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.

1533-5399/2022/07-ART77
https://doi.org/10.1145/3528669

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).

$$\begin{aligned} \mathbb{T} &= \{be_i | 0 \leq i \leq \xi\}, && \text{where } \xi = \max(\text{block height}), \\ \mathcal{L}_t^x &= [ep_t, be_x], && \text{if } t \text{ is included at height } x, \\ \gamma &= |\{be | be \in \mathbb{T} \cap \mathcal{L}_t^x\}|, && \text{number of } t \text{ occurrences.} \end{aligned} \tag{5}$$

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 18^{th} March 2014 to 14^{th} 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 $\mathcal{A}$

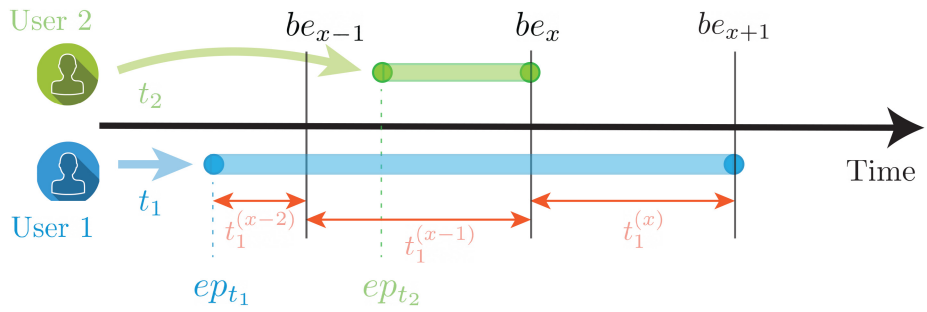

Fig. 1. Two different transactions, t_1 and t_2, issued at time ep_{t_1} and ep_{t_2}, having different lifespan. t_1 will be included in the third block after its inception, at height $x+1$, while t_2 will be immediately included at height x. The number of t_1 occurrences we represent is then $\gamma = 3$, 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:

$$\begin{aligned} S^{(x)} &= \{t | \mathcal{L}(t) = \mathcal{L}_t^y \wedge ep_t < be_{x+1}\}, \quad \forall y > x, \\ S^{(x)} &= [t_1, t_2, \dots t_n] \quad \text{is an ordered set,} \\ \text{where } &\rho_{t_1} \leq \rho_{t_2} \leq \cdots \leq \rho_{t_n}. \end{aligned} \tag{13}$$

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.

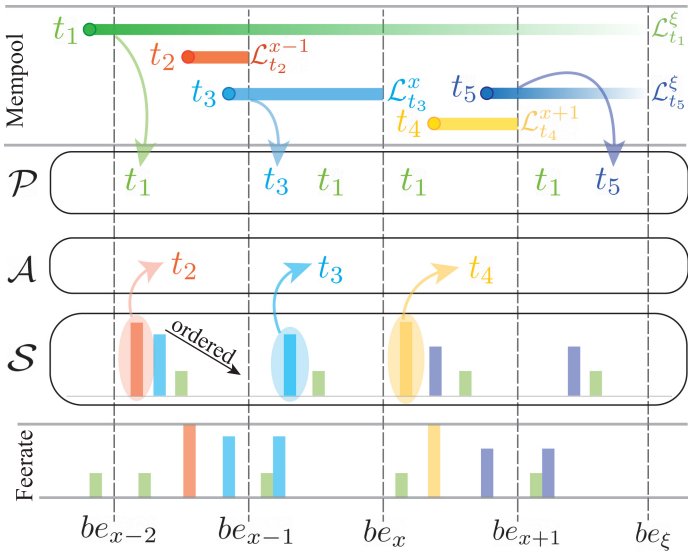

Fig. 2. We consider a subset of transactions, $\{t_1, \ldots, t_5\}$, from time be_{x-3} to be_ξ, to graphically show their path towards inclusion and their belonging in $\mathcal{P}$, $\mathcal{A}$, and S sets. Transactions are inserted in the mempool upon their inception, they carry information about 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/.

