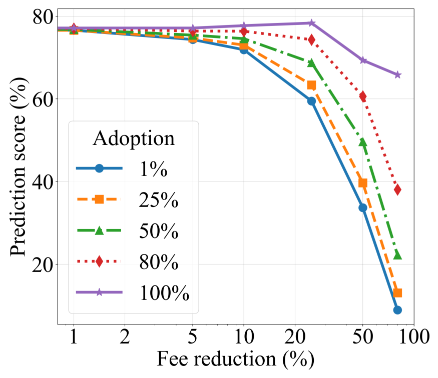
(b) Feb-21



(c) Mar-21



(d) Apr-21



(e) May-21

Figure 7.4: Inclusion prediction score for transactions with reduced fees. In our simulated tests, the adoption rate refers to the proportion of users who have hypothetically adopted our model and subsequently decreased their fees to the percentage levels indicated on the x-axis. It is important to note that these tests do not involve real-life observed data, but rather simulated scenarios.

7.5 Miners' Profit After Fee Reduction

Our fee inclusion model enables users to achieve substantial savings in transaction fees within the Bitcoin network, while maintaining a high likelihood of inclusion. However, a pertinent question emerges regarding potential adverse implications of such fee reduction on miners. To address this question, we assess the extent to which miners are impacted by this fee reduction, using the methodology described in Section 3.1.2.

Purpose We aim to quantify the profit loss for miners resulting from a reduction in transaction fees, as predicted by our model.

Methodology In order to determine miners' profit, we employ the formulas described in Section 3.1.2. Specifically, we calculate the profit under two scenarios: without any fee reduction and with a 10% fee reduction. The latter calculates the profit assuming a 100% adoption rate of our model, which is expected to have a more pronounced negative impact on miners compared to lower adoption rates.

Experimental setup In this evaluation, we perform a profitability analysis specifically for a miner using the Antminer S19 Pro. Our assessment is based on several assumptions: each block comprises 2,500 transactions, and we consider the average transaction fee to be $ 7.5.[3] We analyze the daily profitability of miners in Qatar, where electricity prices are low, and in the US, where electricity prices are average. Our assessment includes various case scenarios, taking into account a range of Bitcoin prices from $ 10,000 to $ 100,000. Additionally, we take into account the block reward of ₿ 3.125, factoring in the imminent halving event that reduces the current block reward of ₿ 6.25. This emphasizes the importance of transaction fees in the overall profitability calculation.

Analysis We examine the daily profit of miners in Qatar and the US, taking into account the specified parameters mentioned earlier. Subsequently, we calculate the variance in profit resulting from a 10% reduction in fees.

Results The analysis conducted, as shown in Table 7.8, reveals that the reduction in fees generally leads to minimal losses for miners. However, in specific scenarios where the profitability approaches zero (such as when the Bitcoin price is at $10,000 and the miner operates in Qatar),

3. based on the yearly average fee in Bitcoin for 2023, sourced from blockchain.com at `https://www.blockchain.com/explorer/charts/fees-usd-per-transaction`

the losses exhibit a relatively higher percentage value (20%), yet remain insignificant in absolute terms. This relatively larger percentage of loss can be attributed to the marginal profit/loss nearing zero. It is crucial to note that despite the notable percentage, the actual impact on daily profit is minimal, amounting to few cents of difference. As the daily profit increases for miners, the impact of this fee reduction on their profit diminishes, with the loss remaining below 1.3%.

Limitations It is important to note that this analysis is conducted through simulation, which means that the expected profit and loss may slightly differ from real-world scenarios. In real-world situations, various factors such as fluctuations in electricity prices, Bitcoin price, transaction fees, and mining probabilities would be considered, potentially impacting the outcomes. Therefore, it is crucial to acknowledge that these simulated results provide an approximation and may vary when accounting for the complexities of actual operational environments.

Summary

In this chapter, we presented the evaluation of our ML model. Firstly, we conducted a comparative analysis of various datasets used in the experiments. We defined metrics such as completeness and freshness to test the quality of these datasets.

e_p (\$/kWh)	Daily profit with \$ 7.5 fee			
Bitcoin price (\$)	10000	30000	60000	100000
Qatar : 0.032	0.54	4.35	10.06	17.68
US : 0.162	-9.59	-5.78	-0.07	7.54
	10% fee deduction			
Qatar	0.43	4.24	9.95	17.57
US	-9.71	-5.89	-0.18	7.42

Table 7.8: The table presents the effects of a 10% fee reduction on miners operating in Qatar and the US across different Bitcoin price variations. The block reward is set at ₿ 3.125, with 2,500 transactions per block, \$ 7.5 fee per transaction, and a single miner using Antminer S19 Pro.

Secondly, we defined evaluation metrics to assess the performance of the model. These metrics included classification accuracy, confusion matrix, and ROC analysis.

Thirdly, we evaluated the model's performance by considering different subsets of features and examining their individual contributions. We also conducted experiments to vary the model's freshness. The classification accuracy of the model reached peaks of 91%. Notably, using our complete set of features resulted in an average accuracy boost of 16% compared to the more common approach of using transaction size and fee alone.

Finally, we analyzed the potential benefits of our solution for end-users, as well as the corresponding potential loss that miners may incur. We demonstrated that a 10% reduction in fees would not negatively impact transaction inclusion. This finding highlights the positive impact our solution can have on transaction costs. Additionally, our analysis demonstrates that implementing a 10% fee reduction results in a minimal impact of less than 1.5% on miners' profits.

In the upcoming chapter, we discuss our results concerning several crucial aspects. These include the suitability of POW-based systems as a low payment scheme, the storage requirements for training the model, feature selection techniques, model cyclicity, transaction sampling approaches, and ethical considerations associated with POW.

8 Discussion

This chapter focuses on the discussions drawn from the analysis performed on Bitcoin. The limitations of the low-payment scheme for users is discussed. The importance of selecting the appropriate set of features for building an accurate prediction model is highlighted, and the comparison between different feature sets is presented. The impact of exogenous events on the accuracy of the model is also discussed. The significance of model cyclicity and the appropriate choice of hyperparameters in boosting the accuracy score is analyzed. Overall, this chapter provides valuable insights into the importance of selecting appropriate parameters and features to build an accurate model that can be used in a POW-based environment.

This dissertation is, to our knowledge, the first work that systematically use modern ML techniques to forecast transaction inclusion. Given this unique contribution, we acknowledge the challenges involved in conducting a comprehensive comparison with existing works. Therefore, interpreting our findings without a genuine real-world case comparison may appear less pertinent. Nevertheless, executing such assessments is a costly endeavor since it necessitates generating actual transactions with a relatively high fee based on the current average fee and subsequently adjusting it based on our model's prediction. Nonetheless, since our model accuracy exceeds 90%, the simulated outcomes demonstrate a high degree of solidity and dependability.

8.1 Bitcoin as Low-Payment Scheme

This dissertation provides insights on how suited POW-based systems, such as Bitcoin, are as low-payment schemes. We can list some reasons on why POW fails on this matter:

1. High Transaction Fees: the negative impact of interblock interval time and block size constraints on the system's throughput, fee markets have emerged. Fees provide a stable income for miners, which in turn leads to overpricing of transaction fees to guarantee immediate inclusion in the blockchain, causing users to pay fees that are two orders of magnitude higher than the recommended one.

2. Scalability Challenges: limitations on block size and frequency limits transaction throughput. As a consequence, it becomes challenging to handle a high volume of low-fee transactions in a timely manner.

3. Confirmation Time: POW systems have confirmation times that can vary depending on factors such as network congestion and mining difficulty. The time required for a transaction to be confirmed and included in a block can range from several minutes to hours. This delay is not suitable for low-payment schemes where quick and near-instantaneous transactions are desired.

4. Environmental Impact: POW algorithms require substantial energy consumption for mining activities. The process of solving computational puzzles in POW systems consumes a significant amount of electricity, leading to a substantial carbon footprint. This environmental impact and energy consumption make POW systems less desirable for low-fee transactions that do not justify the ecological cost.

To address issue number one, our study proposes a transaction inclusion model that extracts the necessary features for accurate classification. By training the model periodically, with a cadence of at least once a month, it is possible to classify incoming transactions into two classes with an accuracy score on average of 89%, which aims to optimize user expenditure.

8.2 Dataset Storage

Figure 8.1 illustrates the storage efficiency of the dataset we have generated. The complete set is somehow prohibitive to store and for that, our approach enables good results while saving a lot of space on disk. Compared to the actual size of the Bitcoin blockchain, our raw dataset $\mathcal{R}$ saves an average of 53.75% of disk space (see Figure 8.1).

Additionally, the training set $\mathcal{X}$, which contains block-epoch-based information, is 54.25% smaller on average than the original blockchain size, and saving up to 88.33% of disk space when smaller portions of the blockchain are analyzed.

We set a threshold of 3,000 blocks (approx. 20 days, or ca. 6 million transactions) for dataset evaluation, as the ResNet models we produce are designed for short-term predictions and are more accurate when generated cyclically. The dataset $\mathcal{C}$ has been found to be the heaviest among all the sets analyzed, with an average disk size of 158% greater than that of the Bitcoin blockchain when analyzing transactions greater than 100,000. For transactions below this threshold, $\mathcal{C}$ is 33% lighter than the Bitcoin blockchain. However, it is important to note that $\mathcal{C}$ is only stored during run-time and not permanently. In conclusion, the dataset $\mathcal{R}$ demonstrated to save 60% of space compared to the Bitcoin blockchain for

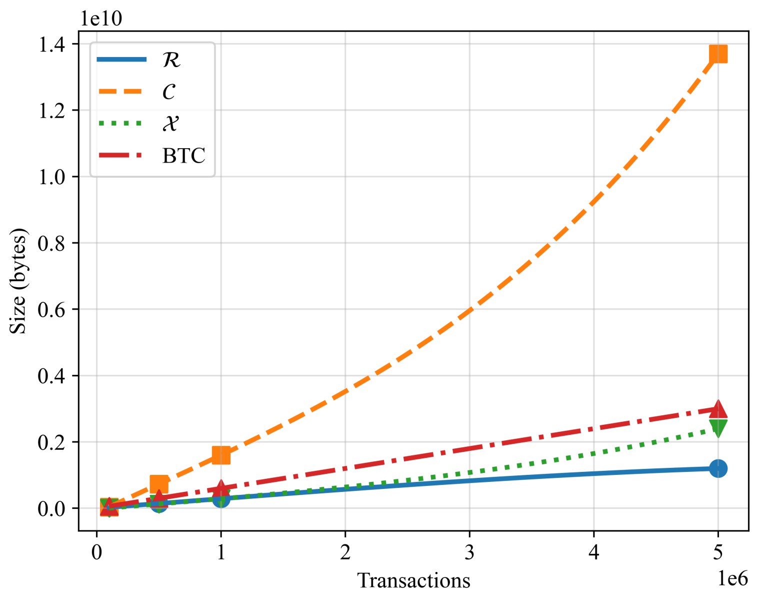


Figure 8.1: The following illustration demonstrates how various datasets we implemented scale in proportion to the size of the Bitcoin blockchain.

5 million transactions.

8.3 Selection of Features

During our experiments we outlined that careful selection of features is important Although more common and simpler fee estimators, which we refer to as primitive in this study, typically perform with an accuracy slightly above 70%. For example the command line call `Bitcoind estimatesmartfee` included in Bitcoin core, which relies solely on past blocks, transactions, and fee rates, our study employs a more comprehensive real-time based approach, with an accuracy score never below 85%.

We analyze the set of engineered features based on our intuition over a period of five months, which we refer to as the complete solution, and compare it with the primitive solution. We demonstrate that our proposed approach leads to a 16% improvement in accuracy score, on average. The accuracy score of the model experiences a downward trend during the months of April and May. This can be attributed to a series of events within the Bitcoin network, including a Bitcoin price drop of 46%, a surge in transaction fees to an all-time high, and an increase in miners' revenue. We argue that these exogenous events impacted the inclusion of transactions.

8.4 Model Cyclicity

The accuracy score of a classifier can be improved by maintaining the completeness and novelty of information in the model, particularly when unexpected events occur. This is demonstrated in Figure 7.2, which shows that classification accuracy drops when older models are used to classify recent data ($\psi \rightarrow 1$), or when information in the test set is incomplete ($\alpha \rightarrow 0$). Accuracy also drops considerably if models prior to the price inversion trend are used to classify more recent data. Two tests in April 2021 are compared in Table 8.1, one with complete data ($\alpha = 1$) and one with incomplete data ($\alpha = 0.5$), both classified with a model from January 2021. Despite data completeness, the mode incorrectly classifies 26% of transactions in v_1 as v_0 (false positive) due to the all-time-high fees in April. When data completeness is reduced to $\alpha = 0.5$, the false positives increase to 41%, while false negatives represent only 5%. In such cases, the model should be trained more frequently than once per month to reduce the number of misclassified transactions caused by deviations from the previous inclusion pattern.

Overall **CM** score, $\psi = 0.95$				
α	0.5		1	
	v_0	v_1	v_0	v_1
v_0	0.95	0.05	0.89	0.11
v_1	0.41	0.59	0.26	0.74

Table 8.1: This table displays the overall confusion matrix score for April 2021 data using the January 2021 model. The ψ value is fixed at 0.95, while two different values of α, 0.5 and 1, are evaluated. The importance of a complete dataset during training is highlighted, and the table demonstrates how classification accuracy can benefit from having complete information.

Figure 7.1 demonstrates that the selected features are crucial for accurate classification, and both parameters are fundamental for boosting the accuracy score. A complete set of information is needed to correctly calculate the offset value and have the right information on the current mempool size. An updated or new dataset helps to have knowledge of the current miner inclusion trend.

8.5 Transactions Sampling

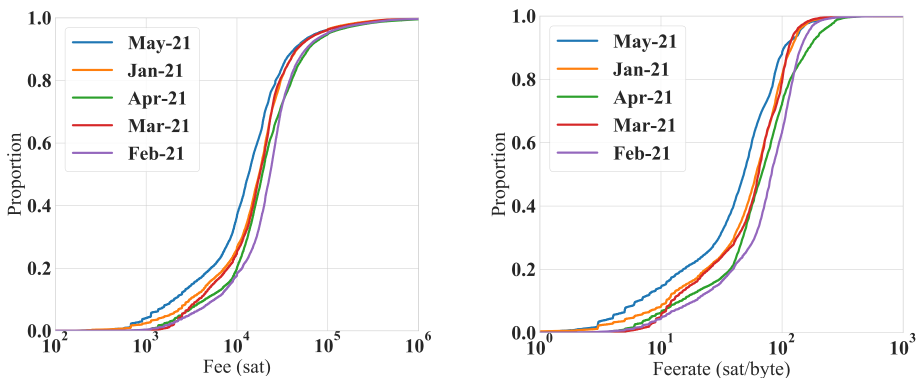


Figure 8.2: Cumulative distribution functions for transaction fees and feerate in our dataset.

Despite obtaining a good classification accuracy score, we observe that the model could be biased towards a specific range of transaction fees. In fact, bias gets into the model through the data that is used for building our classification model [163]. We carry out a supervised approach, thus our model only knows the outcome for transactions occurred in the blockchain. If most users pay a fee greater than the optimal value, the model lacks samples for small transaction fee values.

Figure 8.2 shows the fee and feerate Cumulative Distribution Function during the period of analysis. We observe that after the price drop occurred in late April 2021, fees were considerably lower (May 2021), and that transaction fees between 10^4 and 10^5 sat represent nearly the 75% of the total. Although the recommended transaction feerate should be 1 sat/byte, we observed an extremely low number of accepted transactions that utilize such a low fee rate. Consequently, one might conclude that our model is biased and it is not accurate when predicting transactions with feerate of 1 sat/byte. However, the overall fee trend of approved transactions delineates a pattern itself, meaning that a too low fee will rarely end up in an inclusion, which is in line with our model's outcome. In fact, lower fee transactions are evicted from the network and never included. Our model optimizes expenditure within the range of the already approved transactions, and its scope is not to detect possible evicted transactions, but to determine an inclusion in the next block. Information about eviction is not of particular relevance. Finally, any transaction issuer could consult the model's output in order to trade-off its probability of inclusion with more, or less fee to pay.

8.6 Ethical Concerns of Proof-of-Work

The computational power and energy required to mine results in a high energy consumption as the network scales and gets more secure. The carbon footprint can have negative impact to the environment, contributing to climate change and other environmental problems. Moreover, the cost of energy is a major barrier to entry for smaller miners, and despite the availability of renewable energy sources such as hydroelectric, solar, or wind power, many miners still depend on non-renewable sources like coal or natural gas. This reliance on non-renewable sources negatively impact the environment and it does contribute to greenhouse gas emissions. However, if most miners adopt renewable energy sources, a POW-based blockchain system has the potential to replace the worldwide banking system. Compared to the operational costs of the banking system, the operational costs of POW are relatively low. However, the transaction throughput may become a bottleneck to its adoption.

Another ethical concern related to POW is the unequal distribution of mining power. Mining requires specialized hardware and software, that is expensive to purchase and maintain. Consequently, mining is often more accessible to those with greater financial resources, leading to a concentration of mining power among a small group of individuals or organizations. This can give rich entities undue influence over the network, raising concerns about the overall decentralization and democratic governance of the network.

Mining generates a significant amount of e-waste. The specialized hardware used for mining quickly becomes obsolete as new, more powerful hardware is developed, contributing to the global issue of e-waste and further emphasizing the sustainability issues of POW mining at scale.

Acknowledging these ethical concerns, POW it is still considered the most secure and widely-used consensus mechanism in the blockchain industry. Our dissertation does not aim to question the applicability or sustainability of POW, although we are aware of its limitations. Instead, our study focuses on a widespread phenomenon related to blockchain technology, recognizing that alternative consensus mechanisms may exist that offer solutions to the ethical concerns associated with POW. Nevertheless, we propose an approach to reduce the cost for users in the Bitcoin network by employing transaction inclusion prediction. This reduces the negative burdens associated with the network.

Summary

This chapter examined the limitations of Bitcoin as a low-payment scheme, emphasizing high transaction fees, scalability challenges, confirmation time, and environmental impact. A transaction inclusion model was proposed with periodic training to optimize user expenditure.

The chapter also discussed the efficiency of dataset storage, with the generated dataset saving significant disk space. It highlighted the importance of selecting the right features for accurate prediction models, showing a 16% improvement compared to more simple estimators.

The impact of external events on model accuracy was considered, suggesting that factors like price drops and fee surges affected transaction inclusion. Model cyclicity was emphasized, recommending frequent training to maintain accuracy when faced with deviations.

The chapter also addressed the potential bias in the model towards a specific range of transaction fees. It acknowledged that the model's performance may be affected when predicting transactions with extremely low feerate, however, the model optimizes expenditure within the range of already approved transactions.

In summary, the discussion chapter provided insights into the limitations of Bitcoin as a low-payment scheme, the importance of feature selection, efficient dataset storage, model cyclicity, and ethical concerns associated with POW systems.

9 Concluding Remarks

The high fees reates that haunts POW-based systems at scale, makes ground-breaking system such as Bitcoin less appealing for end-users. This dissertation aims to improve our understanding modeling of the transaction fee mechanism and provide a method for users to optimize their fee expenditure. In this chapter, we provide a comprehensive listing of our contributions, we aim to provide a conclusive affirmation of the key arguments, and provide insights that we have developed throughout the course of this work.

To summarize the focus of this dissertation, our starting point is the thesis that:

Our thesis is that Bitcoin transaction fees can accurately be modeled and predicted using ML methods, improving utility and efficiency for clients using such cryptocurrencies, while maintaining a fair compensation for miners.

We conclude that our research has provided valuable insights into the challenges posed by the scalability bottleneck and high fees in POW-based systems such as Bitcoin. By thoroughly examining the transaction fee mechanism and integrating POW-based blockchains with ML models, we have made significant contributions to improving the utility and efficiency of cryptocurrencies for clients. Our findings demonstrate the efficacy of a formal transaction inclusion model in predicting transaction acceptance, shedding light on the underlying mechanisms and patterns governing miners' decisions. Moreover, our work has the potential to enhance the overall user experience, trust, and reliability of cryptocurrencies, empowering Bitcoin users with precise transaction inclusion predictions and enabling them to make more informed decisions regarding suitable fees. Through the strategic integration of POW-based systems and ResNet, we have paved the way for a more robust and user-centric cryptocurrency ecosystem.

9.1 Summary of Research Methodology and Findings

In Chapter 3, we discuss the principles and rules governing the Bitcoin ecosystem at scale, with a focus on the interdependence between users and miners. We explain how miners make profits, with related cost of mining, and miners' importance in transaction inclusion. The emergence of a fee market in POW-based blockchains is also explored, including the auction schemes that miners could adopt, and how this can lead to fee dynamism in Bitcoin. The chapter also relates these issues to the high fees and overpaying in Bitcoin, which are dictated by the mass adoption of Bitcoin and its inner throughput limitations. Understanding such fee market is important for formalizing a transaction inclusion pattern.

Chapter 4 presents the design and implementation of the Blockchain Analytics System (BAS), which we developed for acquiring and storing a local dataset of the Bitcoin blockchain. The chapter presents the methods and techniques used for data acquisition, including web scraping, APIs, and direct access to the blockchain using Bitcoin Core software. The acquired data is then structured and pre-processed to be suitable for analysis in subsequent chapters.

Chapter 5 discusses the use of time-series data analysis as a tool for predicting future trends. The methodology is based on the collection of time-series data, where transactions are sampled on a monthly basis with a fixed interval, and a notion of relative time is incorporated, represented by block creation epochs. The chapter presents a comprehensive model for the inclusion of transactions in a POW-based blockchain system, with a focus on the factors of revenue and fairness. Revenue serves as an incentive for miners to participate in the network and validate transactions, while fairness ensures equal opportunity for all users to have their transactions included in the blockchain upon paying an adequate fee value. The model takes into account both revenue and fairness as complementary factors for an accurate study and prediction of transaction inclusion.

Chapter 6 presents ML architecture. The chapter covers three critical stages: the ingestion engine, the pre-processing stage, and the ML model. The ingestion engine is responsible for processing and transforming raw data obtained from the blockchain. The pre-processing phase transforms raw data into a suitable form for the analysis, including feature extraction and additional data processing to generate a complete dataset. Finally, the chapter describes the ML model used for the prediction.

Chapter 7 evaluates ML model used in the study. It describes the datasets used for training and testing the model, the evaluation metrics employed, and the results of the analyses. The section reports on the overall accuracy of the model, feature importance, and model cyclicity. The findings of this study offer valuable insights into the effectiveness of the model as well as the appropriateness of the selected features for the intended purpose. Specifically, the study demonstrates the efficiency of the ML model in predicting transaction inclusion, which can potentially serve as a powerful tool for end-users. By utilizing our model, transaction issuers can maintain a high degree of confidence, with a likelihood of inclusion higher than 80%, in the next mined block while saving up to 10% in transaction fees.

9.2 Contributions

After examining the research presented in this dissertation and analyzing our thesis statement, we have drawn the following list of contributions:

1. Our research sheds light on the complexities of the transaction fee mechanism in Bitcoin, and the results of our study provide valuable insights into the underlying mechanisms and patterns governing miners' decisions to include individual transactions.

2. By integrating POW-based blockchains and ML models, we have shown that it is possible to improve the utility and efficiency of cryptocurrencies for clients, and we demonstrate the efficacy of a formal transaction inclusion model in predicting transaction acceptance in the blockchain.

3. Our work has significant implications for improving the overall user

experience and enhancing the trust and reliability of cryptocurrencies, providing Bitcoin users with a notable precision in predicting transaction inclusion and enabling them to better select suitable fees.

In summary, answering our thesis statement we can say that, *yes*, our thorough examination of POW-based blockchains and the application of ML models has indeed demonstrated that the strategic integration of these technologies can improve utility and efficiency for clients using cryptocurrencies, while still ensuring miners to get their revenue. Our research has demonstrated the efficacy of a formal model and ML prototype in predicting transaction inclusion, providing valuable insights into the underlying mechanisms governing miners' decisions and paving the way for improved utility and efficiency of cryptocurrencies for clients. Our work has significant implications for enhancing the trust and reliability of cryptocurrencies and improving the overall user experience by enabling Bitcoin users to better select suitable fees and predict transaction inclusion with notable precision.

9.3 Future Work

To enhance our confidence in the effectiveness of our model, it is crucial to conduct real-world tests of transaction fee reduction. In these tests, transactions would be submitted based on the predictions generated by our model, allowing to calculate the potential cost savings for end-users. This practical validation would provide valuable insights into the actual benefits and performance of our model in a live environment. The primary goal is to determine the extent to which transaction issuers can reduce their fees while still maintaining a high degree of confidence in the inclusion model's output. By conducting such research, it will be possible to quantify the amount of overspending that occurs in the Bitcoin network and contribute to improving the overall user experience.

It is crucial to continue observing blockchain data as the inclusion model may change over time, especially for developers who wish to utilize POW-based systems without overpaying for transaction space in blocks. Therefore, we are currently working on a modified version of the inclusion model that aims to improve the accuracy score. Our approach takes a holistic view of new block alternatives and penalizes transaction offset levels falling in the > 1 MB space of the mempool, thereby orienting the model towards a stronger second-price auction approach.

Furthermore, it would be beneficial to categorize transactions at each block epoch into ranks and incorporate transaction rank as a feature in the prediction model. Additionally, to mitigate training data bias arising from high-fee transactions, it would be valuable to introduce a significant number of low-fee transactions into the actual blockchain and record their latency in the model. By adopting these approaches, we can gain more precise insights into the dynamics of transaction inclusion, leading to the development of a more resilient and effective model that better serves end-users.

References

[1] Alex Tapscott and Don Tapscott. How blockchain is changing finance. *Harvard Business Review*, 1(9):2–5, 2017.

[2] Philip Treleaven, Richard Gendal Brown, and Danny Yang. Blockchain technology in finance. *Computer*, 50(9):14–17, 2017. doi: 10.1109/MC.2017.3571047.

[3] Primavera De Filippi. *Blockchain and the Law: The Rule of Code*. Harvard University Press, 2018. ISBN 9780674985933. doi: 10.4159/9780674985933.

[4] Marko Hölbl, Marko Kompara, Aida Kamišalić, and Lili Nemec Zlatolas. A systematic review of the use of blockchain in healthcare. *Symmetry*, 10(10):470, 2018.

[5] Sabine Kolvenbach, Rudolf Ruland, Wolfgang Gräther, and Wolfgang Prinz. Blockchain 4 education. In *Proceedings of 16th European Conference on Computer-Supported Cooperative Work-Panels, Posters and Demos*. European Society for Socially Embedded Technologies (EUSSET), 2018.

[6] Rodrigo Q. Saramago, Leander Jehl, Hein Meling, and Vero Estrada-Galiñanes. A tree-based construction for verifiable diplomas with issuer transparency. In *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 101–110, 2021. doi: 10.1109/DAPPS52256.2021.00017.

[7] Roi Bar Zur, Ittay Eyal, and Aviv Tamar. Efficient mdp analysis for selfish-mining in blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 113–131, 2020.

[8] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.

[9] Peter R Rizun. A transaction fee market exists without a block size limit. *Block Size Limit Debate Working Paper*, pages 2327–4697, 2015.

[10] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, Sept 2013. doi: 10.1109/P2P.2013.6688704.

[11] David Chaum. Achieving electronic privacy. *Scientific American*, 267(2): 96–101, 1992. ISSN 00368733, 19467087. URL `http://www.jstor.org/stable/24939181`.

[12] Adam Back. Hashcash - a denial of service counter-measure. Published at `http://www.hashcash.org`, 2002.

[13] Hal Finney. Reusable proofs of work. `https://nakamotoinstitute.org/finney/rpow/index.html`, 2004. Accessed: 2022-06-20.

[14] Shaurya Malwa. The first bitcoin transaction was sent to hal finney 12 years ago. `https://decrypt.co/53727/the-first-bitcoin-transaction-was-sent-to-hal-finney-12-years-ago`, 2021. Decrypt, Accessed: 2022-06-23.

[15] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*, pages 357–388. Springer, 2017.

[16] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR and arXiv*, abs/1710.09437, 2017. URL `http://arxiv.org/abs/1710.09437`.

[17] Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall, Cornell.edu*, 1(11), 2014.

[18] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.

[19] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19(1), 2012.

[20] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies.

In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 51–68, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350853. doi: 10.1145/3132747.3132757.

[21] David Schwartz, Noah Youngs, Arthur Britto, et al. The ripple protocol consensus algorithm. *Ripple Labs Inc White Paper*, 5(8):151, 2014.

[22] Brad Chase and Ethan MacBrough. Analysis of the XRP ledger consensus protocol. *CoRR*, abs/1802.07242, 2018. URL `http://arxiv.org/abs/1802.07242`.

[23] Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb. Fast and secure global payments with stellar. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 80–96, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368735. doi: 10.1145/3341301.3359636.

[24] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless BFT consensus through metastability. *CoRR*, abs/1906.08936, 2019. URL `http://arxiv.org/abs/1906.08936`.

[25] Ben Fisch, Joseph Bonneau, Nicola Greco, and Juan Benet. Scaling proof-of-replication for filecoin mining. *Protocol Labs: San Francisco, CA, USA*, 2018.

[26] Juan Benet and Nicola Greco. Filecoin: A decentralized storage network. *Protoc. Labs*, pages 1–36, 2018.

[27] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In Paul Spirakis and Philippas Tsigas, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 282–297, Cham, 2017. Springer International Publishing. ISBN 978-3-319-69084-1.

[28] I Stewart. Proof of burn. bitcoin.it `https://en.bitcoin.it/wiki/Proof_of_burn`, 2012.

[29] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Proof-of-burn. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 523–540, Cham, 2020. Springer International

Publishing. ISBN 978-3-030-51280-4.

[30] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]y. *SIGMETRICS Perform. Eval. Rev.*, 42(3):34–37, dec 2014. ISSN 0163-5999. doi: 10.1145/2695533.2695545.

[31] Bram Cohen and Krzysztof Pietrzak. The chia network blockchain. *chivescoin.org*, 1:1–44, 2019.

[32] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[33] Benjamin Wesolowski. Efficient verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 379–407. Springer, 2019.

[34] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. Pbft vs proof-of-authority: applying the cap theorem to permissioned blockchain. In *Italian Conference on Cyber Security (06/02/18)*, January 2018. URL `https://eprints.soton.ac.uk/415083/`.

[35] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The attack of the clones against proof-of-authority. *CoRR*, abs/1902.10244, 2019. URL `http://arxiv.org/abs/1902.10244`.

[36] Giuseppe Ateniese, Long Chen, Mohammad Etemad, and Qiang Tang. Proof of storage-time: Efficiently checking continuous data availability. Cryptology ePrint Archive, Paper 2020/840, 2020. URL `https://eprint.iacr.org/2020/840`. `https://eprint.iacr.org/2020/840`.

[37] Juan Benet, David Dalrymple, and Nicola Greco. Proof of replication. *Protocol Labs, July*, 27:20, 2017.

[38] Alex Smola and SVN Vishwanathan. Introduction to machine learning. *Cambridge University, UK*, 32(34):2008, 2008.

[39] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. Academic Press, 2009. ISBN 9781597492720.

[40] Hoon Sohn, Charles R Farrar, Norman F Hunter, and Keith Worden. Structural health monitoring using statistical pattern recognition techniques. *J. Dyn. Sys., Meas., Control*, 123(4):706–711, 2001.

[41] A.K. Jain, R.P.W. Duin, and Jianchang Mao. Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000. doi: 10.1109/34.824819.

[42] CD McGillem, JI Aunon, and DG Childers. Signal processing in evoked potential research: applications of filtering and pattern recognition. *Critical reviews in bioengineering*, 6(3):225—265, 1981. ISSN 0731-6984. URL `http://europepmc.org/abstract/MED/7023835`.

[43] Frank Y Shih. *Image processing and pattern recognition: fundamentals and techniques*. John Wiley & Sons, 2010.

[44] King-Sun Fu and Rosenfeld. Pattern recognition and image processing. *IEEE Transactions on Computers*, C-25(12):1336–1346, 1976. doi: 10.1109/TC.1976.1674602.

[45] Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, feb 2006. ISSN 1551-6857. doi: 10.1145/1126004.1126005.

[46] Lina Zhou, Shimei Pan, Jianwu Wang, and Athanasios V. Vasilakos. Machine learning on big data: Opportunities and challenges. *Neurocomputing*, 237:350–361, 2017. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2017.01.026.

[47] Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, and Shuo Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):67, 2016. doi: 10.1186/s13634-016-0355-x.

[48] Alexandra L'Heureux, Katarina Grolinger, Hany F. Elyamany, and Miriam A. M. Capretz. Machine learning with big data: Challenges and approaches. *IEEE Access*, 5:7776–7797, 2017. doi: 10.1109/ACCESS.2017.2696365.

[49] Ziad Obermeyer and Ezekiel J Emanuel. Predicting the future—big data, machine learning, and clinical medicine. *The New England journal of medicine*, 375(13):1216, 2016.

[50] Luigi Tommaso Luppino, Mads Adrian Hansen, Michael Kampffmeyer, Filippo Maria Bianchi, Gabriele Moser, Robert Jenssen, and Stian Normann Anfinsen. Code-aligned autoencoders for unsupervised change detection in multimodal remote sensing images. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13, 2022. doi: 10.1109/TNNLS.2022.3172183.

[51] Kee Yuan Ngiam and Ing Wei Khor. Big data and machine learning algorithms for health-care delivery. *The Lancet Oncology*, 20(5):e262–e273, 2019. ISSN 1470-2045. doi: https://doi.org/10.1016/S1470-2045(19)30149-4.

[52] Ahcéne Boubekki, Jonas Nordhaug Myhre, Luigi Tommaso Luppino, Karl Øyvind Mikalsen, Arthur Revhaug, and Robert Jenssen. Clinically relevant features for predicting the severity of surgical site infections. *IEEE Journal of Biomedical and Health Informatics*, 26(4):1794–1801, 2022. doi: 10.1109/JBHI.2021.3121038.

[53] Bilal Babar, Luigi Tommaso Luppino, Tobias Boström, and Stian Normann Anfinsen. Random forest regression for improved mapping of solar irradiance at high latitudes. *Solar Energy*, 198:81–92, 2020. ISSN 0038-092X. doi: https://doi.org/10.1016/j.solener.2020.01.034.

[54] Bruno Miranda Henrique, Vinicius Amorim Sobreiro, and Herbert Kimura. Literature review: Machine learning techniques applied to financial market prediction. *Expert Systems with Applications*, 124:226–251, 2019.

[55] Wei-Yang Lin, Ya-Han Hu, and Chih-Fong Tsai. Machine learning in financial crisis prediction: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):421–436, 2011.

[56] Sead Fadilpašić. Stop overpaying bitcoin transaction fees. 2019. Accessed: 2020-07-23. `https://bit.ly/2Cy5VtH`.

[57] Enrico Tedeschi, Tor-Arne S Nordmo, Dag Johansen, and Håvard D Johansen. Predicting transaction latency with deep learning in proof-of-work blockchains. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4223–4231. IEEE, 2019.

[58] Enrico Tedeschi, Tor-Arne S. Nordmo, Dag Johansen, and Håvard D. Johansen. On optimizing transaction fees in bitcoin using ai: Investigation

on miners inclusion pattern. *ACM Trans. Internet Technol.*, 22(3), jul 2022. ISSN 1533-5399. doi: 10.1145/3528669.

[59] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[60] Peter Godfrey Smith. Theory and reality: an introduction to the philosophy of science, 2003.

[61] Peter J. Denning. Is computer science science? *Commun. ACM*, 48(4): 27–31, apr 2005. ISSN 0001-0782. doi: 10.1145/1053291.1053309.

[62] Peter J Denning. Computing is a natural science. *Communications of the ACM*, 50(7):13–18, 2007.

[63] D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, Paul R. Young, and Peter J. Denning. Computing as a discipline. *Commun. ACM*, 32(1):9–23, jan 1989. ISSN 0001-0782. doi: 10.1145/63238.63239.

[64] David Lorge Parnas and Paul C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, SE-12(2):251–257, 1986. doi: 10.1109/TSE.1986.6312940.

[65] Enrico Tedeschi. Trading network performance for cash in the bitcoin blockchain. Master's thesis, UiT Norges arktiske universitet, 2017.

[66] Enrico Tedeschi, Håvard D. Johansen, and Dag Johansen. Trading network performance for cash in the bitcoin blockchain. In *Proceedings of the 8th International Conference on Cloud Computing and Services Science, CLOSER 2018, Funchal, Madeira, Portugal, March 19-21, 2018.*, pages 643–650, 2018. doi: 10.5220/0006805906430650.

[67] Dag Johansen, Pål Halvorsen, Håvard Johansen, Håkon Riiser, Cathal Gurrin, Bjørn Olstad, Carsten Griwodz, Åge Kvalnes, Joseph Hurley, and Tomas Kupka. Search-based composition, streaming and playback of video archive content. *Multimedia Tools and Applications*, 61(2):419–445, 2012. doi: 10.1007/s11042-011-0847-5.

[68] D. Johansen, K. Marzullo, and K. Lauvset. An approach towards an agent computing environment. In *Proceedings. 19th IEEE International Conference on Distributed Computing Systems. Workshops on Electronic Commerce and Web-based Applications. Middleware*, pages 78–83, 1999.

doi: 10.1109/ECMDD.1999.776418.

[69] Haakon Riiser, Pål Halvorsen, Carsten Griwodz, and Dag Johansen. Low overhead container format for adaptive streaming. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, MMSys '10, page 193–198, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589145. doi: 10.1145/1730836.1730859.

[70] Håvard D. Johansen, Eleanor Birrell, Robbert van Renesse, Fred B. Schneider, Magnus Stenhaug, and Dag Johansen. Enforcing privacy policies with meta-code. In *Proceedings of the 6th Asia-Pacific Workshop on Systems*, APSys '15, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450335546. doi: 10.1145/2797022.2797040.

[71] Håvard Johansen, André Allavena, and Robbert van Renesse. Fireflies: Scalable support for intrusion-tolerant network overlays. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, page 3–13, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933220. doi: 10.1145/1217935.1217937.

[72] Joakim Aalstad Alslie, Aril Bernhard Ovesen, Tor-Arne Schmidt Nordmo, Håvard Dagenborg Johansen, Pål Halvorsen, Michael Alexander Riegler, and Dag Johansen. Áika: A distributed edge system for ai inference. *Big Data and Cognitive Computing*, 6(2), 2022. ISSN 2504-2289. doi: 10.3390/bdcc6020068.

[73] Tor-Arne S. Nordmo, Aril B. Ovesen, Hårvard D. Johansen, Michael A. Riegler, Pål Halvorsen, and Dag Johansen. Dutkat: A multimedia system for catching illegal catchers in a privacy-preserving manner. In *Proceedings of the 2021 Workshop on Intelligent Cross-Data Analysis and Retrieval*, ICDAR '21, page 57–61, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385299.

[74] Aril Bernhard Ovesen, Tor-Arne Schmidt Nordmo, Håvard Dagenborg Johansen, Michael Alexander Riegler, Pål Halvorsen, and Dag Johansen. File system support for privacy-preserving analysis and forensics in low-bandwidth edge environments. *Information*, 12(10), 2021. ISSN 2078-2489. doi: 10.3390/info12100430.

[75] Rodrigo Q Saramago, Leander Jehl, Hein Meling, and Vero Estrada-Galiñanes. A tree-based construction for verifiable diplomas with issuer transparency. In *2021 IEEE International Conference on Decentralized*

Applications and Infrastructures (DAPPS), pages 101–110. IEEE, 2021.

[76] Racin Nygaard, Hein Meling, and Leander Jehl. Distributed storage system based on permissioned blockchain. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 338–340, 2019.

[77] Petter Olsen, Melania Borit, and Shaheen Syed. Applications, limitations, costs, and benefits related to the use of blockchain technology in the food industry. *Nofima rapportserie*, 2019.

[78] Petter Olsen and Melania Borit. The components of a food traceability system. *Trends in Food Science & Technology*, 77:143–149, 2018.

[79] Petter Olsen and Melania Borit. How to define traceability. *Trends in food science & technology*, 29(2):142–150, 2013.

[80] Ignacio Amores-Sesar, Christian Cachin, and Enrico Tedeschi. When Is Spring Coming? A Security Analysis of Avalanche Consensus. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems (OPODIS 2022)*, volume 253 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:22, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-265-5. doi: 10.4230/LIPIcs.OPODIS.2022.10.

[81] Alex Pellegrini and Luca Zanolini. An algebraic model for quorum systems. *CoRR*, abs/2005.08536, 2020. URL `https://arxiv.org/abs/2005.08536`.

