Future Generation Computer Systems 154 (2024) 413–425

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

EDIT: A data inspection tool for smart contracts temporal behavior modeling and prediction

Andrea De Salve [a,*], Alessandro Brighente [b], Mauro Conti [b]

[a] Institute of Applied Sciences and Intelligent Systems (ISASI) - National Research Council (CNR), Lecce, 73100, Italy
[b] Department of Mathematics - University of Padua, Padua, 35121, Italy

ARTICLE INFO

Keywords:
Blockchain
Machine learning
Data analytics

ABSTRACT

Modeling and predicting the behavior of nodes and users in blockchains provide opportunities for business strategy optimization. Indeed, the number of interactions of a node is strictly related to its balance and its prediction may be used for analytics purposes and investment strategies. However, the amount and diversity of information stored on the blockchain demand advanced tools for the modeling and analysis of blockchain data. Such tools should be able to capture the dynamicity and interaction of multiple independent actors, considering a large number of variables and dynamic interaction graph topologies. This is exacerbated by the use of smart contracts, programs stored in blockchain blocks that bring automation to blockchain's operations and thus increasing the variability of the resulting interaction graphs. Existing modeling methodologies are unable to keep track of all these details, as they are not able to capture the temporal variability of the network.

In this paper, we propose a novel framework for modeling and predicting the behavior of smart contracts on a blockchain. We propose the concept of temporal smart contracts networks, i.e., graphs representing the temporal evolution of interactions and data flow. Our framework allows the creation of temporal smart contract networks with different granularity levels by considering different interaction patterns between smart contracts, externally owned accounts, and internal transactions. Thanks to these graphs, we are able to model features such as the node in degree and amount of ether received by a smart contract, which are directly related to its behavior. We incorporate our modeling approach in Ethereum Data Inspection Tool (EDIT), a novel tool able to model interactions and predict them based on historical data. We test different machine learning models to predict features extracted by EDIT, hence allowing for the prediction of the overall behavior of the smart contract. We test EDIT on the Ethereum blockchain and model several temporal smart contracts networks, which represent the interactions and the data flow resulting from about 4 000 000 consecutive blocks. The evaluation of different real case studies shows that the proposed framework is able to predict, with a mean absolute error close to 1%, the evolution of several interesting properties (e.g., amount of received ether) related to both accounts and smart contracts.

1. Introduction

Distributed Ledger Technology (DLT) has recently undergone rapid growth in both industry and academia thanks to the introduction of the Blockchain. An increasing number of applications and systems in several domains, such as Energy metering [1], Online Social Networks [2], Supply Chains [3], and Healthcare [4], prefer to rely on the blockchain to accomplish their tasks in a trustworthy fashion and independently of the intervention of centralized servers. In several cases, blockchain acts as a decentralized registry that keeps track of all transactions that occurred between different entities, storing relevant and application-dependent data. Nowadays, the blockchain is used to store and manage an ever-increasing variety of information, including the transfer of assets, healthcare records, social media content, and information collected by Internet of Things (IoT) devices. Smart Contracts (SCs) represent one of the biggest steps forward in the development of blockchain technology due to the increased automation level they can provide. SCs are deterministic programs stored in a blockchain that are able to interact with users or other SCs to autonomously execute transactions based on their input parameters [5].

In recent years, analysis and characterization of the Blockchain data have become crucial for various stakeholders and opened new perspectives for forecasting future trends, identifying risks [6], or recognizing specific patterns [7]. The introduction and widespread use of SCs present new interesting challenges concerning their inter-coordination

* Corresponding author.
E-mail addresses: andrea.desalve@cnr.it (A. De Salve), alessandro.brighente@unipd.it (A. Brighente), mauro.conti@unipd.it (M. Conti).

https://doi.org/10.1016/j.future.2024.01.004
Received 13 May 2023; Received in revised form 11 October 2023; Accepted 2 January 2024
Available online 9 January 2024

and relationships. Indeed, SCs can be programmed to interact with existing deployed contracts, forming different types of interaction patterns. In order to analyze the evolution of these patterns established among SCs deployed on SC-enabled blockchain (e.g., Ethereum), an efficient model needs to consider the whole SC life-cycle management. However, state-of-the-art tools for analyzing blockchain data suffer from the lack of an appropriate methodology to reproduce the life-cycle of SCs and their interactions [8]. Due to the nature of the ecosystem, the data stored in the blockchain is immutable because it is not possible to delete transactions. However, a data value can be modified by appending to the blockchain a new block containing the updated value. As a result, the data stored within the blockchain is dynamic: it can change over time based on the users' interactions. Furthermore, existing solutions do not provide the granularity level necessary to characterize different types of SCs such as Externally Owned Account (EOA), and the type of transaction they interact with (internal or external). The development of tools for the analysis of blockchain data bring hence unique challenges due to the temporal dimension introduced by the ecosystem, and they should provide users the ability to investigate both the structure and dynamics of the data. Furthermore, analysis tools should be complemented with a prediction framework capable of providing useful information on the evolution of the SC network and its characterization.

In this paper, we propose a novel approach to model the interactions characterizing the life-cycle of SCs on the Blockchain and show their predictability based on historical data. In order to provide a detailed model of the blockchain ecosystem, our proposed framework uses a state-space diagram to characterize the different stages of a SC's life. Differently from the state-of-the-art where data are represented by a static graph, our model exploits graph formalism to capture both temporal information and internal actions resulting from transaction execution. We name the resulting graph models as *temporal SCs network* due to their ability to represent the temporal evolution and flow of the blockchain data. We incorporate our formalism in EDIT, a tool providing multiple granularity levels thanks to the use of different temporal SCs networks, where vertices and edges represent different sets of SCs and interactions to capture behaviors on a finer or coarser grain. In particular, we provide EDIT with the ability to build three models of temporal SCs networks representing different types of interactions (i.e., transactions, internal messages, or both). We show that, thanks to our developed model, EDIT can predict eight properties that can be used to characterize accounts in Ethereum (e.g., the number of transactions or assets received from a user or a SC). To validate EDIT and show its capabilities, we collected a set of 4 000 000 consecutive blocks from the Ethereum blockchain to create our different graph models and evaluate EDIT's predicting capabilities. We use different machine learning models to develop predictors and compare their achieved prediction Mean Absolute Error (MAE), showing that EDIT can achieve a MAE close to 1%.

Our contributions can be summarized as follows.

- We propose a temporal graph formalization to characterize the lifecycle of a SC. To this aim, we provide an in-depth description of SCs life-cycle, identifying their creator, mode of interactions, state, and end-of-life. We name the resulting temporal graph as *temporal SCs network*.
- We create three different models of temporal SCs network to create different granularity levels. Our three models consider the different interactions between SCs and EOAs, as well as different types of transactions.
- We show that it is possible to predict the behavior of SCs and EOAs based on our formalism. To this aim, we propose EDIT, a framework for the characterization and prediction of SC behavior. Being agnostic to the application scenario but leveraging only interconnections between blockchain elements, EDIT can be readily applied to different use cases.
- We validate EDIT on real-world data collected from the Ethereum blockchain network. We provide insights on the size and shape of the temporal SCs network and test the predicting capabilities of EDIT. Our results show that edit can predict SC features such as the amount of incoming Ether with a MAE close to 1%.

This paper is organized as follows. Section 2 introduces the background concepts related to Ethereum blockchain while Section 3 outlines the dynamic aspects and behavior of smart contracts. The general architecture of the proposed framework and the different internal modules are described in Section 4. Section 5 implements a proof-of-concept of the proposed framework and measures the ability of the framework to predict eight properties of the temporal SC networks by using several algorithms/configurations. The comparison against the state of the art is reported in Section 6. Finally, Section 7 draws the conclusions.

2. Ethereum: A decentralized platform for SCs

We now describe the working principles of the Ethereum Platform (EP) by introducing its main components in Section 2.1. We then define the underlying blockchain of the EP and outline its functioning principles in Section 2.2.

2.1. Ethereum platform: main components

In this section, we provide a detailed model of the EP [9], focusing on the description of the general EP architecture (Section 2.1.1), of the properties of users (Section 2.1.2) and SCs (Section 2.2.2).

2.1.1. The architecture

Ethereum is a decentralized platform [10,11] whose network consists of about 7720 peers,[1] i.e. computer resources distributed all around the world. Each of them contributes to the network in terms of computing, communication, and storage. Each peer is identified through an IP address that may belong to a personal computer, a smartphone, or even a computer cluster. In general, it is assumed that peers do not know and do not trust each other. They can securely communicate with each other by using elliptic curve cryptography. Moreover, their locally stored data is kept synchronized by all the peers participating in the network. In order to join the network, a peer discovery protocol is used: it allows the joining peer to discover at least one of the peers already in the system that is online at the joining time. Each peer of the network corresponds to a miner in the literature related to blockchain [11].

2.1.2. Accounts

Users interact with the EP by creating EOAs, each associated with a pair of keys: the *public* and *private* keys, respectively. The first can be used by other EOA to derive the *account address* of an EOA with that public key. The second key allows a user to access his/her account by signing *transactions*, as described in Section 2.2.2, and it is kept secret. An EOA, contains a *balance* which is expressed in *Ether*, i.e., the crypto-commodity of the Ethereum platform. An EOA is able to send and receive Ether. We highlight that the identity of a user that has generated one or more EOA is not disclosed. That is, no information is maintained in the EP about the association between a user and an EOA it has generated. However, the de-anonymization of the EOAs provides an opportunity to identify users by finding multiple Ethereum addresses that belong to the same identity [12,13] or determining the type of the identity (e.g., exchangers or miners) by analyzing the behavioral characteristics of the account [14,15].

[1] Network Nodes Statistics: https://www.ethernodes.org/network/1 (accessed on 2023).