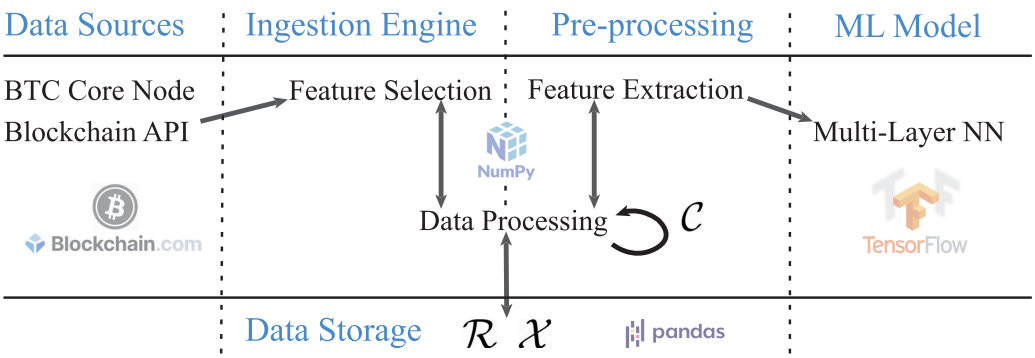

Fig. 3. Data flow in BAS, from the blockchain to the ML framework. $\mathcal{R}$, C, and $\mathcal{X}$ represent different datasets. The first indicates raw data extracted from the blockchain, the second represents a complete, run-time only dataset, while the third one includes all training data.

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:     φ_t ← sin - sou                      ▷ calculate transaction fee
11:     return φ_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 $f : K' \rightarrow \mathbb{R}$ 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

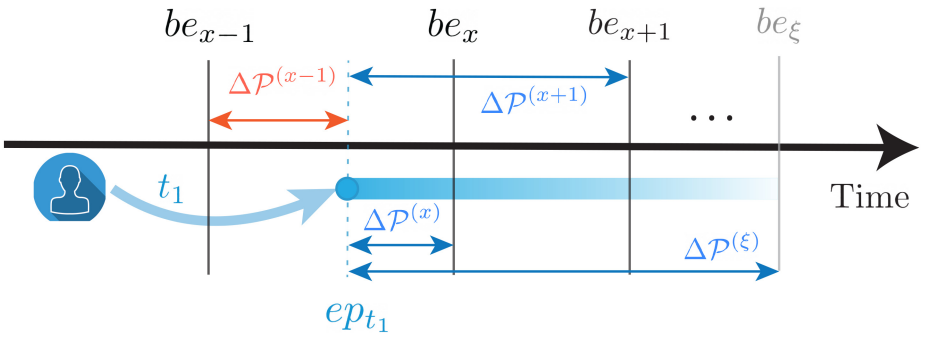

Fig. 4. A transaction t_1 submitted at time ep_{t_1} and yet-to-be included, such that $\mathcal{L}(t_1) = \mathcal{L}^{\xi}_{t_1}$, takes different $\Delta\mathcal{P}$ values over time. For instance, at height $x+1$, $\Delta\mathcal{P}^{(x+1)} = be_{x+1} - ep_{t_1}$. Blue $\Delta\mathcal{P}$ represents 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.

$$\begin{aligned} \Delta\mathcal{P}_t^{(x)} &= \beta_t{}^{(x)} - ep_t, \\ (\beta, ep) &\mapsto \Delta\mathcal{P} = f_{\mathcal{P}}(\beta, ep). \end{aligned} \tag{15}$$

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 $\delta_t^{(x)}$. 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

$$\begin{aligned} \delta_i^{(x)} &= \sum_i^n q_i, \\ \left(S^{(x)}, q\right) &\mapsto \delta = f_\delta\left(S^{(x)}, q\right). \end{aligned} \tag{16}$$

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 $\sum_{i=0,j=0}^{n} ij \sim O(n^2)$.