[82] Christian Cachin, Jovana Micic, and Nathalie Steinhauer. Quick order fairness. *CoRR*, abs/2112.06615, 2021. URL `https://arxiv.org/abs/2112.06615`.

[83] Christian Cachin and Luca Zanolini. Asymmetric byzantine consensus. *CoRR*, abs/2005.08795, 2020. URL `https://arxiv.org/abs/2005.08795`.

[84] Orestis Alpos and Christian Cachin. Consensus beyond thresholds: Generalized byzantine quorums made live. *CoRR*, abs/2006.04616, 2020. URL `https://arxiv.org/abs/2006.04616`.

[85] Marcus Brandenburger, Christian Cachin, and Nikola Knežević. Don't

trust the cloud, verify: Integrity and consistency for cloud object stores. *ACM Trans. Priv. Secur.*, 20(3), jul 2017. ISSN 2471-2566. doi: 10.1145/3079762. URL `https://doi.org/10.1145/3079762`.

[86] Christian Cachin, Angelo De Caro, Pedro Moreno-Sanchez, Björn Tackmann, and Marko Vukolic. The Transaction Graph for Modeling Blockchain Semantics. *Cryptoeconomic Systems*, 0(1), apr 5 2021. https://cryptoeconomicsystems.pubpub.org/pub/cachin-blockchain-semantics.

[87] Ignacio Amores-Sesar, Christian Cachin, and Anna Parker. Generalizing weighted trees: A bridge from bitcoin to ghost. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, AFT '21, page 156–169, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450390828. doi: 10.1145/3479722.3480995.

[88] Ignacio Amores-Sesar, Christian Cachin, and Jovana Mićić. Security Analysis of Ripple Consensus. In Quentin Bramas, Rotem Oshman, and Paolo Romano, editors, *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*, volume 184 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISBN 978-3-95977-176-4. doi: 10.4230/LIPIcs.OPODIS.2020.10.

[89] Enrico Tedeschi. Bitcoin blockchain optimized for machine learning prediction model, 2022. URL `https://doi.org/10.18710/8IKVEU`.

[90] Roger Maull, Phil Godsiff, Catherine Mulligan, Alan Brown, and Beth Kewell. Distributed ledger technology: Applications and implications. *Strategic Change*, 26(5):481–489, 2017.

[91] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[92] Serguei Popov. The tangle. *White paper*, 1(3), 2018.

[93] Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep*, 34, 2016.

[94] Leemon Baird, Mance Harmon, and Paul Madsen. Hedera: A public hashgraph network & governing council. *White Paper*, 1, 2019.

[95] Jeff Kauflin. Hedera hashgraph thinks it can one-up bitcoin and ethereum with faster transactions. `https://bit.ly/3QjR8Df`, 2018.

[96] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* Stanford university, 1979.

[97] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, page 205–220, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595935915. doi: 10.1145/1294261.1294281. URL `https://doi.org/10.1145/1294261.1294281`.

[98] Christian Cachin and Marko Vukolic. Blockchain consensus protocols in the wild. *CoRR*, abs/1707.01873, 2017. URL `http://arxiv.org/abs/1707.01873`.

[99] Ha Nguyen and Linh Do. The adoption of blockchain in food retail supply chain: Case: Ibm food trust blockchain and the food retail supply chain in malta. 2018.

[100] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[101] DL Chaum. Computer systems established, maintained and trusted by mutually suspicious groups (doctoral dissertation). *University of California Berkeley, Berkeley, CA*, 1982.

[102] Alan T Sherman, Farid Javani, Haibin Zhang, and Enis Golaszewski. On the origins and variations of blockchain technologies. *IEEE Security & Privacy*, 17(1):72–77, 2019.

[103] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[104] Fred B Schneider. Byzantine generals in action: Implementing fail-stop

processors. *ACM Transactions on Computer Systems (TOCS)*, 2(2):145–154, 1984.

[105] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[106] Leslie Lamport. *The Part-Time Parliament*, page 277–317. Association for Computing Machinery, New York, NY, USA, 2019. ISBN 9781450372701. URL https://doi.org/10.1145/3335772.3335939.

[107] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 139–147, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. ISBN 978-3-540-48071-6.

[108] John R Douceur. The sybil attack. In *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1*, pages 251–260. Springer, 2002.

[109] Reserve Bank of Australia. Digital currencies. https://www.rba.gov.au/education/resources/explainers/pdf/cryptocurrencies.pdf?v=2022-08-26-13-22-14, 2022. The Reserve Bank of Australia (RBA) is Australia's central bank and derives its functions and powers from the Reserve Bank Act 1959.

[110] Rick Miller. Bitcoin is a cryptocurrency, but is it money? *Forbes*, 2021. Accessed: 2022-08-26.

[111] Nicholas Weaver. Risks of cryptocurrencies. *Commun. ACM*, 61(6): 20–24, may 2018. ISSN 0001-0782. doi: 10.1145/3208095. URL https://doi.org/10.1145/3208095.

[112] Tom M Mitchell and Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[113] Soumya Raychaudhuri, Patrick D. Sutphin, Jeffrey T. Chang, and Russ B. Altman. Basic microarray analysis: grouping and feature reduction. *Trends in Biotechnology*, 19(5):189–193, 2001. ISSN 0167-7799. doi: https://doi.org/10.1016/S0167-7799(01)01599-2.

[114] Anthony McGregor, Mark Hall, Perry Lorier, and James Brunskill. Flow

clustering using machine learning techniques. In *International workshop on passive and active network measurement*, pages 205–214. Springer, 2004.

[115] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. doi: 10.1109/TKDE.2009.191.

[116] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324.

[117] James A Anderson. *An introduction to neural networks*. MIT press, 1995.

[118] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[119] Geoffrey Hinton and Terrence J Sejnowski. *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

[120] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[121] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[122] Anne Håkansson and Ronald Hartung. *Artificial Inteeligence. Concepts, Areas, Techniques and Applications*. Studentlitteratur, 2020. ISBN 9789144125992.

[123] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[124] Yahya Shahsavari, Kaiwen Zhang, and Chamseddine Talhi. A theoretical model for fork analysis in the bitcoin network. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 237–244. IEEE, 2019.

[125] David Easley, Maureen O'Hara, and Soumya Basu. From mining to markets: The evolution of bitcoin transaction fees. *Journal of Financial*

Economics, 134(1):91 – 109, 2019. ISSN 0304-405X. doi: https://doi.org/10.1016/j.jfineco.2019.03.004.

[126] Kerem Kaskaloglu. Near zero bitcoin transaction fees cannot last forever. *The International Conference on Digital Security and Forensics (DigitalSec2014)*, pages 91–99, 2014.

[127] Malte Möser and Rainer Böhme. Trends, tips, tolls: A longitudinal study of Bitcoin transaction fees. In *Financial Cryptography and Data Security: FC 2015.*, number 8976 in LNCS, pages 19–33, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. ISBN 978-3-662-48051-9. doi: 10.1007/978-3-662-48051-9_2.

[128] Soumya Basu, David A. Easley, Maureen O'Hara, and Emin Gün Sirer. Towards a functional fee market for cryptocurrencies. *CoRR*, abs/1901.06830, 2019. URL `http://arxiv.org/abs/1901.06830`.

[129] Vitalik Buterin. First and second-price auctions and improved transaction-fee markets. `https://ethresear.ch/t/first-and-second-price-auctions-and-improved-transaction-fee-markets/2410`, 2018.

[130] Johnnatan Messias, Mohamed Alzayat, Balakrishnan Chandrasekaran, and Krishna P Gummadi. On blockchain commit times: An analysis of how miners choose bitcoin transactions. In *The Second International Workshop on Smart Data for Blockchain and Distributed Ledger (SDBD2020)*, 2020.

[131] Juanjuan Li, Xiaochun Ni, Yong Yuan, and Feiyue Wang. A novel gsp auction mechanism for dynamic confirmation games on bitcoin transactions. *IEEE Transactions on Services Computing*, pages 1–1, 2020. doi: 10.1109/TSC.2020.2994582.

[132] Juanjuan Li, Yong Yuan, and Fei-Yue Wang. A novel gsp auction mechanism for ranking bitcoin transactions in blockchain mining. *Decision Support Systems*, 124:113094, 2019.

[133] Javier Diez-Sierra and Manuel del Jesus. Long-term rainfall prediction using atmospheric synoptic patterns in semi-arid climates with statistical and machine learning methods. *Journal of Hydrology*, 586:124789, 2020. ISSN 0022-1694. doi: https://doi.org/10.1016/j.jhydrol.2020.124789.

[134] Saurav Nanda, Faheem Zafari, Casimer DeCusatis, Eric Wedaa, and Bai-

jian Yang. Predicting network attack patterns in sdn using machine learning approach. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 167–172, 2016. doi: 10.1109/NFV-SDN.2016.7919493.

[135] Abbas Yazdinejad, Hamed HaddadPajouh, Ali Dehghantanha, Reza M. Parizi, Gautam Srivastava, and Mu-Yen Chen. Cryptocurrency malware hunting: A deep recurrent neural network approach. *Applied Soft Computing*, 96:106630, 2020. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2020.106630.

[136] RSJD Baker et al. Data mining for education. *International encyclopedia of education*, 7(3):112–118, 2010.

[137] Ioanna Lykourentzou, Ioannis Giannoukos, Vassilis Nikolopoulos, George Mpardis, and Vassili Loumos. Dropout prediction in e-learning courses through the combination of machine learning techniques. *Computers & Education*, 53(3):950 – 965, 2009. ISSN 0360-1315. doi: https://doi.org/10.1016/j.compedu.2009.05.010.

[138] Indranil Bose and Radha K Mahapatra. Business data mining—a machine learning perspective. *Information & management*, 39(3):211–225, 2001.

[139] Jared Dean. *Big data, data mining, and machine learning: value creation for business leaders and practitioners*. John Wiley & Sons, 2014.

[140] Min Chen, Yixue Hao, Kai Hwang, Lu Wang, and Lin Wang. Disease prediction by machine learning over big data from healthcare communities. *IEEE Access*, 5:8869–8879, 2017. doi: 10.1109/ACCESS.2017.2694446.

[141] Sumeet Dua, U Rajendra Acharya, and Prerna Dua. *Machine learning in healthcare informatics*, volume 56. Springer, 2014.

[142] Salvatore Stolfo, David W Fan, Wenke Lee, Andreas Prodromidis, and P Chan. Credit card fraud detection using meta-learning: Issues and initial results. In *AAAI-97 Workshop on Fraud Detection and Risk Management*, 1997.

[143] Theodore B Trafalis and Huseyin Ince. Support vector machine for regression and applications to financial forecasting. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the*

New Millennium, volume 6, pages 348–353. IEEE, 2000.

[144] Yu Zheng, Like Liu, Longhao Wang, and Xing Xie. Learning transportation mode from raw gps data for geographic applications on the web. In *Proceedings of the 17th international conference on World Wide Web*, pages 247–256. ACM, 2008.

[145] Xiaolei Ma, Haiyang Yu, Yunpeng Wang, and Yinhai Wang. Large-scale transportation network congestion evolution prediction using deep learning theory. *PloS one*, 10(3):e0119044, 2015.

[146] Tom Augspurger, Chris Bartak, Phillip Cloud, Andy Hayden, Stephan Hoyer, Wes McKinney, Jeff Reback, Chang She, Masaaki Horikoshi, Joris Van den Bossche, et al. Pandas: Python Data Analysis Library. software v0.21.0, Pandas community, 2012. URL `http://pandas.pydata.org/`.

[147] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[148] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[149] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[150] Mahmoudreza Tahmassebpour. A new method for time-series big data effective storage. *IEEE Access*, PP:1–1, 05 2017. doi: 10.1109/ACCESS.2017.2708080.

[151] J. Doyne Farmer and John J. Sidorowich. Predicting chaotic time series. *Phys. Rev. Lett.*, 59:845–848, Aug 1987. doi: 10.1103/PhysRevLett.59.845. URL https://link.aps.org/doi/10.1103/PhysRevLett.59.845.

[152] George Foster. Quarterly accounting data: Time-series properties and predictive-ability results. *The Accounting Review*, 52(1):1–21, 1977. ISSN 00014826. URL http://www.jstor.org/stable/246028.

[153] William WS Wei. Time series analysis. In *The Oxford Handbook of Quantitative Methods in Psychology: Vol. 2*. 2006.

[154] James Douglas Hamilton. *Time series analysis*. Princeton university press, 2020.

[155] Wayne A Fuller. *Introduction to statistical time series*, volume 428. John Wiley & Sons, 2009.

[156] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized Blockchains. In *Financial Cryptography and Data Security. FC 2016.*, volume 9604 of *LNCS*, pages 106–125, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. ISBN 978-3-662-53357-4. doi: 10.1007/978-3-662-53357-4_8.

[157] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05, pages 45–58, New York, NY, USA, 2005. ACM. ISBN 1-59593-079-5. doi: 10.1145/1095810.1095816.

[158] Nicolas Houy. The bitcoin mining game. *Available at SSRN 2407834*, 2014.

[159] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT

press, 2016.

[160] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[161] Sotiris B Kotsiantis, Dimitris Kanellopoulos, and Panagiotis E Pintelas. Data preprocessing for supervised leaning. *International journal of computer science*, 1(2):111–117, 2006.

[162] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[163] Jindong Gu and Daniela Oelke. Understanding bias in machine learning. *CoRR*, abs/1909.01866, 2019. URL `http://arxiv.org/abs/1909.01866`.

[164] James Lovejoy and Anne Ouyang. 51% attacks. MIT media lab, 2020. URL `https://dci.mit.edu/51-attacks`. MIT digital currency initiative.

[165] David Siegel, Mark Ly, Greg Fraser, Scott Miller, Caleb Silver, et al. Definition of 51 attack. Investopedia, `https://www.investopedia.com/terms/1/51-attack.asp`, 2017.

[166] Atsushi Kajii and Stephen Morris. The robustness of equilibria to incomplete information. *Econometrica*, 65(6):1283–1309, 1997. ISSN 00129682, 14680262. URL `http://www.jstor.org/stable/2171737`.

List of Symbols

Sign	Description	Unit
B_{day}	Blocks mined in a day	
H	Total hash rate of Bitcoin network	hash/s
M	Revenue of a block. Sum of all ϕ in a block	₿ or $
Q	Block size, $\sim 1.1\,\text{MB}$	bytes
R	Block reward, currently at ₿ 6.25, halved every 210, 000 blocks	₿
S	Set of ordered pending transactions	
$\Delta\mathcal{P}$	It represents the waiting time for a transaction t, before it is included in a block	s
β_t	Block epoch (be) of the block before t's inclusion	
δ_t	Feature that represents the offset, in bytes, of a certain transaction t	bytes
η_{day}	Consumption per day	kWh
η	Cost per hash	kWh/hash
$\langle C \rangle$	Expectation value of a miner's hashing cost per block	₿
$\langle V \rangle$	Expectation value of a miner's revenue per block	₿
$\langle \Pi \rangle$	Expectation value of a miner's profit per block	₿
$\mathbb{E}[f]$	Expected transaction fee value. What is considered to be "a fair amount"	₿
$\mathbb{P}_{orphan}$	Probability that, mined a new block, it gets orphaned	$[0, 1]$
$\mathbb{T}$	Set containing occurrences of block creation times, in epochs	

Sign	Description	Unit
$\mathcal{A}$	Set of temporary approved transactions	
$\mathcal{D}_B$	Locally stored dsataset containing raw fetched blocks	
$\mathcal{D}_T$	Locally stored dsataset containing raw fetched transactions	
$\mathcal{L}_t^x$	Lifespan of a transaction t	
$\mathcal{M}$	Set containing all active miners in Bitcoin	
$\mathcal{P}$	Set of relapsed pending transactions, RPT	
$\mathcal{R}$	Raw dataset, containing the stored portion of the blockchain	
$\mathcal{T}'$	Calculated block interval time	s
$\mathcal{T}$	Interblock time interval, equals to 600 s	s
ϕ	Transaction fee	₿
$\boldsymbol{T}$	Complete block-epoch-based representation of a transaction	
$\boldsymbol{Xte}$	Testing set	
$\boldsymbol{X}$	Training set	
$\boldsymbol{\theta}_t$	Model output array, $\boldsymbol{\theta}_t = [P_t(v_0),\ P_t(v_1)]$	
ρ	Price per byte for block space, or feerate	sat/bytes
BTCP	Bitcoin price in U.S. dollars.	\$/₿
v_0	Class of not approved transactions	
v_1	Class of approved transactions	
ξ	Latest block created	height
be_ξ	Latest block creation time (epoch), $x = -1$	s
be_x	Block creation time (epoch). It represents the epoch of a certain block at height x	s
d_t	Difficulty to solve Proof-of-Work at time t	
e_p	Electricity price	\$/kWh

Sign	**Description**	**Unit**
ep_t	Timestamp of a transaction t. Epoch which represents first appearance of t in the network	s
g_B	Gain in USD for mining a block B	$
ha_t	Transaction hash	
ha_x	Block hash at height x	
h	Miner's individual hash rate	hash/s
inp	Transaction input of the UTXO model. Money received in a specific transaction	₿
l_t	Latency of a transaction t	s
out	Transaction output of the UTXO model. Money sent in a specific transaction	₿
q	Transaction size	bytes
t_B	Number of transaction approved in a block B.	
t_a	Transaction that belongs to the set $\mathcal{A}$ of the Temporarily Approved Transaction	
t_p	Transaction that belongs to the set $\mathcal{P}$ of the Relapsed Pending Transaction	

A Other Definitions

A.1 Bitcoin

Orphaning

The likelihood of a miner, denoted as m, successfully publishing a block decreases if the chosen block is slow to propagate to other miners. In such a scenario, even if m successfully solves a block b, this block may be discarded if a newer block b' mined by another miner m' propagates faster. This phenomenon is referred to as *orphaning*, and the probability of orphaning is determined by the block propagation time τ. As the block creation time follows a Poisson distribution, it can be formalized as:

$$\mathbb{P}_{orphan} = 1 - e^{-\frac{\tau}{T}} \tag{A.1}$$

51% Attack

51% refers to an attack on a POW-based blockchain by an attacker that amasses a majority of the hash rate in the targeted cryptocurrency [164]. Proof-of-Work is intended to make such attack prohibitively expensive, but if accomplished, the attacker could rewrite the blockchain and reverse transactions that are considered settled. For instance, if an attacker tries to double spend its tokens, it will first purchase goods with them. Once the transaction is approved, the attacker forks the blockchain and it uses its 51% hash rate to re-build and overtake the honest chain, and new blocks, including the malicious transactions that spends again the same token, is included and approved by the network. Such attack would not destroy Bitcoin or any other cryptocurrency, but it could provide severe damage to its users, and to the fame of the currency itself [165].

Full Node

Node that stores and processes the entirety of every block, saving locally the entire blockchain.

Light Node

Node that only stores the part of the blockchain it needs.

Bitcoin-cli

Bitcoin-cli is a command-line interface for interacting with a local or remote Bitcoin Core instance, which is a full node implementation of the Bitcoin protocol. It allows users to send commands to the Bitcoin Core daemon (bitcoind) and receive the results. This enables developers to easily query the Bitcoin blockchain, retrieve transaction details, and perform other actions such as sending transactions, or managing wallets.

A.2 Machine Learning

Model Hyperparameters

Our implementation of ResNet consists of three sets of residual blocks, each set consisting of two fully connected (Dense) layers followed by a skip connection (Add) that bypasses the two Dense layers. The first layer in the ResNet is a Dense layer with 256 hidden units and a ReLU activation function. The output of the first Dense layer is passed through three more Dense layers with 256 hidden units each and ReLU activation functions. Then, the output of the first Dense layer is added to the output of the last Dense layer in this set using a skip connection. The output of the first residual block is then passed through two more residual blocks, each consisting of two Dense layers with 64 hidden units and ReLU activation functions. The output of the second residual block is added to the output of the first residual block using another skip connection. The output of the second set of residual blocks is passed through one more residual block with 64 hidden units, and its output is added to the output of the second residual block using another skip connection. The final layers of the network are two Dense layers with 32 and 16 hidden units, respectively, both with ReLU activation functions. The output layer is a Dense layer with two number of units and a softmax activation function, which gives the probability distribution over the possible classes.

The weight initialization of all the Dense layers is done using the He normal initializer, which helps in initializing weights that are well-suited for deep neural networks. Finally, the ResNet is compiled using the categorical cross-entropy loss function, an Adam optimizer, and accuracy as the evaluation metric.

A.3 Statistics

Bayesian Nash equilibrium

In game theory, Bayesian Nash equilibrium (BNE) is a strategy profile that maximizes the expected payoff for each player given a prior beliefs and strategies played by other players. A strategy profile σ is a BNE if and only if for every player i, each one with their prior beliefs and strategies of other players fixed, σ_i maximizes the expected payoff [166].

$$\sigma \text{ is BNE} \iff \forall i, \sum_i \text{payoff}(\sigma_i) \geq \sum_i \text{payoff}(\sigma\prime_i) \tag{A.2}$$

for any possible strategy $\sigma\prime$.

B Publications

This dissertation is based on the work presented in the following five publications:

Publication I

Tedeschi, E., Johansen, H.D. and Johansen, D., 2018. “Trading Network Performance for Cash in the Bitcoin Blockchain”, in CLOSER 2018, *the 8th International Conference on Cloud Computing and Services Science*, pp. 643-650.

Scope

In this paper we perform an initial longitudinal study on the Bitcoin network, where we observe a consistent correlation between fee paid and transactions latency. Using linear regression methods we design a model for fee-cost estimation, and we outline that paying above a certain threshold it will not result in better acceptance performances.

Abstract

Public blockchains have emerged as a plausible cloud-like substrate for applications that require resilient communication. However, sending messages over existing public blockchains can be cumbersome and costly as miners require payment to establish consensus on the sequence of messages. In this paper we analyze the network performance of the Bitcoin public ledger when used as a massaging substrate. We present several real-world observations on its characteristics, transaction visibility, and fees paid to miners; and we propose two models for fee-cost estimation. We find that applications to some extent can improve messaging latency by paying transaction fees. We also suggest that spendings should be kept below 300 Satoshi per byte.

Publication II

E. Tedeschi, T. S. Nordmo, D. Johansen and H. D. Johansen, "Predicting Transaction Latency with Deep Learning in Proof-of-Work Blockchains", 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 4223-4231, `https://doi.org/10.1109/BigData47090.2019.9006228`.

Scope

In this paper we illustrate our first model idea using DNN to predict transaction latency in POW-based blockchains. This solution is based on our previous longitudinal study and on a careful feature selection. This paper defines what we consider the foundations for defining transaction inclusion pattern.

Abstract

Proof-of-work based cryptocurrencies, like Bitcoin, have a fee market where transactions are included in the blockchain according to a first-price auction for block space. Many attempts have been made to adjust and predict the fee volatility, but even well-formed transactions sometimes experience delays and evictions unless an enormous fee is paid. In this paper, we present a novel machine-learning model, solving a binary classification problem, that can predict transaction fee volatility in the Bitcoin network so that users can optimize their fees expenses and the approval time for their transactions. The model's output will give a confidence score whether a new incoming transaction

will be included in the next mined block. The model is trained on data from a longitudinal study of the Bitcoin blockchain, containing more than 10 million transactions. New features that we generate include information on how many bytes were already occupied by other transactions in the mempool, assuming they are ordered by fee density in each mining pool. The collected dataset allows to generate a model for transaction inclusion pattern prediction in the Bitcoin network, hence telling whether a transaction is well formed or not, according to the previous transactions analyzed. With this, we obtain a prediction score for up to 86%.

Publication III

Enrico Tedeschi, Tor-Arne S. Nordmo, Dag Johansen, and Håvard D. Johansen. 2022. "On Optimizing Transaction Fees in Bitcoin using AI: Investigation on Miners Inclusion Pattern". Association for Computing Machinery (ACM) Trans. Internet Technol. 22, 3, Article 77 (August 2022), 28 pages. `https://doi.org/10.1145/3528669`

Scope

In this paper we refine ML methods to obtain a better inclusion prediction for transactions in the Bitcoin network, and we describe new methods for data gathering and manipulation. Data are stored on a time-series like approach and the study is characterized by a formal definition of the model for transaction inclusion pattern. A new series of evaluations and experiments are conducted. The results obtained here represent the main contribution of our dissertation.

Abstract

The transaction-rate bottleneck built into popular proof-of-work-based cryptocurrencies, like Bitcoin and Ethereum, leads to fee markets where transactions are included according to a first-price auction for block space. Many attempts have been made to adjust and predict the fee volatility, but even well-formed transactions sometimes experience unexpected delays and evictions unless a substantial fee is offered. In this paper, we propose a novel transaction inclusion model that describes the mechanisms and patterns governing miners decisions to include individual transactions in the Bitcoin system. Using this

model we devise a ML approach to predict transaction inclusion. We evaluate our predictions method using historical observations of the Bitcoin network from a five month period that includes more than 30 million transactions and 120 million entries. We find that our ML model can predict fee volatility with an accuracy of up to 91%. Our findings enable Bitcoin users to improve their fee expenses and the approval time for their transactions.

Publication IV

Amores-Sesar, I., Cachin, C. and Tedeschi, E., 2022. “When is Spring coming? A Security Analysis of Avalanche Consensus”. `https://doi.org/10.48550/arXiv.2210.03423`.

Scope

In this paper we study the security of a DAG-based blockchain, Avalanche. We provide a detailed formulation of the Avalanche blockchain consensus protocol using pseudo-code, addressing features that were omitted from the original white-paper. Additionally, the paper provides an analysis of the formal properties of Avalanche as a generic broadcast protocol that orders related transactions. The analysis highlights a vulnerability that affects the liveness of the protocol and proposes a potential solution to address the issue. Despite the success of Avalanche, the consensus protocol lacks a complete abstract specification and formal analysis, making this work a valuable contribution to the field.

Abstract

Avalanche is a blockchain consensus protocol with exceptionally low latency and high throughput. This has swiftly established the corresponding token as a top-tier cryptocurrency. Avalanche achieves such remarkable metrics by substituting proof of work with a random sampling mechanism. The protocol also differs from Bitcoin, Ethereum, and many others by forming a Directed Acyclic Graph (DAG) instead of a chain. It does not totally order all transactions, establishes a partial order among them, and accepts transactions in the DAG that satisfy specific properties. Such parallelism is widely regarded as a technique that increases the efficiency of consensus. Despite its success, Avalanche consensus lacks a complete abstract specification and a matching

formal analysis. To address this drawback, this work provides first a detailed formulation of Avalanche through pseudo-code. This includes features that are omitted from the original white-paper or are only vaguely explained in the documentation. Second, the paper gives an analysis of the formal properties fulfilled by Avalanche in the sense of a generic broadcast protocol that only orders related transactions. Last but not least, the analysis reveals a vulnerability that affects the liveness of the protocol. A possible solution that addresses the problem is also proposed.

Publication V - Dataset

Tedeschi, Enrico, 2022, "Bitcoin blockchain optimized for machine learning prediction model", `https://doi.org/10.18710/8IKVEU`, DataverseNO, V1

Scope

This dataset stores part of the Bitcoin blockchain. Blocks are sampled every month and information about transactions and blocks are separated to save disk space and avoid redundancies. This dataset is used in the work presented by Tedeschi et al. [58] which is submitted for review, in order to generate a machine learning model that predicts transaction inclusion. In each month of analysis, there is a block folder and a transaction folder. Information can be merged runtime through 'bhash' attribute (block hash). (2022-02-12)

/C

Source Code

C.1 Blockchain APIs

Listing C.1: RAW BLOCK JSON response of `blockchain.com` APIs at height 600000

```
{
  "hash": "000000...",
  "ver": 536870912,
  "prev_block": "000000...",
  "mrkl_root": "66...",
  "time": 1571443461,
  "bits": 387294044,
  "next_block": ["000000..."],
  "fee": 3764047,
  "nonce": 1066642855,
  "n_tx": 1925,
  "size": 870371,
    ...
  "height": 600000,
  "weight": 2848472,
  "tx": [
    {
      "hash": "93955...",
      "ver": 1,
      "size": 200,
      "fee": 0,
      "relayed_by": "0.0.0.0",
        ...
      "time": 1571443461,
      "block_index": 600000,
      "block_height": 600000,
      "inputs": [ ... ],
      "out": [ ... ]
    },
    { ... },    //More txs in the blocks.
  ]
}
```

C.2 Data Storage

Listing C.2: Transaction info.txt file

```
{"__fileinfo__":
  [
    {"filename": "data/dataset/Feb-21/transactions/Dt0.csv",
      "start": 668405,
      "end": 668514,
      "start_ha": "0000000000000...",
      "end_ha": "0000000000000...",
      "start_time": 1612054039,
      "end_time": 1612116675},
    {"filename": ... }
  ]
}
```

C.3 Data Manipulation

Listing C.3: Calculate transaction fee

```
def transaction_fee(jsontx):
  """ calculates transaction fee based on UTXO model.
  :param jsontx: transaction in json format
  :return: int, transaction fee in satoshi """
  if 'fee' in jsontx:
    fee = jsontx['fee']
  else:
    inputs = 0
    outputs = 0
    for inp in jsontx['inputs']:
      if 'prev_out' in inp:
        inputs += inp['prev_out']['value']
    for out in jsontx['out']:
      outputs += out['value']
    fee = inputs - outputs
  if fee < 0:
    fee = scraper.TransactionPage(jsontx['hash']).
        get_transaction_fee()
  return fee
```

Paper I

Tedeschi, E., Johansen, H.D. & Johansen, D. (2018)

Trading Network Performance for Cash in the Bitcoin Blockchain

CLOSER 2018, the 8th International Conference on Cloud Computing and Services Science, 643-650.

Trading Network Performance for Cash in the Bitcoin Blockchain

Enrico Tedeschi, Håvard D. Johansen and Dag Johansen
UIT The Arctic University of Norway, Tromsø, Norway

Keywords: Bitcoin, Blockchain as a Service, Longitudinal Study, Performance, Transaction Latency.

Abstract: Public blockchains have emerged as a plausible cloud-like substrate for applications that require resilient communication. However, sending messages over existing public blockchains can be cumbersome and costly as miners require payment to establish consensus on the sequence of messages. In this paper we analyze the network performance of the Bitcoin public ledger when used as a massaging substrate. We present several real-world observations on its characteristics, transaction visibility, and fees paid to miners; and we propose two models for fee-cost estimation. We find that applications to some extent can improve messaging latency by paying transaction fees. We also suggest that spendings should be kept below 300 Satoshi per byte.

1 INTRODUCTION

The Blockchain technology used by popular crypto currencies like Bitcoin[1] and Ethereum, are essentially Peer-to-Peer (P2P) broadcast oriented Group Communication Systems (GCSs) (Cheriton and Zwaenepoel, 1985), where all members see all messages, and in the same order. With a built-in fee system that enables operators to make money for storing and forwarding messages, several decentralized P2P blockchain systems have emerged, providing a common ground for mutually distrusting entities to communicate. Due to their promise of highly resiliency, *blockchain-as-a-service* is currently being touted as a promising permissionless cloud-like building block for critical services in, for instance, the finance and health domains.

Unlike traditional multicast oriented GCSs like Horus (van Renesse et al., 1996) and Totem (Moser et al., 1996), blockchains have the unique property that all broadcast messages are kept and made available to all participants, potentially for the system's lifetime. For blockchains, consensus among participants on the total ordering of messages, and hence also consensus on the resulting data-structure or ledger, is achieved through computational puzzles that randomly grant members a time-limited exclusive right to dictate the next batch of messages to be put on the channel.

As with other permissionless systems such as SecureRing (Kihlstrom et al., 1998) and Fireflies (Johansen et al., 2015) that are designed to be highly resilient to intrusions and attacks, providing reliable service by masking Byzantine failures limits scalability. Because blockchains are designed to retain all messages, they are particularly vulnerable to denial-of-service through simple flooding attacks. If an attacker can send messages at an unbounded rate, he can quickly swamp the storage and network capacity of the service. Even benign usage might prove problematic. For instance, if Bitcoin would have the same transaction rate as a VISA circuit, with between 2000 to 56 000 transactions/sec (Croman et al., 2016), its blockchain structure would grow about 1 MB per 3 seconds.

To throttle its blockchain growth rate, Bitcoin adjusts the difficulty of its cryptographic puzzles to match the aggregate mining capacity of the network, towards a target average block creation time of 10 min per block. In combination with its current block size limit of 1 MB, Bitcoin's transfer capacity is roughly a meager 1.667 kBps, or approximately three transactions per second. This capacity is shared between all concurrent clients. Indeed, scalability and network performances are urgent concerns in existing Blockchain-based systems.

This paper presents key observations from our ongoing longitudinal study on the performance of the Bitcoin blockchain. We provide detailed insights and analysis on several important characteristics of Bitcoin, including paid fees and the size of blocks, and show how the rewards to miners have changed over time from a more recent view point compared to earlier studies (Möser and Böhme, 2015; Rizun, 2015).

[1]The Bitcoin currency is denoted BTC or ₿.

Tedeschi, E., Johansen, H. and Johansen, D.
Trading Network Performance for Cash in the Bitcoin Blockchain.
DOI: 10.5220/0006805906430650
In *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018)*, pages 643-650
ISBN: 978-989-758-295-0

Furthermore, we analyze the correlation between the fee paid from a transaction and its *latency*, or the time it takes to become visible in the whole network. We propose two different models to describe how applications should spend money to improve network performance. Although the studies presented in this paper are restricted to the Bitcoin system, we conjecture that our observations are transferable to similar P2P systems that rely on computational expensive proof-of-work for consensus and fee-based incentives.

2 BACKGROUND

This paper considers blockchains as a communication substrate, where a set of *client* or *application* processes communicate by sending and receiving messages. In blockchain systems, a message is often referred to as a transaction, a notation we also adapt in this paper. The blockchain substrate handles all transactions in batches, known as *blocks*. Each block B can be in one of two states: *proposed* or *accepted*, and might contain zero or more transactions from zero or more clients. The system in *permissionless* in that there exist no central authority that coordinate or regulate participation or usage.

A blockchain has one or more *miner* processes that act as the ingress points for transactions submitted by the clients. The set of all miners collaborate to decide on which transactions to admit and their ordering. Every miner has a *mempool* containing the new and unapproved transactions. Applications are free to submit transactions to any miner's mempool, and miners are free to choose which transactions to include in their blocks. Most blockchain systems, including Bitcoin, enforce a strict upper bound Q on the block size, which also limits the number of transactions each block can contain.

The blockchain data structure, often referred to to as a ledger, is maintained by a P2P overlay network where members cooperate to verify and distribute blocks such that each member process has a full replica of the data structure in local storage. The integrity of the data structure is dependent on consensus among the set of correct member processes on which blocks are in the blockchain and their total order. For this, existing blockchain systems, like Bitcoin, use the *Nakamoto consensus protocol*. This protocol relies on members solving computationally complex cryptographic puzzles as proof-of-work for admitting new blocks, commonly referred to as *mining*. Once a miner has solved a puzzle, it broadcasts the block along with a solution to the puzzle so that all other nodes can check its correctness. The block is then tentatively recorded in the blockchain.

In the early days of Bitcoin, it was possible to mine productively with commodity desktop or laptop computers. Nowadays, successful miners use highly specialized hardware called Application Specific Integrated Circuitss (ASICs) (Taylor, 2017), which typically offer up to 100 times improved performance over commodity CPUs and GPUs.

The cost associated with mining was defined by Rizun (2015) to be:

$$\langle C \rangle = \eta h \mathcal{T} \tag{1}$$

Here η is the cost per generated hash, h is the miner's individual hash rate and $\mathcal{T}$ is the block creation time. Hence, the block creation time is directly proportional to the hashing cost. The underlying assumption of proof-of-work consensus is that the high η value, due to the energy cost of mining, will discourage and limit malicious behavior. At the same time, benign participation is promoted by means of incentives: the nodes receive payment as a reward for solving puzzles.

As miners compete to solve the latest puzzle, it may happen that two or more nodes succeed at approximately the same time. This may result in different blocks with different transactions being proposed for the blockchain. Thus, proposing further blocks may result in different chains, often referred to as a fork. To break such ties, Bitcoin adopts the simple rule that the longest consecutive chain of blocks wins. Therefore, a tentative block of transactions may be discarded, which is known as *orphaning*. The recording of a block is only considered permanently accepted after five additional blocks have been added, approximately after 1 hour. Thus, eventually the nodes reach consensus on the ordering of all the blocks on the blockchain.

Given a transaction t from some client c_1 to some other client c_2, we have the following:

Definition 1. *The* commit latency *of a transaction t is the time from when c_1 first proposes a transaction t to a mining pool, to when some block B including t is first mined and permanently accepted.*

Note that the total *end-to-end latency* of t also includes the time it takes for B to be delivered to c_2. However, as blockchain clients must pull the system for updates, the last-hop delivery time will depend to the application specific pull interval. For generality, this paper will therefore only consider the commit latency.

In existing blockchain systems, miners can freely choose which transactions to include when proposing a new block. A client must therefore negotiate with miners to have its transactions included. To entice miners, each transaction t can include a transaction fee t_f paid by the sender to be claimed by the miner whom first successfully include t in an accepted block. Due to the cost of mining, most miners are assumed to behave rationally (Aiyer et al., 2005): following the blockchain protocol, but such that their mining rewards are optimized. Hence, we conjecture that it is possible to use the transaction fee mechanism to improve messaging performance, which will be the focus of the remainder of this paper.

3 OBSERVATIONS

In this section we describe key observations and insights from our studies on the public Bitcoin ledger. These form the basis of our latency-fee models in Section 4. We start this section, however, by describing our method for collecting observational data.

3.1 Methodology

There are several methods that can be used to study Bitcoin. *Real-time analysis* requires setting up and operating one or more full Bitcoin nodes that connect to the P2P network and record traffic. The advantages of this approach is that some of the inner-node communication can sampled, including block propagation time and orphaning rate. However, to obtain usable coverage, multiple geographically dispersed nodes must be set up and injected into the network. Each one must download and store the full ledger and participate in the forwarding of new blocks. At time of writing, the full ledger of data requires 250 GB of storage. This requires significant up-front hardware investments and might potentially disrupt some of the system's characteristics that are under study.

Another approach is to use the *Bitcoin Core application* (van der Laan et al., 2017), which downloads the full ledger into local storage, but without having to run the full P2P protocol. Retrieved data does, however, not include the block propagation time or information from miners, which we require for our studies. Downloading the full ledger can also take significant time (in our case it took four days), and requires significant available disk capacity.

In our studies, we instead adapted a similar methodology to the one used by Möser and Böhme (2015), collecting data from some of the many *online third-party APIs*, made available by various organizations that are already monitoring the Bitcoin system, including tradeblock.com and blockchain.info. Websites like coinbase.com also provide useful information about the money exchange price, and provide an API along with libraries, like *forex-python*,[2] which we used.

[2]https://pypi.python.org/pypi/forex-python

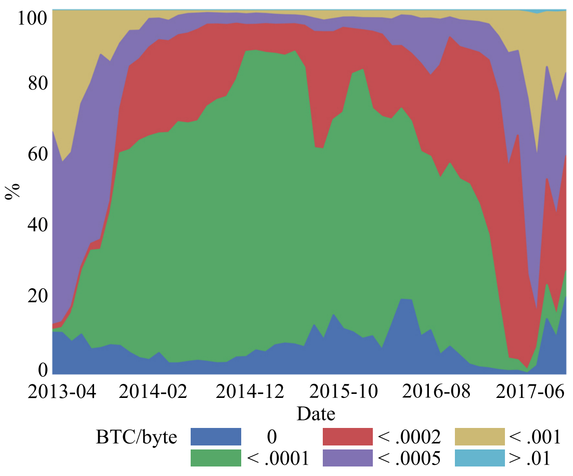


Figure 1: Observed transaction fee (t_f) distribution from 2013 to 2017.

Data was retrieved from these public APIs, and collected as JavaScript Object Notation (JSON) objects stored locally in Pandas data frames (Augspurger et al., 2012). Some data not available in these APIs directly, was instead scraped from the sites' HTML pages. The data was processed and visualized with *Matplotlib* (Hunter et al., 2017) and *Seaborn* (Waskom et al., 2016). This approach enabled us to analyze a considerable part of the blockchain with little up-front investment in computational resources.

For this paper, we elected to study data in the range from April 2013 to September 2017, sampling more than 120 million transactions and 100 000 blocks. Several studies have already been conducted on Bitcoin data before 2013 (Croman et al., 2016; Houy, 2014; Möser and Böhme, 2015; Rizun, 2015), and the popularity of the system before 2011 was low. Interpreting data outside our selected date range would probably not generate new insight. Table 1 names and summarizes the exact segments retrieved and used in this paper.