2.1.3. Smart contracts

SCs are deterministic programs written in *Solidity* [11]. They are compiled into bytecode and executed by the Ethereum Virtual Machine (EVM), hosted by the peers of the network. The data, variables, and functions within SCs are not encrypted and they are public to anyone accessing the blockchain. Internally, the SC could contain three different types of functions: the *constructor*, the *getter*, and the *setter*. The constructor is used only during the creation of an instance of the SC and the caller must pay for its execution. Once a SC has been created, its execution can be activated by both EOA or other contract accounts. The getter is a type of function able to perform computations that do not affect the state of the blockchain. For instance, getters are automatically defined in each SC in order to return the value of state variables. Differently, the setters are functions that change the state of the system, and the caller must pay for their execution. As described in Section 2.2.2, the execution of any SC function can be triggered by EOA (with a transaction) or other contract accounts (with a message).

In order to pay for the execution of the function, the EOA use the Ethereum cryptocurrency, denoted as Ether (ETH). Indeed, each instruction requires a certain amount of *gas* to be executed, which depends on how computationally expensive the operation is. At the time of writing, the median gas price for a unit of gas is 14.9 Gwei.[2]

2.2. The EP blockchain mechanism

We now describe the main components of the blockchain underlying the EP.

2.2.1. The blockchain

The mechanism used to support the EP described in the previous section is the blockchain [16]. We recall that it is an append-only list of blocks linked together by using the values of a cryptographic hash function [17], which is common to all blocks. Specifically, for the EP, each block consists of a header and a set of immutable transactions. Since transactions strongly characterize the EP, they are described in more detail in Section 2.2.2. The block header stores: (i) a piece of summary information that provides means to prove the integrity and validity of the transactions in the block by using a Merkle Patricia Tree [11], and (ii) the hash value pointing to the header of the block preceding in the chain.

Since blocks are added at regular time intervals, the blockchain has a temporal dimension. As a consequence, visiting its blocks from beginning to end provides the entire history of the states of the EP. The blockchain is public to the peers in the network and each peer maintains a synchronized copy of it, according to the protocol described in Section 2.2.3.

2.2.2. Transactions and messages

A transaction is a cryptographically signed data package that can be used by EOAs to transfer ether, create a new contract, or trigger the execution of an existing contract by changing its state. The EOA that creates the transaction has to pay a cost (in *gas*) once the transaction has been included in the blockchain. Such cost is proportional to the computation required for the execution of the SC. The EOA must specify in the transaction how much ether to assign to every unit of gas (*gas price*) and how much gas it is provided for that execution (*gas limit*). The beneficiary of such cost is the miner that successfully includes the transaction in the blockchain. If the execution exceeds the gas limit the contract will be reverted to the original state and all gas spent will not be refunded. Transactions created by EOAs are stored on a temporary list (named *pending transaction list*) available to the miners of the EP for their inclusion in the blockchain via the mining process described in Section 2.2.3. Since the transactions created by an EOA can have different costs and miners are rewarded with ether, the EOA has the possibility to influence the mining process by making some transactions more remunerative than others.

Messages (also named internal transactions) are used by SCs to interact with other contracts. They are similar to transactions, but they are not stored on the blockchain because they are triggered by SCs as a result of the main transaction. As a matter of fact, a transaction can result in the execution of several messages in order to interact with the proper SCs.

2.2.3. Mining

It is essential to describe first how the new potential blocks are created. Each miner selects a list of transactions to be included in a new block from the pending transactions list. In general, the selection policy depends on the gas price that the transaction creator would be willing to pay to the miner for the inclusion of the transaction in a new block.

In order to add a new block to the EP's blockchain, the miners must execute a consensus algorithm, which is a mechanism to discourage bad behavior from users and achieve an agreement about the next block of transactions that will be appended to the blockchain. Each time a new block is appended to the blockchain, the peers of the Ethereum network download it, validate the block and the transactions, and process the transactions in the block.

We can distinguish existing consensus protocols in different categories, depending on the criteria used to reach an agreement. Originally, the EP started block mining by using a Proof of Work (PoW) consensus protocol named Ethash [9] where each miner has to generate a new block whose hashed header value is below a target that is determined by the block difficulty. In such a way, the miner can prove the execution of computational work in a certain interval of time by solving a difficult *crypto-puzzle*. The winner is rewarded for its contribution with a fixed amount of 2 ETH and a variable amount of ethers, which depends on the transaction fees included in the new block. An attacker must control over 51% of the network mining power in order to undermine the security strength of the protocol and subvert the system [18].

In 2022, the EP replaced PoW with a less energy-intensive protocol named Proof of Stake (PoS) where nodes can participate in the consensus algorithm (as *validators*) by sending at least 32 ETH to Ethereum staking contract. A validator is responsible for checking, propagating or creating blocks while the staked ETH is used as a penalty/reward for the behavior of the validator. Time in PoS Ethereum is divided into slots, and one validator is randomly selected to be a block proposer in every slot. The set of validators receives and votes for the next block to be appended on the chain. If the validator's vote is part of the majority, it will be rewarded with an amount of ETH that depends on both the validator's stake balance and the number of validators on the network.

3. Modeling a SC life-cycle via a state-space diagram

In this section, we focus on the dynamic aspects of SCs execution. For this reason, we use a *state-space diagram* to describe the abstract behavior of a SC instance by showing the possible states it can reach.

3.1. Model's overview

SCs in Solidity are similar to data types in programming languages because they characterize the properties of the corresponding instances by specifying their state variables and functions. As shown in Fig. 1, a *state-space diagram* of a SC instance (denoted by Cloud 1) is represented by a directed graph $G(V, E)$, consisting of a set of states and a set of transitions among states. A state represents the execution status of a SC, and each state is denoted by a node (circle) that indicates the name of the state. Instead, a transition is a change in the status of the SC when an event is received (i.e., a transaction or a message) and directed

[2] https://ethgasstation.info/

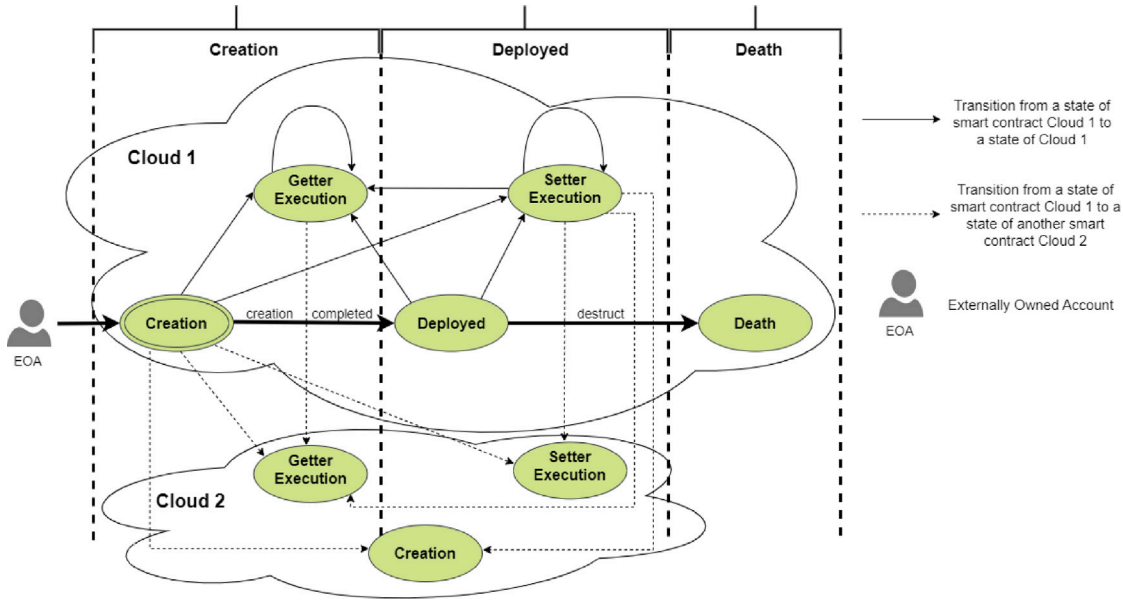


Fig. 1. SC: Life cycle of a generic SC instance Cloud 1.

edges are used to represent the transitions from one state to another. As shown in Fig. 1, the whole life cycle of a generic SC named *Cloud 1* has been divided into three main phases: Creation, Deployed, and Death. Each phase and the actions that can be executed in it can be described as follows.

1. **Creation:** a SC Cloud 1 is created from an EOA by sending to Ethereum a transaction that does not have a recipient. Internally, the transaction contains the code of the contract itself and the arguments to be passed to the constructor of the SC. The SC instance Cloud 1 is initiated as a result of this transaction and the initial state of Cloud 1 is defined by the `Creation` state, where the constructor of the SC is executed only once. The EOA can create several instances of the cloud 1 contract, each having a different address.
2. **Deployed:** In the deployed state, the SC instance Cloud 1 can be used to trigger code execution that either gets or changes the values of the internal state variables of the instance.
3. **Death:** the death state is executed when the Cloud 1 contract is killed using the destruct function.
4. **Getter execution:** this state represents a code execution that does not change the internal state of Cloud 1.
5. **Setter execution:** this state represents a code execution that changes the internal state of Cloud 1.

3.2. State transition within an instance of a contract

We use solid lines to model transitions from a state of Cloud 1 to another internal state of Cloud 1. In particular, we identify the following types of transitions.

1. **Backbone transitions:** Thicker edges represent the relationships among the creation, deployed, and death states. Such edges allow the instance Cloud 1 to reach a new state but they do not allow it to switch back to the starting state.
   After the constructor has executed, the SC instance Cloud 1 is deployed through the `creation completed` transition. The address of Cloud 1 is derived from both the sender address and the nonce of the transaction. Finally, the SC instance Cloud 1 can be destroyed using the `destruct` transition.
2. **Self-loop:** During the getter and setter execution, it is possible that the SC instance Cloud 1 triggers recursively new transitions in order to either get or set other information about the internal state. In contrast to thicker edges, the self-loop edges necessarily indicate that: (*i*) the SC instance Cloud 1 changes its state from the current state to the new state, (*ii*) performs the activity linked to the new state, and (*iii*) returns to the starting state.
3. **Remaining transitions:** The remaining transitions behave as the self-loop transition. In particular, since the constructor is responsible for initializing the internal state of the instance, it can also call back getter/setter functions.
   In addition, a transition from the `Deployed` state to either the `Setter Execution` or the `Getter Execution` state is triggered to execute the corresponding function.