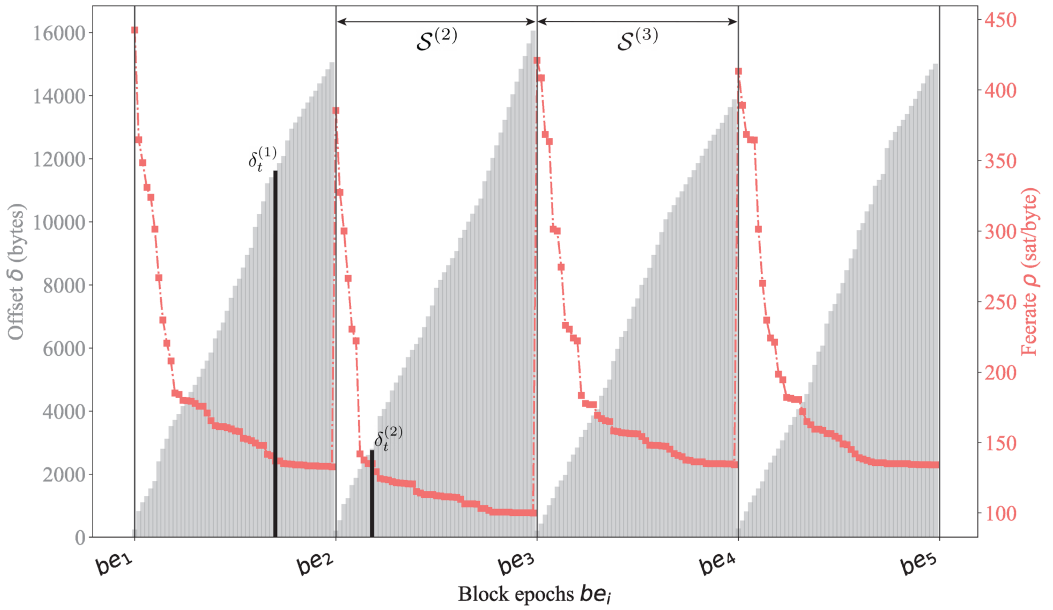

Fig. 5. Data sampled for five consecutive blocks. In each block-epoch slot, transactions are ordered by feerate, then the offset is calculated. We see in this example that the offset value for transaction $t^{(x)}$ changes according to the block-epoch. Offset value for $t^{(1)}$ for instance, is higher 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 ≤ ep_t < be_{x+1}}         ▷ S' contains all transactions occurred between [be_x, be_{x+1}]
3:      S ← ∅
4:      for all t ∈ S' ∪ P^(x) do                 ▷ add transactions that are waiting and are in P^(x)
5:          ρ_t ← f_ρ(φ_t, q_t)                   ▷ calculate feerate of t
6:          S ∪ {t}                               ▷ add t to the set S^(x)
7:      sort(S, ρ)                                ▷ order the set S by feerate in ascending order
8:      for all t ∈ S do
9:          δ_t ← 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:     δ_t ← 0
13:     for all i ∈ S after t do                  ▷ for every transaction in set S, ordered after t
14:         δ_t ← δ_t + q_i
15:     return δ_t
```

ALGORITHM 3: Normalization of $\boldsymbol{X}$

```
1: procedure NORMALIZATION(X, Xte)
2:     μ ← Mean(X)                                ▷ expected value
3:     σ ← Std(X)                                 ▷ standard deviation
4:     X_norm ← (X − μ)/σ                         ▷ set the normalize training set
5:     if Xte then                                ▷ if the algorithm is in testing phase
6:         Xte_norm ← (Xte − μ)/σ                 ▷ set the normalized testing set
7:         return X_norm, Xte_norm
8:     else
9:         return 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 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. $\lvert\mathcal{R}^i\rvert$ 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 $\alpha \longrightarrow 1$ 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 $\mathtt{mo}$ 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: $\psi = \text{sigmoid}(\mathtt{mo})$. 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 $\mathtt{mo} > 0$, and consequently $\psi \longrightarrow 1$. On the other hand, if we train the model a-posteriori and want to test older data, $\mathtt{mo} < 0$ and $\psi \longrightarrow 0$. 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)).

$$\Delta\mathcal{P}_N^{(\gamma)} = \sum_{i=0}^{\gamma} \omega^{(i)} \text{ with,}$$
$$\omega^{(i)} = \begin{cases} 0, & \text{if } \mathbf{t}_i \notin \mathcal{P}^{(i)}, \\ 1, & \text{if } \mathbf{t}_i \in \mathcal{P}^{(i)}. \end{cases} \tag{17}$$

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 (υ_0, υ_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 υ_i, represents the number of $t \in \upsilon_i$ which were correctly classified in class υ_i, while the precision P_i for class υ_i represents the number of data points classified in υ_i which actually belongs to υ_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 $t \in \upsilon_1$ higher than a randomly chosen $t \in \upsilon_0$. 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 υ_0 and υ_1 class points, while if AUC $= 0$, the classifier would be predicting all υ_0 as υ_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 υ_0 was correctly classified. False Positive Rate (*FPR*) is equal to 1 − Specificity, and it represents the proportion of υ_0 that was incorrectly classified.