3.2 Transaction Fees

In this section, we present our observations on how the transaction fees t_f, paid by the clients to the miners, have changed over time. For each transaction t, The fee t_f is the difference between the sum of all input values t_{in} and the sum of every output values t_{ou}. If n is the number of inputs and m is the number of outputs, then we have:[3]

[3]The unit of Equation 2 is ₿.

Table 1: Bitcoin blockchain regions analyzed.

Name	**Start Date**	**End Date**	**Block Height Range**
2009	09-01-2009 03:54:25	08-03-2009 06:31:22	1 – 6710
2011	01-04-2011 19:58:59	09-05-2011 12:58:13	116 167 – 122 876
2013	21-04-2013 03:03:51	01-06-2013 12:37:51	232 333 – 239 042
2015	21-03-2015 04:01:39	06-05-2015 14:37:12	348 499 – 355 208
2017	15-12-2016 18:17:45	19-06-2017 12:04:23	443 599 – 471 951

$$t_f = \sum_{i=1}^{n} t_{in_i} - \sum_{i=1}^{m} t_{ou_i} \tag{2}$$

Figure 1 plots the calculated values for t_f in blocks from 2013 to late 2017, categorized into six payment classes ranging from 0 to 0.01 ฿. As we can see in the figure, in the first half of 2016 fees between 0 and 0.0001 ฿ almost disappeared. Considering that the Bitcoin price raised from less than 1000 USD to more than 5000 USD between mid 2016 and second half of 2017, this is indicative of a huge increment in the monetary value collected by miners. If we compare the Bitcoin price and the fee paid in USD, we see a substantial co-movement, which indicates that ฿ is the dominant unit to consider when deciding about what fee to offer.

Because the number of bytes per transaction can vary in all major blockchain systems today, including Bitcoin, an interesting metric to study is how many ฿ per bytes a transaction t has to offer in payment to miners. This metric is known as the *fee density* t_ρ (Rizun, 2015). For some transaction t, with associated fee t_f and having a payload of t_q bytes, the fee density is defined as:

$$t_\rho = \frac{t_f}{t_q}. \tag{3}$$

Figure 2 plots the observed fee density, similarly to the fee plot in Figure 1. The observed average transaction size t_q is 500 B. At the end of 2017, we see some transactions offering less than 0.0001 ฿ in payment. We observe almost no transactions with fee density $t_\rho = 0$. This indicates that density has become an important metric for miners when deciding whether or not to include a transaction in their next block.

3.3 Transaction Latency

As shown above, a blockchain-based application may attempt to offer various mining fees to improve the commit latency of its messages. In the following we will investigate to what extent we are able to do so in practice.

The experienced commit latency t_l of most transactions can relatively easily be observed in the avaiable data. All transaction are timestamped when

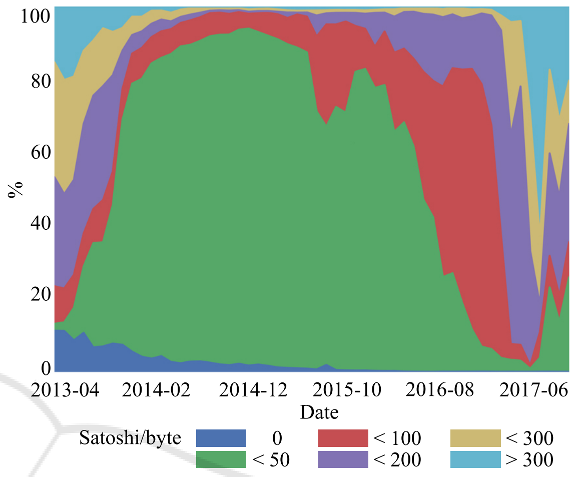


Figure 2: Observed fee density (ρ) distribution from 2013 to 2017.

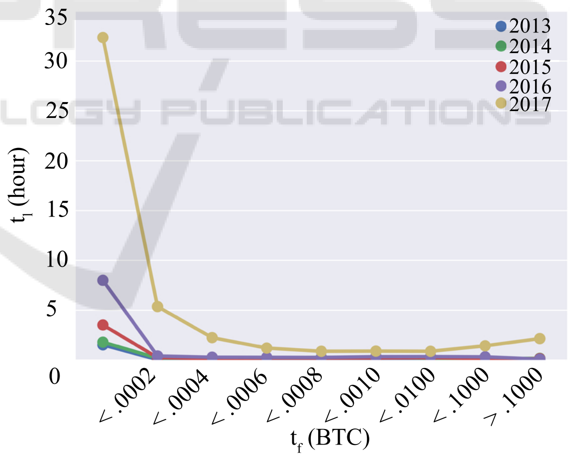


Figure 3: Relation between t_l and t_f grouped by year.

added to a mining pool, and blocks are similarly timestamped when mined.

Let B_{epoch} be the timestamp of some block B containing the transaction t, and let t_{epoch} be the timestamp of when that transaction was first added. Then the commit latency of observed transactions t can simply be calculated from:

$$t_l = B_{epoch} - t_{epoch} \tag{4}$$

Figure 3 plots the observed transaction latency (in hours) against the five fee density classes for each

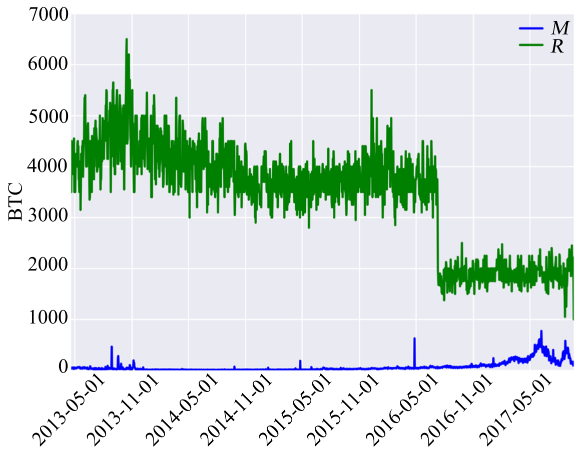


Figure 4: Daily miners revenue divided in block reward R and the sum of transaction fees M.

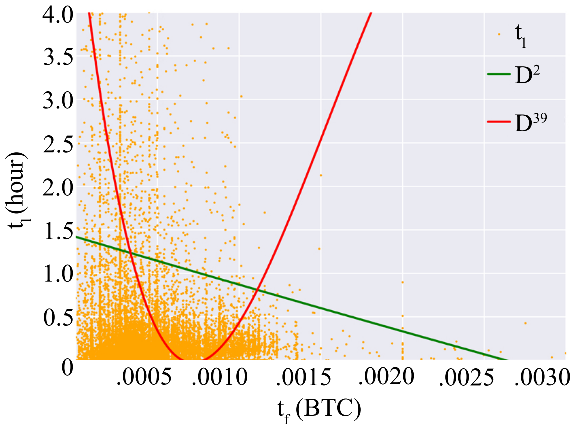


Figure 5: Fee-latency interpolation F with a 2 and 39 degrees polynomials for Bitcoin transactions analyzed in 2017.

year included in our study. In all cases, we observe that paying transaction fees gave a significant boost to latency, and that in 2017 doing so became more important than previously. We also observe the existence of a threshold from where increasing payment has little effect. For the years 2013–2016, the threshold was around 0.0002 ₿, while in 2017 it increased to 0.0006 ₿.

In addition to the total mining reward (M) from all transaction fees in a block, miners also receive a block reward (R) for each mined block. The block reward has historically been an important incentive for miners to produce blocks, regardless of the transaction fees offered by clients. However, the reward mechanism in Bitcoin is designed to halve the size of R every 210 000 blocks. As can be seen in Figure 4, in the period from 2009 to 2013 miners had little considerations for the transaction fees, and relied more on the reward R. In mid 2016, when the block reward was last halved, we observe a clear shift in how the miners profit from their efforts, becoming more influenced by the transaction fees.

This observation is not surprising as we expect most miners to be rational (Aiyer et al., 2005), trying to optimize their profit. With less block reward, rational miners will need to prioritize transactions with a higher fee density over lower ones until the max block size is reached. If the total fee of the included transactions is less than the expected monetary cost of mining the block $\langle C \rangle$, the miner may even opt to wait until a higher density transaction arrives. This can significantly increase experienced commit latency and jitter. We expect M to overcome R by 2020 when the reward is halved again to 6.25 ₿. Hence, for applications that intend to use blockchain as a service for communication, there is a clear potential and need for clever usage of the transaction fee mechanism to optimize the commit latency and the monetary cost of sending messages.

4 MODELS

With data from the 2017 transactions, we generated two models that applications can use when deciding what transaction fees to offer.

4.1 Fee-latency

The first model F describes the expected latency of some transaction t given some transaction fee t_f. We compute two variants of F using polynomial regression: one of degree 2 (F^2) and one of degree 39 (F^{39}). The lower degree regression is used to show the general trend, while the higher degree one is used to show the utility threshold. The resulting regressions are shown in Figure 5. Measured Mean Absolute Error (MAE) was 1.755 for F^2 and 1.7476 for F^{39}. For some given transaction fee x, the function F^2 is given by

$$F^2(x) = 6248x^2 - 555.8x + 1.42 \qquad (5)$$

From the plot of F^2, we we see a clear linear trend that transactions offering higher fees experience lower commit latency, which is what we expected. From F^{39} in Figure 5, we also see a clear threshold at about 0.007 ₿ when the benefits of adding extra fee starts declining.

4.2 Fee Density-latency

As argued in Section 3.2, miners nowadays tend to select transactions based on fee density, rather than solely on the fee amount. We therefore generate a

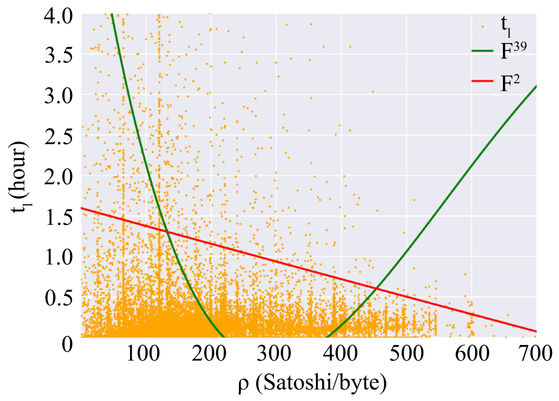


Figure 6: Fee density vs latency interpolation with a 2 and 39 degrees polynomial for Bitcoin transactions analyzed in 2017.

model D that provides the expected latency t_l as a function of the fee density $t_\rho = t_f/t_q$.

Similar to F, we compute two variants of D using polynomial regression: one of degree 2 (D^2) and one of degree 39 (D^{39}). The resulting regressions are shown if Figure 6. Measured MAE was 1.749 for D^2 and 1.837 for D^{39}.

Similarly to F, we also observe in D a clear trend that offering higher fee-density transactions improves commit latency. The threshold for diminishing returns is about 300 Satoshi per byte.[4] Paying a higher fee per byte gives little improvements The calculated polynomial for D^2 is given by:

$$D^2(x) = \frac{5.416}{10^8}x^2 - \frac{2.215}{10^3}x + 1.598 \tag{6}$$

5 DISCUSSIONS

When Bitcoin was first released, one of its strengths was its decentralized P2P architecture. Miners could join the network all over the world, and more miners meant a more robust service.

Over the years though, the opportunities to exchange block and transaction rewards into hard cash enticed more and more people around the world to join the mining effort and make money. This increased the total hashing power of the system, but also increased the difficulty of the proof-of-work puzzle, as Bitcoin is designed to do. Eventually the puzzles became too difficult for most individual casual miners to solve, and people started teaming up into mining pools to share both computational power and profit.

Today only a few large mining pools remain, and the ability of the system to make progress has to a large extent been centralized. Still, the mechanisms controlling mining are governed by marked forces that remain to be exactly described. This may be a difficult task as most large mining pools withhold information about their number of miners, the hardware they use, their profit, their transaction selection criteria, etc. Observational studies, like the one described here, might be the only plausible method for understanding the mechanisms governing these systems.

Towards that end, Table 2 summarizes our findings by listing the effect of changing two key design parameters in Bitcoin: the block size Q and the block creation time $\mathcal{T}$. Decreasing Q might first appear as a good solution for the number of advantages it has. However, its few disadvantages are critical for both performance and scalability. We therefore deem that decreasing Q is ill judged. It is likely much easier to deal with the orphan rate amplification resulting from increasing Q.

For the block creation time $\mathcal{T}$, we have the opposite scenario. An increment in $\mathcal{T}$ will reduce performance as miners will make less profit and thus have less incentive to mine. The only advantage is a lower orphaning rate since blocks will have more time to propagate.

As we can observe from Table 2, the throughput γ increase when either the block size Q is raised or the creation time $\mathcal{T}$ is lowered. According to Croman et al. (2016), the block size should not exceed 4 MB given $\mathcal{T} = 10\,\text{min}$. A good compromise could be to increase the block size limit to 1.5 MB and lower the creation time to 8 min. In that way, the system have a potential throughput capacity of 3.20 kBps and 10 transactions per second.

We cannot find a clear and general relation between Q and t_f in our studies. They seem to relate only when the drastic change in the block size occurs. We do, however, find that from 2013 to 2017 the relation between t_f and t_l became more noticeable day-by-day, having almost an inverse proportionality in the latest data from 2017, as seen in Figure 3. In 2017, zero-fee transactions almost disappeared from the system. This is probably due to the incredibly high commit latency many clients were experiencing at that time, with zero-fee transactions taking up to an average of 33 h to get committed into blocks. At the same time, clients that paid only modest transaction fees, less than 0.0002 ฿, experienced commit latencies of only 5 h, while the ones that paid between 0.0008 ฿ and 0.0010 ฿ expected latencies of less than 1 h. Hence, applications that intend to use blockchains as a service for communication will benefit from having a dynamic fee-latency prediction model, like the one described here, to optimize performance.

[4] 1 Satoshi = 0.00000001 ฿.

Table 2: Scalability and performance tradeoffs when changing block size Q and block creation time $\mathcal{T}$.

	Higher ↑	**Lower ↓**
Q	$+$ improved scalability with more transactions accepted per day $+$ improved commit latency t_l $\pm$ lower fees (good for clients, bad for miners) $-$ orphan rate amplification $-$ increased centralization $-$ congestion concern solved with transaction eviction by miners $-$ no permanent effect	$+$ no transaction spam $+$ no 0-fee transactions $+$ less mining cost $+$ less propagation time $+$ less chance of orphaning $\pm$ higher fees (good for miners bad for clients) $-$ less throughput $-$ higher commit latency t_l
$\mathcal{T}$	$+$ lower orphaning rate $+$ no physical changed needed to support faster inner node communication $-$ lower throughput γ (unless Q is increased) $-$ less scalable (unless Q is increased) $-$ mining profit is confined	$+$ higher throughput γ $+$ system is more scalable $+$ increased mining profit $-$ require faster inner-node communication $-$ exponential increment of orphaning rate

6 CONCLUSION

The Bitcoin blockchain has undoubtedly emerged as a notable substrate and service for communication. The built in mechanism for gaining monetary value by doing useful work for others has clearly moved the underlying architecture from its initial P2P model, to a more centralized model resembling a public cloud-like service. Individual incentives for providing the service is no longer motivated by own needs for it, but rather motivated by the prosperity of earning money.

Although the service provided by the Bitcoin substrate is highly robust, it is also painstakingly slow. Commit latencies of transactions are often measured in hours. Applications that intend to use Bitcoin, or one of its derivatives, as a public cloud-like service for communication, can still improve their messaging performance by adjusting the offered transaction fee to the number of bytes sent. There are, however, clear limits to what can be achieved. For Bitcoin, spending more than 300 Satoshi per byte seems to be ineffective. As the incentives to mine new blocks shift focus from block fees to transaction fees in the years to come, we expect that new schemes for optimizing messaging performance needs to be developed.

ACKNOWLEDGMENT

This work was supported in part by the Norwegian Research Council project number 263248. We would like to thank the anonymous reviewers for their useful insights and comments.

REFERENCES

Aiyer, A. S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.-P., and Porth, C. (2005). Bar fault tolerance for cooperative services. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05, pages 45–58, New York, NY, USA. ACM.

Augspurger, T., Bartak, C., Cloud, P., Hayden, A., Hoyer, S., McKinney, W., Reback, J., She, C., Horikoshi, M., den Bossche, J. V., et al. (2012). Pandas: Python Data Analysis Library. software v0.21.0, Pandas community.

Cheriton, D. R. and Zwaenepoel, W. (1985). Distributed process groups in the V kernel. *ACM Transactions on Computer Systems*, 3(2):77–107.

Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Gün Sirer, E., Song, D., and Wattenhofer, R. (2016). On scaling decentralized Blockchains. In *Financial Cryptography and Data Security. FC 2016.*, volume 9604 of *LNCS*, pages 106–125, Berlin, Heidelberg. Springer Berlin Heidelberg.

Houy, N. (2014). The economics of Bitcoin transaction fees. Working Papers 1407, Groupe d'Analyse et de Théorie Economique (GATE), Université Lyon 2.

Hunter, J., Dale, D., Firing, E., Droettboom, M., et al. (2017). Matplotlib for data plotting. software v2.1.1.

Johansen, H. D., van Renesse, R., Vigfusson, Y., and Johansen, D. (2015). Fireflies: A secure and scalable membership and gossip service. *ACM Transactions on Computer Systems (TOCS)*, 33(2):5:1–5:32.

Kihlstrom, K. P., Moser, L. E., and Melliar-Smith, P. M. (1998). The SecureRing protocols for securing group communication. In *Proc. of the 31st Annual Hawaii International Conference on System Sciences*, pages 317–326. IEEE.

Moser, L. E., Melliar-Smith, P. M., Agarwal, D. A., Budhia, R. K., and Lingley-Papadopoulos, C. A. (1996). Totem: a fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63.

Möser, M. and Böhme, R. (2015). Trends, tips, tolls: A longitudinal study of Bitcoin transaction fees. In *Financial Cryptography and Data Security: FC 2015.*, number 8976 in LNCS, pages 19–33, Berlin, Heidelberg. Springer Berlin Heidelberg.

Rizun, P. R. (2015). A transaction fee market exists without a block size limit. Technical report.

Taylor, M. B. (2017). The evolution of bitcoin hardware. *IEEE Computer*, 50(9):58–66.

van der Laan, W. J., Wuille, P., Andresen, G., et al. (2017). Bitcoin client application. software v0.15.1.

van Renesse, R., Birman, K. P., and Maffeis, S. (1996). Horus: a flexible group communication system. *Communications of the ACM*, 39(4):76–83.

Waskom, M., Botvinnik, O., drewokane, Hobson, P., David, Halchenko, Y., Lukauskas, S., Cole, J. B., Warmenhoven, J., de Ruiter, J., Hoyer, S., Vanderplas, J., Villalba, S., Kunter, G., Quintero, E., Martin, M., Miles, A., Meyer, K., Augspurger, T., Yarkoni, T., Bachant, P., Williams, M., Evans, C., Fitzgerald, C., Brian, Wehner, D., Hitz, G., Ziegler, E., Qalieh, A., and Lee, A. (2016). Seaborn. Software v0.7.1.

Paper II

Tedeschi, E., Nordmo, T.S., Johansen, D. & Johansen, H.D. (2019)

Predicting Transaction Latency with Deep Learning in Proof-of-Work Blockchains

2019 IEEE International Conference on Big Data (Big Data), 4223-4231.

Predicting Transaction Latency with Deep Learning in Proof-of-Work Blockchains

Enrico Tedeschi*, Tor-Arne S. Nordmo*, Dag Johansen* and Håvard D. Johansen*
* UIT: The Arctic University of Norway
Emails: (enrico.tedeschi, tor-arne.s.nordmo, dag.johansen, havard.johansen)@uit.no

Abstract—**Proof-of-work based cryptocurrencies, like Bitcoin, have a fee market where transactions are included in the blockchain according to a first-price auction for block space. Many attempts have been made to adjust and predict the fee volatility, but even well-formed transactions sometimes experience delays and evictions unless an enormous fee is paid. In this paper, we present a novel machine-learning model, solving a binary classification problem, that can predict transaction fee volatility in the Bitcoin network so that users can optimize their fees expenses and the approval time for their transactions. The model's output will give a confidence score whether a new incoming transaction will be included in the next mined block. The model is trained on data from a longitudinal study of the Bitcoin blockchain, containing more than 10 million transactions. New features that we generate include information on how many bytes were already occupied by other transactions in the mempool, assuming they are ordered by fee density in each mining pool. The collected dataset allows to generate a model for transaction inclusion pattern prediction in the Bitcoin network, hence telling whether a transaction is well formed or not, according to the previous transactions analyzed. With this, we obtain a prediction score for up to 86%.**

Index Terms—**Bitcoin, blockchain, longitudinal study, performance, transaction latency, machine learning, neural networks.**

I. Introduction

Bitcoin was intended to provide its users with a low-cost payment scheme, with transaction fees close or equal to zero [1]. In a tragedy of the commons, the cryptocurrency's rising popularity made the inherent throughput limitations of the underlying Proof-of-Work (PoW) scheme for establishing consensus a key scalability bottleneck [2, 3]. The high cost of mining has led to an increased usage of transaction fees as a means for miners to make a profit. Users are experiencing this as delays and performance issues, and the cost of transactions have become nonzero and volatile. Transactions offering low fees are now typically experiencing higher transaction latency and, after 2016, zero-fee transactions were evicted from most of Bitcoin's miners [4]. For Bitcoin, transaction fees are intended to replace miner's minting reward in the long run [5].

As a consequence, a transaction fee market has emerged where transactions are included according to a first-price auction for block space. All bidders (users) submit sealed bids (transaction fee) [6] and the highest bids will have the highest chances of being included in the next block. Although new consensus techniques have been proposed with an indirect scope to also improve scalability, such as Proof-of-Stake (PoS) [7, 8] or Proof-of-Storage (PoSt) [9], Bitcoin cannot easily accommodate for such changes and its users must expect to bid and pay transaction fees to have their transactions included in the blockchain.

This first-price auction for Bitcoin block space is ultimately bad for most users, and more so as the number of daily transactions increases [4]. However, the alternatives of deciding on a common static fee is not possible [5, 10] and providing instant confirmation remains a key challenge [11]. Users are instead left to chose an appropriate payment dynamically when submitting their transactions with no exact formula to optimize expenditure or to control the time it takes for a transaction to be confirmed.

Current transaction fee estimators, such as the one implemented by Bitcoin Core, primarily use historical fee data. These estimators are susceptible to manipulation since miners can arbitrary add transactions to themselves with a high carried fees. Applications following these estimators observe that the fee to be paid is higher than what actually is needed. Other fee estimation algorithms interact with each other in unpredictable ways, which can lead to oscillating fee estimates. Many users end up using their own estimator based on their intuition [6]. This turns out to be a difficult task in practice and in most cases becomes expensive for the users [12] as they often end up with estimators that have the notorious problem of aggregate overpaying. In 2017, poorly designed fee estimators contributed to driving up average Bitcoin fees to over $20 per transaction [4, 6].

In this paper, we propose a transaction inclusion model based on data of historic blocks. Our model is constructed using deep-learning methods such as Neural Networks (NNs) [13], Deep Neural Networks (DNNs), and Residual Neural Networks (ResNets). The aim is to predict whether a certain transaction has a good chance of being included immediately ($\sim$ 10 min) in the next block. Note that we want to provide information on transaction inclusion and not the actual approval time ($\sim$ 6 blocks). This because our assumption is that if a transaction is included in a block, it will be approved regardless. We assume that because, after the inclusion, the waiting time for each additional confirmation is completely independent of the transaction fee, since the transaction has already been included in the blockchain. We develop our model by using data collected daily from the Bitcoin blockchain. We collect and gather longitudinal data, batch them, and for each one we create a prediction model which will be used for the subsequent batch.

II. Background

Due to the cost of mining blocks, most miners are assumed to behave rationally [14], but with individualized policies for inclusion that maximize own profit. These inclusion policies are not publicly known, and they might differ from one miner to the next. We conjecture that miners will depend more on users' fee to keep the network and mining alive, and that they are not only relying on transaction fees to include new transactions. From that, we argue that with the right selection of features, we can generate a model which can predict whether a transaction will be included in the next block. This idea relies on *fee rates* or *fee density* (ρ), and the limitation that miners have with respect to the block size Q, considering the *offset* (δ) of the block space already occupied by previous not-yet-approved transactions.

A. Models

According to the work of Rizun (2015) [2], each mined block has an expected profit $\langle\Pi\rangle$, of

$$\langle\Pi\rangle = \langle V\rangle - \langle C\rangle \tag{1}$$

where $\langle V\rangle$ is the expected revenue for mining and having your block approved in the network, and $\langle C\rangle$ is the expected hashing cost.

If we ignore factors that are not directly limited by the network, such as the monetary hashing cost $\langle C\rangle$, which depend on the individual hashing rate; the block creation time, which is normally distributed with a fixed known mean; and if we also ignore the probability of block orphaning $\mathbb{P}_{orphan}$ due to block propagation delays, which is shown to be mostly dependent on the block size [15], the total expected revenue can be expressed as

$$\langle V\rangle = (R + M)\frac{h}{H} \tag{2}$$

Here h is the individual hashing rate, H is the total hashing power of the network, R is the reward for mining a block, for Bitcoin currently at 12.5 ₿. M is the sum of all the transaction fees t_f in a block including N transactions and is formalized as:

$$M = \sum_{i=1}^{N} t_f^{(i)} \tag{3}$$

Based on the Bitcoin Unspent Transaction Outputs (UTXO) model, if n is the number of transaction inputs in a certain block, and m is the number of transaction outputs, a transaction fee t_f is calculated

$$t_f = \sum_{i=1}^{n} t_{in}^{(i)} - \sum_{j=1}^{m} t_{ou}^{(j)} \tag{4}$$

We can then state, that the expected profit $\langle\Pi\rangle$ gets lower when H increases. Furthermore, R is halved every 210 thousands blocks, making transaction fees, M, the only source of profit for miners.

Permissionless blockchains using PoW are hardly efficient in energy usage, causing miners to be more dependent on t_f as the network scales. If the number of miners scales, then the difficulty raises in order to keep up with the block creation time, having then more energy consumption, hence the fees are higher in order to pay back the miners. On the other hand, if the number of daily transactions scales, there is more competition to be included immediately in the next block, and a rational miner would prioritize higher fee transactions. Increasing the hashing power does not add any better performance related to transaction throughput, while it only increases the overall energy consumption. This is due to the network difficulty d (Equation 5), which allows to mine new blocks, and which is normalized according to $\mathcal{T}' \simeq 2016\,\mathcal{T}$, where $\mathcal{T} = 10\,\text{min}$ is the fixed block creation time. It does not matter then how powerful the network is, and as long as the block size is fixed, it will be impossible to gain any performance. Difficulty d at time t is defined as

$$d_t = \begin{cases} 1, & \text{if } t = 0 \\ d_{t-1}\frac{2016\mathcal{T}}{\mathcal{T}'}, & \text{if } t > 0 \end{cases} \tag{5}$$

where $\mathcal{T}'$ represents the calculated actual time in minutes spent mining the previous 2016 blocks. Consequently, t_f is expected to grow with the network size, making a system based on permissionless PoW blockchain difficult to maintain without a substantial revenue for the miners (i.e., money put into the system by its users). Our work is fundamental to define how a well-formed transaction should look like, so that miners will have their fair revenue, but still such that users have acceptable fee-performance trade-off.

B. Prediction Methods

Machine-learning models have previously been shown effective at finding patterns in data streams in various fields and sectors, ranging from education [16, 17], business and marketing [18, 19], healthcare [20, 21, 22], financial services [23, 24], and transportation [25, 26]. Deep learning is part of a broader family of machine learning methods based on artificial NNs [27, 28, 29]. ResNet models are a variation of deep neural networks, and are implemented with double (or triple) layer skips that contain nonlinearities Rectified Linear Unit (ReLU) and batch normalization in between [30]. These skip connections help with gradient flow in deeper neural networks, which aims to reduce training and test error [31, 32]. The versatility and efficiency of these method has motivated us to use them in this work.

III. Methodology

To perform a prediction based on historic blocks, we need a local instance of the portion of the Bitcoin blockchain that needs to be analyzed. We fetch data from the Bitcoin blockchain and generate a dataset D, containing the features relevant for our model. We wanted to have our instance of the blockchain locally in order to save up space on disk while keeping only the information we needed for the evaluation. We collected data using a third-party APIs,[1] also gathering

[1]https://www.blockchain.com

information about money exchange price with libraries such as *forex-python*.[2]

With our dataset D, we build a prediction model for transaction inclusion pattern using machine-learning models. Having knowledge of PoW-based blockchains, we can then make assumptions on which features might be relevant for the model. From D we derive the feature set ($\mathbb{F}$) for the prediction, and then perform a supervised classification using different deep learning approaches to find the one that performs the best.

Not all the features we wanted were in the information available directly on the Bitcoin blockchain. Some features were generated by gathering information from multiple sources or from multiple features. One example of this typology is Δt_{ep}, which gives information for each transaction t, on how much time is elapsed from the previous block creation time to t's timestamp, t_{ep}. For t_{ep} we refer to the transaction time registered by the API's node, so it does not include information about the real transaction's timestamp, since the latter is not available in Bitcoin.

A. Data Acquisition and Feature Selection

Our feature set $\mathbb{F}$ is formed by both, fetched (Φ) and derived ($\mathbb{D}$) features,

$$\mathbb{F} = \Phi \cup \mathbb{D},$$

where

$$\Phi = \{t_q,\, t_{in},\, t_{ou},\, B_{mi}\},$$

and

$$\mathbb{D} = \{t_f,\, \rho,\, t_{\%},\, \Delta t_{ep},\, \delta\}.$$

For our model, we select the features that have most impact on transaction (t) latency. These are as follows:

t_q Transaction size, in bytes. The block size (Q) restrains the number of transactions that are included in a certain block B, limiting the throughput (γ) and indirectly also the revenue (M). As long as $Q = 1\,\text{MB}$, t_q will always play a role in transaction selection by miners.

t_f The transaction fee directly affects the revenue M of miners. Rational entities then would choose transactions with higher fees when the systems scales.

ρ The fee density is a derived features using the previous two. It will enhance the correlation and the importance of the transaction size and the fee. We also assume that a rational miner is ordering the incoming unapproved transactions by fee density.

$t_{\%}$ The percentage paid in fee matters since it is important to contextualize the fee with the total transaction's amount. If this feature would not matter, it will be impossible to execute small transactions since they will be surmounted by big carrying-fee ones.

Δt_{ep} Waiting time for a transaction t. This is the time elapsed from the transaction timestamp (t_{ep}), until the latest B_{ep}. The time elapsed up to the latest block is relevant since miners can prioritize older transactions over ρ. By intuition, we assume that a transaction can not wait forever for approval if it has paid a fair amount of t_f. This will also avoid that well formed transactions will not lose their space in the block just because some newer transaction with higher fee came right before a newly mined block.

δ Represents the offset in bytes. It defines for each transaction, the amount of bytes already occupied by better unapproved transactions in terms of ρ, in an hypothetical next block. The offset is relevant to give each transaction a place in the future block, assuming that miners are rational and are ordering the incoming transactions by ρ, then they are limited by the block size Q. Greater is the offset fewer are the chances that a transaction is included in the next block.

The features in Φ are directly available on the blockchain. Those in $\mathbb{D}$ are derived. For instance, t_f is calculated as showed in Equation 4, while the fee density ρ of a transaction t is obtained from the following equation:

$$\rho_t = \frac{t_f}{t_q}. \tag{6}$$

Other features were more elaborate and difficult to obtain, like Δt_{ep} or δ. During model training we need to define new concepts in order to contextualize every single transaction analyzed to the moment of their inception. The feature Δt_{ep} is calculated as showed in (7), where $Bs_t^{(i)}$, represents successor/predecessor of a transaction t. The successor is the first block epoch (B_{ep}) mined after t's inception time (t_{ep}) if $i = 0$, while the predecessor is the first previous B_{ep} of t_{ep} if $i = -1$.

$$\begin{aligned} \Delta t_{ep} &= t_{ep} - Bs_t^{(-1)}, \\ \text{where} \quad Bs_t^{(-1)} &\le t_{ep} < Bs_t^{(0)}. \end{aligned} \tag{7}$$

To explain the offset δ, we define the set of transactions S, showed in (8). Then each offset δ_S is a growing number, starting from 0, in a particular set S, where the number of sets goes from 0 to $\max(B_{he})$[3]. In the set S are contained all the transactions whose their t_{ep} is included in a certain timespan, ΔBs_t. We order these transactions in S by their fee density, ρ, and define the offset, δ, as showed in (9) and (10).

$$\begin{aligned} S &= (t^{(0)},\, t^{(1)},\, \ldots,\, t^{(n)}), \\ \text{where} \quad & \rho^{(0)} < \rho^{(1)} < \cdots < \rho^{(n)}, \\ \text{and} \quad & Bs_{tj}^{(-1)} \le t_{ep}^{(j)} < Bs_{tj}^{(0)}, \quad \text{for}\, j = 1, 2, \ldots, n \end{aligned} \tag{8}$$

In Equation 8, for readability, $t^{(j)} \equiv tj$, and $B_{tj}^{(i)}$ represents the successor or predecessor for transaction $t^{(j)}$. Equation 9 represents the offset for a transaction $t^{(n)}$ included in a set S.

$$\delta_S^{(n)} = \begin{cases} t_q^{(j)}, & \text{if } j = 0 \\ \delta_S^{(j-1)} + t_q^{(j)}, & \text{for } j = 1 \text{ to } n \end{cases} \tag{9}$$

[2]https://pypi.python.org/pypi/forex-python

[3]B_{he} represents the block height.

We can then simplify (9) as showed in (10).

$$\delta_S^{(n)} = \sum_{j=0}^{n} t_q^{(j)} \qquad \forall\ S_k, \qquad (10)$$
$$\text{where} \quad 0 \leq k \leq max(B_{he}).$$

B. Prediction Model Hyperparameters and Parameters

Two different models, solving a binary classification problem, are generated and evaluated: DNN and ResNet. The latter, contains skip connections between more distant layers. These skip connections help with gradient flow in deeper neural networks, which aims to reduce training and test error [31, 32]. The output of both models represent the probability for a transaction to be immediately included or not in the next block. The model's output is an array represented with $\boldsymbol{\theta}$, where for each transaction t, $\boldsymbol{\theta}^{(t)}$ is:

$$\boldsymbol{\theta}^{(t)} = [\,P_t(\upsilon_0),\, P_t(\upsilon_1)\,] \qquad (11)$$

and $P_t(\upsilon_i)$ indicates the probability, or confidence, for a transaction t to be labeled in class υ_i. With υ_i we refer to the class in which a transaction t belongs. Our task is a binary classification problem, hence $i \in \{0, 1\}$ and the class indicates whether a transaction t is well formed or not, in order to be approved in the next block. The class υ_1 contains all the transactions that according to the model will be included in the next block, while υ_0 represents the class for transactions which are not good enough to be included in the next block.

An example of DNN representing one of our models is showed in Fig. 1. We changed the number of hidden layers during test, but the activation function used was a ReLU for each node in the network, except for the output layer, where we used a Normalized Exponential Function (or softmax). The weights were initialized with He normalization, which takes into account ReLU and it makes it easier for deep models to converge [33]. The tested models were implemented using Keras[4] with Tensorflow backend.[5]

Parameters that cannot be estimated from data, known as hyperparameters, are set manually by trial and error. This includes the number of hidden layers, number of skip connections, the batch size, and the epoch for each model. The batch size controls the granularity or precision of gradient descent, meaning that the optimization function, so the model internal parameters, are optimized every *batch size* of tuples. The epoch instead, represents the number of times that the learning algorithm will work through the entire training dataset, ideally, getting closer to the optimal solution at every iteration. The model's hyperparameters are configured to optimize the model performance and accuracy.

Our models, using the set of features $\mathbb{F}^6$ (12), aim to take into account how trends and policies for transaction inclusion and eviction change over time. Therefore, if our assumptions on which are the most relevant factors influencing transaction inclusion are correct, we can build accurate models despite the system's dynamicity.

[4]https://keras.io/

[5]https://www.tensorflow.org/api_docs

IV. OBSERVATIONS

The purpose of our observations is to make sure that our models can give an accurate prediction on what the transaction inclusion pattern will be on newer transactions. This will help users in making decisions regarding the transaction fee to pay, knowing how likely that transaction will be included in the blockchain. The metrics used for the test include accuracy, precision, and recall. We perform our tests on transactions registered on the Bitcoin blockchain from 6^{th} April 2019 until 23^{rd} May 2019. We take into account almost 20 million of transactions over ~ 7 thousand blocks.

For testing, DNN and ResNet models are used. Tests also include different set of features, which in our case are $\mathbb{F}^5$ and $\mathbb{F}^6$, defined as:

$$\begin{aligned} &\mathbb{F}^5 = \{t_q,\, t_f,\, \rho,\, t_{\%},\, \Delta t_{ep}\}, \\ &\mathbb{F}^6 = \mathbb{F}^5 \cup \{\delta\}. \end{aligned} \qquad (12)$$

Depending on the cardinality of the feature set, $|\mathbb{F}^c| = c$, our model names will identify the model's type and the feature set used, for instance, DNN6 identifies the deep neural network model having $\mathbb{F}^6$ as feature set.

Data from diverse type measurements can have different scale and the biggest can dominate the model's algorithm. Therefore, during pre-processing phase we normalize our training and test dataset ($\boldsymbol{X}$ and $\boldsymbol{Xte}$) to have an homogeneous dataset, based on the mean and variance of the features rather than single values (Algorithm 1). We also noticed that the number of transactions belonging to the two classes, υ_0 and υ_1, is always unbalanced, with an average score of 35% for class υ_0 and 65% for class υ_1. Because of that, we decided to test our models with both, balanced, and unbalanced but weighted classes. We refer to the balanced models with (*), for instance, DNN* or ResNet*.

Algorithm 1 Normalization of $\boldsymbol{X}$

```
1: procedure NORMALIZATION($\boldsymbol{X}$, $\boldsymbol{Xte}$)
2:     $\mu \leftarrow$ Mean($\boldsymbol{X}$)                      ▷ expected value
3:     $\sigma \leftarrow$ Std($\boldsymbol{X}$)                    ▷ standard deviation
4:     $\boldsymbol{X}_{norm} \leftarrow (\boldsymbol{X}-\mu)/\sigma$          ▷ normalizing training set
5:     $\boldsymbol{Xte}_{norm} \leftarrow (\boldsymbol{Xte}-\mu)/\sigma$      ▷ normalizing testing set
6:     return $\boldsymbol{X}_{norm}, \boldsymbol{Xte}_{norm}$
```

For performance evaluations we use the holdout method, which belongs to cross validation [34, 35, 36] class. Holdout method results optimal when the dataset contains n elements, where $n \rightarrow \infty$. Plus, the holdout method has a lower computational overhead if compared with leave-one-out and k-fold cross validation methods, and it still keeps the training and test set independent, unlike happens in residual methods [35]. We then train a model with 85% of the total training set, we compute the metrics on the remaining 15% (validation set), where the model is adjusted over all training data, and then we compute the metrics on test set, which is a complete new set, with newer transactions.

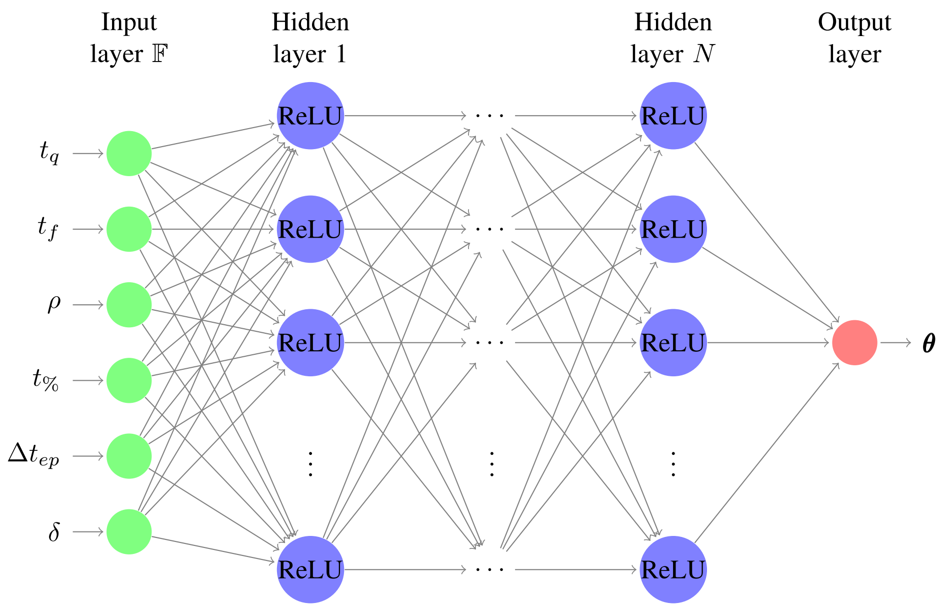


Fig. 1: Neural Network representing our DNN6 model. For each hidden layer a ReLU function is used, the number of hidden layers varies between different tests.