3.3. State transition to other SCs

During its execution, the SC instance Cloud 1 may interact with other instances of a SC Cloud 2 that have the same or different type. We use dotted lines to represent the external transitions that involve a state of another SC instance Cloud 2. In Ethereum, by executing the constructor or `Setter Execution` the instance Cloud 1 can call back the creation of another SC instance Cloud 2. Indeed, the constructor and the functions of the SC can be called recursively, but the depth of the recursion is limited (to about 1024) because the gas that can be forwarded in each call and the total number of slots available on the stack is limited. Furthermore, during the creation and the `Setter Execution`, the SC instance Cloud 1 can invoke the `Setter Execution` of Cloud 2 and get values from internal state of Cloud 2 (thought the `Getter Execution`). Finally, the `Getter Execution` state allows the Cloud 1 instance to interact with the same state of Cloud 2.

3.4. Use case of SCs and its lifecycle

In the following, we provide a detailed model of a general voting system implemented by using SCs. The system allows authorized users to vote only one proposal, takes care of counting the votes received by each proposal, and selects the winning proposal. Fig. 2 shows the behavior of the system and the relationships among the SC instances that compose the system. In particular, the voting system consists of

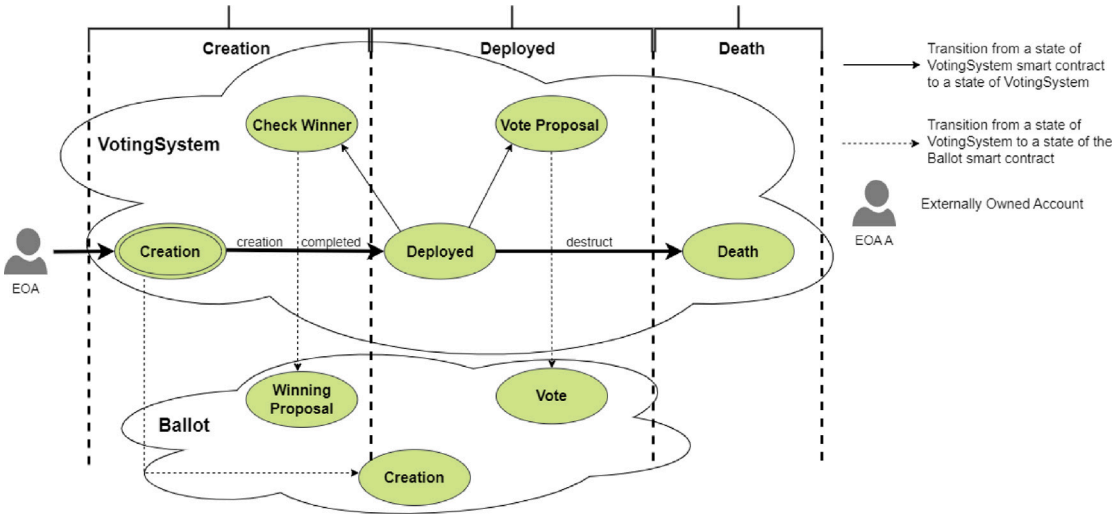


Fig. 2. Use Case: Life cycle of a generic SC instance implementing a voting system.

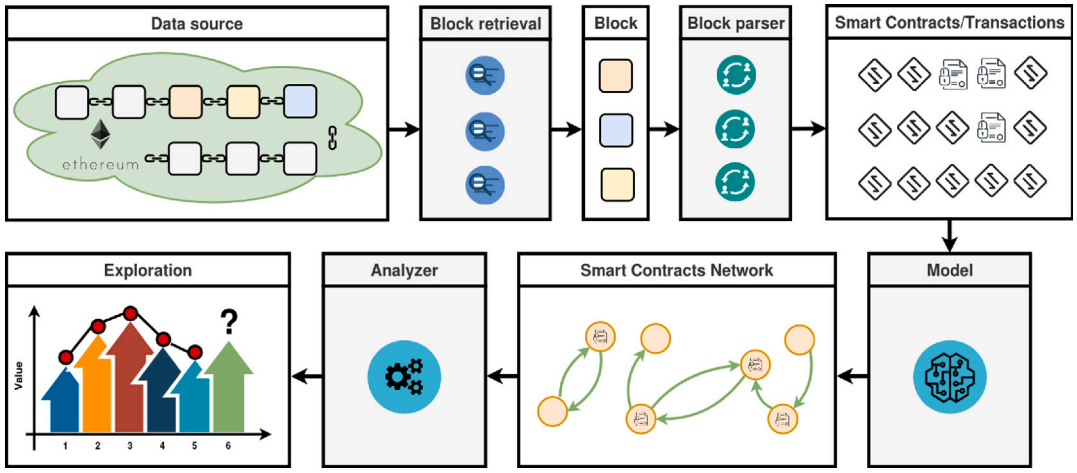


Fig. 3. General architecture of the proposed framework.

two SCs: the `VotingSystem` and the `Ballot` SC. The former allows users to define new poll and creates one `Ballot` contract for each poll. Once deployed, the `VotingSystem` allows users to vote for a proposal (Vote Proposal) and collects the preferences given by voters by exploiting the `Ballot` SC. Finally, the SC allows users to retrieve the winner of the poll (Check Winner) by getting from the `Ballot` SC the proposal with the largest number of votes.

4. EDIT: Ethereum Data Inspection Tool

In this section, we present our proposed Ethereum Data Inspection Tool (EDIT), by describing its architecture (Section 4.1) and the graph-model used to represent data (Section 4.2).

4.1. Architecture and working principle

Fig. 3 describes the high-level functionalities of the proposed system by showing the components and the data transformations involved during the analysis of the blockchain from a given source. The arrows in the Figure describe the data and execution flow resulting from the union of the different components. In particular, the input arrow of each component indicates the data received as input by the component, while the output arrow indicates the data returned by applying the functionality of the component to the input data. In the following, we provide a detailed description of each of our proposed components.

Data source. We design our framework such that different *data sources* can be considered to retrieve in parallel the Ethereum blockchain. Examples of data sources that can be used to retrieve information from the Ethereum blockchain are: block explorer platforms (e.g., Etherscan and Blockchair), Ethereum clients having archive node's capabilities to store state for every block since the genesis block (e.g., Geth and Hyperledger Besu), public datasets available on the Cloud (e.g., Ethereum in BigQuery provided by Google Cloud) and on database servers (e.g., MongoDB supports JSON-like documents, which are compatible with the Ethereum data format).

By default, the framework accesses blocks of the Ethereum main network (known as *mainnet*), but it can be configured to obtain information from other data sources based on the Ethereum protocol (such as, *Ropsten Testnet*,[3] *Rinkeby Testnet*,[4] or *GoChain Mainnet*[5]).

This component collects the set of blocks to be analyzed from the different *data sources*. In addition to the data sources, we enable the framework to accept as input the number of the *initial block* and the number of the *final block* to be recovered. By default, the *block retrieval* component exploits a progressive retrieval technique to collect the

[3] Ropsten Testnet: https://ropsten.etherscan.io/.
[4] Rinkeby Testnet: https://www.rinkeby.io/#stats.
[5] GoChain Mainnet: https://explorer.gochain.io.

whole set of blocks to be analyzed from the data sources, but different data retrieval approaches can also be implemented. In particular, starting from the *initial block* up to the *final block*, the *block retrieval* component requests one block at a time to each *data source*. The output of the block retrieval component is a list of blocks that belong to the input interval.

Block parser. Each of the blocks selected by the *block retrieval* component is taken as an input parameter by the *block parser* component, i.e., an interactive module that extracts relevant information from the block. For each block, the output returned by the *block parser* component is a list of both SCs and transactions involving such SCs.

Model. This component models blockchain information as a graph or a network, as described in Section 3. The resulting graph consists of nodes and edges representing the main entities of the Ethereum protocol, i.e., SCs, EOA, and transactions.

The SC model can be made more expressive by including additional levels of detail, e.g., edge labels, weights, or balance of each node in the networks. It is important to note that, as opposed to other systems (see Table 5), we consider also the time dimension in this component. In general, we refer to such a graph as a *SC network*.

Analyzer. Once the model has been built, the *Analyzer* component takes it as an input to analyze the *SC network*. The component includes a number of metrics from graph theory, data mining, and social network analysis. In particular, we design the proposed framework to support community detection, computation of network distances, flows, and importance measures (degree centrality, PageRank, HITS, etc.). This analysis enables the development of exploration/prediction models, which consider several dimensions of the *SC network* and help decision-making processes and business intelligence systems.

4.2. Graph-based data modeling

The model component uses a graph formalism to model blockchain information obtained from the data source as a *SC network*. We model a SC network $G(V, E)$ as a multigraph consisting of a set of vertices V and a set of directed edges E among vertices.

The set of vertices $V = SC \cup EOA$ of the network G consists of the SCs SC and the EOAs EOA collected from the data source. In particular, each vertex $v_i \in V$ could be either a SC $v_i \in SC$ or an EOA $v_i \in EOA$ and it is identified by its addresses i.

The set of edges $E = Tnx \cup inTnx$ of the SC network G can represent both transactions between EOAs and SCs (Tnx) or transactions between two different SCs ($inTnx$), also known as internal transactions. Each directed edge $\vec{t_i} \in E$ consists of some basic information $\vec{t_i} = (s_i, d_i)$, i.e., the unique identifier i of the transaction, the source address s_i, and the destination address d_i. The transaction hash is used to uniquely identify the corresponding edge $\vec{t_i} \in E$.

In the case of $\vec{t_i} \in Tnx$, the source s_i of $\vec{t_i}$ must necessarily be an EOA while the destination d_i must be a SC, i.e., $s_i \in EOA$ and $d_i \in SC$. Instead, in the case of $\vec{t_i} \in inTnx$, i.e., $\vec{t_i}$ is a transition in the life cycle of the SC and the source s_i of $\vec{t_i}$ must necessarily be a SC, i.e., $s_i \in SC$, while the destination d_i could be either an EOA or a SC, i.e., $d_i \in EOA \cup SC$. The SC network G can represent different edges $\vec{t_i} = (s_i, d_i) \in E$ and $\vec{t_j} = (s_j, d_j) \in E$ that have the same source $s_i = s_j$ and the same destination $d_i = d_j$ because the EOAs and SCs can start multiple independent transactions over time. In the case of an internal transition of the state-space diagram (see Section 3), the corresponding edge $\vec{t_i} \in inTnx$ also contains a reference to the parent transaction hash that triggered it. The framework allows to enrich at different granularity level the graph representation by considering several pieces of information obtained from the data source. In particular, the value being transacted and the value of the fee in *ether* are both information that can be attached to edges $\vec{t_i} \in E$ of the SC network $G(V, E)$.