Table 3. Confusion Matrix Model

		Actual values 0	Actual values 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_χ Sets

Classification accuracy for K^n_χ						
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 $K^n_\chi \subset K_\chi$. 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_χ identifies only fetched features such as transaction size and fee. This information is used by many fee predictors, with poor results about fee overpaying, $K^1_\chi = [\phi, q]$; (2) *Fairness assumptions*, the set K^2_χ excludes features based on the revenue principle, so a miner needs to only be fair in order to include transactions in the next block, $K^2_\chi = [\phi, q, \Delta\mathcal{P}, \Delta\mathcal{P}_N]$; (3) *Revenue assumptions*, the set K^3_χ excludes features based on the fairness principle, to monitor how much revenue impacts the transaction inclusion pattern, $K^3_\chi = [\phi, q, \rho, \delta, \delta_N]$; (4) *Assumptions-based*, the set K^4_χ includes only extracted features, this evaluation aims at verifying the reliability of our initial assumptions, including both, fairness and revenue concepts, $K^4_\chi = [\rho, \Delta\mathcal{P}, \Delta\mathcal{P}_N, \delta, \delta_N]$.

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_χ classification accuracy since it appears to be smaller than the one of K^3_χ. This highlights the importance of the fairness concept in order to delineate transaction inclusion. In the month of April, we observe

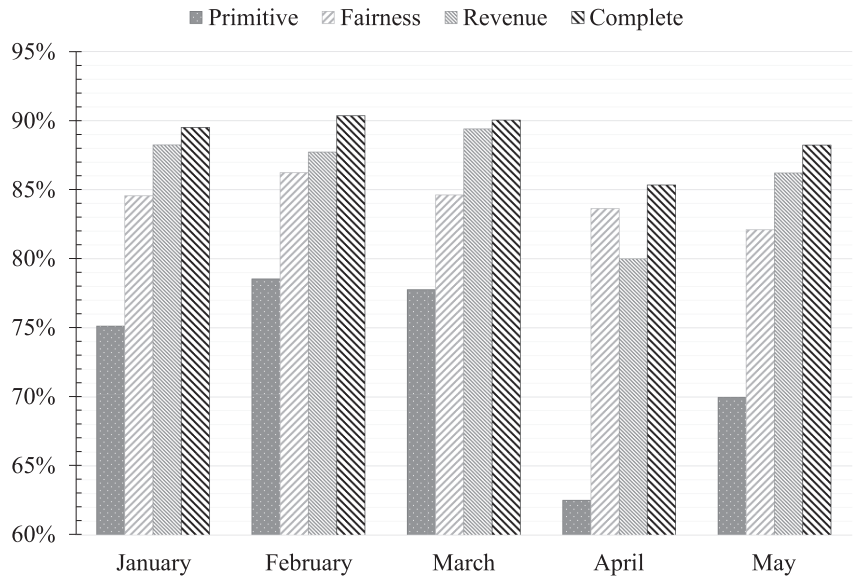

Fig. 6. Classification accuracy results for each one of our assumption sets, from primitive to complete, for each month of analysis from January 2021 until May 2021. The model accuracy increases considerably if a complete feature set is used, compared to the basic idea of using only transaction size 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 $\psi \rightarrow 0.5$ and $\alpha \rightarrow 1$, 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