The metrics used to test our models, over the binary classification problem (υ_0, υ_1), are based on the confusion matrix ($\boldsymbol{A}$) defined as:

$$\boldsymbol{A} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}, \tag{13}$$

where a_{ij} is the number of elements which truly belong to i but were classified in j. The metrics obtained from $\boldsymbol{A}$, are represented in Equation 14 and defined as follows:

R_i Recall for class υ_i. Tells the number of $t \in \upsilon_i$ which were correctly classified in class υ_i.

P_i Precision for class υ_i. Tells the number of data points classified in υ_i which actually belongs to υ_i.

A Overall accuracy. How many elements were rightly classified.

$$\begin{aligned} R_i &= \frac{a_{ii}}{\sum_{j=0}^{1} a_{ij}}, \quad P_i = \frac{a_{ii}}{\sum_{j=0}^{1} a_{ji}}, \\ A &= \frac{\sum_{i=0}^{1} a_{ii}}{\sum_{i=0}^{1}\sum_{j=0}^{1} a_{ij}} \end{aligned} \tag{14}$$

For clarity, we will represents these metrics with their value in percentage score in relation with the total transactions analyzed.

We train our models by batching the same time frame into twenty or five days of transactions, tested over the next fifteen or five days. Our purpose other than having an high score in accuracy, is (1) to test different feature sets ($\mathbb{F}^5$ and $\mathbb{F}^6$), (2) enhance differences using both models (DNN and ResNet), and (3) highlight how the class balancing can affect our results. The next subsections describes two different analysis and it underlines model's performance following our metrics and points (1) (2) (3).

A. *Twenty Days Analysis*

The first analysis is done over a training time period of twenty days and a test time of fifteen days. Data are trained considering transactions from 6^{th} of April 2019 to 26^{th} of April 2019, and tested for the next following days until 11^{th} May 2019. In Fig. 2a the scores of our metrics for two different models, ResNet and DNN, are represented respectively with dashed and continuous lines, while two different feature set, $\mathbb{F}^5$ and $\mathbb{F}^6$, are listed with light blue and red colors. From Fig. 2a we can see that the two different models have almost an identical trend if the same input space is considered.

Analyzing point (1), when the input space does not include δ ($\mathbb{F}^5$), both models have an accuracy score which is $\simeq 14\%$ lower than $\mathbb{F}^6$ models. We see that ResNet5* has more difficulties in classifying transactions which the real class is υ_1, having $R_1 \simeq 65\%$, while it performs better in classifying transactions when their real label is υ_0, with $R_0 \simeq 75\%$. Without δ the model has no information on the block size limitation, so the reason of overpopulating υ_0 might be that it classifies in υ_1 only extremely profitable transactions, ignoring the fact that is better to have lower fee transactions rather than no transaction at all. By adding δ, having $\mathbb{F}^6$ as input space, accuracy is higher and the biggest improvement in the model is related to R_1, that goes from $\sim 65\%$ to $\sim 85\%$. This means that the model significantly improved in assigning transactions $t \in \upsilon_1$ to the their right class υ_1. Finally, looking at Fig. 2b, the increment from a two dimensional feature space, $\mathbb{F}^2 = \{t_q, t_f\}$, to a six dimensional one, $\mathbb{F}^6$, is significant in terms of accuracy. A big increment comes with Δt_{ep}, where both models go from an accuracy of 67% to 70%. However, the biggest increment comes for both models when the feature δ is added, with an increment of $\simeq 10\%$. The scores for both models are constant to $\sim 66\%$ with $\mathbb{F}^3 = \mathbb{F}^2 \cup \{\rho\}$ and $\mathbb{F}^4 = \mathbb{F}^3 \cup \{t_{\%}\}$.

Referring to point (2), we notice a small boost in accuracy if ResNet is used, the difference is less than 1%, but since there is no computational ovrehead in running ResNet instead of DNN, we preferred the first to the latter. In all the metrics, ResNet6* scores above 80%, with an overall accuracy of 83.32%, while DNN6* is less precise in assigning $t \in \upsilon_0$ to their right class υ_0, having a total accuracy of 83.13%.

In this analysis, we show the importance for both models to have knowledge of the block space Q, thanks to δ. The training set entropy with such new information is significantly higher.

Both models weakness is R_0, which means that it is harder to correctly classify transactions which should not be immediately included. However, the R_0 score for both models is $\sim 80\%$, which means that only less than 20% of transactions $t \in \upsilon_0$ are not correctly classified.

B. Five Days Analysis

In the five days analysis we run the same models over a batched dataset, for a total time frame of one month. Each batch identifies transactions occurred in five days, the training is performed for each batch, and the test is done over the following five days. The method is straightforward, if the training set is represented by $\boldsymbol{X}$, and the dataset of the labels as $\boldsymbol{Y}$, then we train and test each batch as showed in Algorithm 2. For each batch we build a model, then we run it with data from the next batch $\boldsymbol{X}_{i+1}$, obtaining in this way the predicted values $\hat{\boldsymbol{Y}}$ for the i^{th} batch, $\hat{\boldsymbol{Y}}_i$.

Algorithm 2 Training Batched Dataset $\boldsymbol{X}$

```
1: procedure TRAINBATCH($\boldsymbol{X}$, $\boldsymbol{Y}$ n_batches)
2:     for $0 \leq i < n\_batches$ do
3:         ResNet$_i$ ← Train($\boldsymbol{X}_i$, $\boldsymbol{Y}_i$)
4:         $\hat{\boldsymbol{Y}}_i \leftarrow$ ResNet$_i$($\boldsymbol{X}_{i+1}$)
```

If we observe Fig. 3, accuracy (A) varies from 80% til 86%, and both models have a similar trend. However, to focus on (3), we notice that ResNet6* ability to correctly classify positive samples is same as its ability to correctly classify negative samples, which makes ResNet6* a more balanced model than ResNet6. $P_{0(1)}$[6] for both models have a similar trend, which means that, independently from the class balancing, ResNet has good precision for classes, since most of the classified transactions in $\upsilon_{0(1)}$ actually belong to $\upsilon_{0(1)}$. Even if the precision is high for ResNet6, with scores above 80%, the recall $R_{0(1)}$ show that in ResNet6 the good $P_{0(1)}$ is caused by the significantly higher number of $t \in \upsilon_1$ over $t \in \upsilon_0$. The misclassification mostly occurs for transactions which belong to υ_0 but are classified in υ_1 instead, therefore, rightly classified transactions in υ_1, showed by P_1, outnumber misclassified transactions, resulting in good precision score. We can tell by observing Fig. 3 that ResNet6 has trouble in classifying $t \in \upsilon_0$, since R_0 has a score that goes from 61% to 71%, while in ResNet6* it goes from 73% to 83%.

[6]with the nomenclature $P_{0(1)}$ we refer to P_0 and P_1

The precision is in line with the accuracy, between 80% and 90%, which means that, despite having the same days for training and test, which might drag accuracy down, the model is still confident on rightly classifying most of the transactions. We also tested DNN6 and DNN6* and they resulted to have a lower accuracy in all tests by at least 1%.

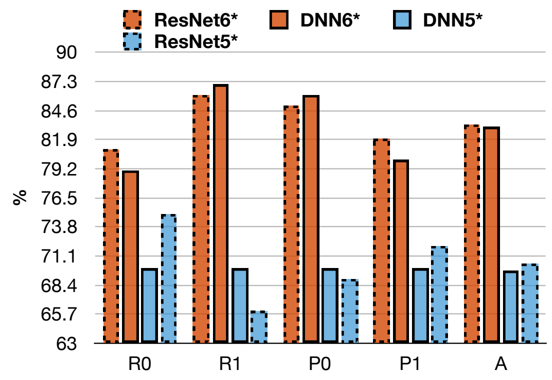
V. DISCUSSIONS

This paper wants to be the beginning of a broader project, a project which aims to educate users on how they can spend their fee in a PoW-based blockchain, in a way to optimize their expenses while miners or workers, are guaranteed a fair revenue for their work. The first step was then to create a prediction model based on historic blocks, which enables the implementation of future systems who wants to include such model in order to educate users on transaction fee.

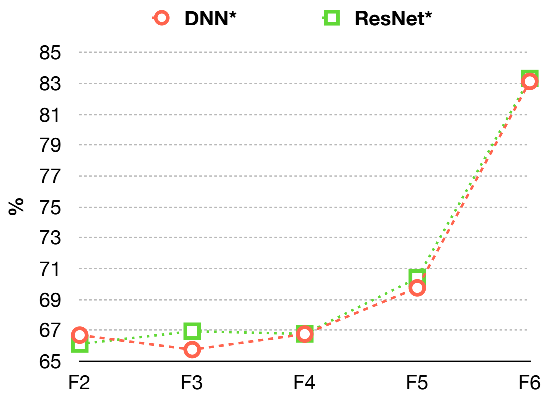
The inclusion pattern estimation problem is proved to be a not trivial one, since it depends on transaction fee estimation. the latter involves some unpredictable factors, such as supply unpredictability, demand unpredictability and different requirements from users. The supply is predictable only in the long run, having approximately 2 MB of space every 10 minutes, while is unpredictable over shorter time. Having a Poisson distribution over space supply means that we might have one block over a hundred discovered within 7 seconds from the previous, and one in a hundred discovered after 45 minutes the previous [37]. Also the demand is more predictable in the long run, since there is some cyclicity in transaction flow. However, it is hard to predict the demand on a shorter period, and since transaction fees will change following the demand, fee estimators will suffer of that.

Our model, since it includes the offset δ, aims to learn the demand unpredictability over time, and it tries to give an accurate confidence based on the network demand. We do not consider in our model the network supply, since the model predicts whether a transaction has probability of being inserted in the next block, independently on when the next block will be mined.

We gather data and build our dataset with the assumption to have a complete blockchain, including every transaction present in it for a particular time frame. Because of that it will be possible to generate the expected δ for every transaction. Due to this assumption, the cost of calculating δ is computationally high in a considerable large dataset, having for twenty days analysis a total training time of 6 hours, where 3 hours are used for calculating δ. The process of fetching δ over a five days dataset takes from 1 to 2 hours, while the total time of training goes up to 2 hours and 40 minutes. However, the offset gives a boost in the model accuracy of $\sim 15\%$, thus it will not be possible to exclude it from the analysis, even if the model will train faster. Furthermore, our local instance of the blockchain saves 44% of the space on disk, making it a valuable dataset even for other purposes.

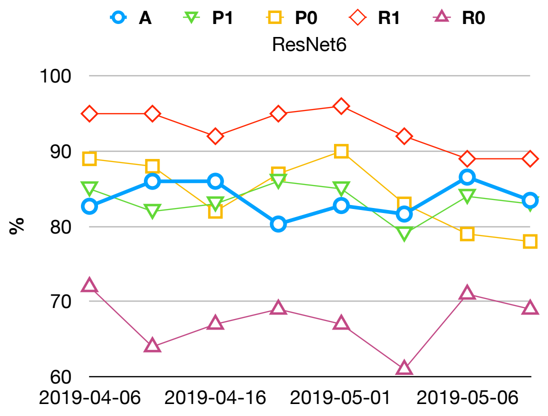


(a) Recall, precision and accuracy of four different models (DNN5*, ResNet5*, DNN6*, ResNet6*) trained from 6^{th} April 2019 to 26^{th} April 2019 and tested on transactions until 11^{th} May 2019. The red bars indicate $\mathbb{F}^6$ models while the blue ones $\mathbb{F}^5$. The dashed bars represent ResNet while the continuous DNN.

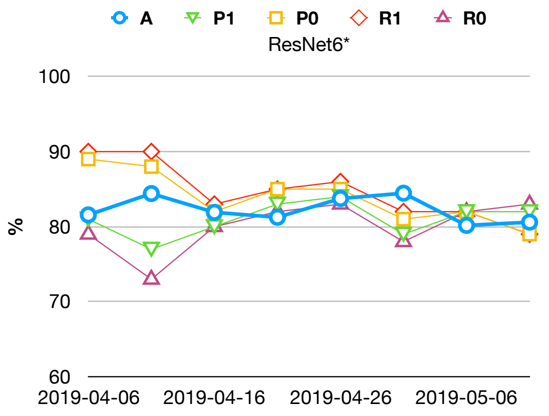


(b) Graph showing the improvement in accuracy for different models, DNN and ResNet, when the feature set is incremented from $\mathbb{F}^2$ to $\mathbb{F}^6$.

Fig. 2: Twenty days (2a) and features analysis (2b).



(a) Without class balancing



(b) Class balancing

Fig. 3: ResNet6 and ResNet6* models over one month, tested in batches of five days training and five days test. Precision, recall and accuracy are represented for each batch, from 6^{th} April 2019 til 6^{th} May 2019. The left figure represents the score using the ResNet6 model with no class balancing, hence, classes are only weighted. The figure on the right instead, represents the score for a balanced model, ResNet6*.

The two models, DNN and ResNet, got very similar results; we assume that the cause of it is the simplicity of the data distribution, which means that the loss function surface remains smooth, therefore with our data, it is way more important to select the right feature set rather than a particular model. The reason for the accuracy ceiling, not higher than $\sim 86\%$, might be caused by data noise and related to the unpredictability factors mentioned above.

We also need to consider that the accuracy is calculated by rounding off the probability output value to either 0 or 1, thus a transaction might get classified in v_1 with a confidence of only 51%. An user will get the confidence instead, so with 51% confidence, the user might reconsider its fee. In our analysis, misclassified transactions got a quite low confidence if compared to the ones which were rightly classified.

Future improvements of the model have been thought. First of all, we assume that miners have different policies for transaction inclusion, then we want to include another feature, B_{mi}, that indicates which miner mined a certain block. In this way an user can have different confidences, each one representing the probability of inclusion if the next block is mined by B_{mi}. The output confidences will be paired together with the probability $P(B_{mi})$, which indicates the probability for B_{mi} to mine the next block, given the historical record for instance, of the last two months of miner's activity.

Even if we do something slightly different than purely estimating transaction fee, it would not be difficult to get the expected approval time of a transaction from our model, since it depends directly on the inclusion pattern. Because of that, it will be possible to benchmark the system with a simple fee and volume strategy currently in use, such the one implemented in Bitcoin Core.

VI. Conclusions

The current Bitcoin transaction fee market is based on a first-price auction principle, which is not an optimal solution for PoW-based blockchain systems. Such fee markeds make transaction inclusion for users a complex task that often ends up disadvantaging them by paying an unfair amount of fee. However, transaction fees are still the primary motivation for encouraging mining. Without it, soon the miners will not have any other revenue, It is therefore important to balance miners profit with a fair, but not too high, fee paid by users.

Our proposed model does not want to be a fee estimator that impose a fee that will be paid to miners. Instead, our goal is to learn from previous blocks in order to give users a confidence on how much well formed their transactions are. In a way that, users can use this information to trade their fee with better latency in the network. Our assumption is that miner does not select a transaction only by its carried fee. We select several other features we believe that have impact on how miners include transactions. We also assume that transactions are ordered based on their fee density in a miner's mempool. To each transaction we assign a successor and a predecessor as explained in Section III, then from the retrieved dataset we derive our new features, *offset* and *delta*, and finally add those to our training dataset.

We propose two machine-learning models, DNN6*, and ResNet6*, both having as input space $\mathbb{F}^6$ set, with the purpose of helping users in understanding the transaction fee trend related to the auction fee market. By studying the transaction inclusion pattern, users can optimize their expenses with no loss in transaction latency, and still miners can get their expected revenue without big losses.

Considering the difficulty of having a pattern when the creation time is a randomized process, and the miner's policy of inclusions are unknown, we obtained significant results, thus confirming the importance of some features we believed to be relevant. While analyzing more than 10 million transactions for the twenty days analysis and batches of 1.5 million for the five days analysis, we obtained an accuracy between 80% and 90%, even though we did not use the confidence given by the model as users might do, but a rounded off value as explained in Section V, hence, the information an user can get by using the model it goes beyond model's accuracy score. Therefore, by following the confidence of these models, users can perform their transactions in order to have a trade off between the fee paid and the confidence of $P(v_1)$.

Acknowledgement

This work is funded in part by Research Council of Norway project numbers 263248 and 275516.

References

[1] S. Nakamoto *et al.*, "Bitcoin: a peer-to-peer electronic cash system." https://bitcoin.org/bitcoin.pdf, 2008.

[2] P. R. Rizun, "A transaction fee market exists without a block size limit," *Block Size Limit Debate Working Paper*, 2015.

[3] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*, pp. 1–10, Sept 2013.

[4] E. Tedeschi, H. D. Johansen, and D. Johansen, "Trading network performance for cash in the bitcoin blockchain," in *Proceedings of the 8th International Conference on Cloud Computing and Services Science, CLOSER 2018, Funchal, Madeira, Portugal, March 19-21, 2018.*, pp. 643–650, 2018.

[5] M. Möser and R. Böhme, "Trends, tips, tolls: A longitudinal study of Bitcoin transaction fees," in *Financial Cryptography and Data Security: FC 2015.*, no. 8976 in LNCS, (Berlin, Heidelberg), pp. 19–33, Springer Berlin Heidelberg, 2015.

[6] H. Qureshi, "Blockchain fees are broken. here are 3 proposals to fix them.," Jun 2019.

[7] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 51–68, ACM, 2017.

[8] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[9] M. Samaniego and R. Deters, "Blockchain as a service for iot," in *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 433–436, IEEE, 2016.

[10] N. Houy, "The economics of Bitcoin transaction fees," Working Papers 1407, Groupe d'Analyse et de Théorie Economique (GATE), Université Lyon 2, 2014.

[11] Y. Zhu, R. Guo, G. Gan, and W. Tsai, "Interactive incontestable signature for transactions confirmation in bitcoin blockchain," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, pp. 443–448, June 2016.

[12] S. Basu, D. Easley, M. O'Hara, and E. Gün Sirer, "The old fee market is broken, long live the new fee market," Jan 2019.

[13] S. Haykin, *Neural networks*, vol. 2. Prentice hall New York, 1994.

[14] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, "Bar fault tolerance for cooperative services," in *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05, (New York, NY, USA), pp. 45–58, ACM, 2005.

[15] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized Blockchains," in *Financial Cryptography and Data Security. FC 2016.*, vol. 9604 of *LNCS*, (Berlin, Heidelberg), pp. 106–125, Springer Berlin Heidelberg, 2016.

[16] R. Baker *et al.*, "Data mining for education," *International encyclopedia of education*, vol. 7, no. 3, pp. 112–118, 2010.

[17] I. Lykourentzou, I. Giannoukos, V. Nikolopoulos, G. Mpardis, and V. Loumos, "Dropout prediction in e-learning courses through the combination of machine learning techniques," *Computers & Education*, vol. 53, no. 3, pp. 950 – 965, 2009.

[18] I. Bose and R. K. Mahapatra, "Business data mining—a machine learning perspective," *Information & management*, vol. 39, no. 3, pp. 211–225, 2001.

[19] J. Dean, *Big data, data mining, and machine learning: value creation for business leaders and practitioners*. John Wiley & Sons, 2014.

[20] M. Chen, Y. Hao, K. Hwang, L. Wang, and L. Wang, "Disease prediction by machine learning over big data from healthcare communities," *Ieee Access*, vol. 5, pp. 8869–8879, 2017.

[21] S. Dua, U. R. Acharya, and P. Dua, *Machine learning in healthcare informatics*, vol. 56. Springer, 2014.

[22] M. Riegler, M. Lux, C. Griwodz, C. Spampinato, T. de Lange, S. L. Eskeland, K. Pogorelov, W. Tavanapong, P. T. Schmidt, C. Gurrin, D. Johansen, H. Johansen, and P. Halvorsen, "Multimedia and medicine: Teammates for better disease detection and survival," in *Proc. of the the 2016 ACM on Multimedia Conference*, MM '16, pp. 968–977, ACM, 2016.

[23] S. Stolfo, D. W. Fan, W. Lee, A. Prodromidis, and P. Chan, "Credit card fraud detection using meta-learning: Issues and initial results," in *AAAI-97 Workshop on Fraud Detection and Risk Management*, 1997.

[24] T. B. Trafalis and H. Ince, "Support vector machine for regression and applications to financial forecasting," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 6, pp. 348–353, IEEE, 2000.

[25] Y. Zheng, L. Liu, L. Wang, and X. Xie, "Learning transportation mode from raw gps data for geographic applications on the web," in *Proceedings of the 17th international conference on World Wide Web*, pp. 247–256, ACM, 2008.

[26] X. Ma, H. Yu, Y. Wang, and Y. Wang, "Large-scale transportation network congestion evolution prediction using deep learning theory," *PloS one*, vol. 10, no. 3, p. e0119044, 2015.

[27] Y. Bengio, A. C. Courville, and P. Vincent, "Unsupervised feature learning and deep learning: A review and new perspectives," *CoRR*, vol. abs/1206.5538, 2012.

[28] J. Schmidhuber, "Deep learning in neural networks: An overview," *CoRR*, vol. abs/1404.7828, 2014.

[29] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[32] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Advances in Neural Information Processing Systems*, pp. 6389–6399, 2018.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[34] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," pp. 1137–1143, Morgan Kaufmann, 1995.

[35] S. Theodoridis and K. Koutroumbas, *Pattern Recognition, Fourth Edition*. Academic Press, 4th ed., 2008.

[36] S. Geisser, *Predictive inference: an introduction*. CRC Press, 2017.

[37] J. Newbery, "An introduction to bitcoin core fee estimation," *Bitcoin Tech Talk*, 2017.

Paper III

Tedeschi, E., Nordmo, T.A.S., Johansen, D. & Johansen, H.D. (2022)

On Optimizing Transaction Fees in Bitcoin using AI: Investigation on Miners Inclusion Pattern.

ACM Transactions on Internet Technology, 22(3), 77.

On Optimizing Transaction Fees in Bitcoin using AI: Investigation on Miners Inclusion Pattern

ENRICO TEDESCHI, TOR-ARNE S. NORDMO, DAG JOHANSEN, and HÅVARD D. JOHANSEN, UiT The Arctic University of Norway

The transaction-rate bottleneck built into popular proof-of-work (PoW)-based cryptocurrencies, like Bitcoin and Ethereum, leads to fee markets where transactions are included according to a first-price auction for block space. Many attempts have been made to adjust and predict the fee volatility, but even well-formed transactions sometimes experience unexpected delays and evictions unless a substantial fee is offered. In this article, we propose a novel transaction inclusion model that describes the mechanisms and patterns governing miners decisions to include individual transactions in the Bitcoin system. Using this model we devise a Machine Learning (ML) approach to predict transaction inclusion. We evaluate our predictions method using historical observations of the Bitcoin network from a five month period that includes more than 30 million transactions and 120 million entries. We find that our ML model can predict fee volatility with an accuracy of up to 91%. Our findings enable Bitcoin users to improve their fee expenses and the approval time for their transactions.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Feature selection**; • **Information systems** → Specialized information retrieval; • **Computer systems organization** → *Peer-to-peer architectures;*

Additional Key Words and Phrases: Bitcoin, blockchain, fee market, first price auction, transaction inclusion, cryptocurrencies, proof-of-work, machine learning

ACM Reference format:
Enrico Tedeschi, Tor-Arne S. Nordmo, Dag Johansen, and Håvard D. Johansen. 2022. On Optimizing Transaction Fees in Bitcoin using AI: Investigation on Miners Inclusion Pattern. *ACM Trans. Internet Technol.* 22, 3, Article 77 (July 2022), 28 pages.
https://doi.org/10.1145/3528669

1 INTRODUCTION

Blockchain protocols, like Bitcoin and Ethereum, secured in mid-2020 about 80% of the cryptocurrency market cap [55]. Most of the current blockchain implementations that use a permissionless **Proof-of-Work** (**PoW**)-based chain, come with some non-negligible side effects. Besides their substantial environmental impact [39], they suffer from a non-trivial scalability problem: the low throughput (transactions approved per second) [11, 30, 41, 43]. The substrate consensus protocol (Proof-of-Work (PoW)) limits the throughput upper bound with two important parameters: the *block size* (time constraint), and the *interblock interval time* (frequency constraint). Changes to

This research was funded in part by The Research Council of Norway under grant numbers 270053 and 263248.
Authors' addresses: E. Tedeschi, T.-A. S. Nordmo, D. Johansen, and H. D. Johansen, Institutt for Informatikk, Fakultet for Naturvitenskap og Teknologi, UiT Norges Arktiske Universitet, Postboks 6050 Langnes, N-9037 TROMSØ (NO); emails: enrico.tedeschi@uit.no, tor-arne.s.nordmo@uit.no, dag.johansen@uit.no, havard.johansen@uit.no.

either of those two parameters to improve throughput demand proportional improvement in the block-propagation time, which is limited by the current P2P network substrate [3, 30].

The Bitcoin system was intended to provide users with a low-cost payment scheme, with transaction fees close to or equal to zero [37]. In a tragedy of the commons, the cryptocurrency's rising popularity made the physical throughput limitations of the underlying PoW scheme a key scalability bottleneck [13, 42]. The high cost of mining has led to an increased usage of transaction fees as a means for miners to make a profit.

Transaction fees and the behavior of miners became a relevant subject of study after the affirmation of Bitcoin as the primary cryptocurrency. The trend of a long-established PoW-based blockchain started to move towards a fee-oriented market [36], where a *rational* (or greedy) miner aims at optimizing its profits by following a certain *norm* while selecting new incoming transactions from its mempool. This is known as the *fee-per-bytes* dequeuing policy, where transactions are being scheduled for inclusion according to their *feerate*. While fee predictors still adopt the notorious fee-per-bytes dequeuing policy, miners somehow deviate from the norm [35]. This has some serious economic implications for users, *fee-overpaying* is rather the norm and first-price auction markets have failed to provide users with stable prices [7].

Deciding on a common static fee is impossible in practice [27, 36], and users are instead forced to either choose an appropriate payment dynamically or rely on current fee estimators when submitting their transactions, with no exact formula to optimize expenditure or to control the time it takes for a transaction to be confirmed [54]. Most users end up using their own estimator based on their own intuition [40]. This turns out to be a very difficult task, and in most cases expensive for users [6]. Furthermore, many existing estimators have the notorious problem of aggregate overpaying, while adopting a higher fee than necessary [19]. In 2017, poorly designed fee estimators contributed to driving up average Bitcoin fees to over $ 20 per transaction [40, 47].

The strategic behavior of miners and users over time made the role of transaction fees in Bitcoin change from a mining-based structure to a market-based ecology [16]. This makes PoW-based blockchain systems unpredictable at scale. Thus, their non-deterministic outcome for transaction fees induces blockchain-based cryptocurrencies to not be suited in the near future, as a global, shared, and decentralized currency. The outcome of such systems must be predictable, and the risk of overpaying should be reduced to zero.

Enhancing our knowledge of how Bitcoin's first-price auction and transaction fee mechanisms work in practice is important to make such systems more useful and predictable under various network congestion circumstances or independently of the fees paid by other users. In this article, we, therefore, propose a novel transaction inclusion model that describes the fee-market mechanisms of the Bitcoin system. We show how the overpaying problem is a consequence of miners' rational (and unexpected) behaviors, define the concept of fee market, and outline its main side effects. Using our transaction inclusion model we devise a **Machine Learning** (**ML**) approach based on **Deep Neural Networks** (**DNNs**) that can predict miners' behavior while selecting new transactions for inclusion. We evaluate our prediction method using historical observations of the Bitcoin blockchain from a five month period that includes more than 30 million transactions and 120 million entries. Our findings enable Bitcoin users to improve their fee expenses and approval time for their transactions.

The main contributions of this article are as follows:

— A formal definition of a *transaction inclusion model* (or pattern).
— A dataset containing Bitcoin transactions sampled every month, from January 2021, until May 2021 [46].
— A multi-layer **Neural Network** (**NN**) model to predict the pattern we formalize in this article.

The remainder of the article is organized as follows. In Section 2, we describe related works, including studies on transaction fees and miners' behavior. We also analyze relevant works on ML models for pattern prediction. In Section 3, we describe the Bitcoin fee market, its purpose for the users, and how miners' rational behavior relates to overpayment. Next, our transaction inclusion model is described in Section 4, its implementation is explained in Section 5, and the evaluation is presented in Section 6. Section 7 discusses our findings and Section 8 concludes.

2 RELATED WORKS

The high fee issue in Bitcoin is well known. Already in 2014 the study of Kaskaloglu [29] claims that an increase in transaction fees of Bitcoin is inevitable, and they identify the reasons to be: (1) cost of mining, (2) risk of 51% attack. The study also discusses the parameters affecting the problem of determining the right fee for Bitcoin, and it considers how these parameters are changing in a dynamic ecosystem of miners, investors, and users of the Bitcoin network. Later studies started to analyze miners' behavior with respect to transaction fees. In 2015 Möser and Böhme [36] acknowledge the role of fees as a key aspect of the system's stability. They provide empirical evidence of agents' behavior concerning their payment of transaction fees, together with several regime shifts, caused by changes in the default client software. The study shows the trend of a long-established PoW-based blockchain, moving towards a fee-oriented market.

As fees increase, miner behavior become an interesting subject of study and research. In 2019, the study of Basu et al. [7] outlines that first-price auction markets have failed to provide users with stable prices for their services, and historical analysis shows that Bitcoin users could have saved $ 272,528,000 in transaction fees, while miners could have reduced the variance of fee income by an average factor of 7.4 times. Furthermore, the study of Messias et al. [35] shows that miners somehow deviate from the first-price auction norm, while fee predictors still adopt the notorious fee-per-bytes dequeuing policy. This has some serious economic implications for users; fee-overpaying is rather the norm, and Messias et al. [35] show that in June 2019 more than 30% of transactions were offering a feerate two orders of magnitude higher than the minimum recommended.

Improving on the current first-price auction in the Bitcoin environment is neither a simple nor straightforward matter, and the study of Basu et al. [7] outlines how the design of such a mechanism is challenging. As a matter of fact, miners can use any criteria for including transactions and can manipulate the results of the auction after seeing the proposed fees. Basu et al. [7] introduce a mechanism inspired by a **generalized second-price (GSP)** auction which identifies a bidder willing to pay only more than the (K + 1)-th highest bid, where K is the number of transactions included in a block. Another version of the GSP auction model is presented by Li et al. [31, 32], where they use a novel *rank-by-cost* rule to order transactions. The cost is calculated by the user-submitted fee and the waiting time. With this approach, they show that the daily saved fees for users are up to ₿ 24.5985 on average. Our study will take the Generalized Second Price (GSP) approach into account since we include a relative time-bound notion of transaction, based on block-epoch, which tells how much a transaction is waiting to be approved.

Easley et al. [16] investigate the role that transaction fees play in the Bitcoin blockchain, changing from a mining-based structure, to a market-based ecology. They examine exogenous and endogenous features of Bitcoin. They show how waiting times arise and how they are influenced both by endogenous transactions fees and by exogenous dynamic constraints imposed by the Bitcoin protocol. Furthermore, they highlight how exogenous structural constraints influence the dynamics of user participation on the blockchain. Such features make a PoW-based blockchain system unpredictable at scale, the uncertainty of non-inclusion for transactions makes their issuers insecure, potentially causing overpaying. In their work, they argue that increasing transaction fees will increase the number of miners, but this, in turn, will trigger increases in the difficulty level to control the creation rate of new blocks, thereby raising the costs to miners. More revenue does not necessarily mean more profit, and in a competitive market, this would not lead to an overall increase in compensation. Finally, they argue that transaction fees alone are unlikely to solve the challenges facing the Bitcoin blockchain. This is the reason why our approach offers an alternative by considering fees as *one of the means* for transaction inclusion.

The use of ML models to predict patterns from large sets of data has been already adopted. From hydrology [14] to network attacks [38], and finally the work of Yazdinejad et al. [52] utilizes a large amount of data for hunting cryptocurrencies malware threats. When we analyze the pattern for transaction inclusion we deal with an enormous amount of heterogeneous data, and we need to establish an order out of it. A ML-based approach enables us to make predictions and decisions using sample data (training data). They are widely used by enterprises in different sectors, i.e., education [5, 33], business and marketing [8, 12], healthcare [9, 15], financial services [44, 49], and transportation [34, 53]. An important prerequisite to train an ML model is to have a large and diverse dataset. The public and well-established Bitcoin blockchain fits perfectly for this purpose.

3 THE FEE MARKET IN BITCOIN

Bitcoin can transition from its current operational mode where transactions fees are, in practice, no more than a complementary tip to the miners, to a situation where a *fee market* effectively regulates all traffic. With such a fee market, low-fee transactions might potentially remain pending for hours, days, or even weeks waiting for approval and inclusion by a miner. As the energy demands of the Bitcoin network increase and mining becomes more expensive, a transition to such a fee market becomes more evident.

Bitcoin is highly energy inefficient by design. Its PoW scheme secures records in the blockchain only after a certain amount of computational work has been carried out by the *prover* (or miner). This mechanism is however essential as it prevents double-spending and Sybil attacks. This work must be hard—yet feasible—on the miner side, but easy to check for the *verifier* as a form of a **nondeterministic polynomial time** (**NP**) decision problem. In Bitcoin, the prospect of making profit via mining has built up the common interest in joining the PoW protocol. However, a set by-design parameter such as the *interblock time interval*,[1] made any speed up in the mining process—which would cause more revenue—impossible, despite the network hashing power increasing. As a consequence, mining became more expensive and miners needed to reshape their way of making profits.

Each mined block has an expected profit $\langle \Pi \rangle$ (Equation (1)), where $\langle V \rangle$ is the expected revenue for mining (Equation (2)), and $\langle C \rangle$ (Equation (3)) are the expenses, or cost of mining [42].

$$\langle \Pi \rangle = \langle V \rangle - \langle C \rangle, \tag{1}$$

$$\langle V \rangle = (R + M)\frac{h}{H}(1 - \mathbb{P}_{orphan}), \tag{2}$$

$$\langle C \rangle = \eta h \mathcal{T}. \tag{3}$$

Miners' revenue $\langle V \rangle$ is inversely proportional to the total hashing power of the Bitcoin network H, but directly related to three main factors: the reward and transaction fees $(R + M)$, the individual hashing power (h), and the probability of not orphaning[2] the block just mined $(1 - \mathbb{P}_{orphan})$. The reward R for mining a block is, as of writing, ₿6.25. The value M represents the sum of all the

[1]The average time required for the system to mine a new block.
[2]Detached or Orphaned blocks are valid blocks that are not part of the main chain. They can occur when two miners produce blocks at similar times, and one block gets discarded because of a higher propagation delay.

transaction fees included in the mined block. Increasing the number of miners will substantially increase H, and consequently it will reduce $\langle V \rangle$. Miners' expenses $\langle C \rangle$ are directly proportional to the individual hashing rate (h), the cost per hash (η), and the expected time to mine a block ($\mathcal{T}$). It is evident that the system becomes more expensive for miners as the number of them scales up, and the inherent throughput limitation of the system causes new incoming transactions to compete for space in the blocks, as the number of transactions scales up. Because of that, miners can change their behavior for transaction inclusion, in a profit-oriented manner.

The nontrivial scalability problem in Bitcoin, and its scalability bottleneck, resides in the *low throughput.*[3] It depends directly on two factors: (1) the *block size* Q, and (2) the *interblock interval time* $\mathcal{T}$, which allows approximately from three to seven transactions per second. This upper bound is hindered by an exogenous property known as *difficulty* (Equation (4)). The difficulty raises or drops in order to maintain a target block creation time of $\mathcal{T} \simeq 600$ sec, and it is normalized according to $\mathcal{T}'$, which is selected as the mean value of the past 2,016 block creation times. Equation (4) defines difficulty d_x at block height x. Because $\mathcal{T}$ is fixed, in order to avoid this throughput bottleneck one remedy is to increase the block size Q. Unfortunately, the blockchain's distributed nature does not allow to arbitrarily change either of these parameters [30], and miners have the hallowed power to unilaterally order and select transactions [28]. Furthermore, because the network is congested most of the time [35], this creates a competition for transaction inclusion, that escalates in a *first-price auction market.*

$$d_x = \begin{cases} 1 & \text{if } x = 0 \\ d_{x-1}\mathcal{T}/\mathcal{T}' & \text{if } x > 0. \end{cases} \tag{4}$$

In Equation (2) we see that a miner's profit depends on R and M. The block reward R is halved every 210,000 blocks, leaving M—the sum of transaction fees—the only reliable source of profit left in the long term. In a first-price auction market, a bidder tries to raise his bet in order to beat any other competitor. In absence of any other source of profit, transaction fees are competing with each other as bidders in a first-price auction market. The well-known adopted strategy by miners to choose a bidder, is the fee-per-bytes dequeuing policy, and it is widely considered to be the *norm* for prioritizing transactions [35]. In this policy, transactions are ordered by their *feerate*—transaction fee divided by its size in bytes (Equation (6))—and then included accordingly. However, Eyal and Sirer [18] show that the Bitcoin mining protocol is not incentive-compatible. Greedy miners can benefit from a first-price auction scheme while the burden is being borne by users. The individual competition of a first-price auction scheme is harmful for Bitcoin, where its market design engages K identical items to be sold—K number of transactions included in the next mined block—to bidders who each want one item at most. The study outlines how a GSP auction market could benefit user expenditures. In this scheme, the K highest bids each win an item (inclusion in a block) and bidders all pay the (K + 1)-th highest bid. If fee estimators use the first-price auction mechanism to calculate transaction fee then overpaying is likely to occur.

A first-price auction market is demonstrably unsuitable for large-scale blockchains based on PoW. The market does not provide a stable coin price, so transaction fees are unpredictable and their variance is enormous, capacity and demand do not always meet, forcing users to overpay for their space in the block, to avoid being left out. We consider two main issues coming from the fee market legacy: *fees unpredictability*, and *users overpaying*. Our purpose is to build a model which defines a *transaction inclusion pattern*, based on ML techniques such as multi-layer Neural Network (NN). The model wants to track both the block size and the mempool size, therefore it

[3]Transactions committed per second (t/s).

lies outside the group of first-price auction predictors. Furthermore, we include features which are not fee-based, so that the model does not rely its knowledge on a second-price auction scheme alone.

Our previous study shows that it is possible, using ML techniques, to analyze and define patterns in big data [48], and the public and well-established Bitcoin blockchain fits perfectly for our purpose. The new proposed model is able to make a binary decision on transaction inclusion—*will a transaction be included in the next mined block?*—thanks to some carefully chosen *engineered features*, based on assumptions we make in Section 4.

4 TRANSACTION INCLUSION MODEL

Here we analyze, define and present our model for transaction inclusion. In Section 4.1, we explain how transaction data is treated, and why we base our observations on a time-series-like approach. Consequently, we analyze two main factors that can alter inclusion. We call them *revenue* (Section 4.2) and *fairness* (Section 4.3), and they serve to ensure an equilibrium of user's and miner's participation. These two definitions are complementary for an accurate study and prediction, therefore the holistic view for transaction inclusion is to be sought in both fairness and revenue concepts.