Starting from this definition, we propose three different models for SC networks, which we exploit for the analysis by the proposed framework. We define these models as follows.

- $G1(V, E)$ where the set of edges $E = Tnx$ represents only the transactions between EOAs and SCs (Tnx) while the set of vertices $V = SC \cup EOA$ consists of both SCs and EOAs.
- $G2(V, E)$ where the set of edges $E = Tnx \cup inTnx$ represents both internal transactions ($inTnx$) and transactions between EOAs and SCs (Tnx) while the set of vertices $V = SC \cup EOA$ consists of both SC and EOAs.
- $G3(V, E)$ where the set of edges $E = inTnx$ represents only the internal transactions between SCs ($inTnx$). As a result, the set of vertices $V = SC$ consists of SCs.

4.2.1. Modeling the blockchain's time dimension

Very often, current analyses and modeling tools for the Ethereum blockchain assume that networks are represented by a static graph. However, in real life, the SC network G is dynamic due to different transactions occurring in different blocks [19]. For this reason, some edges in G are present only at specific time instants.

To allow for the full understanding of the temporal properties and the evolution of the *SC network*, detailed information on the temporal sequences of transactions must be considered by the model component. As described in the Ethereum documentation,[6] the blocks on the blockchain represent units of time and the blockchain itself has a temporal dimension. Indeed, the sequence of blocks on the chain represents the entire history of the system at discrete time points. Such temporal dimension can be included also in the SC network by attaching temporal information on the edges E of the SC graph $G(V, E)$. For this reason, the model component associates to each edge $\vec{t_i} \in E$ of the SC network $G(V, E)$ the block number b_i, from which the corresponding transaction was retrieved. The block number is a positive integer that indicates the length of the blockchain by uniquely identifying each block and it increases after the addition of a new block. Since the proposed framework accepts as input the number of the initial block n_{init} and the number of the final block n_{fnl} to be recovered, we denote by $l = n_{\text{fnl}} - n_{\text{init}} + 1$ the total number of consecutive blocks represented by the SC network $G(V, E)$.

Given a generic block number $B \in [\text{final block}, \text{initial block})$, we denote by G_T^B a snapshot of the SC network $G(V, E)$ representing the events on the T consecutive blocks $\{B, B+1, \ldots, B+T-1\}$ starting from block B. The total number T of consecutive blocks to be considered in a snapshot G_T^B must meet the following inequalities: $0 < T < l$ and $B + T - 1 \leq l$. The snapshot $G_T^B = (V_T^B, E_T^B)$ is a multigraph which consists of the directed edges $E_T^B = \{\vec{t_i} \in E | b_i \in [B, B + T - 1]\}$, corresponding to transactions of the SC network $G(V, E)$ occurred in the considered blocks interval $[B, B+T-1]$. Instead, the set $V_T^B = \{s_i \in V | \vec{t_i} \in E_T^B\} \cup \{d_i \in V | \vec{t_i} \in E_T^B\}$ consists of the vertices of the SC network G which are involved as sources or destinations of the edges E_T^B.

In order to capture the evolution and the dynamic aspects of the SC network $G(V, E)$, we further increase the capability of the proposed framework by allowing the model component to return a temporal version of G. The temporal SCs network for a number T of consecutive blocks is denoted by G_T and it consists of $L = \lceil l/T \rceil$ different consecutive snapshots of G, denoted as

$$G_T = [G_T^B, G_T^{B+T}, G_T^{B+2T}, \ldots, G_T^{B+L-T}] \tag{1}$$

where the block number B is equal to the initial block number n_{init} to be recovered. The purpose of this model is to enable the analysis of the system over time by focusing only on the events happening between entities at specific time windows [19].

5. Evaluation on Ethereum temporal data analysis

We provide a proof-of-concept implementation, providing the experimental setup in Section 5.1 and data retrieval procedure in Section 5.2. We then show how EDIT provides possibilities for building models in Section 5.3, describe the identified features in Section 5.4, and show the predictability of the identified features in Section 5.5.

[6] https://ethereum.org/en/developers/docs/accounts/

Table 1
Characteristics of the dataset and properties of the graphs derived from the dataset.

Dataset			
Blocks	0-3 999 999	Days	710
Start time	Feb-13-2016	End time	Jul-09-2017
Property	**G1**	**G2**	**G3**
Nodes	1 341 312	20 857 641	19 830 017
Edges	11 664 930	77 186 567	65 521 532
EOA	877 210	20 026 460	19 301 205
SCs	464 102	831 181	528 812
SC Calls	11 256 650	22 528 523	11 271 768
SC Creation	408 280	881 957	473 677
Funds Transfer	–	53 776 087	53 776 087

5.1. Experimental setup

We use the Ethereum main net as a reference blockchain because of its popularity and the availability of consolidated development tools. We first configure the framework to exploit Hyperledger Besu [20], EthereumJ,[7] and Etherscan[8] as data sources.

Hyperledger Besu and EthereumJ are both Java-based Ethereum client that can be configured to run on the main net. Both provide JSON-RPC APIs to monitor nodes in an Ethereum network and can be hence used for both the block retrieval and the block parser components. Instead, Etherscan is an online block explorer platform which can be queried to retrieve block information from different blockchains.

The *block retrieval* component is configured to collect the SCs and transaction information of the Ethereum main net from the different data sources, while the *block parser* components is responsible for extracting the information necessary to build the model.

The *model* component and the *analysis* components are developed by using the Java Universal Network/Graph Framework,[9] i.e., a widely adopted open-source library providing standard algorithms for graph/network modeling, analysis, and visualization. In particular, such components are configured to return and analyze our three different proposed SC network models (see Section 4.2): $G1$, $G2$, and $G3$.

5.2. Data retrieval

To model the temporal properties of the SC network, we have to retrieve from the Ethereum main net a sequence of l contiguous blocks. However, syncing the framework with a complete snapshot of the blockchain data will result in high resource consumption because of the storage, bandwidth, and computation power necessary to manage terabytes of data. For this reason, in this paper, we focused on the retrieval and analysis of the first 4 000 000 blocks stored on the blockchain: from block 0, the genesis block, to block 3 999 999. As shown in Table 1, the miners take 710 days to include the previously selected blocks on the blockchain and the time interval ranges from Feb-13-2016 to Jul-09-2017.

The model component uses such data to build models $G1, G2$, and $G3$ representing three different SC networks. Each model exposes different properties which depend on the blockchain data. In particular, $G2$ is the largest graph model generated by the proposed system, consisting of more than 20 million nodes, the majority of which (96%) represent EOA. The number of edges in $G2$ exceeds 77 million edges and they include the creation of a SC (SC Creation - 1%), funds transfer between parties (Funds Transfer - 70%), and transaction with a SC (SC Calls - 29%). Instead, graph $G3$ consists of about 19 million nodes, the majority of which (97%) represent EOA. The number of edges is about 65 million and they include only internal transition of SCs.

[7] EthereumJ: https://github.com/ethereum/ethereumj.
[8] Etherscan: http://etherscan.io/.
[9] JUNG: http://jung.sourceforge.net/.

Table 2
Summary of the granularity level and the size of the resulting target set for each temporal SCs network.

Parameter	G_1^T	G_2^T	G_3^T
consecutive blocks (T)		50000	
consecutive blocks of activity		50	
target set (SC)	121	132	15
target set (EOA)	16	16	0

In particular, the majority of the edges are related to funds transfer between parties (82%), transactions with a SC (17%), and creation of a SC (1%). Finally, $G1$ is the smallest graph model and it consists of about 1 million nodes and 11 million edges. Most of the nodes in $G1$ (65%) belong to the EOAs while edges consist of SCs creation (3%) and calls (97%).

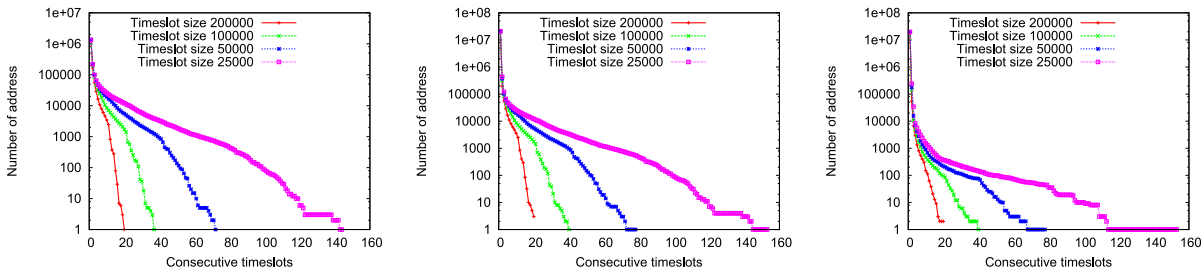
5.3. Model building

In order to capture the dynamic aspects of the SCs, we build for each graph $G1$, $G2$, and $G3$, a temporal SCs network G_1^T, G_2^T, and G_3^T consisting of a sequence of L graphs. Each of them represents the events that occurred on T consecutive blocks of the Ethereum blockchain. Since each temporal SCs network consists of a sequence of L graphs, it can provide different levels of granularity, depending on the number T of consecutive blocks in each graph. In order to select the proper granularity level of the temporal SCs network, we set different numbers T of consecutive blocks from {2 000 000, 1 000 000, 50 000, 25 000} in order to obtain a batch of temporal SC networks, each consisting of a sequence of L graphs in {20, 40, 80, 160}. For example, the temporal SCs network $G_1^2$00000 consists of a sequence of 20 graphs representing interactions that occur among Ethereum entities for different subchains of blocks. In order to increase the effectiveness of the data and make them more suitable for the prediction task, we analyze the activity of each Ethereum addresses by investigating how often they interact with the blockchain. Fig. 4 shows the number of addresses that have been involved in transactions with other addresses in at least C consecutive graphs of each temporal SCs network. As we expected, most Ethereum addresses perform very few transactions and interact occasionally with the blockchain, regardless of the granularity level of the temporal SCs network.