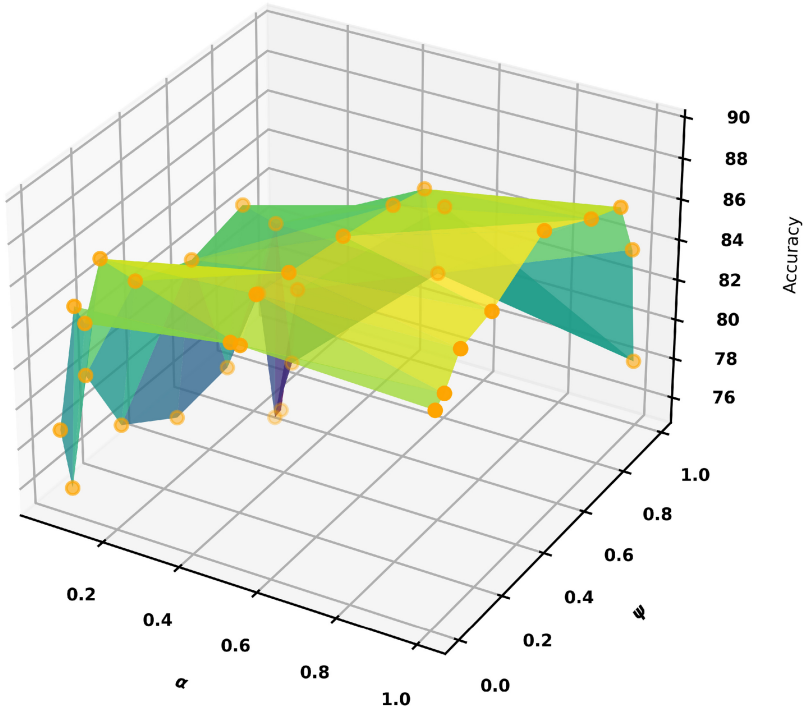

Fig. 7. Classification accuracy value (z-axis) for various models, tested with different hyperparameters setup. The y-axis, with the ψ parameter, represents how much a certain model is updated, while the x-axis indicates the model's completeness (α parameter). The importance of model cyclicity is highlighted 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 υ_0 and υ_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,

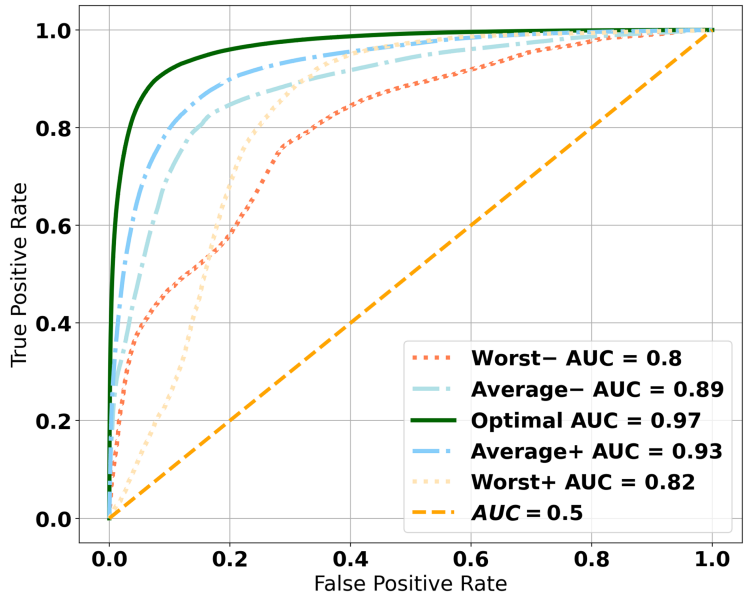

Fig. 8. Five tests to measure AUC-ROC in an optimal, average, and worst-case. The optimal test is run with $\alpha = 1$ and $\psi = 0.5$, identified with a green continuous line. The two average tests are run with $\alpha = 0.2$ and $\psi = [0.12, 0.88]$, represented with a dot-dashed light blue lines. 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, $\psi \rightarrow 1$, or when information in the test set is not complete, $\alpha \rightarrow 0$. 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.

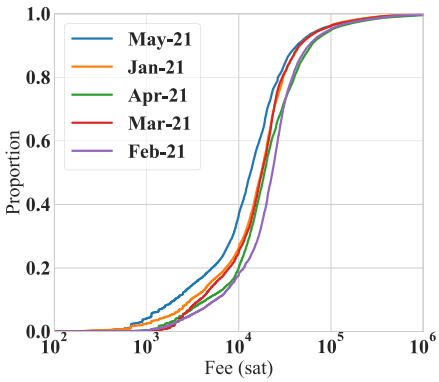

Fig. 9. CDF of transaction fees in our dataset.

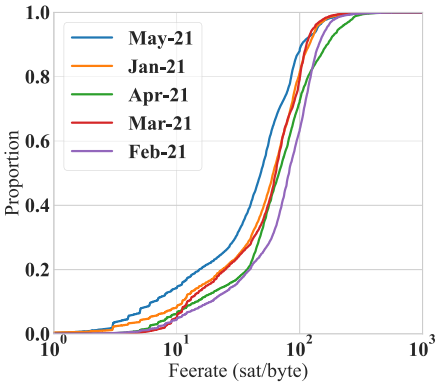

Fig. 10. CDF of transaction feerate in our dataset.

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