4.1 Observational Approach

Time-series data analyses have proven to be useful for predicting future trends [20–22, 24, 45, 51]. Our observational approach follows a methodology that strongly depends on time-series data collection, since we sample Bitcoin transactions monthly with a fixed interval, although we add a notion of relative time to it: the *block creation*. The idea is that a transaction carries different information throughout the time that it is pending, and therefore our approach follows a *block-epoch-based* collection, where a transaction changes part of its carried information every time there is a block creation. The relative time interval is then defined between two block creation epochs. Each record t (or transaction), at time x, is uniquely identified with the pair (ha_t, be_x), as the hash of t and the block time epoch at height x. We refer to a transaction t at height x as $t^{(x)}$, and this represents an instance of the transaction t during the time slot $[be_x, be_{x+1}]$. For this study, we want to track transactions over time to have information about network saturation and waiting time at each block epoch. In order to delineate the time slots used for the analysis, we define the *timeline-set* $\mathbb{T}$, which contains all the block time epochs. We consider a transaction *lifespan* $\mathcal{L}(t)$ as the time in seconds that goes from the moment the first node sees t (ep_t), until t is included in a mined block. If the mined block is at height x, then we define $\mathcal{L}(t)$ as $\mathcal{L}_t^x$. A transaction t, therefore, has as many occurrences in our local dataset as the number of blocks it sees before being included. We define $\mathbb{T}$, $\mathcal{L}_t^x$, and the number of t occurrences in a dataset (γ) in Equation (5).

$$\begin{aligned}
\mathbb{T} &= \{be_i | 0 \le i \le \xi\}, &&\text{where } \xi = \max(\text{block height}),\\
\mathcal{L}_t^x &= [ep_t, be_x], &&\text{if } t \text{ is included at height } x,\\
\gamma &= |\{be | be \in \mathbb{T} \cap \mathcal{L}_t^x\}|, &&\text{number of } t \text{ occurrences.}
\end{aligned} \tag{5}$$

4.2 Revenue for Miners

Difficulty is an exogenous property of Bitcoin. It grows in relation with the total network hashing power H, and it makes scaling of miners extremely expensive.[4] The blockchain is secure if a set of malicious miners has less computational power than the honest nodes together, putting Bitcoin

[4]Difficulty increases/decreases because the pool of miners solves the PoW puzzle faster/slower. So if the number of miners increases, they are consuming more while producing the same amount of blocks.

security directly proportional to the growth of H. A high degree of security then hinders a low and cheap maintenance scheme for miners, low transaction fees, and therefore a deterministic outcome of the system. Consequently, we conjecture that *a miner's main purpose is its revenue.*

Referring to Equations (2)–(3), revenues get lower as H increases. A miner then could either: (1) reduce its costs, (2) increase its hashing power h, (3) reduce chances of orphaning $\mathbb{P}_{orphan}$, (4) increase the amount of fee it gets ($M + R$). In order to reduce the miner's costs, the exogenous properties $\mathcal{T}$ and η cannot be changed,[5] so that a miner should decrease its hashing power, which would conflict with point 2. The probability of block orphaning depends on the block propagation delay, which is affected by the block size [11]. A rational miner could make its blocks lighter in order to spread faster to reduce orphaning rate. However, this only partially increases the revenue, since fewer transactions means less fee as reward, besides decreasing the overall network throughput. Furthermore, orphaning rate has been calculated to occur only three times every 1,000 blocks,[6] so it has little impact on miner revenue. If we now consider that R is halved every 210,000 blocks, then M is the only source of profit left. Most miners are assumed to behave rationally [2], implementing individualized and unknown policies for transaction inclusion that aim at maximizing their profit. Because miners can arbitrarily select transactions, revenue is their purpose and the sum of transaction fees is their main endogenous source of profit, we assume that *a transaction is selected rationally, and its inclusion strongly depends on its fee.*

Although we consider transaction fees to play a key role in inclusion, the concept of revenue does not refer exclusively to that. While inserting transactions into a block, every miner increases its possible revenue, but it also decreases his block propagation rate, which undermines the probability to earn any reward at all. In fact, the time needed for propagation in order to reach consensus among participants depends on block size [26]. If the block size Q is fixed to 1 MB, then a rational miner might attempt to optimize the number of transactions paying more fee within Q, by calculating the ratio between transaction fee ϕ and transaction size q, called feerate (ρ) (Equation (6)). Dequeuing feerate policy is generally considered to be the norm [35] among miners, and we assume that *feerate is the bedrock of miners' revenue.*

$$\rho = \phi/q. \tag{6}$$

Despite many fee estimators include feerate in their evaluations,[7] they still fail to avoid overpaying. Analyses show [35] that on average, 50%–70% of transactions offer feerates two orders of magnitude higher than the recommended minimum,[8] and that 88% of all Bitcoin transaction inputs pay higher fees than necessary [17]. Because of this, we conjecture that revenue is not the only metric to study and analyze. We assume that a miner also needs to be fair and give space to those transactions that are waiting for a longer time, in order to maintain system sustainability.

4.3 Fairness for Users

The concept of fairness considers those transactions that, upon payment of a fee, are waiting unfairly long because some newer, higher-fee transactions joined the network. To explain this view we define the set $\mathcal{P}$ (Definition 4.2) as the set of ***relapsed pending transaction (RPT)***. We define the *block-before-inclusion* (β_t) in respect of a transaction t, as the epoch of the last block, mined before t's inclusion. If the latest block epoch is represented as be_ξ, where ξ is the last block

[5]Normally distributed with a fixed known mean (600 seconds in Bitcoin).

[6]0.31% is the probability of orphaning on data collected from blockchain.com for every block occurred from 18th March 2014 to 14th June 2016, and stored at https://cutt.ly/YvUvMz5.

[7]E.g., bitcoin fees https://bitcoinfees.earn.com.

[8]Recommended minimum feerate is 10^{-5} BTC/kB ≡ 1, 000 sat/kB ≡ 1 sat/byte, according to Bitcoin Core.

height, this results in:

$$\beta_t = \begin{cases} be_{x-1}, & \text{if } t \text{ is included in block at epoch } be_x, \\ be_{\xi}, & \text{if } t \text{ is yet to be included.} \end{cases} \tag{7}$$

We also extend the block-before-inclusion concept to the block-epoch approach, and the β_t at height x for a transaction $t^{(x)}$ is ${\beta_t}^{(x)} = be_x$, where for height x we always refer to the time slot $[be_x, be_{x+1}]$. It is important to define in our model the principle of fairness since a transaction cannot wait forever for approval, upon payment of a rightful *expected fee.*

Definition 4.1 ($\mathbb{E}[f]$). The expected fee value is the amount of fee a transaction should pay according to its size q, if we consider the minimum feerate requirements of 1 sat/byte.

$$\mathbb{E}[f] = \rho' q, \text{ where } \rho' \geq 1 \text{ sat/byte.} \tag{8}$$

We now assume that if a rational miner is fair, it does not only include transactions with a high feerate, but it also considers those in the set $\mathcal{P}$.

Definition 4.2 ($\mathcal{P}$). We identify $\mathcal{P}^{(y)}$ as the set of relapsed pending transaction (RPT) at height y. The set contains transactions that in their lifespan (included at be_y), have seen at least one block creation, have not been included up until height y, and have a fee equal or greater than the expected one, such that:

$$\begin{aligned} \mathcal{P}^{(y)} = \{t | ep_t - {\beta_t}^{(y)} < 0 \quad & \wedge \\ be_y < be_x, \text{ if } \mathcal{L}(t) = \mathcal{L}_t^x \quad & \wedge \\ \phi_t \geq \mathbb{E}[f_t]\}. & \end{aligned} \tag{9}$$

We say t is an RPT transaction at height x, if $t^{(x)} \in \mathcal{P}^{(x)}$.

We consider a more generic definition of $\mathcal{P}$ as the union of all the block-epoch-based $\mathcal{P}$-sets: $\bigcup_{i=0}^{\xi} \mathcal{P}^{(i)}$. Following this concept, we assume that, when a transaction appears multiple times in $\mathcal{P}$, it will have more chances of being included in the next mined block.

4.4 Model Formalization

An efficient transaction inclusion model should consider the aforementioned concepts and it should monitor, at the time of training and testing, the current network state. For this purpose, it is important to keep track, of each transaction analyzed, its moment of inclusion in a new block. The set $\mathcal{A}$ contains **temporarily approved transactions** (**TATs**) (Definition 4.3). The latter is useful in phases of training and testing, in order to have knowledge of when a transaction is about to be included, monitoring at the same time any other type of information carried by such a transaction.

Definition 4.3 ($\mathcal{A}$). We identify $\mathcal{A}^{(y)}$ as the set of TAT at height y. The set contains all the yet-to-be-included transactions that are selected for inclusion at height y, during $[be_y, be_{y+1}]$ time slot, and included then at height $y + 1$. We say t is a temporarily approved transaction (TAT) at height y if $t^{(y)} \in \mathcal{A}^{(y)}$.

$$\mathcal{A}^{(y)} = \left\{ t | \mathcal{L}(t) = \mathcal{L}_t^{y+1} \right\}. \tag{10}$$

Figure 1 shows two instances of a block-epoch-based transaction, t_1 and t_2, initiated by User 1 and User 2, respectively. Transaction t_1 carries different information according to where it is located, and we name it differently, e.g., $t_1^{(x-2)}$, $t_1^{(x-1)}$, and $t_1^{(x)}$. The belonging of t_1 and t_2 to sets $\mathcal{A}$

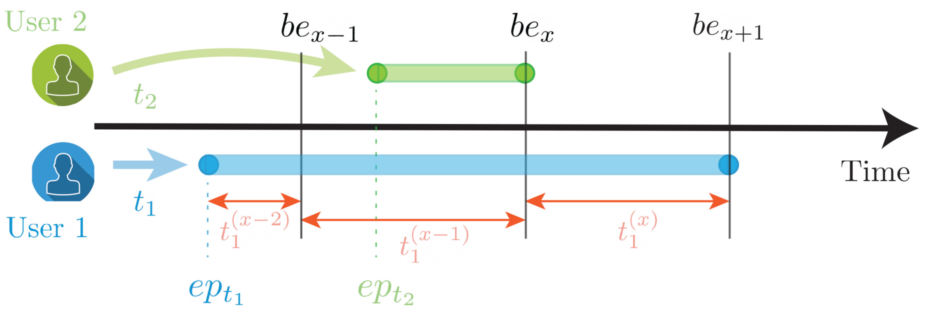


Fig. 1. Two different transactions, t_1 and t_2, issued at time ep_{t_1} and ep_{t_2}, having different lifespan. t_1 will be included in the third block after its inception, at height $x + 1$, while t_2 will be immediately included at height x. The number of t_1 occurrences we represent is then $\gamma = 3$, while t_2 has $\gamma = 1$ occurrence.

and $\mathcal{P}$ changes over time, and we formalize it in Equations (11) and (12). Considering that lifespan for t_1 is $\mathcal{L}(t_1) = \mathcal{L}_{t_1}^{x+1}$, then:

$$\begin{aligned} t_1^{(x-2)} &\notin \mathcal{P}^{(x-2)} \wedge \notin \mathcal{A}^{(x-2)}, \\ t_1^{(x-1)} &\in \mathcal{P}^{(x-1)} \wedge \notin \mathcal{A}^{(x-1)}, \\ t_1^{(x)} &\in \mathcal{P}^{(x)} \wedge \in \mathcal{A}^{(x)}. \end{aligned} \tag{11}$$

Similarly, since lifespan for t_2 is $\mathcal{L}(t_2) = \mathcal{L}_{t_2}^{x}$, we have:

$$t_2^{(x-1)} \notin \mathcal{P}^{(x-1)} \wedge \in \mathcal{A}^{(x-1)}. \tag{12}$$

As can be observed in Section 4.2, each miner tries to optimize their profit by including transactions with a higher feerate first, and this is also widely accepted to be the norm. A rational miner will then order pending transactions by feerate (ρ). Our transaction inclusion model must take that into account, so we define the ordered set of pending transactions S, formalized in Definition 4.4.

Definition 4.4 (S). We define S at block height x as the set of ordered pending transactions, $S^{(x)}$. The set includes all non-approved transactions at time be_x, ordered by their feerate in ascending order, formally:

$$\begin{aligned} S^{(x)} &= \left\{t | \mathcal{L}(t) = \mathcal{L}_t^y \wedge ep_t < be_{x+1}\right\}, \quad \forall y > x, \\ S^{(x)} &= [t_1, t_2, \dots t_n] \quad \text{is an ordered set,} \\ &\text{where } \rho_{t_1} \leq \rho_{t_2} \leq \dots \leq \rho_{t_n}. \end{aligned} \tag{13}$$

Our model uses information about sets $\mathcal{P}$, $\mathcal{A}$, and S to make decisions on transactions inclusion. Knowledge on revenue and fairness (Sections 4.2–4.3) from sets $\mathcal{P}$ and S is fundamental to determine whether or not a transaction belongs to the set $\mathcal{A}$ over time. The model uses information from the aforementioned sets, to obtain new features through a process of feature extraction (Section 5.2). Figure 2 shows how the $\mathcal{P}$ $\mathcal{A}$ S ecosystem changes during time, and it considers five different transactions carrying contrasting feerate values. In the next section, we explain data retrieval, elaboration, and features engineering, and we finally build and test the model for transaction inclusion.

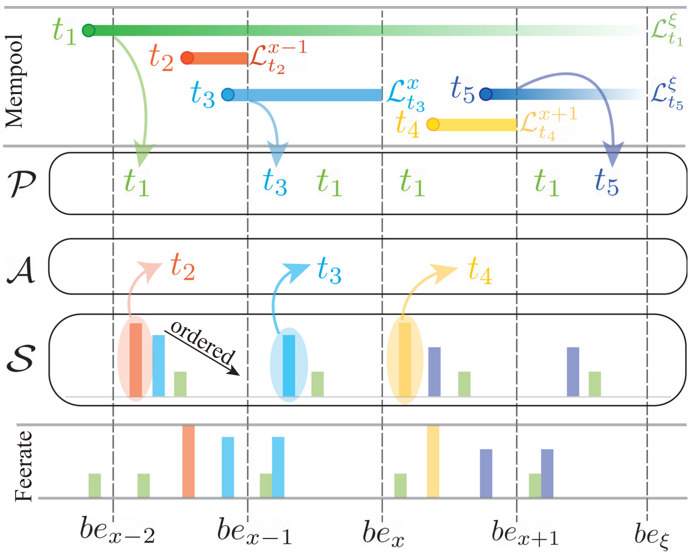


Fig. 2. We consider a subset of transactions, $\{t_1, \ldots, t_5\}$, from time be_{x-3} to be_ξ, to graphically show their path towards inclusion and their belonging in $\mathcal{P}$, $\mathcal{A}$, and S sets. Transactions are inserted in the mempool upon their inception, they carry information about their feerate, and in different time-epochs they can belong or not to the sets we have defined. In S, transactions are ordered by feerate, $\mathcal{P}$ gives a relative time-view of how long they are waiting, and finally, they belong to $\mathcal{A}$ if they are about to be included.

5 METHODOLOGY

We store data locally to have an instance of the Bitcoin blockchain always accessible and save on average up to 68%[9] of the disk space required, by removing redundancies, keeping the essential information for predictions, and adding newly extracted features. We collect data using third-party APIs[10] and gather information about money exchange prices with *forex-python*[11] libraries. Furthermore, we collect information to verify data correctness by setting up our own node using Bitcoin Core.[12] Once data is retrieved and relevant information based on revenue and fairness is stored, we extract new features, and we perform supervised classification using a Deep Neural Network (DNN) model, making it possible to analyze a considerable part of the blockchain with little up-front investment in computational resources. We divide this Section into Data Acquisition (Section 5.1), Feature Selection and Extraction (Section 5.2), and Prediction Model (Section 5.3).

5.1 Data Acquisition

Retrieving data from Bitcoin Core can often be cumbersome and time-consuming. The process of calculating transaction fees, for instance, is intricate and it requires checking all the spent outputs in different transactions, due to Bitcoin being based on the **Unspent Transaction Outputs (UTXO)** model. Using the procedure listed in Algorithm 1, it takes on average 30 seconds to fetch

[9]For instance, we manage to store 1.3 GB of information from the actual Bitcoin blockchain, in only 0.4 GB. See Table 1 and 2 for more information.
[10]https://www.blockchain.com.
[11]https://pypi.python.org/pypi/forex-python.
[12]https://bitcoin.org/en/bitcoin-core/.

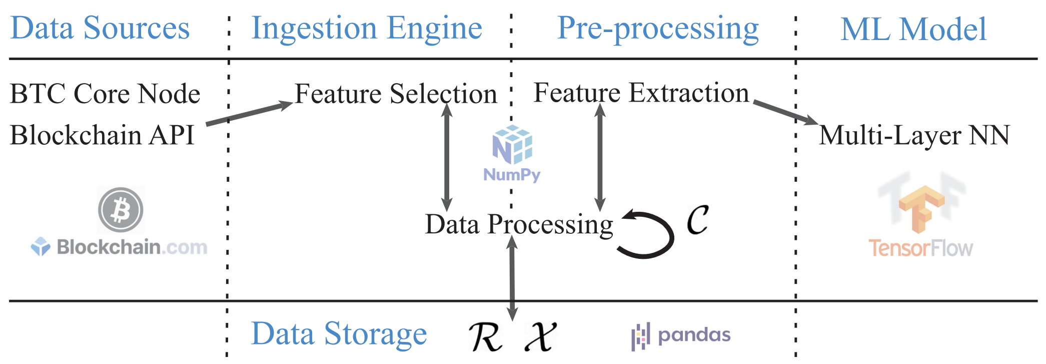


Fig. 3. Data flow in BAS, from the blockchain to the ML framework. $\mathcal{R}$, $\mathcal{C}$, and $\mathcal{X}$ represent different datasets. The first indicates raw data extracted from the blockchain, the second represents a complete, run-time only dataset, while the third one includes all training data.

information about fees contained in a single block. We have developed our own system for data retrieval, **Blockchain Analytics System** (**BAS**), built over a Python back-end. In BAS, transaction fees are obtained using APIs from blockchain.com, which is faster than using Bitcoin Core[13] and more convenient for our purpose. Blockchain Analytics System (BAS) fetches and stores only targeted data, which makes it faster than retrieving or querying the entire blockchain. Furthermore, it saves resources up-front, making data more accessible for future queries.

ALGORITHM 1: Our approach to get information on a single transaction fee using Bitcoin Core.

```
1:  procedure GETTXFEE(t)                         ▷ in: json from getrawtransaction query of Bitcoin Core
2:      sin, sou ← 0                              ▷ initialized sum of transaction input and output
3:      for all in ∈ t['vin'] do                  ▷ for each input in t
4:          txin ← getrawtransaction(in['txid'])  ▷ call using bitcoin-cli and local blockchain
5:          vin ← txin['vout'][in['vout']]['value']  ▷ in['txid'] is the index of the spent output in in
6:          sin ← sin + vin                       ▷ update transaction input
7:      for all ou ∈ t['vout'] do                 ▷ for each output in t
8:          vou ← ou['value']
9:          sou ← sou + vou                       ▷ update transaction output
10:     $\phi_t$ ← sin - sou                      ▷ calculate transaction fee
11:     return $\phi_t$
```

Figure 3 shows the data process in BAS from the blockchain to the ML model. Data is retrieved using third-party APIs[10] and our Bitcoin Core node running on Azure. JSON messages compose most of the data exchange from data sources to the ingestion engine, and whenever data is missing, raw HTML data is fetched and parsed. Data are then processed in the ingestion engine and saved locally in the data storage using Pandas [4]. While data are being processed, the ingestion engine/pre-processing is selecting/extracting features which are eventually stored locally as NumPy text files [50], and used as training sets for the ML model, which uses a TensorFlow [1] back-end with Keras [10] modules. We will discuss which features are selected and why in Section 5.2.

[13]More than $10x$ faster. In 30 seconds, using APIs from blockchain.com, we fetch information on transactions included in 10 blocks.

Our local data view separates blocks and transactions, such that their information is stored in different datasets. This is to avoid redundancies and to save disk space, since some block information is deep-seated in every transaction, and therefore, repeated thousands of times in one single block. Hence, we consider $\mathcal{D}_T$ and $\mathcal{D}_B$ as datasets containing, respectively, raw transactions and raw blocks, meaning that information is stored as it was fetched, before being elaborated or engineered. Every transaction is linked to a block with a many-to-one relation using block hash (ha) as unique key, $r : \mathcal{D}_T \rightarrow \mathcal{D}_B$ where $\forall ha' \in \mathcal{D}_T \exists ha'' \in \mathcal{D}_B$ st $ha' = ha''$. We then construct the *raw dataset* as $\mathcal{R} = \mathcal{D}_B \bowtie \mathcal{D}_T$. Each row of $\mathcal{R}$ represents an instance of a raw transaction t, at time of its approval. Each column in $\mathcal{R}$ identifies values of a specific feature, and a column key, represented as k_i, is the feature name at i-th position. We consider $K_{\mathcal{R}}$ (more generically K) to be the set containing all the keys in $\mathcal{R}$.

5.2 Feature Selection and Extraction

The feature engineering process is based on fairness and revenue concepts outlined in Section 4. We distinguish between feature selection and extraction and refer to the former as a subset of attributes already present at the time of fetching (e.g., transaction size), while the latter are generated during the pre-processing phase. They are intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps in order to determine a transaction inclusion pattern. Ingestion engine and pre-processing create a local, run-time only, version of $\mathcal{R}$, which we call complete dataset C. Each row of C represents an instance of a feature vector $\boldsymbol{t}$, which identifies a transaction during a well-defined time slot. We define then $\boldsymbol{T}$ as the *complete* block-epoch-based representation of a transaction $t \in \mathcal{R}$, since it contains information belonging to the same transaction for its entire lifespan $\mathcal{L}(t)$.

Definition 5.1 (Feature Vector). A feature vector $\boldsymbol{t}$ is a list of keys (k) that identify a block-epoch-based transaction $\boldsymbol{T}$ in a specific time slot, and it is defined as $\boldsymbol{t} = [k_1, k_2, \ldots, k_n]$ with $|K| = n$.

Definition 5.2 (Complete Transaction $\boldsymbol{T}$). A Multivariate Time Series (MTS) $\boldsymbol{T} = [\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_\gamma]$ is an ordered set of feature vectors $\boldsymbol{t}$, and consists on γ different univariate time-series with $\boldsymbol{t} \in \mathbb{R}^n$, $\forall \boldsymbol{t} \in \boldsymbol{T}$, and $|K| = n$.

C is used to create the training/test datasets $\mathcal{X}$, where $|C| > |\mathcal{X}| > |\mathcal{R}|$ and $K_{\mathcal{X}} \subset K_C \subset K$. To extract new features we apply a function to some values in the feature set K. Formally, to generate a new key k' we define $f : K' \longrightarrow \mathbb{R}$ st: $K' \subset K$. Pre-processing layer and ingestion engine exchange and update the information so that $\mathcal{R}$ contains newly generated features.

5.2.1 Fee-Functions: f_ϕ and f_ρ. The fee-functions include revenue-based techniques which calculate transaction fee and feerate. Transaction fee is a relevant information for our model, and it is calculated using the function f_ϕ, which resembles Algorithm 1, and it is based on transactions' inputs and outputs. If, a transaction t has n inputs and m outputs, we formalize Equation (14) for calculating transaction fee.

$$\phi = \sum_{i=1}^{n} in_i - \sum_{j=1}^{m} ou_j. \tag{14}$$

We identify the fee-function f_ϕ as: $(in, ou) \mapsto \phi = f_\phi(in, ou)$. Always based on the revenue principle, the feerate-function f_ρ wants to get information on transaction feerate. As outlined in Equation (6) represents the way feerate is calculated, formally, we define f_ρ as: $(\phi, q) \mapsto \rho = f_\rho(\phi, q)$.

5.2.2 Pending-txs-Function: $f_{\mathcal{P}}$. This fairness-based function aims at quantifying the belonging of a transaction t to the set $\mathcal{P}$, to see whether such a transaction has chance of getting into $\mathcal{A}$ anytime soon. The function $f_{\mathcal{P}}$ generates feature $\Delta\mathcal{P}$, described in Equation (15), and graphically

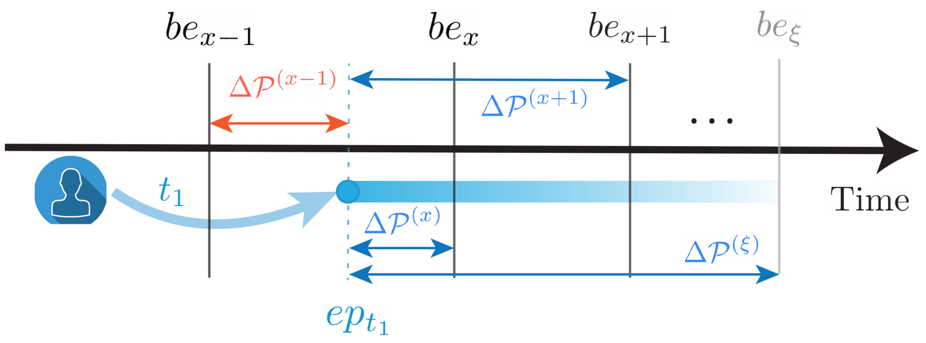

be_{x-1}
be_x
be_{x+1}
be_ξ
$\Delta\mathcal{P}^{(x-1)}$
$\Delta\mathcal{P}^{(x+1)}$
t_1
Time
$\Delta\mathcal{P}^{(x)}$
$\Delta\mathcal{P}^{(\xi)}$
ep_{t_1}

Fig. 4. A transaction t_1 submitted at time ep_{t_1} and yet-to-be included, such that $\mathcal{L}(t_1) = \mathcal{L}^{\xi}_{t_1}$, takes different $\Delta\mathcal{P}$ values over time. For instance, at height $x + 1$, $\Delta\mathcal{P}^{(x+1)} = be_{x+1} - ep_{t_1}$. Blue $\Delta\mathcal{P}$ represents a positive number, while the red $\Delta\mathcal{P}^{(x-1)}$ indicates a negative value.

explained in Figure 4. Specifically, $\Delta\mathcal{P}^{(x)}$ represents how much a transaction $t^{(x)}$ is waiting, from its inception, until the nearer block epoch be_x. We note that if $t^{(x)} \notin \mathcal{P}^{(x)}$ then $\Delta\mathcal{P}^{(x)}$ has a negative value, which is the case of $\Delta\mathcal{P}^{(x-1)}$ in Figure 4.

$$\begin{aligned} \Delta\mathcal{P}^{(x)}_t &= {\beta_t}^{(x)} - ep_t, \\ (\beta, ep) \mapsto \Delta\mathcal{P} &= f_{\mathcal{P}}(\beta, ep). \end{aligned} \tag{15}$$

In the case a transaction is not included yet, we calculate $\beta_t = be_\xi$. With the feature $\Delta\mathcal{P}$, we assume that a transaction can not wait forever for approval if it has paid a fair amount of fee. Furthermore, since we know γ to be the number of $\boldsymbol{t}$ instances in $\boldsymbol{T}$, we can also quantify time, in relation to the number of blocks that $\boldsymbol{T}$ has seen before being approved, so to consider both, an absolute view of time in seconds, and a relative view in γ-occurrences. Consequently, a block-epoch-based transaction has γ-different instances of $\Delta\mathcal{P}$.

5.2.3 *Offset-Function: f_δ.*

The offset-function f_δ is a revenue-based solution. It orders pending transactions according to their feerate, taking into account block space limitations. This function generates a new feature, the *offset*, represented with δ. For each block-epoch-based transaction $\boldsymbol{t}$, its offset value at height x, indicates the amount of bytes already occupied in the block space, from unapproved transactions with a higher feerate, and it is represented as $\delta^{(x)}_t$. The offset then is relevant to give each transaction a place in the future block, greater is the offset and fewer are the chances that $t \in \mathcal{A}^{(x)}$. If we now consider the set S at height x, containing n transactions ordered by feerate, the offset value at any index i is given by

$$\begin{aligned} \delta^{(x)}_i &= \sum_i^n q_i, \\ \left(S^{(x)}, q\right) &\mapsto \delta = f_\delta\left(S^{(x)}, q\right). \end{aligned} \tag{16}$$

As with $\Delta\mathcal{P}$, for each block-epoch-based transaction, there are γ different occurrences of offset values. Algorithm 2 lists how to calculate the offset for transactions in $S^{(x)}$. If compared to other features, this one requires more computational overhead, and the procedure DEFINES in Algorithm 2 counts a time complexity of $O(n^2)$, where n is the number of transactions in $S^{(x)}$. The OFFSET execution time (Algorithm 2) is upper bounded to $O(n)$, the latter procedure gets executed n times, and consequently, the total number of operations is derived from $\sum^n_{i=0,j=0} ij \sim O(n^2)$.

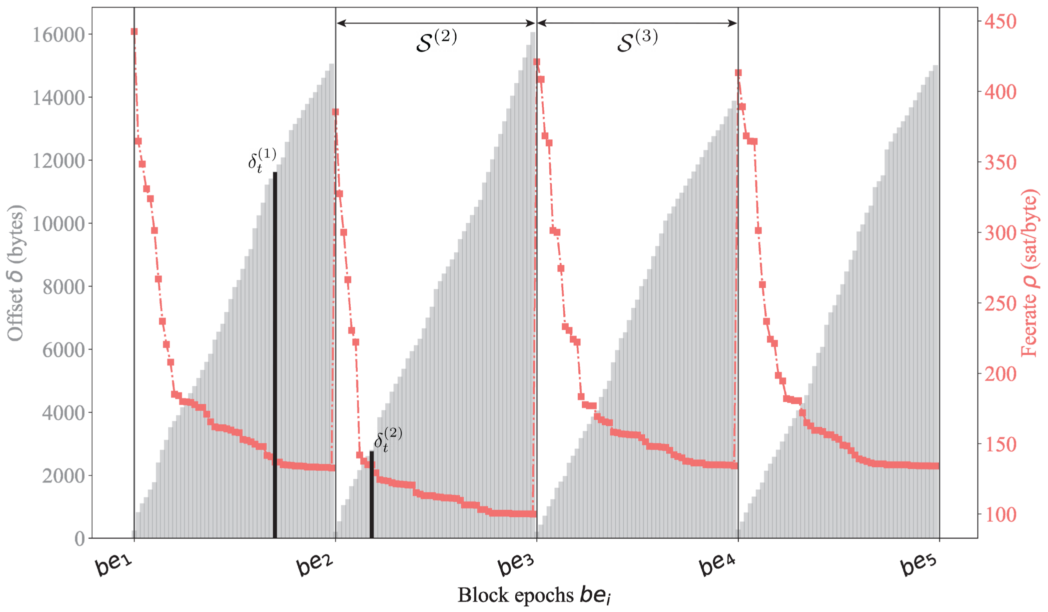


Fig. 5. Data sampled for five consecutive blocks. In each block-epoch slot, transactions are ordered by feerate, then the offset is calculated. We see in this example that the offset value for transaction $t^{(x)}$ changes according to the block-epoch. Offset value for $t^{(1)}$ for instance, is higher than its offset at epoch $x = 2$, of $t^{(2)}$. Thus, $\delta_t^{(1)} \gg \delta_t^{(2)}$.

Figure 5 shows the offset trend over time, for five consecutive blocks. In each block-epoch time frame, from be_1 to be_5, transactions waiting to be approved are ordered in the mempool by feerate, and then their relative offset value is calculated. For instance, transaction $t^{(x)}$ appears in the pool at height $x = 1$ and $x = 2$, with different offset values. Same transactions can then carry different information based on their block-epoch snapshot, thus, adding valuable knowledge about network status, at every block-epoch time slot.

5.3 Prediction Model

The main purpose of the prediction model is to define whether or not a transaction t at height x is likely to be included in $\mathcal{A}^{(x)}$. As explained in Section 5.2, the ML model gets as input a training/test set $\mathcal{X}$, which is the set identifying both the training and the test datasets, respectively $\boldsymbol{X}$ and $\boldsymbol{Xte}$. Following definitions of $\boldsymbol{t}$ and $\boldsymbol{T}$ (5.1–5.2), we represent the training/test dataset $\mathcal{X}$ as an M-dimensional MTS collection of pairs $(\boldsymbol{T}_i, \mathbf{Y}_i)$, represented as $\mathcal{X} = \{(\boldsymbol{T}_1, \mathbf{Y}_1), (\boldsymbol{T}_2, \mathbf{Y}_2), \ldots, (\boldsymbol{T}_M, \mathbf{Y}_M)\}$, where $\mathbf{Y}_i$ is the corresponding one-hot label vector to a certain transaction $\boldsymbol{T}_i$. For a value $\gamma_i = |\boldsymbol{T}_i|$, the one hot label vector $\mathbf{Y}_i$ is a vector of length γ_i where each element $j \in [1, \gamma_i]$ is equal to 1 if $\boldsymbol{t}_j$ represents the time slot of $\boldsymbol{T}$'s inclusion, and 0 otherwise.

The ML framework outputs a model for transaction inclusion. The latter will be used later on to predict whether or not a certain transaction will be included in the next mined block. Models need to be updated and trained/tested at least every week, in order for them to be accurate, and to have enough knowledge of the current network status. When the trained model gets a transaction t as input, it outputs a vector $\boldsymbol{\theta}_t$, which represents the confidence for each transaction to be included or not, in the next mined block. Since this is a binary classification, we represent $\boldsymbol{\theta}_t = [P_t(v_0),\ P_t(v_1)]$,

ALGORITHM 2: Defining S and δ

```
1:  procedure DEFINES($x$)                    ▷ add and order pending transactions in $S^{(x)}$
2:      $S' = \{t | be_x \leq ep_t < be_{x+1}\}$    ▷ $S'$ contains all transactions occurred between $[be_x, be_{x+1}]$
3:      $S \leftarrow \varnothing$
4:      for all $t \in S' \cup \mathcal{P}^{(x)}$ do    ▷ add transactions that are waiting and are in $\mathcal{P}^{(x)}$
5:          $\rho_t \leftarrow f_\rho(\phi_t, q_t)$    ▷ calculate feerate of $t$
6:          $S \cup \{t\}$    ▷ add $t$ to the set $S^{(x)}$
7:      sort($S$, $\rho$)    ▷ order the set $S$ by feerate in ascending order
8:      for all $t \in S$ do
9:          $\delta_t \leftarrow$ OFFSET($t$, $S$)    ▷ set the offset for every transaction in $S$
10:
11: procedure OFFSET($t$, $S$)    ▷ index $i$ is the place of $t$ in the set $S$
12:     $\delta_t \leftarrow 0$
13:     for all $i \in S$ after $t$ do    ▷ for every transaction in set $S$, ordered after $t$
14:         $\delta_t \leftarrow \delta_t + q_i$
15:     return $\delta_t$
```

ALGORITHM 3: Normalization of $\boldsymbol{X}$

```
1:  procedure NORMALIZATION($\boldsymbol{X}$, $\boldsymbol{Xte}$)
2:      $\mu \leftarrow$ Mean($\boldsymbol{X}$)    ▷ expected value
3:      $\sigma \leftarrow$ Std($\boldsymbol{X}$)    ▷ standard deviation
4:      $\boldsymbol{X}_{norm} \leftarrow (\boldsymbol{X}-\mu)/\sigma$    ▷ set the normalize training set
5:      if $\boldsymbol{Xte}$ then    ▷ if the algorithm is in testing phase
6:          $\boldsymbol{Xte}_{norm} \leftarrow (\boldsymbol{Xte}-\mu)/\sigma$    ▷ set the normalized testing set
7:          return $\boldsymbol{X}_{norm}, \boldsymbol{Xte}_{norm}$
8:      else
9:          return $\boldsymbol{X}_{norm}$
```

where v_0 is the class representing a *non-inclusion* in the next block, while v_1 represents an *inclusion*. In other words, $\boldsymbol{\theta}_t$ represents the probability, $P(v_i)$, of t to fall in the class v_i, with $i \in \{0, 1\}$.

We perform a supervised classification, which means that we know a-priori the outcome of transactions in $\boldsymbol{X}$, and use this information to test the model accuracy during the training phase, which is the purpose of labels $\mathbf{Y}$. Part of $\mathcal{X}$ is then used for testing, and $\boldsymbol{Xte}$ represents the respective testing set of $\mathcal{X}$. In our model implementations, we dynamically change the number of hidden layers during the validation phase. We use a **Rectified Linear Unit** (**ReLU**) function for each node in the NN, except for the output layer where we use a Normalized Exponential Function (or softmax). The weights are initialized with He normalization, which takes into account Rectified Linear Unit (ReLU) and makes it easier for deep models to converge [25]. Parameters for data normalization are set before the training phase, the set $\boldsymbol{X}$ is normalized using its expected value and standard deviation, as showed in Algorithm 3. If we are in the testing phase, then the algorithm normalizes the testing set $\boldsymbol{Xte}$, according to the mean and standard deviation of $\boldsymbol{X}$. In order to balance the set and be unbiased in the training phases, data normalization is a necessary measure to be adopted since features have different orders of magnitude.

Parameters of the DNN classifier which cannot be estimated from data, known as hyperparameters, are set manually by trial and error, including the number of hidden layers, the number of skip connections, the batch size, and the number of epochs for each model. The batch size controls the

Table 1. Disk Space Occupied by Instances of Different Sets if they Contain Information About 100k, 500k, 1M, and 5M Transactions

Disk space of different datasets				
Dataset	100k txs	500k	1M	5M
$\mathcal{R}$	~29 MB	~145 MB	~290 MB	~1.2 GB
$\mathcal{C}$	~40 MB	~730 MB	~1.6 GB	~13.7 GB
$\mathcal{X}$	~7 MB	~130 MB	~288 MB	~2.4 GB
₿	~60 MB	~300 MB	~600 MB	~3 GB

The last row shows the corresponding space taken by the same amount of transaction in the actual Bitcoin blockchain. The sizes reported in this table consider a worst-case scenario for our datasets and an optimal one for the Bitcoin blockchain.

granularity or precision of gradient descent, so the model internal parameters are optimized for every *batch size* of tuples. The number of epochs instead represents the number of times that the learning algorithm will work through the entire training dataset, ideally getting closer to the optimal solution at every iteration. The model's hyperparameters are configured in order to optimize the model performance and accuracy. In the next section we present experiments and evaluations of the ML model described so far.

6 EVALUATION

In this section we present the evaluation of our ML model. In Section 6.1, we describe the datasets used for both training and testing, and the selection of features. In Section 6.2, we list our evaluation metrics and what we define as *model accuracy*. Finally, in Section 6.3, we show the results of our analyses in terms of overall accuracy, the importance of the selected features, and model cyclicity, that is, how much information of the current network status can influence model prediction if the model is updated cyclically.

6.1 Experimental Setup

We conducted experiments on a large scale with data from the Bitcoin blockchain. We analyzed more than 30 million transactions across 15 thousand blocks between January 2021 and May 2021. In Sections 5.1–5.2, we introduced datasets $\mathcal{R}$, $\mathcal{C}$, and $\mathcal{X}$. Table 1 shows the required space on the disk of those three datasets, when they store information about 100 thousand to 5 million transactions. The raw dataset $\mathcal{R}$ contains information about blocks and transactions as it was collected from the blockchain. Nothing is added, and redundant or irrelevant information is discarded in order to save disk space. The dataset $\mathcal{C}$ is used to generate the block-epoch-based training/test dataset $\mathcal{X}$, and considering $\mathcal{C}$'s demanding storage requirements at scale, only a copy of the lighter dataset $\mathcal{X}$ is stored locally. The dataset $\mathcal{C}$ as can be observed, has a superlinear growth. This is as expected if we consider that $\mathcal{C}$ stores all transactions in $\mathcal{R}$, plus it keeps track of block-epoch dependent features described in Section 5.2, and finally it has knowledge about $\mathcal{P}$ and $\mathcal{S}$ sets.

We fetch data and store locally different $\mathcal{R}$ instances, one per each month of evaluation, and list them in Table 2. For each period we fetch the same number of blocks (3,010 in our analysis), and for every dataset $\mathcal{R}^i$ we create a prediction model based on the inclusion pattern defined in Section 4. A $\mathcal{C}^i$ dataset is generated at run-time, and from it, new information and features are extracted. Finally, data from $\mathcal{C}^i$ are selected to form the training/test dataset of features $\mathcal{X}^i$. For every period i, we train one model with the first 50%-occurred transactions in $\mathcal{R}^i$. Each tested set is labeled with hyperparameters defining how *complete* and *new* the considered set is. We define these hyperparameters as α and ψ. We identify $\alpha = |\boldsymbol{Xte}|/|\boldsymbol{X}|$, as the fraction of transactions

Table 2. Tests and Evaluations are Performed on these Raw Time-Series Datasets

Raw dataset for each period				
i	Date	$\lvert\mathcal{R}^i\rvert$	$\mathcal{R}^i$ size	₿ price
1	January	6.5M	1.09 GB	\$29 k–\$35 k
2	February	6.7M	1.12 GB	\$33 k–\$56 k
3	March	6.2M	1.05 GB	\$49 k–\$58 k
4	April	6.2M	1.04 GB	\$58 k–\$56 k
5	May	5.2M	877 MB	\$58 k–\$36 k

3,010 blocks are analyzed every month. $|\mathcal{R}^i|$ is the number of transactions analyzed in each set, while $\mathcal{R}^i$ size identifies the set's space on disk. We also include the Bitcoin price at time of evaluation to discuss any correlation between model prediction and coin price at that time.

used for testing over the total number of points used for training e.g., if $\alpha = 0.5$ the amount of transactions used for testing is half of the one used for training. This value is important to have an accurate prediction when live information is not available. In our case, we preferred to conduct experiments using millions of older transactions fetched and stored locally. In this way, we know a-priori their inclusion, and consequently, it is easier to evaluate large datasets quicker. As $\alpha \rightarrow 1$ the set becomes *complete*, and the offset value constructed for testing is close-enough to the one used for training, therefore we reduce false-positives points if $\alpha \ll 1$ or false negatives if $\alpha \gg 1$. We say that a complete set has an accurate view of the mempool size over time. The other hyperparameter, ψ, represents the distance (time-wise) of test set from training. Let `mo` be the difference between the test month and the training month, the value ψ is normalized through a sigmoid function so to have a bound of $[0, 1]$, such that: $\psi = \text{sigmoid}(\texttt{mo})$. If we use training and test data from the same month, we have $\psi = \text{sigmoid}(0) = 0.5$, and we consider the model *new*. If the test is done with an older model, we have $\texttt{mo} > 0$, and consequently $\psi \rightarrow 1$. On the other hand, if we train the model a-posteriori and want to test older data, $\texttt{mo} < 0$ and $\psi \rightarrow 0$. This parameter indicates how new the transactions tested are compared to the ones trained. Therefore ψ measures how updated the model is in relation to the current network status. Furthermore, information on ψ is useful later on, when model cyclicity is analyzed.