In order to obtain a consistent set of Ethereum addresses that can be used as a target in our prediction task, we select the temporal SCs network with granularity level $T = 50000$ as a reference model for our evaluation and we focused on the set of Ethereum addresses that interact with the blockchain in at least 50 consecutive graphs. In particular, the resulting target sets will be used as input for the prediction task and they consist of 136 addresses for G_1^{50000}, 148 addresses for G_2^{50000}, and 15 addresses for G_3^{50000}. Table 2 summarizes the number of EOA and SCs that will be used as targets for the prediction task and the parameters used by the framework to provide the selected granularity level.

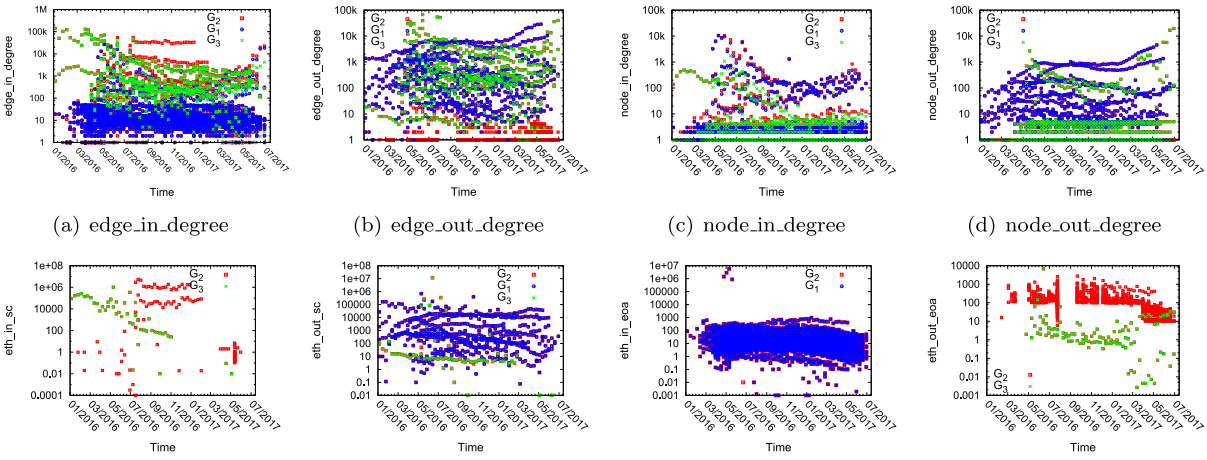
5.4. Properties of the temporal SCs network

The temporal SCs network supports a broad range of measures that can be used to characterize EOA and SCs. For example, some basic measures include counting the number of incoming/outgoing edges or counting the vertices that interact with a specific vertex. Other complex measures that can be computed on the proposed temporal SCs networks are those based on the connectivity between vertices, the importance of a vertex, the reciprocity of interactions between vertices, and the similarity between vertices. In the following, we focus on basic measures that are able to capture interesting and predictable properties related to EOA and SCs. In particular, as shown in Table 3, the properties computed on the node u of each temporal SCs network measure the number of incoming edges of u ($edge\_in\_degree_u$) over time, the number of outgoing edges of u ($edge\_out\_degree_u$), the distinct number of



(a) G_1^T (b) G_2^T (c) G_3^T

Fig. 4. Number of addresses who have been involved in transaction with other address (y-axis) in at least c consecutive graphs (x-axis) of the temporal SCs network.



(a) edge_in_degree (b) edge_out_degree (c) node_in_degree (d) node_out_degree

(e) eth_in_sc (f) eth_out_sc (g) eth_in_eoa (h) eth_out_eoa

Fig. 5. Properties of the nodes of each temporal SCs network.

Table 3
Properties of the temporal SCs networks.

Properties	Description
edge_in_degree	Number of incoming edges
edge_out_degree	Number of outgoing edges
node_in_degree	Distinct number of nodes connected by incoming edges
node_out_degree	Distinct number of nodes connected by outgoing edges
eth_in_sc	Total number of ETH received from SCs
eth_out_sc	Total number of ETH sent to SCs
eth_in_eoa	Total number of ETH received from EOA
eth_out_eoa	Total number of ETH sent to EOA

nodes connected by incoming edges to u ($node\_in\_degree_u$), the distinct number of nodes connected by outgoing edges to u ($node\_out\_degree_u$), the total number of ETH received by u from SCs ($eth\_in\_sc_u$), the total number of ETH sent from u to SCs ($eth\_out\_sc_u$), the total number of ETH received by u from EOAs ($eth\_in\_eoa_u$), and the total number of ETH sent from u to EOAs ($eth\_out\_eoa_u$).

Fig. 5 shows how the values of the aforementioned properties computed on all the nodes of the different temporal SCs networks distribute over time. In particular, the scatter plots of Figs. 5(b), 5(a), 5(c), and 5(d) are related to the structural properties of the nodes and they expose high variability in their values regardless of the type of the temporal SCs network, suggesting that the considered nodes are heterogeneous in their distribution. Instead, the scatter plots of Figs. 5(e), 5(f), 5(g), and 5(h) consider the values of ETHs transferred between different types of nodes and the collection of points appears to be very clustered. Furthermore, Figs. 5(e) and 5(h), which respectively show the number of ETHs received from SCs and sent to EOA by each user, do not include data of the temporal SCs network G_1. The reason for this is that G_1 represents only transactions between EOA and SCs, and as a result, transactions initiated by SCs are not taken into account. Instead, Fig. 5(g), which shows the total number of ETHs received from EOA, does not include the temporal SCs network G_3 because such model considers only internal transactions between SCs.

5.5. Prediction of temporal SCs properties

In our proposed framework, we use the properties of the temporal SCs networks to learn several prediction algorithms having different configurations, which are summarized in Table 4.

Among the several algorithms available in the literature, we consider widely known prediction models based on K-Nearest Neighbors (K-NN) [23], Linear Regression (LR), Multilayer Perceptron (MP), Regression Tree (REPTree), Support Vector Machine for Regression (SMO) [21,22], Random Tree (RTree) [24], Forest of Random Trees (RForest) [25], Decision List Regressor (MD5) [26], and Decision Table Regressor (DT) [27]. Our framework has been configured to utilize 90% of the data-trace related to a user for the training phase and the remaining 10% for testing. The training set is used to derive a set of lagged features (at most 12) capturing temporal information and periodicity of the target measure (e.g., AM indicator, day of the week, month, quarter, etc.). The performance of each algorithm is presented in terms of MAE between the true values and the predicted values.

Fig. 6 compares the performance of the different algorithms by showing the distribution of the normalized MAE obtained by all the configurations in order to predict the properties (see Table 3) on different smart contracts temporal network models (see Table 2). The box on each plot denotes the region between the first and third quartiles while the horizontal line represents the median value. The error bar on each box includes values that lie within 1.5 times the interquartile range.

Table 4
Algorithms and configurations used by the framework.

Algorithm	Configuration
K-nearest neighbors (K-NN)	c_1: $K = 2$ c_2: $K = 4$ c_3: $K = 16$ c_4: $1 \leq K \leq 16$ using hold-one-out
Linear Regression (LR)	c_1: M5 model tree attribute selection c_2: No attribute selection c_3: Greedy attribute selection c_4: Greedy attribute selection; no collinearity filer
Multilayer Perceptron (MP)	c_1: hidden layers = 1; nodes = (#lagged features+1)/2 c_2: hidden layers = 1; nodes = 1 c_3: hidden layers = 3; nodes = #lagged features+1,#lagged features,(#lagged features+1)/2 c_4: hidden layers = 2; nodes = #lagged features+1,(#lagged features+1)/2
Regression Tree (REPTree)	c_1: leaf instances = 2; split variance = 0.01; pruning = 3 c_2: leaf instances = 2; split variance = 0.001, pruning = 5 c_3: leaf instances = 2; split variance = 0.001, pruning =3 c_4: leaf instances = 4; split variance = 0.001, pruning = 3
SVM Regressor (SMO)	c_1: polynomial kernel; [21] implementation c_2: normalized polynomial kernel; [21] implementation c_3: Radial Basis Function (RBF) kernel; [21] implementation c_4: RBF kernel; [22] implementation
Random Tree (RTree)	c_1: split variance = 0.001 c_2: split variance = 0.001; random tie-breaking c_3: split variance = 0.001; use backfitting c_4: split variance = 0.0001
Random Trees Forest (RForest)	c_1: bag size = 50%; #trees = 50 c_2: bag size = 50%; #trees = 100 c_3: bag size = 100%; #trees = 50 c_4: bag size = 100%; #trees = 100
Decision List Regressor (MD5)	c_1: pruned rules; smoothed predictions c_2: unpruned rules; smoothed predictions c_3: pruned rules; unsmoothed predictions c_4: unpruned rules; unsmoothed predictions
Decision Table Regressor (DT)	c_1: Best first attributes selection, forward search c_2: Best first attributes selection, bi-directional search c_3: Greedy stepwise attribute selection c_4: Best first attribute selection; forward search; use nearest neighbor

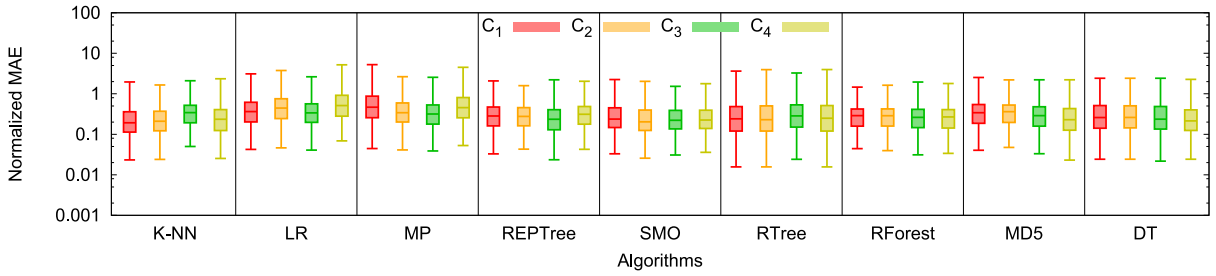
As shown by Fig. 6(a), prediction errors span a small range for G_1^T while for models G_2^T and G_3^T the intervals on the prediction errors are larger. Regarding the temporal SC network G_1^T, the K-NN algorithm has the best performance (normalized MAE 0.193) for configuration c_1 where the number of nearest neighbors K is set to 2, but there were no significant differences with other algorithms. Instead, SMO achieves good overall performance (normalized MAE 0.203) on configuration c_2, which uses the implementation proposed in [21] and uses normalized polynomial kernel. The Regression Tree model (REPTree) outperforms the other algorithms on the configuration c_3 with Normalized MAE of 0.235, where the minimum number of instances per leaf is 2, the minimum variance for split is 0.001, and the reduced error pruning factor is 3. The configuration c_4 attains the highest level of precision (normalized MAE 0.214) with DT algorithm using best-first forward search to select the optimal feature subsets and nearest neighbor to predict instance values.