The features we select to be trained (from C to $\mathcal{X}$), are partially fetched from the blockchain and some others are engineered as seen in Section 5.2. Following our assumptions, $\mathcal{X}$ should contain information about fairness and revenue (Sections 4.2–4.3). We train our models with the following features: $K_{\mathcal{X}} = [\phi, q, \rho, \Delta\mathcal{P}, \delta, \Delta\mathcal{P}_N, \delta_N]$, where $\Delta\mathcal{P}_N$ and δ_N are normalized values for respectively, $\Delta\mathcal{P}$ and δ. The first one represents the waiting time $\Delta\mathcal{P}$, as the number of created blocks a certain transaction $\boldsymbol{t}_\gamma$ has seen from its inception (Equation (17)).

$$\Delta\mathcal{P}_N^{(\gamma)} = \sum_{i=0}^{\gamma} \omega^{(i)} \text{ with,} \qquad \omega^{(i)} = \begin{cases} 0, & \text{if } \mathbf{t}_i \notin \mathcal{P}^{(i)}, \\ 1, & \text{if } \mathbf{t}_i \in \mathcal{P}^{(i)}. \end{cases} \tag{17}$$

The second one, δ_N, is the offset normalized with the maximum block space (~1.1 MB), so that δ_N is a percentage which tells how much the mempool is already occupied by richer transactions (Equation (18)).

$$\delta_N^{(x)} = \frac{\delta^{(x)} \times 100}{Q}. \tag{18}$$

6.2 Evaluation Metrics

Evaluating our ML model is an essential part of this study. While classification accuracy is still our main evaluation metric, sometimes it is not enough to truly judge our model. For this, we also consider a confusion matrix and area under curve (AUC) evaluation. We briefly describe them and then present our results in Section 6.3.

6.2.1 Classification Accuracy. When we evaluate our model, we initially refer to its broad-accuracy-value as the so-called *classification accuracy*, which is the ratio of number of correct predictions to the total number of input samples:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}.$$

This metric is immediate and easy to calculate, and we use it as a general measure for comparing accuracy between different models. However, classification accuracy does not always represent an accurate model evaluation since it works well only when a homogeneous class distribution is studied. Our classes proved themselves to be quite unbalanced, and since we did not want to reduce the number of sampled data, we opted to include other metrics for evaluating the model.

6.2.2 Confusion Matrix. Confusion matrix allows to visualize how accurate our model is to make predictions over the binary classification problem (v_0, v_1). We refer to the following definition of confusion matrix **CM**, also formalized in Table 3:

$$\begin{aligned} \mathbf{CM} = \mathbf{Fr}^{-1} \cdot \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}, \\ \text{where} \quad \mathbf{Fr}_{2,2} = [b_{ii}], \quad b_{ii} = \sum_j a_{ij}. \end{aligned} \tag{19}$$

a_{ij} is the number of elements that truly belongs to i but were classified in j. The metrics we use, obtained from **CM**, are the *recall* and the *precision*. The recall R_i of a specific class v_i, represents the number of $t \in v_i$ which were correctly classified in class v_i, while the precision P_i for class v_i represents the number of data points classified in v_i which actually belongs to v_i.

$$R_i = \frac{a_{ii}}{\sum_{j=0}^{1} a_{ji}}, \quad P_i = \frac{a_{ii}}{\sum_{j=0}^{1} a_{ij}}. \tag{20}$$

6.2.3 Area Under Curve. AUC is a widely used metric for binary classification problems. Area Under Curve (AUC) represents the probability that our model will rank a randomly chosen $t \in v_1$ higher than a randomly chosen $t \in v_0$. AUC has a range value of $[0, 1]$, and the greater the value, the better is the classifier performance. If AUC = 1 then the classifier is able to perfectly distinguish between the two classes, if AUC = 0.5 the model is not able to distinguish between v_0 and v_1 class points, while if AUC = 0, the classifier would be predicting all v_0 as v_1 and vice-versa. **Area Under Curve of Receiver Operator Characteristic** (**AUC-ROC**) is a very important metric for our evaluations, and specifically, we calculate Area Under Curve of Receiver Operator Characteristic (AUC-ROC) following the scheme in Table 3, and according to

$$\begin{aligned} TPR = \frac{TP}{TP + FN}, \quad FNR = \frac{FN}{TP + FN}, \\ TNR = \frac{TN}{TN + FP}, \quad FPR = \frac{FP}{TN + FP}. \end{aligned} \tag{21}$$

True Positive Rate (*TPR*) identifies R_1, or *Sensitivity*, True Negative Rate (*TNR*) represents the *Specificity*, so what proportion of the negative class v_0 was correctly classified. False Positive Rate (*FPR*) is equal to 1 − Specificity, and it represents the proportion of v_0 that was incorrectly classified.

Table 3. Confusion Matrix Model

		Actual values	
		0	**1**
Predicted values	**0**	True Negative	False Negative
	1	False Positive	True Positive

From this model, we base the calculation of AUC-ROC.

Table 4. Classification Accuracy for each Month, from January Until May

Classification accuracy							
α	ψ	January	February	March	April	May	**Overall**
1	0.5	90.07%	90.9%	91.08%	85.52%	88.29%	89.17%

The overall accuracy represents the average classification accuracy for the whole period of analysis. The optimal case scenario for hyperparameters α and ψ is presented, in this way the model is complete and updated in relation to the data tested.

Finally, False Negative Rate (*FNR*) indicates what proportion of the positive class was incorrectly classified. The representation of the **Receiver Operator Characteristic** (**ROC**) curve and its consequent AUC, has *FPR* on x-axis and *TPR* on y-axis.

6.3 Analyses

In Section 6.3.1, we present the overall classification accuracy of our classifier, following the evaluation metrics defined in Section 6.2. We list classification accuracy for each month of the evaluation, from January 2021 until May 2021, with an optimal scenario of $\alpha = 1$ and $\psi = 0.5$. A confusion matrix representing the whole period is also presented. In Section 6.3.2, we discuss the importance of selecting the right features, how accuracy changes with or without certain information, and how the results might diverge if the wrong assumptions are made. Also in these experiments, hyperparameters are set as an optimal case scenario. Finally, Section 6.3.3 outlines the importance of keeping the model updated. Different hyperparameters are evaluated and an AUC-ROC score for each of them is presented and compared.

6.3.1 Overall Accuracy. We present here the overall classification accuracy for each month of training, by setting hyperparameters with their optimal values of $\alpha = 1$, $\psi = 0.5$. We show the overall classification accuracy for each month in Table 4, while the **CM** for all the points analyzed from January until May, of over 30 million transactions, is represented in Table 5. We observe that during the whole training/test period, the overall accuracy of the model is above 90% for three out of five months, while the model struggles between April and May, during the coin price plunge (discussed in Section 7). Despite that, the model appears to be solid even in our worst-case scenarios, and the confusion matrix in Table 5 shows that the model correctly classifies 91% of transactions in v_0, and 88% in v_1.

Table 5. Overall **CM** Score for Test Ran between January 2021 and May 2021, with Parameters $\alpha = 1$ and $\psi = 0.5$

Overall **CM** score		
	v_0	v_1
v_0	0.91	0.09
v_1	0.12	0.88

This matrix shows how the 30+ million transactions were classified using five different models, one for each month, with a complete and updated view. False negative transactions represents 9% of the negative ones, while false positive 12% of the positive ones.

Table 6. Classification Accuracy for Different $K_{\mathcal{X}}$ Sets

Classification accuracy for $K^n_{\mathcal{X}}$						
n : **type**	January	February	March	April	May	**Overall**
1 : **Primitive**	75.13%	78.54%	77.77%	62.52%	69.97%	72.78%
2 : **Fairness**	84.57%	86.24%	84.63%	83.64%	82.11%	84.23%
3 : **Revenue**	88.24%	87.73%	89.4%	80%	86.21%	86.31%
4 : **Complete**	89.51%	90.36%	90.04%	85.35%	88.23%	88.69%

Each set of selected features represents a different assumption we made in the analysis phase. From the concept of inclusion as merely transaction fee and transaction size, to the concepts of fairness and revenue we explained.

6.3.2 Selected Features. In order to validate our assumptions, we verify and test how important some features are in predicting transaction inclusion. In the following results we show the classification accuracy and confusion matrix for trained models with only a specific subset of features $K^n_{\mathcal{X}} \subset K_{\mathcal{X}}$. We evaluate four different sets of features where $n = [1, 2, 3, 4]$, and each number identifies a different type of evaluation: (1) *Primitive information*, the set $K^1_{\mathcal{X}}$ identifies only fetched features such as transaction size and fee. This information is used by many fee predictors, with poor results about fee overpaying, $K^1_{\mathcal{X}} = [\phi, q]$; (2) *Fairness assumptions*, the set $K^2_{\mathcal{X}}$ excludes features based on the revenue principle, so a miner needs to only be fair in order to include transactions in the next block, $K^2_{\mathcal{X}} = [\phi, q, \Delta\mathcal{P}, \Delta\mathcal{P}_N]$; (3) *Revenue assumptions*, the set $K^3_{\mathcal{X}}$ excludes features based on the fairness principle, to monitor how much revenue impacts the transaction inclusion pattern, $K^3_{\mathcal{X}} = [\phi, q, \rho, \delta, \delta_N]$; (4) *Assumptions-based*, the set $K^4_{\mathcal{X}}$ includes only extracted features, this evaluation aims at verifying the reliability of our initial assumptions, including both, fairness and revenue concepts, $K^4_{\mathcal{X}} = [\rho, \Delta\mathcal{P}, \Delta\mathcal{P}_N, \delta, \delta_N]$.

The experiments performed for this purpose have a set up of $\alpha = 1$ and $\psi = 0.5$, classification accuracy results are shown in Table 6, and the related plot is presented in Figure 6. The DNN classifier struggles when the Bitcoin price drops drastically over the month, e.g., in April. In fact, we observe that when only primitive information is used, the accuracy drops 15% from the previous month, while with complete information the accuracy drop is 5% only. The impact of our assumptions on predicting transaction inclusion is relevant. The model can perform from 12% to 14% better on average, if fairness and revenue principles are applied *separately*, to data that most fee predictors use, and up to 23% if they are combined. To be noted is the variance of $K^2_{\mathcal{X}}$ classification accuracy since it appears to be smaller than the one of $K^3_{\mathcal{X}}$. This highlights the importance of the fairness concept in order to delineate transaction inclusion. In the month of April, we observe

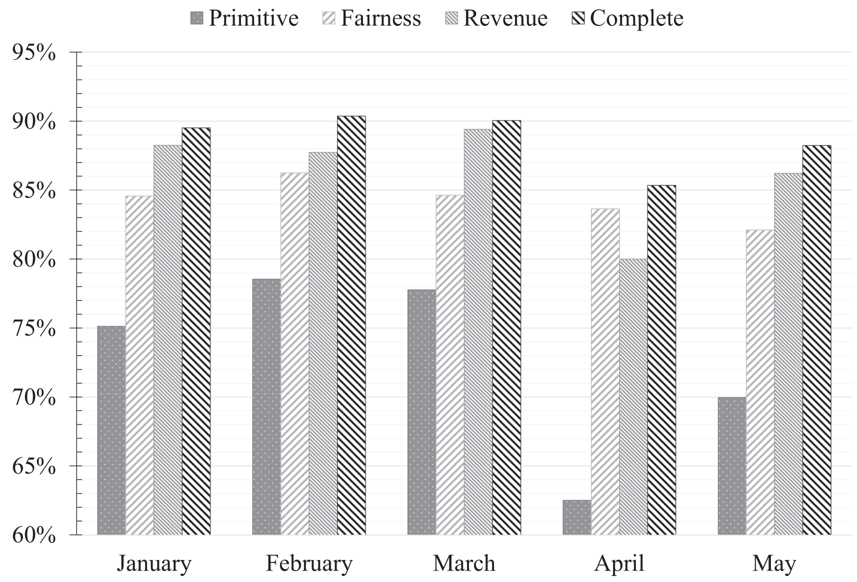


Fig. 6. Classification accuracy results for each one of our assumption sets, from primitive to complete, for each month of analysis from January 2021 until May 2021. The model accuracy increases considerably if a complete feature set is used, compared to the basic idea of using only transaction size and transaction fee as features.

an inversion of the Bitcoin price uptrend, and miner revenue was at its all-time-high. This outlines how miners are deviating from the revenue principle more than the fairness one.

6.3.3 *Model Cyclicity.* In this paragraph, we deviate from the optimal hyperparameter settings used so far, in order to highlight how much model classification accuracy could benefit if well-formed data are used for training. Well-formed data creates what we call an *updated* model, or a model trained cyclically over time, and this information should be *complete* ($\alpha = 1$, for a correct offset value) and *new* ($\psi = 0.5$, transactions should be tested with a relatively recent model). We note that, if we deviate from the mean value of $\psi = 0.5$ towards 1, the test sets occur after the training, while if the value ψ tends to 0, the model used is newer than the transactions tested. Despite this being an unrealistic scenario, we want to monitor the model's behavior with both, older and newer transactions. We test for each month, all the possible combinations between the values $\alpha = [0.05, 0.1, 0.5, 1]$, and $\psi = [0.01, 0.05, 0.12, 0.26, 0.5, 0.73, 0.88, 0.95, 0.98]$. Results are showen in Figure 7, where each point represents the average classification accuracy over five months if hyperparameters are changed according to α (x-axis) and ψ values (y-axis). The plot shows that the accuracy is higher (yellow) when $\psi \longrightarrow 0.5$ and $\alpha \longrightarrow 1$, but as the offset precision diminishes (lower α), then model cyclicity (ψ) becomes less important. Here is outlined how much the combination of both hyperparameters is significant, ψ becomes relevant when it has the right information about the mempool size, provided by the right choice of α. In this way, the offset contextualizes the model over time, and without an accurate calculation of it, the model would just predict according to the fairness concept, which seems to be less time-dependent than offset (Figure 6, Fairness bar).

Figure 8 shows the AUC-ROC curves for five tests performed with different hyperparameter values, chosen to be representative for *optimal*-, *average*-, and *worst*-case scenarios. The parameters chosen are represented in Table 7. We identify two average cases, and two worst cases, describing if the testing occurred before (−) or after (+) the training. In the optimal case, the model can

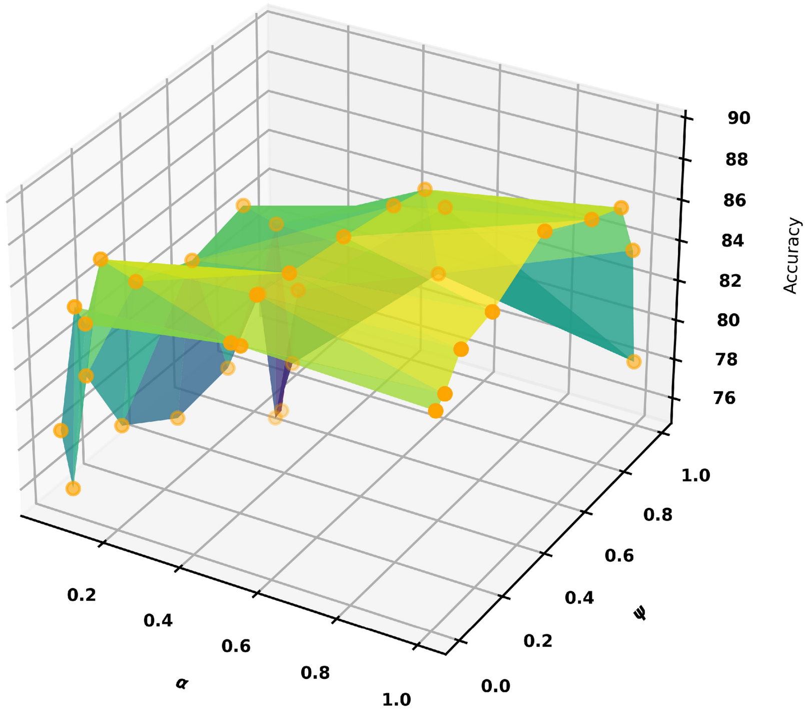


Fig. 7. Classification accuracy value (z-axis) for various models, tested with different hyperparameters setup. The y-axis, with the ψ parameter, represents how much a certain model is updated, while the x-axis indicates the model's completeness (α parameter). The importance of model cyclicity is highlighted in order to improve classification accuracy.

distinguish between classes with an area under the curve of 0.97, which is a solid classification result. Even if we accept an *FPR* of 10%, the *FNR* does not go higher than 10%, and if we accept an *FPR* of 20%, the *FNR* is kept below 5%.

7 DISCUSSION

The extensive analysis we performed on Bitcoin, outlined difficulties in ensuring a low-payment scheme for users. We explained in Section 3 how the interblock interval time and the block size constraints negatively affect the system's throughput. Fee markets emerge as a means to provide a stable income for miners. This leads fee estimators to overprice transaction fees in order to guarantee an immediate inclusion in the blockchain. However, this results in users having to pay a fee two orders of magnitude higher than the recommended one. To optimize user expenditure we define our view of a transaction inclusion model and then save data locally for the analysis. The inclusion model formalized and described in Sections 4–5 is fundamental to extract the right features for an accurate classification. With this, if models are cyclically trained at least once a month, it is possible to collocate new incoming transactions in classes v_0 and v_1, with an accuracy score on average of 89%.

As Table 1 shows, the dataset we generate is efficient in terms of disk space even in the worst-case scenario for our datasets (Table 2 lists the actual dataset sizes for our analysis). If the raw dataset ($\mathcal{R}$) size is compared to the actual blockchain size, then $\mathcal{R}$ saves on average 54% of disk space. Furthermore, we notice that the training set $\mathcal{X}$ containing block-epoch-based information,

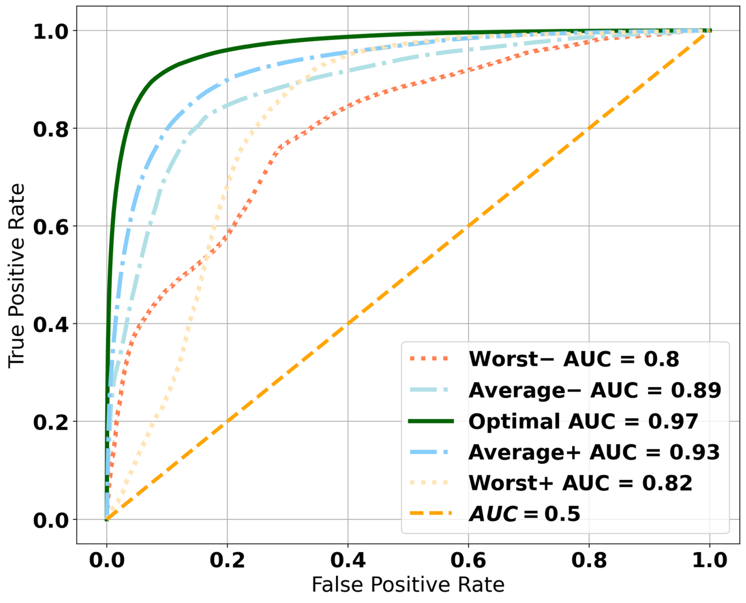


Fig. 8. Five tests to measure AUC-ROC in an optimal, average, and worst-case. The optimal test is run with $\alpha = 1$ and $\psi = 0.5$, identified with a green continuous line. The two average tests are run with $\alpha = 0.2$ and $\psi = [0.12, 0.88]$, represented with a dot-dashed light blue lines. The two worst-case tests are run with $\alpha = 0.04$, $\psi = [0.02, 0.98]$, and they are represented with dotted yellow and red lines. When AUC = 0.5, the classifier is not able to distinguish anymore between positive and negative class points.

Table 7. AUC-ROC Results for Different Hyperparameters

Evaluations on model cyclicity				
type	α	ψ	Accuracy	**AUC-ROC**
Worst-	0.04	0.02	78.63%	0.82
Average-	0.2	0.12	84.71%	0.92
Optimal	1	0.5	91.08%	0.97
Average+	0.2	0.88	81.25	0.89
Worst+	0.04	0.98	70.25	0.8

The optimal evaluation is the one having a 100% complete model, and the test is performed within the same month of training. An average evaluation model is 20% complete and there is a two months deviation from training and testing. The worst evaluation case is when the model is only 4% complete and there is four months deviation between training and testing.

is 54.4% lighter on average than the blockchain original size, with 89% of disk space saved, if smaller datasets are taken into account. For dataset evaluation, we set a 3,000 blocks threshold (~20 days, or ~6 million transactions) because the DNN models we produce are for short-term prediction and, therefore, are more accurate if generated cyclically. The complete set $\langle C \rangle$ is the heavier one, although we never store it locally and it is used only to generate its lighter version $\mathcal{X}$ for the model's training. In Table 2, we represent time-series observational data of the raw datasets $\mathcal{R}^i$, and we observe that the size on disk is 71.5% lower than the corresponding Bitcoin blockchain size.

Results of our experiments are presented in Section 6, highlighting the importance of selecting and generating the right set of features. Building a model using only data fetched from the blockchain, which is the solution adopted by many fee estimators, is trivial. We label this as the

Table 8. Overall **CM** Score for April 2021 Data, using the January 2021 Model

Overall **CM** score, $\psi = 0.95$				
α	0.5		1	
	v_0	v_1	v_0	v_1
v_0	0.95	0.05	0.89	0.11
v_1	0.41	0.59	0.26	0.74

Information of two **CM** tables is represented, the ψ is kept at 0.95 while two different values of α are evaluated, 0.5 and 1. This table shows the importance of having a complete dataset during training, and how classification accuracy can benefit from it.

primitive solution. During the five months of analysis, we test models with the set of engineered features based on our intuition. We call it a *complete* solution, and then we compare it with the primitive one. On average, we obtain an improvement in the accuracy score of 16%. Most importantly, we notice the downtrend of our model accuracy score during the months of April and May. We conjecture that this is caused by a series of events that occurred during these months. Initially, there was a Bitcoin price inversion-trend and its price dropped 46%.[14] Following, more transactions saturated the mempool[15] and transaction fees reached a new all-time-high.[16] Miners revenue reached an all-time-high between April 14th and May 10th,[17] leading us to consider that revenue stopped being miners' main means of inclusion. However, despite relevant and unpredictable exogenous events, our complete solution never fell below 85.35% of accuracy score.

Another fundamental aspect that boosts classifier accuracy score is model cyclicity. Figure 7 shows the importance of keeping the information in the model *complete* and *new*, especially when unexpected events occur. We can evince from the plot that classification accuracy drops when older models are used to classify more recent data, $\psi \longrightarrow 1$, or when information in the test set is not complete, $\alpha \longrightarrow 0$. Also, the accuracy score drops considerably if models prior to the price inversion trend are used to classify more recent data. For instance, Table 8 shows confusion matrices of two different tests. Both represent data fetched in April 2021 and classified with a model from January 2021. One dataset is complete ($\alpha = 1$), while the other one is not ($\alpha = 0.5$). Despite data completeness, the all-time-high fees in April make the model incorrectly classify 26% of transactions in v_1 while they belonged to v_0 (false positive). That means the model classifies more transactions as *included*, while they are not. If we then reduce α to 0.5 the false positives increase to 41%, while the false negative represents only 5%. When such events occur, the model ought to be trained more frequently than once per month. This will reduce the number of misclassified transactions due to some deviations from the previous inclusion pattern. In Figure 6 we show that our intuitions on the selected features are correct, and they are crucial for an accurate classification. Both hyperparameters resulted to be fundamental for boosting the accuracy score. A complete set is needed to have the right information on the mempool size in order to correctly calculate the offset value. An updated (or new) dataset helps to have knowledge of the current miner inclusion trend.

Despite the fact that we obtain a good classification accuracy score, we note that the model could be biased towards a specific range of transaction fees. In fact, bias gets into the model through the data that is used for building the ML model [23]. We carry out a supervised approach, thus our

[14]Bitcoin price dropped from \$ 63,000 to \$ 34,000 starting on April 17th https://www.coindesk.com/price/bitcoin.
[15]Mempool count https://www.blockchain.com/charts/mempool-count.
[16]With an average of \$ 60 per transaction on April 21st. https://www.blockchain.com/charts/transaction-fees-usd.
[17]Miners revenue reached 80 million per day https://www.blockchain.com/charts/miners-revenue.

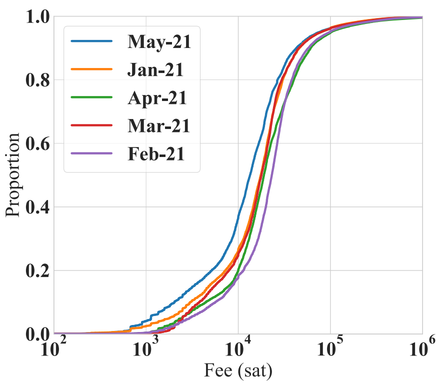


Fig. 9. CDF of transaction fees in our dataset.

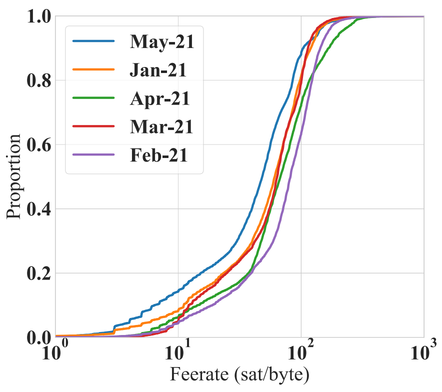


Fig. 10. CDF of transaction feerate in our dataset.

model only knows the outcome for transactions that occurred in the blockchain. If most users pay a fee greater than the optimal value, the model lacks samples for small transaction fee values. Figures 9 and 10 show the fee and feerate **Cumulative Distribution Function** (**CDF**) during the period of analysis. We observe that after the price drop occurred in late April 2021, fees were considerably lower (May 2021), and that transaction fees between 10^4 and 10^5 sat[18] represent nearly the 75% of the total. The recommended feerate should be 1 sat/byte, however, we notice that nearly 0% of transactions have a feerate that low. Consequently, an educated guess is that our model is biased and it is not accurate when predicting transactions with 1 sat/byte of feerate. However, the overall fee trend of approved transactions delineates a pattern itself, meaning that a too low fee will rarely end up in an inclusion, which is in line with our model's outcome. In fact, lower fee transactions are evicted from the network and never included. Our model optimizes expenditure within the range of the already approved transactions and its scope is not to detect possible evicted transactions but to determine an inclusion in the next block. Information about eviction is not of particular relevance. Finally, any transaction issuer could consult the model's output in order to tradeoff its probability of inclusion with more, or less fee to pay.

7.1 Future Work

We are currently working on expanding this work with some experiments related to user's expenditure. The goal is to measure how much an issuer can lower its given fee, by keeping the output confidence of the inclusion model high. By doing so, we can quantify the actual overspending of the Bitcoin network, and we can contribute to improving the overall user experience.

The inclusion model we refer to might change during time. It is useful then to keep on observing blockchain data in order to make new conjectures on inclusion pattern, if we want to use PoW-based systems without overpaying for transaction space in the blocks. We are currently working on a modified version of the aforementioned inclusion model which aims at boosting accuracy score. This approach includes a holistic view of new block alternatives, and considers transaction offset level being penalized if they fall in the > 1 MB space of the mempool. In this way the model has a stronger second-price auction orientation. Furthermore, we aim to organize transactions at every block epoch into ranks, and include transaction rank as a feature for the prediction model. Finally, to reduce training data bias from high-fee transactions, a big enough number of low-fee transactions can be added to the actual blockchain and their latency registered by the model.

[18] 1 satoshi = 0.00000001 Bitcoin.

8 CONCLUSION

The unpredictable fee market in PoW-based blockchains, like Bitcoin and Ethereum, leads to unexpected transaction delays and evictions unless a substantial fee is offered. This has serious economic implications for the end users, as overpaying transaction fees and unstable prices become the norm.

In this article, we present a systematic study of the transaction fee mechanism in Bitcoin and show that the generic information available is not sufficient to delineate a pattern for transaction acceptance. By analyzing the blockchain data for mechanisms and patterns governing miners' decisions to include individual transactions, we devise a novel formal transaction inclusion model that is based on fairness and revenue principles. We show the applicability of our formal model by using it to construct a DNN prototype that predicts transaction inclusion. Despite the limitations of delineating a pattern when the block creation time is a randomized process and the miner's policies of inclusion are unknown, we obtained promising results. When training on more than 30 million transactions over a five months period, we obtained an overall average accuracy of 89%, in spite of the price inversion trend and with peaks up to 91%. The model is therefore capable of predicting transaction inclusion with a notable precision, enabling Bitcoin users to better select a suitable fee paid and probability of transaction inclusion.

ACKNOWLEDGMENTS

We thank Christian Cachin and Ignacio Amores of the Cryptology and Data Security group at the University of Bern for their helpful comments and feedback.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, PeteWarden, MartinWattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). Retrieved April 2021 from https://www.tensorflow.org/. Software available from tensorflow.org

[2] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. 2005. BAR fault tolerance for cooperative services. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles.* ACM, New York, NY, 45–58. DOI: https://doi.org/10.1145/1095810.1095816

[3] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy.* 375–392. DOI: https://doi.org/10.1109/SP.2017.29

[4] Tom Augspurger, Chris Bartak, Phillip Cloud, Andy Hayden, Stephan Hoyer, Wes McKinney, Jeff Reback, Chang She, Masaaki Horikoshi, Joris Van den Bossche, et al. 2012. *Pandas: Python Data Analysis Library.* software v0.21.0. Pandas community. Retrieved from http://pandas.pydata.org/.

[5] RSJD Baker et al. 2010. Data mining for education. *International Encyclopedia of Education* 7, 3 (2010), 112–118.

[6] Soumya Basu, David Easley, Maureen O'Hara, and Emin Gün Sirer. 2019. The old fee market is broken, long live the new fee market. *Hacking Distributed* (2019). Retrieved from https://bit.ly/3gUQaLc.

[7] Soumya Basu, David Easley, Maureen O'Hara, and Emin Sirer. 2018. Towards a functional fee market for cryptocurrencies. *CoRR* abs/1901.06830. http://arxiv.org/abs/1901.06830.

[8] Indranil Bose and Radha K. Mahapatra. 2001. Business data mining—a machine learning perspective. *Information & Management* 39, 3 (2001), 211–225.

[9] Min Chen, Yixue Hao, Kai Hwang, Lu Wang, and Lin Wang. 2017. Disease prediction by machine learning over big data from healthcare communities. *IEEE Access* 5 (2017), 8869–8879. DOI: 10.1109/ACCESS.2017.2694446

[10] François Chollet et al. 2015. Keras. Retrieved April 2021 from https://github.com/fchollet/keras.

[11] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. 2016. On scaling decentralized Blockchains. In *Financial Cryptography and Data Security.* Springer, Berlin, 106–125. DOI:https://doi.org/10.1007/978-3-662-53357-4_8

[12] Jared Dean. 2014. *Big Data, Data Mining, and Machine Learning: Value Creation for Business Leaders and Practitioners.* John Wiley & Sons.

[13] C. Decker and R. Wattenhofer. 2013. Information propagation in the Bitcoin network. In *Proceedings of the IEEE P2P 2013.* 1–10. DOI:https://doi.org/10.1109/P2P.2013.6688704

[14] Javier Diez-Sierra and Manuel del Jesus. 2020. Long-term rainfall prediction using atmospheric synoptic patterns in semi-arid climates with statistical and machine learning methods. *Journal of Hydrology* 586 (2020), 124789. DOI:https://doi.org/10.1016/j.jhydrol.2020.124789

[15] Sumeet Dua, U. Rajendra Acharya, and Prerna Dua. 2014. *Machine Learning in Healthcare Informatics.* Springer.

[16] David Easley, Maureen O'Hara, and Soumya Basu. 2019. From mining to markets: The evolution of bitcoin transaction fees. *Journal of Financial Economics* 134, 1 (2019), 91–109. DOI:https://doi.org/10.1016/j.jfineco.2019.03.004

[17] Mark Erhardt. 2021. 88% of all BTC transfers are overpaying transaction fees. Retrieved February 11, 2021 from https://bit.ly/32CJE73.

[18] Ittay Eyal and Emin Gün Sirer. 2014. Majority is not enough: Bitcoin mining is vulnerable. In *Proceedings of the International Conference on Financial Cryptography and Data Security.* Springer, 436–454.

[19] Sead Fadilpašić. 2019. Stop Overpaying Bitcoin Transaction Fees. (2019). Retrieved July 23, 2020 from https://bit.ly/2Cy5VtH.

[20] J. Doyne Farmer and John J. Sidorowich. 1987. Predicting chaotic time series. *Physical Review Letters* 59 (1987), 845–848. DOI:https://doi.org/10.1103/PhysRevLett.59.845

[21] George Foster. 1977. Quarterly accounting data: Time-series properties and predictive-ability results. *The Accounting Review* 52, 1 (1977), 1–21. Retrieved from http://www.jstor.org/stable/246028.

[22] Wayne A. Fuller. 2009. *Introduction to Statistical Time Series.* John Wiley & Sons.

[23] Jindong Gu and Daniela Oelke. 2019. Understanding bias in machine learning. arXiv:1909.01866. Retrieved from https://arxiv.org/abs/1909.01866.

[24] James Douglas Hamilton. 2020. *Time Series Analysis.* Princeton University Press.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision.* 1026–1034.

[26] Nicolas Houy. 2014. The Bitcoin mining game. *Available at SSRN 2407834* (2014).

[27] Nicolas Houy. 2014. *The Economics of Bitcoin Transaction Fees.* Working Papers 1407. Groupe d'Analyse et de Théorie Economique (GATE), Université Lyon 2. Retrieved May 2021 from http://EconPapers.repec.org/RePEc:gat:wpaper:1407.

[28] Ari Jules, Ittay Eyal, and Mahimna Kelkar. 2021. Miners, Front-Running-as-a-Service Is Theft. Retrieved April 28, 2021 from https://bit.ly/3a8UPZs.

[29] Kerem Kaskaloglu. 2014. Near zero bitcoin transaction fees cannot last forever. In *the International Conference on Digital Security and Forensics (DigitalSec'14).* 91–99.

[30] Aleksandar Kuzmanovic. 2019. Net neutrality: Unexpected solution to blockchain scaling. *Communications of the ACM* 62, 5 (2019), 50–55.

[31] Juanjuan Li, Xiaochun Ni, Yong Yuan, and Feiyue Wang. 2020. A novel GSP auction mechanism for dynamic confirmation games on Bitcoin transactions. *IEEE Transactions on Services Computing* (2020), 1–1. DOI: 10.1109/TSC.2020.2994582

[32] Juanjuan Li, Yong Yuan, and Fei-Yue Wang. 2019. A novel GSP auction mechanism for ranking Bitcoin transactions in blockchain mining. *Decision Support Systems* 124 (2019), 113094.

[33] Ioanna Lykourentzou, Ioannis Giannoukos, Vassilis Nikolopoulos, George Mpardis, and Vassili Loumos. 2009. Dropout prediction in e-learning courses through the combination of machine learning techniques. *Computers & Education* 53, 3 (2009), 950–965. DOI:https://doi.org/10.1016/j.compedu.2009.05.010

[34] Xiaolei Ma, Haiyang Yu, Yunpeng Wang, and Yinhai Wang. 2015. Large-scale transportation network congestion evolution prediction using deep learning theory. *PloS one* 10, 3 (2015), e0119044.

[35] Johnnatan Messias, Mohamed Alzayat, Balakrishnan Chandrasekaran, and Krishna P. Gummadi. 2020. On blockchain commit times: An analysis of how miners choose Bitcoin transactions. *The 2nd International Workshop on Smart Data for Blockchain and Distributed Ledger (SDBD'20).*

[36] Malte Möser and Rainer Böhme. 2015. Trends, tips, tolls: A longitudinal study of Bitcoin transaction fees. In *Proceedings of the Financial Cryptography and Data Security: FC 2015.* Springer, Berlin, 19–33. DOI:https://doi.org/10.1007/978-3-662-48051-9_2

[37] Satoshi Nakamoto et al. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008).

[38] Saurav Nanda, Faheem Zafari, Casimer DeCusatis, Eric Wedaa, and Baijian Yang. 2016. Predicting network attack patterns in SDN using machine learning approach. In *Proceedings of the 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks.* 167–172. DOI:https://doi.org/10.1109/NFV-SDN.2016.7919493

[39] Karl J. O'Dwyer and David Malone. 2014. Bitcoin mining and its energy footprint. (2014). Retrieved April 2021 from http://karlodwyer.com/publications/pdf/bitcoin_KJOD_2014.pdf.

[40] Haseeb Qureshi. 2019. Blockchain fees are broken. Here are 3 proposals to fix them. *Hacker Noon* (2019). Retrieved from https://haseebq.com/blockchain-fees-are-broken/.

[41] Zhijie Ren, Kelong Cong, Johan Pou welse, and Zekeriya Erkin. 2017. Implicit consensus: Blockchain with unbounded throughput. Retrieved June 29, 2017 from https://bit.ly/34VEzcD.

[42] Peter R. Rizun. 2015. A transaction fee market exists without a block size limit. *Block Size Limit Debate Working Paper* (2015), 2327–4697.

[43] Peter R. Rizun. 2016. Subchains: A technique to scale bitcoin and improve the user experience. *Ledger* 1 (2016), 38–52. Retrieved from https://www.bitcoinunlimited.info/resources/subchains.pdf.

[44] Salvatore Stolfo, David W. Fan, Wenke Lee, Andreas Prodromidis, and P. Chan. 1997. Credit card fraud detection using meta-learning: Issues and initial results. In *Proceedings of the AAAI-97 Workshop on Fraud Detection and Risk Management.*

[45] Mahmoudreza Tahmassebpour. 2017. A new method for time-series big data effective storage. *IEEE Access* PP (2017), 1–1. DOI:https://doi.org/10.1109/ACCESS.2017.2708080

[46] Enrico Tedeschi. 2022. Bitcoin blockchain optimized for machine learning prediction model. (2022). DOI:https://doi.org/10.18710/8IKVEU

[47] Enrico Tedeschi, Håvard D. Johansen, and Dag Johansen. 2018. Trading network performance for cash in the bitcoin blockchain. In *Proceedings of the 8th International Conference on Cloud Computing and Services Science.* 643–650. DOI:https://doi.org/10.5220/0006805906430650

[48] Enrico Tedeschi, Tor-Arne S. Nordmo, Dag Johansen, and Håvard D. Johansen. 2019. Predicting transaction latency with deep learning in proof-of-work blockchains. In *Proceedings of the 2019 IEEE International Conference on Big Data.* IEEE, 4223–4231.

[49] Theodore B. Trafalis and Huseyin Ince. 2000. Support vector machine for regression and applications to financial forecasting. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, Vol. 6. IEEE, 348–353.

[50] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. 2019. SciPy 1.0–fundamental algorithms for scientific computing in python. *Nature Methods* 17 (2020). DOI: 10.1038/s41592-019-0686-2

[51] William W. S. Wei. 2006. Time series analysis. In *Proceedings of the Oxford Handbook of Quantitative Methods in Psychology: Vol. 2.*

[52] Abbas Yazdinejad, Hamed HaddadPajouh, Ali Dehghantanha, Reza M. Parizi, Gautam Srivastava, and Mu-Yen Chen. 2020. Cryptocurrency malware hunting: A deep recurrent neural network approach. *Applied Soft Computing* 96 (2020), 106630.

[53] Yu Zheng, Like Liu, Longhao Wang, and Xing Xie. 2008. Learning transportation mode from raw gps data for geographic applications on the web. In *Proceedings of the 17th International Conference on World Wide Web.* ACM, 247–256.

[54] Y. Zhu, R. Guo, G. Gan, and W. Tsai. 2016. Interactive incontestable signature for transactions confirmation in bitcoin blockchain. In *Proceedings of the 2016 IEEE 40th Annual Computer Software and Applications Conference.* 443–448. DOI:https://doi.org/10.1109/COMPSAC.2016.142

[55] Roi Bar Zur, Ittay Eyal, and Aviv Tamar. 2020. Efficient MDP analysis for selfish-mining in blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies.* 113–131.

Received August 2021; revised February 2022; accepted March 2022

Paper IV

Amores-Sesar, I., Cachin, C. & Tedeschi, E.

When is Spring coming? A Security Analysis of Avalanche Consensus

(Manuscript).

When is Spring coming? A Security Analysis of Avalanche Consensus

Ignacio Amores-Sesar
University of Bern[*]
ignacio.amores@unibe.ch

Christian Cachin
University of Bern[*]
christian.cachin@unibe.ch

Enrico Tedeschi
The Artic University of Norway[†]
enrico.tedeschi@uit.no

10th October, 2022

Abstract

Avalanche is a blockchain consensus protocol with exceptionally low latency and high throughput. This has swiftly established the corresponding token as a top-tier cryptocurrency. Avalanche achieves such remarkable metrics by substituting proof of work with a random sampling mechanism. The protocol also differs from Bitcoin, Ethereum, and many others by forming a directed acyclic graph (DAG) instead of a chain. It does not totally order all transactions, establishes a partial order among them, and accepts transactions in the DAG that satisfy specific properties. Such parallelism is widely regarded as a technique that increases the efficiency of consensus.

Despite its success, Avalanche consensus lacks a complete abstract specification and a matching formal analysis. To address this drawback, this work provides first a detailed formulation of Avalanche through pseudocode. This includes features that are omitted from the original whitepaper or are only vaguely explained in the documentation. Second, the paper gives an analysis of the formal properties fulfilled by Avalanche in the sense of a generic broadcast protocol that only orders related transactions. Last but not least, the analysis reveals a vulnerability that affects the liveness of the protocol. A possible solution that addresses the problem is also proposed.

1 Introduction

The Avalanche blockchain with its fast and scalable consensus protocol is one of the most prominent alternatives to first-generation networks like Bitcoin and Ethereum that consume huge amounts of energy. Its AVAX token is ranked 14th according to market capitalization in August 2022 [8]. Avalanche offers a protocol with high throughput, low latency, excellent scalability, and a lightweight client. In contrast to many well-established distributed ledgers, Avalanche is not backed by proof of work. Instead, Avalanche bases its security on a deliberately metastable mechanism that operates by repeatedly sampling the network, guiding the honest parties to a common output. This allows Avalanche to reach a peak throughput of up to 20’000 transactions per second with a latency of less than half a second [28].

This novel mechanism imposes stricter security constraints on Avalanche compared to other networks. Traditional Byzantine fault-tolerant consensus tolerates up to a third of the parties to be corrupted [23] and proof-of-work protocols make similar assumptions in terms of mining power [12, 11]. Avalanche, however, can tolerate only up to $O(\sqrt{n})$ malicious parties. Furthermore, the transactions in the “exchange chain” of Avalanche (see below) are not totally ordered, in contrast to most other cryptocurrencies, which implement a form of atomic broadcast [5]. As the protocol is structured around a

[*]Institute of Computer Science, University of Bern, Neubrückstrasse 10, 3012 CH-Bern, Switzerland.
[†]Institute of Computer Science, The Artic University of Norway, Hansine Hansens vei 54, 6050 NO-Langnes, Norway.

directed acyclic graph (DAG) instead of a chain, it permits some parallelism. Thus, the parties may output the same transactions in a different order, unless these transactions causally depend on each other. Only the latter must be ordered in the same way.

The consensus protocol of a blockchain is of crucial importance for its security and for the stability of the corresponding digital assets. Analyzing such protocols has become an important topic in current research. Although Bitcoin appeared first without formal arguments, its security has been widely understood and analyzed meanwhile. The importance of proving the properties of blockchain protocols has been recognized for a long time [7].

However, there are still protocols released today without the backing of formal security arguments. The Avalanche whitepaper [28] introduces a family of consensus protocols and offers rigorous security proofs for some of them. Yet the Avalanche protocol itself and the related Snowman protocol, which power the platform, are not analyzed. Besides, several key features of this protocol are either omitted or described only vaguely.

In this paper, we explain the Avalanche consensus protocol in detail. We describe it abstractly through pseudocode and highlight features that may be overlooked in the whitepaper (Sections 3–4). Furthermore, we use our insights to formally establish safety properties of Avalanche. Per contra, we also identify a weakness that affects its liveness. In particular, Avalanche suffers from a vulnerability in how it accepts transactions that allows an adversary to delay targeted transactions by several orders of magnitude (Section 5), which may render the protocol useless in practice. The problem results from dependencies that exist among the votes on different transactions issued by honest parties; the whitepaper does not address them. The attack may be mounted by a single malicious party with some insight into the network topology. Finally, we suggest a modification to the Avalanche protocol that would prevent our attacks from succeeding and reinstantiate liveness of the protocol (Section 6). This version, which we call *Glacier*, restricts the sampling choices in order to break the dependencies, but also eliminates the parallelism featured by Avalanche.

The vulnerability has been acknowledged by the Avalanche developers. The deployed version of the protocol differes however from the protocol in the whitepaper in a crucial way. It implements another measure that prevents the problem, as we explain as well (in Appendix B).

2 Related work

Despite Avalanche's tremendous success, there is no independent research on its security. Recall that Avalanche introduces the "snow family" of consensus protocols based on sampling [28, 3]: Slush, Snowflake, and Snowball. Detailed proofs about liveness and safety for the snow-family of algorithms are given. The Avalanche protocol for asset exchange, however, lacks such a meticulous analysis. The dissertation of Yin [31] describes Avalanche as well, but does not analyze its security in more detail either.

Recall that Nakamoto introduced Bitcoin [22] without any formal analysis. This has been corrected by a long line of research, which established the conditions under which it is secure (e.g., by Garay, Kiayias, and Leonardos [12, 13] and by Eyal and Sirer [11]).

The consensus mechanisms that stand behind the best-known cryptocurrencies are meanwhile properly understood. Some of them, like the proof-of-stake protocols of Algorand [14] and the Ouroboros family that powers the Cardano blockchain [16, 9], did apply sound design principles by first introducing and analyzing the protocols and only later implementing them.

Many others, however, have still followed the heuristic approach: they released code first and were confronted with concerns about their security later. This includes Ripple [2, 1] and NEO [30], in which several vulnerabilities have been found, or Solana, which halted multiple times in 2021–2022. Stellar comes with a formal model [20], but it has also been criticized [17].

Protocols based on DAGs have potentially higher throughput than those based on chains. Notable examples include PHANTOM and GHOSTDAG [26], the Tangle of IOTA (`www.iota.org`), Conflux [19], and others [15]. However, they are also more complex to understand and susceptible to a wider range of attacks than those that use a chain. Relevant examples of this kind are the IOTA protocol [21], which has also failed repeatedly in practice [29] and PHANTOM [26], for which a vulnerability has been shown [18] in an early version of the protocol.

3 Model

3.1 Avalanche platform

We briefly review the architecture of the Avalanche platform [3]. It consists of three separate built-in blockchains, the *exchange* or *X-Chain*, the *platform* or *P-Chain*, and the *contract* or *C-Chain*. Additionally there are a number of subnets. In order to participate in the protocols and validate transactions, a party needs to stake at least 2'000 AVAX (about 50'000 USD in August 2022 [8]).

The *exchange chain* or *X-Chain* secures and stores *transactions* that trade digital assets, such as the native AVAX token. This chain implements a variant of the *Avalanche consensus protocol* that only partially orders the transactions and that is the focus of this work. All information given here refers to the original specification of Avalanche [28].

The *platform chain* or *P-Chain* secures *platform primitives*; it manages all other chains, designates parties to become validators or removes them again from the validator list, and creates or deletes wallets. The P-Chain implements the *Snowman* consensus protocol: this is a special case of Avalanche consensus that always provides total order, like traditional blockchains. It is not explained in the whitepaper and we do not describe it further here.

The *C-Chain* hosts *smart contracts* and runs transactions on an Ethereum Virtual Machine (EVM). It also implements the Snowman consensus protocol of Avalanche and totally orders all transactions and blocks.

3.2 Communication and adversary

We now abstract the Avalanche consensus protocol and consider a network of n parties $\mathcal{N} = \{p, q, \dots\}$ that communicate with each other by sending messages. An adversary may *corrupt* up to f of these parties and cause them to behave *maliciously* and diverge arbitrarily from the protocol. Non-corrupted parties are known as *honest*, messages and transactions sent by them are referred to as *honest*. Analogously, corrupted parties send *malicious* transactions and messages. The parties may access a low-level functionality for sending messages over authenticated point-to-point links between each pair of parties. In the protocol, this functionality is accessed by two events *send* and *receive*. Parties may also access a second low-level functionality for broadcasting messages through the network by gossiping, accessed by the two events *gossip* and *hear* in the protocol. Both primitives are subject to network and timing assumptions. We assume *partial synchrony*, as in the original Avalanche whitepaper [28]. Messages are delivered according to an exponential distribution, that is, the amount of time between the sending and the receiving of a message follows an exponential distribution with unknown parameter to the parties. However, messages from corrupted parties are not affected by this delay and will be delivered as fast as the adversary decides. This model differs from the traditional definition of partial synchrony [10], since the adversary does not possess the ability to delay honest messages as it pleases.

3.3 Abstractions

The *payload transactions* of Avalanche are submitted by users and built according to the *unspent transaction output* (*UTXO*) model of Bitcoin [22]. A payload transaction *tx* contains a set of *inputs*, a set of *outputs*, and a number of digital signatures. Every input refers to a position in the output of a transaction executed earlier; this output is thereby *spent* (or *consumed*) and distributed among the outputs of *tx*. The balance of a user is given by the set of unspent outputs of all transactions (UTXOs) executed by the user (i.e., assigned to public keys controlled by that user). A payload transaction is valid if it is properly authenticated and none of the inputs that it consumes has been consumed yet (according to the view of the party executing the validation).

Blockchain protocols are generally formalized as *atomic broadcast*, since every party running the protocol outputs the same ordered list of transactions. However, the transaction sequences output by two different parties running Avalanche may not be exactly the same because Avalanche allows more flexibility and does not require a total order. Avalanche only orders transactions that causally depend on each other. Thus, we abstract Avalanche as a *generic broadcast* according to Pedone and Schiper [24], in which the total-order property holds only for *related* transactions as follows.

Definition 1. Two payloads tx and tx' are said to be *related*, denoted by $tx \sim tx'$, if tx consumes an output of tx' or vice versa.

Our generic broadcast primitive is accessed through the events $broadcast(tx)$ and $deliver(tx)$. Similar to other blockchain consensus protocols, it defines an "external" validity property and introduces a predicate V that determines whether a transaction is valid [6].

Definition 2. A payload tx satisfies the validity predicate of Avalanche if all the cryptographic requirements are fulfilled and there is no other delivered payload with any input in common with tx.

For the remainder of this work, we fix the external validation predicate V to check the validity of payloads according to the logic of UTXO mentioned before.

Since Avalanche is a randomized protocol, the properties of our broadcast abstraction need to be fulfilled only with all but negligible probability.

Definition 3. A protocol solves *validated generic broadcast* with validity predicate V and relation $\sim$ if it satisfies the following conditions, except with negligible probability:

Validity. If a honest party *broadcasts* a payload transaction tx, then it eventually *delivers* tx.

Agreement. If a honest party *delivers* a payload transaction tx, then all honest parties eventually *deliver* tx.

Integrity. For any payload transaction tx, every honest party *delivers* tx at most once, and only if tx was previously *broadcast* by some party.

Partial order. If honest parties p and q both *deliver* payload transactions tx and tx' such that $tx \sim tx'$, then p *delivers* tx before tx' if and only if q *delivers* tx before tx'.

External validity. If a honest party *delivers* a payload transaction tx, then $V(tx) = \text{TRUE}$.

Note that different instantiations of the relation $\sim$ transform the generic broadcast primitive into well-known primitives. For instance, when no pair of transactions are related, generic broadcast degenerates to reliable broadcast. Whereas when every two transactions are related, generic broadcast transforms into atomic broadcast. In our context, broadcasting corresponds to submitting a payload transaction to the network, whereas delivering corresponds to accepting a payload and appending it to the ledger.

The Avalanche protocol augments payload transactions to *protocol transactions*. A protocol transaction additionally contains a set of *references* to previously executed protocol transactions, together with further attributes regarding the execution. A protocol transaction in the implementation contains a batch of payload transactions, but this feature of Avalanche is ignored here, since it affects only efficiency. Throughout this paper, *transaction* refers to a protocol transaction, unless the opposite is indicated, and *payload* means simply a payload transaction.

A transaction references one or multiple previous transactions, unlike longest-chain protocols, in which each transaction has a unique parent [22]. An execution of the Avalanche protocol will therefore create a directed acyclic graph (DAG) that forms its ledger data structure.

Given a protocol transaction T, all transactions that it references are called the *parents* of T and denoted by $parents(T)$. The parents of T together with the parents of those, recursively, are called the *ancestors* of T, denoted by $ancestors(T)$. Analogously, the transactions that have T as parent are called

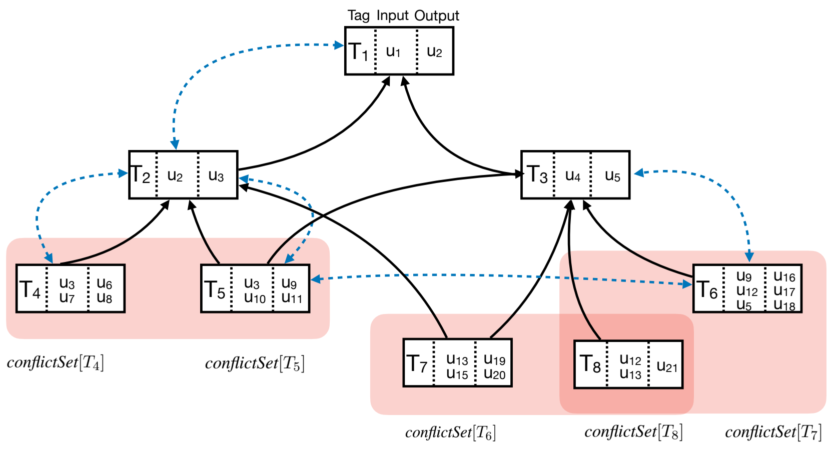


Figure 1. The UTXO model, conflicting transactions, and related transactions in Avalanche. The eight transactions are labeled $T_1, ..., T_8$. Each transaction is divided into three parts: the left part is a tag T_i to identify the transaction, the middle part is its set of inputs, and the right part is its set of outputs. The solid arrows indicate the references added by the protocol, showing the parents of each transaction. For instance, T_5 references T_2 and T_3 and has them as parents. The dashed double-arrows indicate related transactions. For example, T_5 and T_2 are related because u_3 is created by T_2 and consumed by T_5. The conflict sets are denoted by the shaded (red) rectangles. As illustrated, conflict sets can be symmetric, as for T_4 and T_5, where the conflict sets are identical (*conflictSet*$[T_4]$ = *conflictSet*$[T_5]$) or asymmetric, as for T_6, T_7, and T_8 where *conflictSet*$[T_6] \cup$ *conflictSet*$[T_7]$ = *conflictSet*$[T_8]$.

the *children* of T and are denoted by *children*(T). Finally, the children of T together with their recursive set of children are called the *descendants* of T, denoted by *descendants*(T).

Note that two payload transactions tx_1 and tx_2 in Avalanche that consume the same input are not related, unless the condition of Definition 1 is fulfilled. However, two Avalanche payloads consuming the same output *conflict*. For each transaction T, Avalanche maintains a set *conflictSet*$[T]$ of transactions that conflict with T.

4 A description of the Avalanche protocol

Avalanche's best-known quality is its efficiency. Permissionless consensus protocols, such as those of Bitcoin and Ethereum, are traditionally slow, suffer from low throughput and high latency, and consume large amounts of energy, due to their use of proof-of-work (PoW). Avalanche substitutes PoW with a random sampling mechanism that runs at network speed and that has every party adjust its preference to that of a (perceived) majority in the system. Avalanche also differs from more traditional blockchains by forming a DAG of transactions instead of a chain.

4.1 Overview

Avalanche is structured around its *polling* mechanism. In a nutshell, party u repeatedly selects a transaction T and sends a *query* about it to k randomly selected parties in the network. If a majority of them send a positive reply, the query is successful and the transaction contributes to the security of other transactions. Otherwise, the transaction is still processed but does not contribute to the security of any other transactions. Then the party selects a new transaction and repeats the procedure. A bounded number of such polls may execute concurrently. Throughout this work the terms "poll" and "query" are interchangeable.

In more detail, the protocol operates like this. Through the *gossip* functionality, every party is aware of the network membership $\mathcal{N}$. A party locally stores all those transactions processed by the network that it knows. The transactions form a DAG through their references as described in the previous section.

Whenever a user submits a payload transaction *tx* to the network, the user actually submits it through a party u. Then, u randomly selects a number of leaf nodes from a part of the DAG known as the *virtuous frontier*; these are the leaf nodes that are not part of any conflicting set. Party u then extends *tx* with references to the selected nodes and thereby creates a transaction T from the payload transaction *tx*. Next, u sends a QUERY message with T to k randomly, according to stake, chosen parties in the network and waits for their replies in the form of VOTE messages. When a party receives a query for T and if T and its ancestors are *preferred*, then the party replies with a positive vote. The answer to this query depends exclusively on the status of T and its *ancestors* according to the local view of the party that replies. Moreover, the definition of *preferred* is non-trivial and will be explained further below. If the polling party receives more than $\alpha > \frac{k}{2}$ positive votes, the poll is defined to be successful.

Every party u running the Avalanche protocol sorts transactions of its DAG into conflict sets.

Definition 4. The *conflict set* *conflictSet*$[T]$ of a given transaction T is the set of transactions that have an input in common with T (including T itself).

Note that even if two transaction T and T' consume one common transaction output and thus conflict, their conflict sets *conflictSet*$[T]$ and *conflictSet*$[T']$ can differ, since T may consume outputs of further transactions. (In Figure 1, for example, T_8 conflicts with T_6 and T_7, although T_7 conflicts with T_8 but not with T_6.)

Decisions on accepting transactions are made as follows. For each of its conflict sets, a party selects one transaction and designates it as *preferred*. This designation is parametrized by a *confidence value* $d[T]$ of T, which is updated after each transaction query. If the confidence value of some conflicting transaction T^* surpasses $d[T]$, then T^* becomes the preferred transaction in the conflict set.

It has been shown [27, 28] that regardless of the initial distribution of such confidence values and preferences of transactions, this mechanism converges. For the transactions of one conflict set considered in isolation, this implies that all honest parties eventually prefer the same transaction from their local conflict sets. (The actual protocol has to respect also dependencies among the transactions; we return to this later.)

To illustrate this phenomenon, assume that there exist only two transactions T and T' and that half of the parties prefer T, whereas the other half prefers T'. This is the worst-case scenario. Randomness in sampling breaks the tie. Without loss of generality, assume that parties with preferred transaction T are queried more often. Hence, more parties consider T as preferred as a consequence. Furthermore, the next time when a party samples again, the probability of hitting a party that prefers T is higher than hitting one that prefers T'. This is the "snowball" effect that leads to ever more parties preferring T until every party prefers T.

This preferred transaction is the candidate for acceptance and incorporation into the ledger. The procedure is parametrized by a confidence counter for each conflict set, which reflects the probability that T is the preferred transaction in the local view of the party. The party increments the confidence counter whenever it receives a positive vote to a query on a descendant of T; the counter is reset to zero whenever such a query obtains a negative vote. When this counter overcomes a given threshold, T is accepted and its payload is added to the ledger. We now present a detailed description of the protocol and refer to the pseudocode in Algorithm 1–4.

4.2 Data structures

The information presented here has been taken from the whitepaper [28], the source code [4], or the official documentation [3].

Notation. We introduce the notation used in the remaining sections including the pseudocode. For a variable a and a set $\mathcal{S}$, the notation $a \overset{R}{\leftarrow} \mathcal{S}$ denotes sampling a uniformly at random from $\mathcal{S}$. We frequently use hashmap data structures: A hashmap associates keys in a set $\mathcal{K}$ with values in $\mathcal{V}$ and is denoted by *HashMap*$[\mathcal{K} \to \mathcal{V}]$. For a hashmap $\mathcal{F}$, the notation $\mathcal{F}[K]$ returns the entry stored under key $K \in \mathcal{K}$; referencing an unassigned key gives a special value $\perp$.

We make use of timers throughout the protocol description. Timers are created in a stopped state. When a timer has been started, it produces a *timeout* event once after a given duration has expired and then stops. A timer can be (re)started arbitrarily many times. Stopping a timer is idempotent.

Global parameters. We recall that we model Avalanche as run by an immutable set of parties $\mathcal{N}$ of size n. There are more three global parameters: the number k of parties queried in every poll, the majority threshold $\alpha > \frac{k}{2}$ for each poll, and the maximum number *maxPoll* of concurrent polls.

Local variables. Queried transactions are stored in a set $\mathcal{Q}$, the subset $\mathcal{R} \subset \mathcal{Q}$ is defined to be the set of *repollable* transactions, a feature that is not explained in the original paper [28]. The number of active polls is tracked in a variable *conPoll*. The parents of a transaction are selected from the *virtuous frontier*, $\mathcal{VF}$, defined as the set of all *non-conflicting* transactions that have no known descendant and whose ancestors are preferred in their respective conflict sets. A transaction is non-conflicting if there is no transaction in the local DAG spending any of its inputs. For completeness, we recall that conflicting transactions are sorted in *conflictSet[T]* formed by transactions that conflict with T, i.e., transactions which have some input in common with T.

Transactions bear several attributes related to queries and transaction preference. A *confidence value* $d[T]$ is defined to be the number of positive queries of T and its descendants. Given a conflict set *conflictSet*$[T]$, the variable *pref*[*conflictSet*$[T]$], called *preferred transaction*, stores the transaction with the highest confidence value in *conflictSet*$[T]$. The variable *last*[*conflictSet*$[T]$] denotes which transaction was the preferred one in *conflictSet*$[T]$ after the most recent update of the preferences. The preferred transaction is the candidate for acceptance in each conflict set, the acceptance is modeled by a counter *cnt*[*conflictSet*$[T]$]. Once accepted, a transaction remains the preferred one in its conflict set forever.

4.3 Detailed description

Each transaction does through three phases during the consensus protocol: query of transactions, reply to queries, and update of preferences. All of the previous phases call the same set of functions.

Functions. The function *updateDAG*(T) sorts the transactions in the corresponding conflict sets. The function *preferred*(T) (L 98) outputs TRUE if T is the preferred transaction in its conflict set and FALSE otherwise. The function *stronglyPreferred(T)* (L 100) outputs TRUE if and only if T, and everyone of its ancestors is the preferred transaction in its respective conflict set.

The function *acceptable*(T) (L 102) determines whether T can be accepted and its payload added to the ledger or not. Transaction T is considered accepted when one of the two following conditions is fulfilled:

- T is the unique transaction in its conflict set, all the transactions referenced by $\mathcal{T}$ are considered accepted, and *cnt*[*conflictSet*$[T]$] is greater or equal than β_1.
- *cnt*[*conflictSet*$[T]$] is greater or equal than β_2.

Finally, the function *updateRepollable*() (L 106) updates the set of repollable transactions. A transaction T is repollable if T has already been accepted; or all its ancestors are preferred, a transaction in its conflict set has not already been accepted, and no parent has been rejected

Transaction query. A party in Avalanche progresses only by querying transactions. In each of these queries, party u selects a random transaction T (L 33), from the set of transactions that u has not previously queried by u. Then, it samples a random subset $\mathcal{S}[T] \subset \mathcal{N}$ of k parties from the set of parties running the Avalanche protocol and sends each a $[\text{QUERY}, T]$ message. In the implementation of the protocol, u performs *numPoll* simultaneous queries. The repoll functionality (L 33–48) consists of performing several simultaneous transactions. When u does not know of any transaction that has not been queried, u queries a transaction that has not been accepted yet. The main idea behind this functionality is to utilize the network when this is not saturated. The repoll functionality (L 33–48) constitutes one of the most notable changes from Avalanche's whitepaper [28].

Query reply. Whenever u receives a query message with transaction T, party u replies with a message $[\text{VOTE}, u, T, \mathit{stronglyPreferred(T)}]$ containing the output of the binary function *stronglyPreferred(T)* according to its local view (L 100).

Update of preferences. Party u collects the replies $[\text{VOTE}, v, T, \mathit{stronglyPreferred(T)}]$, and counts the number of positive votes. On the one hand, if the number of positive votes overcomes the threshold α (L 53), the query is considered successful. In this case party u loops over T and all its ancestors T', increasing the confidence level $d[T']$ by one. If T' is the preferred transaction in its conflict set, then party u increases the counter for transaction $\mathit{cnt}[\mathit{conflictSet}[T']]$ by one. Subsequently, u checks whether T' has also previously been the preferred transaction in its conflict set. And when T' is not the preferred transaction according to the most recent query, party u will set the counter to one (L 53–67), in order to ensure that $\mathit{cnt}[\mathit{conflictSet}[T']]$ correctly reflects the number of consecutive successful queries of descendants of T'.

On the other hand, if u receives more than $k - \alpha$ negative votes, party u loops also over T and its ancestors, and sets their counters $\mathit{cnt}[\mathit{conflictSet}[T']]$ to zero as if to indicate that T' and the other transactions should not be accepted yet. (L 68–73).

Acceptance of transactions. Party u accepts transaction T when its counter $\mathit{cnt}[\mathit{conflictSet}[T]]$ reaches a certain threshold β_1 or β_2. If T is the only transaction in its conflicting set and all its parents have already been accepted, then u accepts T if $\mathit{cnt}[\mathit{conflictSet}[T]] \geq \beta_1$, otherwise u waits until the counter overcomes a higher value β_2.

No-op transactions. The local DAG is modified whenever a poll is finalized. In particular, only the queried transaction ans its ancestors are modified. Avalanche makes use of *no-op transactions* to modify all the transactions in the DAG. After finalizing a poll, party u queries the network with all the transactions in the virtuous frontier whose state has not been modified, in a sequential manner.

4.4 Life of a transaction

We follow an honest transaction T through the protocol. The user submits the payload transaction *tx* to some party u, then u adds references *refs* to the payload transaction, creating a transaction $T = (\mathit{tx}, \mathit{refs})$. These references point to transactions in the virtuous frontier $\mathcal{VF}$. Transaction T is then *gossiped* through the network and added to the set of known transactions $\mathcal{T}$(L 22–28). Party u may also *hear* about new transactions through this gossip functionality. Whenever this is the case, u add the transaction to its set of known transactions $\mathcal{T}$ (L 29–32).

Party u eventually selects T to be processed. When this happens, u *samples* k random parties from the network and stores them in $\mathcal{S}[T]$. Party u *queries* parties in $\mathcal{S}[T]$ with T and starts a timer $\mathit{timeout}[T]$. T is added to $\mathcal{Q}$ (L 33–48).

Parties queried with T reply with the value of the function $\mathit{stronglyPreferred}(T)$ (L 100). This function answers positively (TRUE) if T is *strongly preferred*, i.e., if T and all of its ancestors are the preferred transaction inside each respective conflict set. A negative answer (FALSE) is returned if either T or any of its ancestors fail to satisfy these conditions.

Party u then stores the answer from party v to the query in the variable $\mathit{votes}[T][v]$.

- If u receives more than α positive votes, u runs over all the ancestors of T. If the ancestor T' was the most recent (or "last") preferred transaction in its conflict set, its counter is increased by one. Otherwise, T' becomes the most recent preferred transaction and its counter is reset to one (L 53–67).

- If u receives at least $k - \alpha$ FALSE votes, u resets the counter for acceptance of all its ancestors *cnt*$[T'] \leftarrow 0$ (L 68–73).
- If timer *timeout*$[T]$ is triggered before the query is completed, the query is aborted instead. The votes are reset and every party is removed from the set $\mathcal{S}[T]$, so no later reply can be considered (L 80–83).

In parallel to the previous procedure, party u may perform up to *conPoll* concurrent queries of different transactions.

Once T has been queried, it awaits in the local view of party u to be accepted. Since by assumption T is honest, *conflictSet*$[T] = \{T\}$. Hence T is accepted when *cnt*$\big[$*conflictSet*$[T]\big]$ reaches β_1, if all the ancestors of T are already accepted, or β_2 otherwise (L 102–104). We recall that *cnt*$[$*conflictSet*$[T]]$ is incremented whenever a query involving a descendant of T is successful. However, when a non-descendant of T is queried, it may trigger a no-op transaction (L 35) that is a descendant of T.

If there is no new transaction waiting to be queried, i.e., $\mathcal{T} \setminus \mathcal{Q}$ is empty, the party proceeds with a *repollable* transaction (L 40–42). A *repollable* transaction is one that has not been previously accepted but it is a candidate to be accepted (L 106–110).

5 Security analysis

Avalanche deviates from the established PoW protocols and uses a different structure. Its security guarantees must be assessed differently. The bedrock of security for Avalanche is random sampling.

5.1 From Snowball to Avalanche

The Avalanche protocol family includes Slush, Snowflake, and Snowball [28] that implement single-decision Byzantine consensus. Every party *proposes* a value and every party must eventually *decide* the

Algorithm 1 Avalanche (party u), state

```
    Global parameters and state
 1: $\mathcal{N}$                                      // set of parties
 2: $maxPoll \in \mathbb{N}$                            // maximum number of concurrent polls, default value 4
 3: $k \in \mathbb{N}$                                  // number of parties queried in each poll, default value 20
 4: $\alpha \in \{\lceil \frac{k+1}{2} \rceil, ..., k\}$   // majority threshold for queries, default value 15
 5: $\beta_1 \in \mathbb{N}$                            // threshold for early acceptance, default value 15
 6: $\beta_2 \in \mathbb{N}$                            // threshold for acceptance, default value 150
 7: $\mathcal{T} \leftarrow \emptyset$                  // set of known transactions
 8: $\mathcal{Q} \subset \mathcal{T} \leftarrow \emptyset$   // set of queried transactions
 9: $\mathcal{R} \subset \mathcal{Q} \leftarrow \emptyset$   // set of repollable transactions
10: $\mathcal{D} \subset \mathcal{T} \leftarrow \emptyset$   // set of no-op transactions to be queried
11: $\mathcal{VF} \subset \mathcal{Q} \leftarrow \emptyset$  // set of transactions in the virtuous frontier
12: $conPoll \in \mathbb{N} \leftarrow 0$               // number of concurrent polls performed
13: $conflictSet : HashMap[\mathcal{T} \to 2^{\mathcal{T}}]$   // conflict set
14: $\mathcal{S} : HashMap[\mathcal{T} \to \mathcal{N}]$   // set of sampled parties to be queried with a transaction
15: $votes : HashMap[\mathcal{T} \times \mathcal{N} \to \{\text{FALSE}, \text{TRUE}\}]$   // variable to store the replies of queries
16: $d : HashMap[\mathcal{T} \to \mathbb{N}]$           // confidence value of a transaction
17: $pref : HashMap[2^{\mathcal{T}} \to \mathcal{T}]$   // preferred transaction in the conflict set
18: $last : HashMap[2^{\mathcal{T}} \to \mathcal{T}]$   // preferred transaction in the last query
19: $cnt : HashMap[2^{\mathcal{T}} \to \mathbb{N}]$     // counter for acceptance of the conflict set
20: $accepted : HashMap[\mathcal{T} \to \{\text{FALSE}, \text{TRUE}\}]$   // indicator that a transaction is accepted
21: $timer : HashMap[\mathcal{T} \to \{timers\}]$       // timer for the query of transactions
```

Algorithm 2 Avalanche (party u), part 1

```
22: upon $broadcast(tx)$ do
23:     if $V(tx)$ then
24:         $T \leftarrow (tx, \mathcal{VF})$                    // up to a maximum number of parents
25:         $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$
26:         $accepted[T] \leftarrow$ FALSE
27:         $updateDAG(T)$
28:         gossip message $[\text{BROADCAST}, T]$

29: upon hearing message $[\text{BROADCAST}, T]$ do
30:     if $T \notin \mathcal{T}$ do
31:         $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$
32:         $accepted[T] \leftarrow$ FALSE

33: upon $conPoll < maxPoll$ do
34:     $conPoll \leftarrow conPoll + 1$
35:     if $\mathcal{D} \neq \emptyset$ then                         // prefer no-op transactions
36:         $T \leftarrow$ least recent transaction in $\mathcal{D}$
37:     else if $\mathcal{T} \setminus \mathcal{Q} \neq \emptyset$ then            // take any not yet queried transaction
38:         $T \overset{R}{\leftarrow} \mathcal{T} \setminus \mathcal{Q}$
39:         $d[T] \leftarrow 0$
40:     else                  // all transaction queried already, take one of them
41:         $updateRepollable()$
42:         $T \overset{R}{\leftarrow} \mathcal{R}$
43:     $\mathcal{S}[T] \leftarrow sample(\mathcal{N} \setminus \{u\}, k)$     // sample $k$ parties randomly according to stake
44:     send message $[\text{QUERY}, T]$ to all parties $v \in \mathcal{S}[T]$
45:     $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\bot, \mathcal{VF} \setminus \{T\})\}$            // create a no-op transaction
46:     start $timer[T]$                                          // duration $\Delta_{query}$
47:     $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{T\}$
48:     $updateDAG(T)$

49: upon receiving message $[\text{QUERY}, T]$ from party $v$ do
50:     send message $[\text{VOTE}, u, T, stronglyPreferred(T)]$ to party $v$

51: upon receiving message $[\text{VOTE}, v, T, w]$ such that $v \in \mathcal{S}[T]$ do        // $w$ is the vote
52:     $votes[T, v] \leftarrow w$                                  // $w \in \{\text{FALSE}, \text{TRUE}\}$
```

Algorithm 3 Avalanche (party u), part 2

```
53: upon $\exists T \in \mathcal{T}$ such that $\left|\{v \in \mathcal{S}[T] \mid votes[T,v] = \text{TRUE}\}\right| \geq \alpha$ do    // query of $T$ is successful
54:     stop $timer[T]$
55:     $votes[T,*] \leftarrow \bot$                                  // remove all entries in $votes$ for $T$
56:     $S[T] \leftarrow [\,]$                                           // reset $S$ for $T$
57:     $d[T] \leftarrow d[T] + 1$
58:     for $T' \in ancestors(T)$ do                                   // all ancestors of $T$
59:         $d[T'] \leftarrow d[T'] + 1$
60:         if $d[T'] > d[pref[conflictSet[T']]]$ then
61:             $pref[conflictSet[T']] \leftarrow T'$
62:         if $T' \neq last[conflictSet[T']]$ then
63:             $last[conflictSet[T']] \leftarrow T'$
64:             $cnt[conflictSet[T']] \leftarrow 1$
65:         else
66:             $cnt[conflictSet[T']] \leftarrow cnt[conflictSet[T']] + 1$
67:     $conPoll \leftarrow conPoll - 1$

68: upon $\exists T \in \mathcal{T}$ such that $\left|\{v \in \mathcal{S}[T] \mid votes[T,v] = \text{FALSE}\}\right| > k - \alpha$ do    // query of $T$ failed
69:     stop $timer[T]$
70:     $votes[T,*] \leftarrow \bot$                                  // remove all entries in $votes$ for $T$
71:     $S[T] \leftarrow [\,]$                                           // reset $S$ for $T$
72:     for $T' \in ancestors(T)$ do                                   // all ancestors of $T$
73:         $cnt[conflictSet[T']] \leftarrow 0$

74: upon $\exists T \in \mathcal{T}$ such that $acceptable(T) \wedge \neg accepted[T]$ do    // $T$ can be accepted
75:     $(tx, parents) \leftarrow T$
76:     if $V(tx)$ then
77:         $accepted[T] \leftarrow \text{TRUE}$
78:         deliver $tx$

80: upon timeout from $timer[T]$ do                                 // not enough votes on $T$ received
81:     $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{T\}$
82:     $votes[T,*] \leftarrow \bot$                                  // remove all entries in $votes$ for $T$
83:     $S[T] \leftarrow [\,]$                       // do not consider more votes from this query
```

Algorithm 4 Avalanche, auxiliary functions

```
84: function updateDAG($T$)
85:     $\mathcal{VF} \leftarrow$ set of non-conflicting leaves in the DAG
86:     $conflictSet[T] \leftarrow \emptyset$
87:     for $T' \in \mathcal{T}$ such that $T' \neq T$ and $T'$ has a common input with $T$ do
88:         $conflictSet[T] \leftarrow conflictSet[T] \cup \{T'\}$
89:         $conflictSet[T'] \leftarrow conflictSet[T'] \cup \{T\}$
90:     if $conflictSet[T] = \emptyset$ then                    // $T$ is non-conflicting
91:         $pref[conflictSet[T]] \leftarrow T$
92:         $last[conflictSet[T]] \leftarrow T$
93:         $cnt[conflictSet[T]] \leftarrow 0$
94:     $conflictSet[T] \leftarrow conflictSet[T] \cup \{T\}$

95: function getParents($T$)
96:     $(tx, parents) \leftarrow T$
97:     return $parents$                                   // set of parents stored in $T$

98: function preferred($T$)
99:     return $T \overset{?}{=} pref[conflictSet[T]]$

100: function stronglyPreferred($T$)
101:     return $\bigwedge_{T' \in ancestors(T)} preferred(T')$

102: function acceptable($T$)
103:     return $\big(\big|conflictSet[T]\big| = 1 \wedge cnt\big[conflictSet[T]\big] \geq \beta_1\big) \wedge \bigwedge_{T' \in\, parents(T)} acceptable(T')$
104:         $\vee\ cnt\big[conflictSet[T]\big] \geq \beta_2$

105: function isRejected($T$)
     return $\exists T' \in \mathcal{T}$ such that $\forall T' \in conflictSet[T] \setminus \{T\} : acceptable(T')$

106: function updateRepollable()
107:     $\mathcal{R} \leftarrow \emptyset$
108:     for $T \in \mathcal{T}$ do
109:         if $acceptable(T) \vee \bigwedge_{T' \in\, parents(T)} stronglyPreferred(T') \wedge \neg isRejected(T')$ then
110:             $\mathcal{R} \leftarrow \mathcal{R} \cup \{T\}$
```

same value for an instance. The Avalanche protocol itself provides a "payment system" [28, Sec. V]; we model it here as generic broadcast.

The whitepaper [28] meticulously analyzes the three consensus protocols. It shows that as long as $f = O(\sqrt{n})$, the consensus protocols are live and safe [28] based on the analysis of random sampling [25]. On the other hand, an adversary controlling more than $\Theta(\sqrt{n})$ parties may have the ability to keep the network in a bivalent state.

However, the Avalanche protocol itself is introduced without a rigorous analysis. The most precise statement about it is that *"it is easy to see that, at worst, Avalanche will degenerate into separate instances of Snowball, and thus provide the same liveness guarantee for virtuous transactions"* [28, p. 9]. In fact, it is easy to see that this is *wrong* because every vote on a transaction in Avalanche is linked to the vote on its ancestors. The vote on a descendant T' of T depends on the state of T.

We address this situation here through the description in the previous section and by giving a formal description of Snowball in Appendix A. We notice that one can isolate single executions of Snowball that occur inside Avalanche as follows. Consider an execution of Avalanche and a transaction T and define an *equivalent* execution of Snowball as the execution in which every party u proposes the value 1 if T is preferred in their local view, proposes 0 if another transaction is, and does not propose otherwise. Every party also selects the same parties in each round of snowball and for a query with T, for a query with a transaction that conflicts with T, or for any query with a descendant of these two.