The graph G_2^T appears to be the largest temporal smart contracts network considered in the experiments and the proposed framework achieves similar or worse performances to those of G_1^T and G_3^T. As shown in Fig. 4(b), except for configuration c_4, the algorithms that achieve lower errors are the same as G_1^T, i.e., K-NN with a normalized MAE of 0.373 for configuration c_1, SMO with a normalized MAE of 0.380, and REPTree for configuration c_3 with a normalized MAE of 0.421. Instead, for configuration c_4, the SMO algorithm exposes the lowest normalized MAE of 0.365 and it is configured to use Radial Basis Function (RBF) kernel and the implementation proposed in [22].

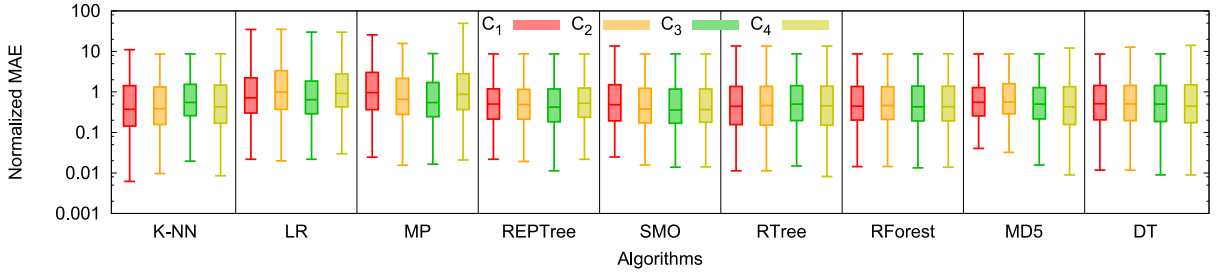
The graph G_3^T results to be the smallest temporal SC network that can be modeled via our framework and the results obtained from the predictions are shown in Fig. 4(c). The lowest prediction errors on configuration c_1 is obtained by the DT algorithm which uses best-first forward search to select the optimal feature subsets, resulting in a normalized MAE of 0.356. A similar error (0.364) is also achieved by the RForest algorithm on the configuration c_1, where the size of each bag is 50% the training set size and the number of trees in the random forest is 50. In the case of configuration c_2, the best accuracy is achieved by the K-NN algorithm with a number of nearest neighbors K set to 4, resulting in a normalized MAE of 0.213. Finally, the SMO algorithm achieves impressive performance on both configurations c_3 (with a normalized MAE of 0.155) and c_4 (with a normalized MAE of 0.194), which exploit the same RBF kernel function, but they are based on different implementations ([21,22], respectively).

The performance of the algorithms have been examined and assessed based on the specific accuracy achieved by each configuration on the different properties of the temporal smart contracts networks. Fig. 7 shows the detailed distribution of the MAE resulting from the best algorithm/configuration on all the properties of the temporal smart contracts network G_1^T, G_2^T, and G_3^T.

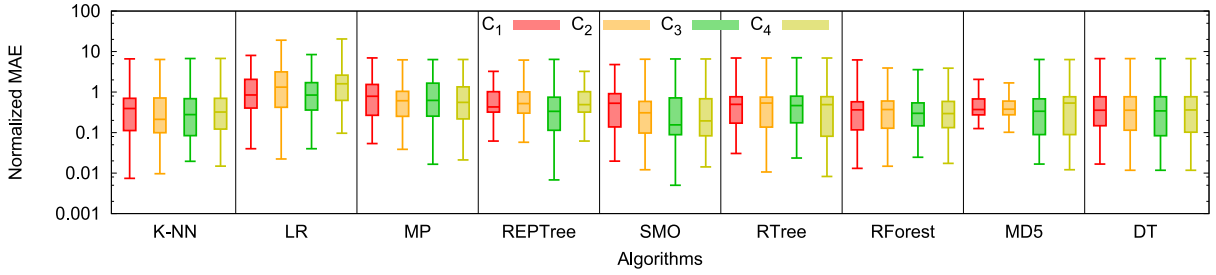
Regarding the prediction of the *egde_in_degree* property of the nodes, the K-NN algorithm exposes the lowest median MAE on all the temporal SC networks. In particular, the K-NN algorithm configured with a number of nearest neighbors K of 4 is able to reach a median MAE of 0.128 on G_1^T, 0.132 on G_2^T, and 0.136 on G_3^T. The same algorithm's configuration for K-NN outperforms all the others also on the prediction of the *edge_out_degree* property of the temporal smart contracts network G_3^T, resulting in a median MAE of 0.213. Instead, in the case of the temporal smart contracts network G_1^T and G_2^T, the prediction of the *edge_out_degree* property proved to be very accurate with the RForest and the RTree algorithms respectively. Indeed, the RForest algorithm



(a) G_1^T



(b) G_2^T



(c) G_3^T

Fig. 6. Performances achieved by different algorithms and configurations on the temporal smart contracts network G_1^T, G_2^T, and G_3^T where $T = 50\,000$.

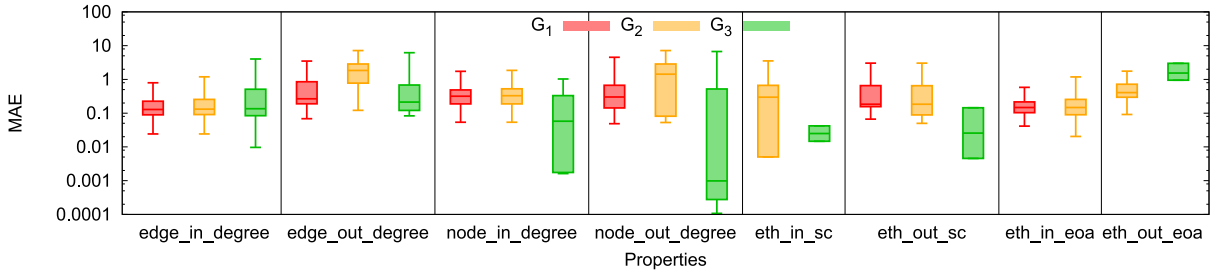


Fig. 7. Performances achieved by the framework on individual properties of the temporal smart contracts network G_1^T, G_2^T, and G_3^T where $T = 50\,000$.

obtains the best median MAE of 0.268 on G_1^T with the configuration c_4, where the size of each bag is 100% the training set size and the number of trees in the random forest is 100. A median MAE of 1.842 is achieved by the configuration c_4 of the RTree algorithm (with a spit variance of 0.0001) for the prediction of the *edge_out_degree* property on the temporal smart contracts network G_2^T.

The *node_in_degree* property of the nodes in the temporal smart contracts network G_1^T and G_2^T is predicted with a median MAE of 0.317 and 0.329, respectively, by using the configuration c_4 of the DT algorithm where nearest neighbor and best-first forward search is used to select the optimal feature subsets. As concern the prediction of the *node_in_degree* on the temporal smart contracts network G_3^T, the SMO algorithm has the lowest median MAE of 0.063 when the RBF kernel function and the implementation proposed in [22] is used.

The forecasting of the *node_out_degree* property on the temporal smart contracts network G_1^T, G_2^T, and G_3^T, is very accurate by using the SMO, RTree and the MP algorithm, respectively. In particular, the SMO algorithm achieves the lowest median MAE of 0.302 by using a polynomial kernel (configuration c_1) while the RTree algorithm has a median MAE of 1.429 if configured for breaking ties randomly and using a split variance of 0.001 (configuration c_2). Finally, the MP algorithm is the most accurate on temporal smart contracts network G_3^T and it achieves a median MAE of 0.001 by using 3 different hidden layers (configuration c_3).

As explained in Section 5.4, the *eth_in_sc* and the *eth_out_eoa* properties are defined for temporal smart contracts network G_2^T and G_3^T. Regarding the *eth_in_sc* property, the K-NN algorithm configured to use respectively 4 (configuration c_2) and 2 (configuration c_3) nearest neighbors performs successfully on both G_2^T (median MAE 0.298) and

Table 5
Summary of the capabilities of the considered analysis tools.

Approach	Modeling	Transactions data			Accounts	Tasks	Year	API
		Tx	*intTx*	*time*				
Ethslurp [28]	Table	✓	✗	✗	EOA/SC	Monitoring	2018	CLI
SC-Watch [29]	Table	✓	✗	✗	SC	Monitoring	2019	CLI
Eventeum [30]	Table	✓	✗	✗	EOA/SC	Monitoring	2018	REST
EthQL [31]	Table	✓	✗	✗	EOA/SC	Monitoring, exploratory analysis	2019	GraphQL
TrueBlocks [32]	Table	✓	✗	✗	EOA/SC	Monitoring, exploratory analysis	2018	C++
Presto-Ethereum [33]	Table	✓	✗	✗	EOA/SC	Monitoring, exploratory analysis	2019	SQL
B. M. et al. [34]	Table	✓	✗	✗	EOA/SC	Exploratory analysis	2017	(no)SQL
Dune Analytics [35]	Table	✓	✓	✗	EOA/SC	Monitoring, exploratory analysis	2019	SQL
SmartAnvil [36]	Graph	✓	✓	✗	SC	Monitoring, semantic Analysis	2019	GraphQL
C. T. et al. [37]	Graph	✓	✓	✗	EOA/SC	Exploratory analysis	2020	–
W. J., et al. [38]	Graph	✓	✗	✓	EOA/SC	phishing classification	2020	–
C. S. et al. [39]	Graph	✓	✗	✓	EOA	time behavior analysis	2021	–
L. D. et al. [40]	Temporal ego-network	✓	✗	✓	EOA/SC	temporal link prediction	2020	Python
B. H. et al. [41]	Temporal ego-network	✓	✗	✓	EOA/SC	evolution analysis	2022	–
TTAGN [42]	Temporal Graph	✓	✗	✓	EOA	phishing classification	2022	–
Q. B. et al. [43]	Temporal Graph	✓	✓	✓	EOA/SC	evolution analysis	2022	–
Z. L. et al. [44]	Temporal Graph	✓	✓	✓	EOA/SC	evolution analysis and community continuation	2021	–
EDIT	**Temporal Graph**	✓	✓	✓	**EOA/SC**	**evolution analysis and prediction**		

G_3^T (median MAE 0.028). The K-NN algorithm using the hold-one-out method to select the number K of nearest neighbors (configuration c_4) outperforms the other algorithms on the prediction of the *eth_out_eoa* property in G_3^T, resulting in a median MAE of 1.545. Instead, the SMO with RBF kernel (configuration c_3) is one of the most accurate algorithms in predicting the *eth_out_eoa* property in G_2^T and it achieves a median MAE of 0.407.

The *eth_out_sc* property on both G_1^T and G_2^T and the *eth_in_eoa* property on G_2^T is predicted with a median MAE of about 0.184 by the SMO algorithm configured with the normalized polynomial kernel (configuration c_2). Instead, the forecasting of the *eth_out_sc* property on G_3^T proved to be very accurate (median MAE of 0.0743) by using DT algorithm based on best-first forward-search features selection (configuration c_1). Finally, the *eth_in_eoa* property of the temporal smart contracts network G_1^T is predicted with a median MAE of 0.147 by the K-NN algorithm configured to use 4 nearest neighbors (configuration c_1).

6. Related work and comparison

In this section, we present a synoptic state-of-the-art of the software tools and solutions proposed for analysis of the blockchain data by both academia and enterprises. Each of them is categorized in terms of the following capabilities:

Modeling: the general formalism used to model both data extracted from the blockchain and their relationships, such as table, graph, or temporal graph.

Transactions Data: the ability of the approach to consider different types of transactions data, i.e., normal transactions (*Tx*), internal transactions (*inTx*), and time of the transactions (*time*).

Accounts: the entities considered in the model (e.g., EOAs or SCs).

Tasks: the task/problem that the framework or designed solution intends to solve.

Year: year in which each analysis tool was proposed.

API: the type of interfaces provided by the tool.

Table 5 considers the tools proposed in the survey by Khan et al. [45] and extends them to provide an analysis of the capabilities each tool based on the aforementioned features. We notice that a large part of the solutions proposed in the current literature focus on retrieving data from the blockchain and indexing them in a database in order to monitor events [28–30,36], make exploratory analyses by using advanced mechanisms for querying data [34,37] or both [31–33,35].

As concern the first task, Smart-contract-watch [29] is a tool for monitoring SC based on generated transactions and events. This is done by sending requests to an Ethereum node via JSON RPC calls. Indeed, Eventeum [30] and EthSlurp [28] focused on both SC and EOA. In particular, Eventeum [30] is a server that allows to listen for specific event emissions from the Ethereum network. Currently, it supports the notification of both events emitted by a SC and transactions with a specific hash, sender or recipient address. Instead, EthSlurp [28] allows to retrieve, to parse and to explore Ethereum blockchain transactions from any Ethereum address (SC or EOA) by exploiting public data sources (such as Etherscan.io).

As concern the task of exploratory analysis, Bartoletti M. et al. [34] proposed a framework supporting the retrieval of blockchain transaction and the integration of such data with external information (such as exchange rates, address tags, and protocol identifiers) from other sources. The view of the data, which are parsed and organized either in a SQL or NoSQL database, can be analyzed by using the query language while the framework is released as an open-source Scala library.

EthQL [31], TrueBlocks [32], and Presto-Ethereum [33] are among the solutions that allow both monitoring and to make simple exploratory analysis on the normal transaction data while Dune Analytics [35] considers also internal transaction. Indeed, EthQL [31] and TrueBlocks [32] are tools that connect to an Ethereum client through standard JSON-RPC API and expose endpoints that allow to query, and to visualize human-readable Ethereum data from the Ethereum blockchain. Presto-Ethereum [33] is a distributed SQL query engine where servers are responsible for executing tasks and fetch data from Ethereum blockchain, managing workers and collecting results. Instead, Dune Analytics [35] allows to retrieve, query, and visualize human-readable Ethereum data related to both normal and internal transactions.

Graph modeling has been initially introduced to perform static analysis on smart contracts or to characterize money transfers between accounts. For example, SmartAnvil [36] is an open-source platform that is responsible for static analysis of SCs and for accessing deployed contracts, transactions, and blocks. The static analysis of the solidity code is performed by using both bytecode reverse engineering and abstract syntax trees. Finally, it offers a query language to navigate and access transactions and deployed contracts. Instead, Chen, Ting, et al. [46] leveraged graph analysis to characterize money transfer, SC creation, and SC invocation. The methodology used for the analyzing exploits different graphs to characterize both normal and internal transactions among EOA and SCs. The graph analysis step computes

different metrics on the entire static graphs (namely, the degree distribution, clustering, degree correlation, page rank, weak/strong ties, and assortativity).

In contrast to the proposed approach, the introduction of the temporal dimension in graph modeling has been mainly used to perform time behavior analysis [39] and for classification tasks [38,42]. Indeed, authors of [39] consider transactions that involve Ethereum Non-Fungible Tokens (ERC-721) and model them as a multi-directed weighted graph. An individual NFT transaction graph is created for each token, and time behavior analysis is realized by considering the timestamps at which transactions are made. The main task involves the analysis of the exchanged volume of transactions and the identification of the major owners of tokens.

Indeed, in [38] a graph is used to classify Ethereum addresses as phishing scams or not while time is used to infer transition probability for random walk-based network embedding method. Instead, authors of [42] exploits LSTM model to encode the ordered sequence of normal transactions occurred between EOAs into a vector representation, reducing graph dimensionality for phishing address detection.

The approaches in [40,41] differ from the proposed solution because they exploit the temporal graph to represent ego-network related to a specific account, i.e., the network made up of a central account (*ego*), its neighbors, and the normal transactions between them. In [40] temporal ego network is obtained by using random walk-based graph representation with different sampling strategies for selecting the edge to traverse and the accuracy of each strategy is evaluated in the context of the temporal link prediction. Instead, authors of [41] provides temporal analysis of different ego networks related to ICO, mining, exchange, ponzi, phishing accounts.

Similar to our approach, the authors of [43] further advance the temporal analysis of the Ethereum blockchain by proposing three different temporal graph models: user-to-user, contract-to-contract, and user-to-contract. Each model is based on both external and internal transactions. However, the time window is fixed to 180 days and shifted with a granularity of 45 days while the temporal network is used to analyze the evolution of the degree distribution, the number of transactions, the recurrent interaction patterns (or *motifs*), and the temporal variations of the system (*burstiness*). Finally, authors of [44] focused on community continuation prediction by considering the network of all internal transactions, the network of all internal transactions between smart contracts, the external transaction graph, and the tokens transactions graph. Unlike the proposed framework, they studied temporal variations of the four networks, considering yearly, 6-monthly, 3-monthly, and monthly graphs. Prediction of the survival of a community in a specific month is performed based on data acquired from three consecutive months.

Comparison against the related works reveals that the proposed framework is more comprehensive and flexible than the existing tools that mainly focus on retrieving and indexing data from the blockchain for monitoring or exploratory analysis purposes. Furthermore, the proposed approach introduces the temporal dimension in graph modeling, which allows us to capture the dynamic evolution of the blockchain network and its entities over time. Another important difference is that the proposed approach does not rely on fixed time windows or sampling strategies, but adapts to the varying activity and complexity of the blockchain network by enabling various types of tasks on the temporal graph.

7. Conclusion

Blockchain has gained popularity in recent years, becoming one of the most used distributed databases which store different types of information and manage them with a decentralized computation model. This paper investigates how blockchain affects the interactions and the flow of information among entities, with the aim of providing a general framework for analyzing the evolution of blockchain data. The proposed approach combines the interaction patterns derived from the blockchain with temporal information in order to model and predict interesting properties of the system. The framework has been implemented and tested on the Ethereum blockchain and the accuracy in predicting different blockchain properties is measured.

To further improve performance metrics, we intend to explore other properties of the data stored on the blockchain, such as the influence of entities and the strength of the ties, and to apply advanced prediction models based on Deep Learning.

CRediT authorship contribution statement

Andrea De Salve: Conceptualization, Coding, Writing. **Alessandro Brighente:** Conceptualization, Writing. **Mauro Conti:** Conceptualization, Revision.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Alessandro Brighente reports financial support was provided by European Union.

Data availability

The data is publicly available on the Ethereum blockchain.

Acknowledgments

This work was partially supported by the project “PRE-CUBE-PREdizione, PREvenzione, PREdisposizione” - CUP G69J18001170007 and by the project “Secure and TRaceable Identities in Distributed Environments (STRIDE)” included in the Spoke 5 of the Research and Innovation Program PE00000014, “SEcurity and RIghts in the CyberSpace (SERICS)”, under the National Recovery and Resilience Plan, funded by the European Union - NextGenerationEU.