Lemma 1. *If party u delivers an honest transaction in Avalanche, then u decided 1 in the equivalent execution of Snowball with threshold β_1. Furthermore, u delivers a conflicting transaction in Avalanche, then u decides 1 in Snowball with threshold β_2.*

Proof. By construction of the Avalanche and Snowball protocols in the whitepaper [28], the counter for acceptance of value 1 in Snowball is always greater or equal than the counter for acceptance in Avalanche. Since a successful query in Avalanche implies a successful query in Snowball, if an honest transaction in Avalanche is delivered, the counter in the equivalent Snowball instance is at least β_1. Analogously, if a conflicting transaction in Avalanche is delivered, then the counter in Snowball is at least β_2. Hence, a party in Snowball would decide 1 with the respective thresholds. $\square$

Looking ahead, we will introduce a modification of Avalanche that ensures the complete equivalence between Snowball and Avalanche. We first assert some safety properties of the Avalanche protocol.

Theorem 2. *Avalanche satisfies integrity, partial order, and external validity of a generic broadcast for payload transactions under relation $\sim$ and UTXO-validity.*

Proof. The proof is structured by property:

Integrity. We show that every payload is delivered at most once. A payload *tx* may potentially be delivered multiple times in two ways: different protocol transactions that both carry *tx* may be accepted or *tx* is delivered multiple times as payload of the same protocol transaction.

First, we consider the possibility of accepting two different transactions T_1 and T_2 carrying *tx*. Assume that party u accepts transaction T_1 and party v accepts transaction T_2. By definition, T_1 and T_2 are *conflicting* because they spend the same inputs. Using Lemma 1, party u and v decide differently in the equivalent execution in Snowball, which contradicts agreement property of the Snowball consensus [28].

The second option is that one protocol transaction T that contains *tx* is accepted multiple times. However, this is not possible either because *tx* is delivered only if *accepted*$[T] = \text{FALSE}$; variable *accepted*$[T]$ is set to TRUE when transaction T is accepted (L 74–78).

Partial order. Avalanches satisfies partial order because no payload is valid unless all payloads creating its inputs have been delivered (L 74–78). Transactions T and T' are related according to Definition 1 if and only if T has as input (i.e., spends) at least one output of T', or vice versa. This implies that related transactions are delivered in the same order for any party.

External validity. The external validity property follows from L 74, as a payload transaction can only be delivered if it is valid, i.e., its inputs have not been previously spent and the cryptographic requirements are satisfied.

□

Theorem 2 shows that Avalanche satisfies the safety properties of a generic broadcast in the presence of an adversary controlling $O(\sqrt{n})$ parties. A hypothetical adversary controlling more than $\Omega(\sqrt{n})$ parties could violate safety. It is not completely obvious how an adversary could achieve that. Such an adversary would broadcast two conflicting transactions T_1 and T_2. As we already discussed, and also explained in the whitepaper of Avalanche [28], such an adversary can keep the network in a bivalent state, so the adversary keeps the network divided into two parts: parties in part $\mathcal{P}_1$ consider T_1 preferred, and parties in part $\mathcal{P}_2$ prefer T_2. The adversary behaves as preferring T_1 when communicating with parties is $\mathcal{P}_1$ and as preferring T_2 when communicating with parties in $\mathcal{P}_2$. Eventually, a party $u \in \mathcal{P}_1$ will query only parties in $\mathcal{P}_1$ or the adversary for β_2 queries in a row. Thus, u will accept transaction T_1. Similarly, a party $v \in \mathcal{P}_2$ will eventually accept transaction T_2. Party u will *deliver* the payload contained in T_1 and v the payload contained in T_2, hence violating agreement.

An adversary controlling at most $O(\sqrt{n})$ can also violate agreement, but the required behavior is more sophisticated, as we explain next.

5.2 Delaying transaction acceptance

An adversary aims to prevent that a party u accepts an honest transaction T. A necessary precondition for this is $\mathit{cnt}[\mathit{conflictSet}[T]] \geq \beta_1$. Note that whenever a descendant of T is queried, $\mathit{cnt}[\mathit{conflictSet}[T]]$ is modified. If the query is successful (L 53), then $\mathit{cnt}[\mathit{conflictSet}[T]]$ is incremented by one. If the query is unsuccessful, $\mathit{cnt}[\mathit{conflictSet}[T]]$ is reset to zero. Remark, however, $\mathit{cnt}[\mathit{conflictSet}[T]]$ cannot be reset to one as a result of another transaction becoming the preferred in $\mathit{conflictSet}[T]$ (L 62) because T is honest, as there exist no transaction conflicting with T.

Our adversary thus proceeds by sending to u a series of cleverly generated transactions that reference T. We describe these steps that will delay the acceptance of T.

1. **Preparation phase.** The adversary submits conflicting transactions T_1 and T_2. For simplicity, we assume that she submits first T_1 and then T_2, so the preferred transaction in both conflict sets will be T_1. The adversary then waits until the target transaction T is submitted.
2. **Main phase.** The adversary repeatedly sends malicious transactions referencing the target T and T_2 to u. These transactions are valid but they reference a particular set of transactions.
3. **Searching phase.** Concurrently to the main phase, the adversary looks for transactions containing the same payload as T. If some are found, she references them as well from the newly generated transactions.

For simplicity, we assume that the adversary knows the acceptance counter of T at u, so she can send a malicious transaction whenever T is close to being accepted. In practice, she can guess this only with a certain probability, which will degrade the success rate of the attack. We also assume that the query of an honest transaction is always successful, which is the worst case for the adversary.

After u submits T, the adversary starts the main phase of the attack. If u queries an honest transaction $\hat{T}$, and if $\hat{T}$ references a descendant of T, then $\mathit{cnt}[\mathit{conflictSet}[T]]$ increases by one. If it does not, then $\hat{T}$ may cause u to submit a no-op transaction referencing a descendant of T. Hence, honest transactions always increase $\mathit{cnt}[\mathit{conflictSet}[T]]$ by one, this is the worst case for an adversary aiming to delay the acceptance of T.

If u queries a malicious transaction $\hat{T}$, honest parties reply with their value of $\mathit{stronglyPreferred}(\hat{T})$. Since T_2 is an ancestor of $\hat{T}$ and not the preferred transaction in its conflict set (as we have assumed that T_1 is preferred), all queried parties return FALSE. Thus, u sets acceptance counter of every ancestor of $\hat{T}$ to zero (L 68), in particular, $\mathit{cnt}[\mathit{conflictSet}[T]] \leftarrow 0$. However, since $\hat{T}$ does not reference

Algorithm 5 Liveness attack: Delaying transaction T

```
    Initialization
111:    create two conflicting transactions $T_1$ and $T_2$
112:    gossip two messages $[\textsc{broadcast}, T_1]$ and $[\textsc{broadcast}, T_2]$
113:    $\mathcal{A} \leftarrow \emptyset$

114: upon hearing message $[\textsc{broadcast}, T]$ do                    // target transaction
115:    $\mathcal{A} \leftarrow \{T\}$

116: upon $cnt[conflictSet[T]] = \lfloor \frac{\beta_1}{2} \rfloor$ in the local view of $u$ do
117:    create $\hat{T}$ such that $T_2 \in ancestors(\hat{T})$ and for all $T' \in \mathcal{A}$, also $T' \in ancestors(\hat{T})$
118:    send message $[\textsc{broadcast}, \hat{T}]$ to party $u$        // pretend to gossip the message

119: upon hearing message $[\textsc{broadcast}, \tilde{T}]$ such $\tilde{T}$ and $T$ contain the same payload do
120:    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\tilde{T}\}$
```

the virtuous frontier, u submits a no-op transaction that references a descendant of T, thus increasing $cnt[conflictSet[T]]$ to one.

We show that when the number of transactions is low, in particular when $|\mathcal{T} \setminus \mathcal{Q}| \leq 1$ for every party, then Avalanche may lose liveness.

Theorem 3. *Avalanche does not satisfy validity nor agreement of generic broadcast with relation $\sim$ with one single malicious party if $|\mathcal{T} \setminus \mathcal{Q}| \leq 1$ for every party.*

Proof. We consider again the adversary described above that targets T and u.

- **Validity.** Whenever $cnt[conflictSet[T]]$ in the local view of u reaches $\lfloor \frac{\beta_1}{2} \rfloor$, the adversary sends a malicious transaction to party u, who immediately queries it (since $|\mathcal{T} \setminus \mathcal{Q}| \leq 1$). It follows that u sets $cnt[conflictSet[T]]$ to zero and increases it intermediately afterwards, due to a no-op transaction. This process repeats indefinitely over time and prevents u from delivering the payload in T.
- **Agreement.** Assume that an honest party broadcasts the payload contained in T. The adversary forces a violation of agreement by finding honest parties u and v such that $cnt[conflictSet[T]] = \beta_1 - 1$ at v and $cnt[conflictSet[T]] < \beta_1 - 1$ at u (such parties exist because in the absence of an adversary, as $cnt[conflictSet[T]]$ increases monotonically over time). The adversary then sends an honest transaction T_h that references T to v and a malicious transaction T_m, as described before, to u. On the one hand, party v queries T_h, increments $cnt[conflictSet[T]]$ to β_1, accepts transaction T, and delivers the payload. On the other hand, party u queries T_m and sets $cnt[conflictSet[T]]$ to one. After that, the adversary behaves as discussed before. Notice that v has delivered the payload within T but u will never do so.

$\square$

An adversary may thus cause Avalanche to violate validity and agreement. For this attack, however, the number of transactions in the network must be low, in particular, $|\mathcal{T} \setminus \mathcal{Q}| \leq 1$. In July 2022, the Avalanche network processed an average of 647238 transactions per day[1]. Assuming two seconds per query, four times the value observed in our local implementation, the recommended values of 30 transactions per batch, and four concurrent polls, the condition $|\mathcal{T} \setminus \mathcal{Q}| \leq 1$ is satisfied 88% of the time.

However, the adversary still needs to know the value of the counter for acceptance of the different parties.

[1]https://subnets.avax.network/stats/network

5.3 A more general attack

We may relax the assumption of knowing the acceptance counters and also send the malicious transaction to more parties through gossip. After selecting a target transaction, the adversary continuously gossips malicious transactions to the network instead of sending them only to one party. For analyzing the performance of this attack, our figure of merit will be the number of transactions to be queried by an honest party (not counting no-ops) for confirming the target transaction T. The larger this number becomes, the longer it will take the party until it may accept T. We assume that $\mathcal{T} \setminus \mathcal{Q} \neq \emptyset$ and that a fraction γ of those transactions are malicious at any point in time[2]. A non-obvious implication is that the repoll function never queries the same transaction twice.

Lemma 4. *Avalanche requires every party to query at least β_1 transactions before accepting transaction T in the absence of an adversary.*

Proof. The absence of an adversary carries several simplifications. Firstly, there are no conflicting transactions, thus every transaction is the preferred one in its respecting conflict set and every query is successful. Secondly, due to the no-op transactions, the counter for acceptance of every transaction in the DAG is incremented by one after each query. Finally, a transaction T is accepted when its counter for acceptance reaches β_1, since the counter of the parent of any transaction reaches β_1 strictly before T (L 102). □

Lemma 5. *The average number of queried transactions before accepting transaction T in the presence of the adversary, as described in the text, is at least*

$$\beta_1 + \frac{1 + (2 + \beta_1\gamma)(1-\gamma)^{\beta_1} - (1-\gamma)^{2\beta_1}(1+\beta_1\gamma)}{\gamma(1-\gamma)^{\beta_1}(1-(1-\gamma)^{\beta_1})}.$$

Proof. We recall that in the worst-case scenario for the adversary, the query of an honest transaction increments the counter for acceptance of the target transaction T by one, while the query of a malicious transaction, effectively, resets the counter for acceptance to one, as a result of a no-op transaction.

Let a random variable W denote the number of transactions queried by u until T is accepted, and let $X \in \{0, 1\}$ model the outcome of the following experiment. Party u samples transactions until it picks a malicious transaction or until it has sampled $\beta_1 - 1$ honest transactions. In the first case, X takes the value zero, and otherwise, X takes the value one. By definition, X is a Bernoulli variable with parameter $p = (1-\gamma)^{(\beta_1 - 1)}$. Thus, the number of attempts until X returns one is a random variable Y with geometric distribution, $Y \sim \mathcal{G}(p)$, with the same parameter p. We let W_a be the random variable denoting the number of queried transactions per attempt of this experiment. The expected number of failed attempts is $\mathrm{E}[Y] = \frac{1}{(1-\gamma)^{\beta_1}}$. Furthermore, the probability that an attempt fails after sampling exactly k transactions, for $k \leq \beta_1$, is

$$\mathrm{P}[W_a = k | X = 0] = \frac{\gamma(1-\gamma)^{k-1}}{1-(1-\gamma)^{\beta_1}}.$$

Thus, the expected number of transactions per failed attempt can be expressed as

$$\mathrm{E}[W|X=0] = \frac{1-(1-\gamma)^{\beta}_1(1+\beta_1\gamma)}{\gamma(1-(1-\gamma)^{\beta_1})}. \tag{1}$$

The expected number of transaction queried during a successful attempt is at least β_1 by Lemma 4. Finally, the total expected number of queried transactions can be written as the expected number of transaction per failed attempt multiplied by the expected number of failed attempts plus the expected number of transactions in the successful attempt,

$$\mathrm{E}[W] = \mathrm{E}[W_a|X=0] \cdot (\mathrm{E}[Y] - 1) + \mathrm{E}[W_a|X=1] \cdot 1. \tag{2}$$

[2]Avalanche may impose a transaction fee for processing transactions. However, since the malicious transactions cannot be delivered, this mechanism does not prevent the adversary from submitting a large number of transactions.

From equations (1) and (2) and basic algebra, we obtain

$$\mathrm{E}[W] = \beta_1 + \frac{1 + (2 + \beta_1\gamma)(1-\gamma)^{\beta_1} - (1-\gamma)^{2\beta_1}(1+\beta_1\gamma)}{\gamma(1-\gamma)^{\beta_1}(1-(1-\gamma)^{\beta_1})}.$$

□

This expression is complex to analyze. Hence, a graphical representation of this bound is given in Figure 2. It shows the expected smallest number of transactions to be queried by an honest party (not counting no-ops) until it can confirm the target transaction T. The larger this gets, the more the protocol loses liveness. It is relevant that this bound grows proportional to $\frac{1}{(1-\gamma)^{\beta_1}}$, i.e., exponential in acceptance threshold β_1 since $(1-\gamma) < 1$.

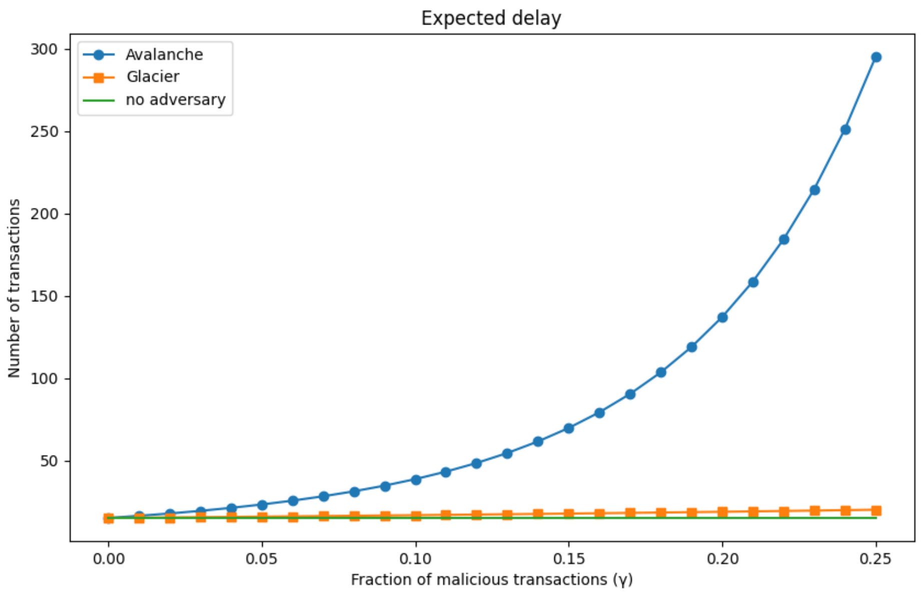


Figure 2. Expected delay in number of transactions needed to confirm a given transaction with acceptance threshold $\beta_1 = 15$, the recommended value [3], and assuming that the queries of honest transactions are successful. The (green) horizontal line shows β_1, the expected delay without attacker. The (blue) dotted line represents the expected confirmation delay in Avalanche depending on the fraction of malicious transactions. The (orange) squared line denotes the delay in Glacier (Section 6).

The Avalanche team has acknowledged our findings and the vulnerability in the abstract protocol. The protocol deployed in the actual network, however, differs from our formalization in a way that should prevent the problem. We describe the deployed version of Avalanche in Appendix B. The next section describes another variation of Avalanche that provably eliminates the problem.

6 Fixing liveness with Glacier

The adversary is able to delay the acceptance of an honest transaction T because T is directly influenced by the queries of its descendants. Note the issuer of T has no control over its descendants according to the protocol. A unsuccessful query of a descendant of T carries a negative consequence for the acceptance of T, regardless of the status of T inside its conflict set. This influence is the root of the problem described earlier. An immediate, but inefficient remedy might be to run one Snowball consensus instance for each transaction. However, this would greatly degrade the throughput and increase the latency of the protocol, as many more messages would be exchanged.

We propose here a modification, called *Glacier*, in which an unsuccessful query of a transaction T carries negative consequences *only* for those of its ancestors that led to negative votes and caused the query to be unsuccessful. Our protocol is shown in Algorithm 6. It specifically modifies the voting protocol and adds to each VOTE message for T a list L with all ancestors of T that are not preferred in their respective conflict sets (L 123–127). When party u receives a negative vote like $[\text{VOTE}, v, T, \text{FALSE}, L]$, it performs the same actions as before. Additionally, it increments a counter for each ancestor T^* of T to denote how many parties have reported T^* as not preferred while accepting T (L 135). If u receives a positive vote, the protocol remains unchanged.

If the query is successful because u receives at least α positive votes on T, then it proceeds as before (Algorithm 3, L 53). But before u declares the query to be unsuccessful, it furthermore waits until having received a vote on T from all k parties sampled in the query (L 137). When this is the case, u only resets the counter for acceptance of those ancestors T^* of T that have been reported as non-preferred by more than $k - \alpha$ queried parties (L 141–143). If T^* is preferred by at least α parties, however, then u increments its confidence level as before (L 145).

Algorithm 6 Modifications to Avalanche (Algorithm 1–4) for Glacier (party u)

```
    State
121:    $nonpref : HashMap[\mathcal{T} \times \mathcal{T} \to \mathbb{N}]$    // number of votes on $T$ reporting that $T'$ is not preferred

122:upon receiving message $[\text{QUERY}, T]$ from party $v$ do                    // replaces L 49
123:    $L \leftarrow [\,]$                                   // contains the non-preferred ancestors of $T$
124:    for $T' \in ancestors(T)$ do
125:        if $\neg preferred(T')$ then
126:            append $T'$ to $L$
127:    send message $[\text{VOTE}, v, T, stronglyPreferred(T), L]$ to party $v$

128:// replaces code at L 51
129:upon receiving message $[\text{VOTE}, v, T, w, L]$ from a party $v \in \mathcal{S}[T]$ do          // $w$ is the vote
130:    $votes[T, v] \leftarrow w$
131:    for $T' \in L$ do
132:        if $nonpref[T, T'] = \perp$ then
133:            $nonpref[T, T'] \leftarrow 1$
134:        else
135:            $nonpref[T, T'] \leftarrow nonpref[T, T'] + 1$

136:// replaces code at L 68
137:upon $\exists T \in \mathcal{T}$ such that $\big|votes[T, v]\big| = k \wedge \Big|\{v \in \mathcal{S}[T] \mid votes[T, v] = \text{FALSE}\}\Big| > k - \alpha$ do
138:    stop $timer[T]$
139:    $votes[T, *] \leftarrow \perp$                                // remove all entries in $votes$ for $T$
140:    $S[T] \leftarrow [\,]$                                          // reset the HashMap $S$
141:    for $T'$ such that $nonpref[T, T'] \neq \perp$ do                       // all ancestors of $T$
142:        if $nonpref[T, T'] > k - \alpha$ then
143:            $cnt[conflictSet[T']] \leftarrow 0$
144:        else // $nonpref[T, T'] \leq \alpha$
145:            $cnt[conflictSet[T']] \leftarrow cnt[conflictSet[T']] + 1$
146:    $nonpref[T, *] \leftarrow \perp$
```

Considering the adversary introduced in Section 5.3, a negative reply to the query of a descendant of the target transaction T does not carry any negative consequence for the acceptance of T here. In particular, the counter for acceptance of transaction T is never reset, even when a query is unsuccessful, because T is the only transaction in its conflicting set, then always preferred. Thus, transaction T will be accepted after β_1 successful queries, if all its parents are accepted, or β_2 successful queries if they are not accepted. Assuming that queries of honest transactions are successful, on average $\frac{\beta}{1-p}$ transactions are required to accept T for $\beta \in [\beta_1, \beta_2]$ depending on the state of the parents of T. For simplicity we assume that the parents are accepted, thus, the counter needs to achieve the value β_1. If this were not the case, then it is sufficient to substitute β_1 with β in the upcoming expression. Avalanche requires on average $\beta_1 + \frac{1+(2+\beta_1\gamma)(1-\gamma)^{\beta_1}-(1-\gamma)^{2\beta_1}(1+\beta_1\gamma)}{\gamma(1-\gamma)^{\beta_1}(1-(1-\gamma)^{\beta_1})}$ transactions to accept T by Lemma 5. The assumption that the query of honest transactions is always successful is more beneficial to Avalanche than to Glacier, since in Avalanche such a query resets the counter for acceptance of T. But in Glacier, the query simply leaves the counter as it is. The value of the acceptance threshold β_1 is also more beneficial for Avalanche since the number of required transactions increases linearly in Glacier and exponentially in Avalanche. Figure 2 shows a comparison of both expressions.

In Glacier, the vote for a transaction is independent of the vote of its descendant and ancestors, even if a query of a transaction carries an implicit query of all its ancestors. Thus, Lemma 1 can be extended.

Lemma 6. *Party u delivers a transaction T with counter for acceptance* cnt[conflictSet[T]] $\geq \beta_1$ *in Glacier if and only if u decides 1 in the equivalent execution of Snowball with threshold* cnt[conflictSet[T]].

Proof. Consider a transaction T in the equivalent execution of Snowball. The counter for acceptance of the value 1 in Snowball is always the same as the counter for acceptance of transaction T in Glacier because of the modifications introduced by Glacier. Thus, following the same argument as in Lemma 1, transaction T is accepted in Glacier with counter *cnt*[*conflictSet*[T]] if and only if 1 is decided with counter *cnt*[*conflictSet*[T]] in the equivalent execution of Snowball. $\square$

Theorem 7. *The Glacier algorithm satisfies the properties of generic broadcast in the presence of an adversary that controls up to $\mathcal{O}(\sqrt{n})$ parties.*

Proof. Lemma 1 is a a special case of Lemma 6. Theorem 2 shows that Lemma 1 and the properties of Snowball (Appendix A) guarantee that Avalanche satisfies integrity, partial order, and external validity. In the same way, Lemma 6 guarantees that Glacier satisfies these same properties. Thus, it is sufficient to prove that Glacier satisfies validity and agreement.

Validity. Assume that an honest party broadcasts a payload *tx*. Because the party is honest, the transaction T containing *tx* is valid and non-conflicting. In the equivalent execution of Snowball, every honest party that proposes a value proposes 1. Hence, using the validity and termination properties of Snowball, every honest party eventually decides 1. Using Lemma 6, every honest party eventually delivers *tx*.

Agreement. Assume that an honest party delivers a payload transaction *tx* contained in transaction T. Using Lemma 6, an honest party decides 1 in the equivalent execution of Snowball. Because of the termination and agreement properties of Snowball, every honest party decides 1. Using Lemma 6 again, every honest party eventually delivers payload *tx*

We conclude that Glacier satisfies the properties of generic broadcast. $\square$

With the modification to Glacier, Avalanche can be safely used as the basis for a payment system. The only possible concern with Glacier could be a decrease in performance compared to Avalanche. However, Glacier does not reduce the performance but rather improves it. Glacier only modifies the update in the local state of party u after a query has been unsuccessful. The counter of acceptance of a given transaction T in Glacier implementation is always greater or equal than its counterpart in Avalanche. This follows because a reset of *cnt*[*conflictSet*[T]] in Glacier implies the same reset in Avalanche. Such a reset in Glacier occurs if the query of a descendant of T fails and T was reported as non-preferred by more than $k - \alpha$ parties, whereas in Avalanche it is enough if the query of the descendant failed. In Avalanche, *cnt*[*conflictSet*[T]] is incremented if the query of a descendant of T succeeds, and the same occurs in Glacier. Thus, *cnt*[*conflictSet*[T]] in Glacier is always greater or equal than in Avalanche. We recall that a transaction is accepted when *cnt*[*conflictSet*[T]] reaches a threshold depending on some conditions of the local view of the DAG, but these are identical for Glacier and Avalanche. Hence, every transaction that is accepted in Avalanche is accepted in Glacier with equal or smaller latency. This implies not only that the latency of Glacier is smaller than the latency of Avalanche, but also that the throughput of Glacier is at least as good as the throughput of Avalanche.

7 Conclusion

Avalanche is well-known for its remarkable throughput and latency that are achieved through a metastable sampling technique. The pseudocode we introduce captures in a compact and relatively simple manner the intricacies of the system. We show that Avalanche, as originally introduced, possesses a vulnerability allowing an adversary to delay transactions arbitrarily. We also address such vulnerability with a modification of the protocol, Glacier, that allows Avalanche to satisfy both safety and liveness.

The developers of Avalanche have acknowledged the vulnerability, and the actual implementation does not suffer from it due to an alternative fix. Understanding this variant of Avalanche remains open and is subject of future work.

Acknowledgments

This work has been funded by the Swiss National Science Foundation (SNSF) under grant agreement Nr. 200021_188443 (Advanced Consensus Protocols).

References

[1] AMORES-SESAR, I., CACHIN, C., AND MICIC, J. Security analysis of ripple consensus. In *OPODIS* (2020), vol. 184 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 10:1–10:16.

[2] ARMKNECHT, F., KARAME, G. O., MANDAL, A., YOUSSEF, F., AND ZENNER, E. Ripple: Overview and outlook. In *TRUST* (2015), vol. 9229 of *Lecture Notes in Computer Science*, Springer, pp. 163–180.

[3] AVA LABS, INC. Avalanche documentation. `https://docs.avax.network/`.

[4] AVA LABS, INC. Node implementation for the Avalanche network. `https://github.com/ava-labs/avalanchego`.

[5] CACHIN, C., GUERRAOUI, R., AND RODRIGUES, L. E. T. *Introduction to Reliable and Secure Distributed Programming (2. ed.).* Springer, 2011.

[6] CACHIN, C., KURSAWE, K., PETZOLD, F., AND SHOUP, V. Secure and efficient asynchronous broadcast protocols. In *CRYPTO* (2001), vol. 2139 of *Lecture Notes in Computer Science*, Springer, pp. 524–541.

[7] CACHIN, C., AND VUKOLIC, M. Blockchain consensus protocols in the wild (keynote talk). In *DISC* (2017), vol. 91 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 1:1–1:16.

[8] Coinmarketcap: Today's cryptocurrency prices by market cap. `https://coinmarketcap.com/`, 2022.

[9] DAVID, B., GAZI, P., KIAYIAS, A., AND RUSSELL, A. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)* (2018), vol. 10821 of *Lecture Notes in Computer Science*, Springer, pp. 66–98.

[10] DWORK, C., LYNCH, N. A., AND STOCKMEYER, L. J. Consensus in the presence of partial synchrony. *J. ACM 35*, 2 (1988), 288–323.

[11] EYAL, I., AND SIRER, E. G. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography* (2014), vol. 8437 of *Lecture Notes in Computer Science*, Springer, pp. 436–454.

[12] GARAY, J. A., KIAYIAS, A., AND LEONARDOS, N. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)* (2015), vol. 9057 of *Lecture Notes in Computer Science*, Springer, pp. 281–310.

[13] GARAY, J. A., KIAYIAS, A., AND LEONARDOS, N. The bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO (1)* (2017), vol. 10401 of *Lecture Notes in Computer Science*, Springer, pp. 291–323.

[14] GILAD, Y., HEMO, R., MICALI, S., VLACHOS, G., AND ZELDOVICH, N. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP* (2017), ACM, pp. 51–68.

[15] KEIDAR, I., KOKORIS-KOGIAS, E., NAOR, O., AND SPIEGELMAN, A. All you need is DAG. In *PODC* (2021), ACM, pp. 165–175.

[16] KIAYIAS, A., RUSSELL, A., DAVID, B., AND OLIYNYKOV, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO (1)* (2017), vol. 10401 of *Lecture Notes in Computer Science*, Springer, pp. 357–388.

[17] KIM, M., KWON, Y., AND KIM, Y. Is stellar as secure as you think? In *EuroS&P Workshops* (2019), IEEE, pp. 377–385.

[18] LI, C., LI, P., XU, W., LONG, F., AND YAO, A. C. Scaling nakamoto consensus to thousands of transactions per second. *CoRR abs/1805.03870* (2018).

[19] LI, C., LI, P., ZHOU, D., YANG, Z., WU, M., YANG, G., XU, W., LONG, F., AND YAO, A. C. A decentralized blockchain with high throughput and fast confirmation. In *USENIX Annual Technical Conference* (2020), USENIX Association, pp. 515–528.

[20] LOKHAVA, M., LOSA, G., MAZIÈRES, D., HOARE, G., BARRY, N., GAFNI, E., JOVE, J., MALINOWSKY, R., AND MCCALEB, J. Fast and secure global payments with stellar. In *SOSP* (2019), ACM, pp. 80–96.

[21] MAMACHE, H., MAZUÉ, G., RASHID, O., BU, G., AND POTOP-BUTUCARU, M. Resilience of IOTA consensus. *CoRR abs/2111.07805* (2021).

[22] NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. Whitepaper, `https://bitcoin.org/bitcoin.pdf`, 2009.

[23] PEASE, M. C., SHOSTAK, R. E., AND LAMPORT, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (1980), 228–234.

[24] PEDONE, F., AND SCHIPER, A. Generic broadcast. In *DISC* (1999), vol. 1693 of *Lecture Notes in Computer Science*, Springer, pp. 94–108.

[25] ROSS, S. *Stochastic Processes*, second ed. Wiley, 1996.

[26] SOMPOLINSKY, Y., WYBORSKI, S., AND ZOHAR, A. PHANTOM GHOSTDAG: a scalable generalization of nakamoto consensus: September 2, 2021. In *AFT* (2021), ACM, pp. 57–70.

[27] TAN, W. Y. On the absorption probabilities and absorption times of finite homogeneous birth-death processes. *Biometrics 32*, 4 (1976), 745–752.

[28] TEAM ROCKET, YIN, M., SEKNIQI, K., VAN RENESSE, R., AND SIRER, E. G. Scalable and probabilistic leaderless BFT consensus through metastability. e-print, arXiv:1906.08936 [cs.CR], 2019.

[29] WANG, B., WANG, Q., CHEN, S., AND XIANG, Y. Security analysis on tangle-based blockchain through simulation. *CoRR abs/2008.04863* (2020).

[30] WANG, Q., YU, J., PENG, Z., BUI, V. C., CHEN, S., DING, Y., AND XIANG, Y. Security analysis on dbft protocol of NEO. In *Financial Cryptography* (2020), vol. 12059 of *Lecture Notes in Computer Science*, Springer, pp. 20–31.

[31] YIN, M. *Scaling the Infrastructure of Practical Blockchain Systems*. PhD thesis, Cornell University, USA, 2021.

A The Snowball protocol

The Snowball protocol is the Byzantine-resistant protocol introduced together with Avalanche in the whitepaper [28]. We shortly summarize this protocol in Algorithm 7.

Snowball possesses almost the same structure as Avalanche, but it is a protocol for consensus, not for broadcast. This Byzantine consensus primitive is accessed through the events *propose*(b) and *decide*(b). Any party is allowed to propose a value $b \in \{0, 1\}$. Since Snowball is a probabilistic algorithm, the properties of our abstraction need to be fulfilled only with all but negligible probability.

Definition 5. A protocol solves *Byzantine consensus* if it satisfies the following conditions, except with negligible probability:

Validity: If all parties are honest and *propose* the same value v, then no honest party decides a value different from v; furthermore, if some process decides v, then v was proposed by some process.

Termination: Every honest party eventually *decides* some value.

Integrity: No honest party *decides* twice.

Agreement: No two honest parties *decide* differently.

At the beginning of the protocol, party u may propose a value b, if u does not propose $b = \perp$. Snowball is structured in rounds around the same sample mechanism as Avalanche. Party u starts a round by sampling k parties at random and querying them for the value they are currently considering for b. If the value of a queried party is undefined, the queried party adopts the value that it is been queried with and replies accordingly. If more than α votes for value b' are collected (L 165), the query is finalized and u updates its local state by incrementing $d[b']$. If b' is the same value as the value b in the local view of u, the counter for acceptance is incremented. However, if $b' \neq b$ and $d[b'] > d[b]$, the party updates its local value b and resets the counter to zero. Party u finishes consensus when the acceptance counter reaches the value β.

Considering the adversary introduced in Section 3.2 controlling up to $O(\sqrt{n})$ parties, it can be shown that Snowball satisfies the properties of Byzantine consensus.

Theorem 8. *Snowball satisfies Byzantine consensus.*

Proof. The proof is provided in the Appendix A of the whitepaper of Avalanche [28]. □

Algorithm 7 Snowball (party u)

```
    Global parameters and state
147: $\mathcal{N}$                                    // set of parties
148: $newRound \in \{\text{FALSE}, \text{TRUE}\}$       // boolean variable indicating when to start a round
149: $decided \in \{\text{FALSE}, \text{TRUE}\}$        // boolean variable indicating when to finish the protocol
150: $k \in \mathbb{N}$                                 // number of parties queried in each poll
151: $\alpha \in \mathbb{N}$                            // majority threshold for queries
152: $cnt \in \mathbb{N}$                               // counter for acceptance
153: $\beta \in \mathbb{N}$                             // threshold for acceptance
154: $\mathcal{S} : HashMap[\mathcal{T} \to \mathcal{N}]$   // set of sampled parties to be queried
155: $d : HashMap[\{0,1\} \to \mathbb{N}]$              // confidence value of a transaction
156: $votes : HashMap[\{0,1\} \to \mathbb{N}]$          // number of votes for a value

    Algorithm
157: upon $propose(b)$ do
158:     $decided \leftarrow \text{FALSE}$
159:     $newRound \leftarrow \text{TRUE}$

160: upon $newRound \land \neg decided$ do               // still not decided
161:     $newRound \leftarrow \text{FALSE}$
162:     if $b \neq \perp$ then
163:         $\mathcal{S} \leftarrow sample(\mathcal{N} \setminus \{u\}, k)$   // sample $k$ random parties
164:         send message $[\text{QUERY}, b]$ to all parties $v \in \mathcal{S}$

165: upon $votes[b'] \geq \alpha$ do                    // $b' = 0$ or $b' = 1$
166:     $d[b'] \leftarrow d[b'] + 1$
167:     if $b = b'$ then                               // the outcome of the query is the same as our proposal
168:         $cnt \leftarrow cnt + 1$
169:     else
170:         if $d[b'] > d[b]$ then
171:             $b \leftarrow b'$
172:             $cnt \leftarrow 0$
173:     $newRound \leftarrow \text{TRUE}$

174: upon $n = k \land votes[0] < \alpha \land votes[1] < \alpha$ do   // there is no majority for any value
175:     $cnt \leftarrow 0$
176:     $newRound \leftarrow \text{TRUE}$

177: upon receiving message $[\text{QUERY}, b']$ from party $v$ do
178:     if $b = \perp$ then
179:         $decided \leftarrow \text{FALSE}$
180:         $b \leftarrow b'$
181:     send message $[\text{VOTE}, b]$ to party $v$        // reply with the local value of $b$

182: upon receiving message $[\text{VOTE}, b^*]$ from a party $v \in \mathcal{S}$ do   // collect the vote $b^*$
183:     $votes[b^*] \leftarrow votes[b^*] + 1$

184: upon $cnt = \beta$ do                               // there is enough confidence for $B$
185:     $decide(b)$
186:     $decided \leftarrow \text{TRUE}$
```

B The Avalanche protocol as implemented

The actual implementation of Avalanches addresses the liveness problems differently from Glacier and works as follows. Consider the protocol in Section 4. In the implementation, a party u queries k parties with transaction T as before. When some party v is queried with T, then v first adds T to its local view. Then it replies with a VOTE message, but instead of including a binary vote, it sends the whole virtuous frontier according to its local view. Party u collects the responses as in Section 4 and stores the virtuous frontiers received in the VOTE messages. Then it counts how many queried parties have reported some transaction T', or a descendant of T', as part of their virtuous frontier in the variable $ack[T, T']$. If more than α parties have reported T', then u adds T' to the set $\mathcal{G}[T']$ and updates its state, as for a successful query in the original protocol (L 209–215). For the remaining transactions, i.e., all the transactions in $\mathcal{Q}$ outside $\mathcal{G}[T']$, the counter is set to zero (L 217–219); this is equivalent to the effect of a negative query in the original protocol.

This fix addresses the liveness issue shown in Section 5.2 since a potential adversary loses the ability to submit a transaction that causes a reset of the acceptance counter of an honest transaction. As explained in Section 3.2, the adversary could reset the counter for acceptance of honest transactions by simply submitting a transaction T referring to the target transaction T and both transactions of a double spending T_1 and T_2 because T is not strongly preferred. However, in the implemented protocol, the parties reply with the virtuous frontier regardless of the queried transaction. Due to this, the adversary cannot influence the reply of the queried parties by creating malicious transactions. A detailed analysis of this protocol is beyond the scope of this work, however.

Algorithm 8 Modifications to Avalanche (Algorithm 1–4) in the implementation (party u)

```
    State
187:$ack : HashMap[\mathcal{T} \times \mathcal{T} \rightarrow \mathbb{N}]$        // number of votes on $T$ reporting that $T'$ is not preferred
188:$\mathcal{G} : HashMap[\mathcal{T} \times \mathcal{N} \rightarrow 2^{\mathcal{T}}]$        // ancestors of positively reported transactions

189:// replaces code at L 49
190:upon receiving message $[\textsc{Query}, T]$ from party $v$ do
191:    if $T \notin \mathcal{T}$ then        // party $u$ sees $T$ for the first time
192:        $updateDAG(T)$
193:    send message $[\textsc{Vote}, u, T, \mathcal{VF}]$ to party $v$

194:// replaces code at L 51
195:upon receiving message $[\textsc{Vote}, v, T, \mathcal{VF}']$ from a party $v \in \mathcal{S}[T]$ do        // $\mathcal{VF}'$ is the new vote
196:    $votes[T, v] \leftarrow \mathcal{VF}'$
197:    for $T' \in votes[T, v]$ do        // build the ancestors of the reported transactions
198:        $\mathcal{G}[T, v] \leftarrow \mathcal{G}[T, v] \cup ancestors(T')$
199:    for $T' \in \mathcal{G}[T, v]$ do        // count the number of parties that reported $T'$
200:        if $ack[T, T'] = \perp$ then
201:            $ack[T, T'] \leftarrow 1$
202:        else
203:        $ack[T, T'] \leftarrow ack[T, T'] + 1$

204:// replaces code at L 53 and L 68
205:upon $\exists T \in \mathcal{T}$ such that $\big|\{votes[T, v]\}\big| = k$        // every queried party has replied
206:    stop $timer[T]$
207:    $votes[T, *] \leftarrow \perp$        // remove all entries in votes for $T$
208:    for $T' \in \mathcal{Q}$ do
209:        if $ack[T, T'] \geq \alpha$ then
210:            $d[T'] \leftarrow d[T'] + 1$
211:            if $d[T'] > d[pref[conflictSet[T']]]$ then
212:                $pref[conflictSet[T']] \leftarrow T'$
213:            if $T' \neq last[conflictSet[T']]$ then
214:                $last[conflictSet[T']] \leftarrow T'$
215:                $cnt[conflictSet[T']] \leftarrow 1$
216:            else
217:                $cnt[conflictSet[T']] \leftarrow cnt[conflictSet[T']] + 1$
218:        else
219:            $cnt[conflictSet[T']] \leftarrow 0$
220:    $ack[T, *] \leftarrow \perp$        // remove all entries in ack for $T$
221:    $\mathcal{G}[T, *] \leftarrow \perp$        // remove all entries in $\mathcal{G}$ for $T$
```