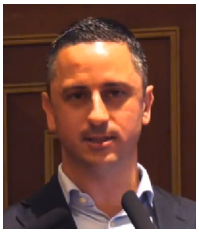
References

[1] Qiang Wang, Min Su, Integrating blockchain technology into the energy sector—from theory of blockchain to research and application of energy blockchain, Comp. Sci. Rev. 37 (2020) 100275.
[2] Le Jiang, Xinglin Zhang, Bcosn: A blockchain-based decentralized online social network, IEEE Trans. Comput. Soc. Syst. 6 (6) (2019) 1454–1466.
[3] Andreas Kamilaris, Agusti Fonts, Francesc X. Prenafeta-Boldú, The rise of blockchain technology in agriculture and food supply chains, Trends Food Sci. Technol. 91 (2019) 640–652.
[4] Anton Hasselgren, Katina Kralevska, Danilo Gligoroski, Sindre A. Pedersen, Arild Faxvaag, Blockchain in healthcare and health sciences—a scoping review, Int. J. Med. Inf. 134 (2020) 104040.
[5] Weiqin Zou, David Lo, Pavneet Singh Kochhar, Xuan-Bach Dinh Le, Xin Xia, Yang Feng, Zhenyu Chen, Baowen Xu, Smart contract development: Challenges and opportunities, IEEE Trans. Softw. Eng. 47 (10) (2019) 2084–2106.
[6] Vatsal Patel, Sutharshan Rajasegarar, Lei Pan, Jiajun Liu, Liming Zhu, EvanGCN: Evolving graph deep neural network based anomaly detection in blockchain, in: International Conference on Advanced Data Mining and Applications, Springer, 2022, pp. 444–456.
[7] Muhammad Saad, Jinchun Choi, DaeHun Nyang, Joongheon Kim, Aziz Mohaisen, Toward characterizing blockchain-based cryptocurrencies for highly accurate predictions, IEEE Syst. J. 14 (1) (2019) 321–332.
[8] Wenhan Hou, Bo Cui, Ru Li, A survey on blockchain data analysis, in: 2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC, IEEE, 2021, pp. 357–365.
[9] Gavin Wood, Ethereum: a secure decentralised generalised transaction ledger, 2014.
[10] Andrew S. Tanenbaum, Maarten Van Steen, Distributed Systems: Principles and Paradigms, Prentice-Hall, 2007.
[11] Gavin Wood, et al., Ethereum: A secure decentralised generalised transaction ledger, Ethereum Project Yellow Paper 151 (2014) 1–32.
[12] Ferenc Béres, István A. Seres, András A. Benczúr, Mikerah Quintyne-Collins, Blockchain is watching you: Profiling and deanonymizing ethereum users, in: 2021 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS, IEEE, 2021, pp. 69–78.

[13] Friedhelm Victor, Address clustering heuristics for ethereum, in: Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February (2020) 10–14 Revised Selected Papers, 2020, pp. 617–633.
[14] Yue Gao, Jinqiao Shi, Xuebin Wang, Ruisheng Shi, Zelin Yin, Yanyan Yang, Practical deanonymization attack in ethereum based on p2p network analysis, in: 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking, ISPA/BDCloud/SocialCom/SustainCom, IEEE, 2021, pp. 1402–1409.
[15] Xiao Liu, Zaiyang Tang, Peng Li, Song Guo, Xuepeng Fan, Jinbo Zhang, A graph learning based approach for identity inference in dapp platform blockchain, IEEE Trans. Emerg. Top. Comput. (2020).
[16] Melanie Swan, Blockchain: Blueprint for a New Economy, O'Reilly Media, Inc., 2015.
[17] Phillip Rogaway, Thomas Shrimpton, Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance, in: International Workshop on Fast Software Encryption, Springer, 2004, pp. 371–388.
[18] Mauro Conti, E. Sandeep Kumar, Chhagan Lal, Sushmita Ruj, A survey on security and privacy issues of bitcoin, IEEE Commun. Surv. Tutor. 20 (4) (2018) 3416–3452.
[19] Petter Holme, Jari Saramäki, Temporal networks, Phys. Rep. 519 (3) (2012) 97–125.
[20] Hyperledger. Hyperledger Besu. https://besu.hyperledger.org/en/stable/. (Online; Accessed 16 June 2021).
[21] S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, K.R.K. Murthy, Improvements to the smo algorithm for svm regression, in: IEEE Transactions on Neural Networks, 1999.
[22] A.J. Smola, B. Schoelkopf, A tutorial on support vector regression, in: NeuroCOLT2 Technical Report NC2-TR-1998-030, 1998.
[23] D. Aha, D. Kibler, Instance-based learning algorithms, Mach. Learn. 6 (1991) 37–66.
[24] Sushilkuma Kalmegh, Analysis of weka data mining algorithm reptree, simple cart and randomtree for classification of indian news, Int. J. Innov. Sci. Eng. Technol. 2 (2) (2015) 438–446.
[25] Leo Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[26] Geoffrey Holmes, Mark Hall, Eibe Frank, Generating rule sets from model trees, in: Twelfth Australian Joint Conference on Artificial Intelligence, Springer, 1999, pp. 1–12.
[27] Ron Kohavi, The power of decision tables, in: 8th European Conference on Machine Learning, Springer, 1995, pp. 174–189.
[28] EthSlurp. EthSlurp. http://ethslurp.com. [Online; accessed].
[29] SCW. smart-contract-watch. https://github.com/Neufund/smart-contract-watch. [Online; accessed].
[30] Eventeum. Eventeum. https://github.com/ConsenSys/eventeum. [Online; accessed].
[31] Ethql. Ethql. https://github.com/ConsenSys/ethql. [Online; accessed].
[32] TrueBlocks. TrueBlocks.io. http://trueblocks.io. [Online; accessed].
[33] Presto. Presto. https://prestosql.io/. [Online; accessed].
[34] Massimo Bartoletti, Stefano Lande, Livio Pompianu, Andrea Bracciali, A general framework for blockchain analytics, in: Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, ACM, 2017, p. 7.
[35] Dune Analytics. Dune Analytics. https://www.duneanalytics.com/. [Online; accessed].
[36] SmartAnvil. SmartAnvil. https://github.com/smartanvil. [Online; accessed].
[37] Ting Chen, Zihao Li, Yuxiao Zhu, Jiachi Chen, Xiapu Luo, John Chi-Shing Lui, Xiaodong Lin, Xiaosong Zhang, Understanding ethereum via graph analysis, ACM Trans. Internet Technol. (TOIT) 20 (2) (2020) 1–32.
[38] Jiajing Wu, Qi Yuan, Dan Lin, Wei You, Weili Chen, Chuan Chen, Zibin Zheng, Who are the phishers? phishing scam detection on ethereum via network embedding, IEEE Trans. Syst. Man Cybern. Syst. 52 (2) (2020) 1156–1166.
[39] Simone Casale-Brunet, Paolo Ribeca, Patrick Doyle, Marco Mattavelli, Networks of ethereum non-fungible tokens: A graph-based analysis of the erc-721 ecosystem, in: 2021 IEEE International Conference on Blockchain, Blockchain, IEEE, 2021, pp. 188–195.
[40] Dan Lin, Jiajing Wu, Qi Yuan, Zibin Zheng, Modeling and understanding ethereum transaction records via a complex network approach, IEEE Trans. Circuits Syst. II 67 (11) (2020) 2737–2741.
[41] Baoying Huang, Jieli Liu, Jiajing Wu, Quanzhong Li, Hao Lin, Temporal analysis of transaction ego networks with different labels on ethereum, in: IEEE International Symposium on Circuits and Systems, ISCAS 2022, Austin, TX, USA, May 27 - June 1 2022, IEEE, 2022, pp. 3517–3521.
[42] Sijia Li, Gaopeng Gou, Chang Liu, Chengshang Hou, Zhenzhen Li, Gang Xiong, TTAGN: temporal transaction aggregation graph network for ethereum phishing scams detection, in: Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, Lionel Médini (Eds.), WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April (2022) 25-29, ACM, 2022, pp. 661–669.
[43] Qianlan Bai, Chao Zhang, Nianyi Liu, Xiaowei Chen, Yuedong Xu, Xin Wang, Evolution of transaction pattern in ethereum: A temporal graph perspective, IEEE Trans. Comput. Soc. Syst. 9 (3) (2022) 851–866.
[44] Lin Zhao, Sourav Sen Gupta, Arijit Khan, Robby Luo, Temporal analysis of the entire ethereum blockchain network, in: Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, Leila Zia (Eds.), WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April (2021) 19-23, ACM / IW3C2, 2021, pp. 2258–2269.
[45] Arijit Khan, Graph analysis of the ethereum blockchain data: A survey of datasets, methods, and future work, in: 2022 IEEE International Conference on Blockchain, Blockchain, IEEE, 2022, pp. 250–257.
[46] Ting Chen, Yuxiao Zhu, Zihao Li, Jiachi Chen, Xiaoqi Li, Xiapu Luo, Xiaodong Lin, Xiaosong Zhange, Understanding ethereum via graph analysis, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 1484–1492.

Andrea De Salve is a researcher at the Institute of Applied Sciences and Intelligent Systems (ISASI) of the National Research Council (CNR) in Italy. He received his Ph.D. in Computer Science from the University of Pisa and has extensive experience working in both industry and academia. He is involved in several collaborations with different research institutes, including IIT-CNR, UNIPI, and UNIPA. His research focuses on the security, privacy, and trust of systems based on centralized or decentralized architectures, and he is actively involved in EU-funded and Italian research projects related to different aspects of the previous research topics.

Alessandro Brighente is Assistant Professor (RTDA) at the University of Padova. He obtained in Ph.D. in Information Engineering from the University of Padova in 2021. He was visiting researcher at Nokia Bell Labs, Stuttgart, and University of Washington, Seattle, in 2019 and 2022, respectively. He served as TPC for several international conferences, including ESORICS, and WWW. He has been guest editor for IEEE Transactions in Industrial Informatics and for Elsevier's Computers and Security. He is part of several industrial and research projects. His current research interests include security and privacy in cyber–physical systems, the Internet of Things, and Blockchain.



Mauro Conti is Full Professor at the University of Padua, Italy. He obtained his Ph.D. from Sapienza University of Rome, Italy, in 2009. He has been awarded with a Marie Curie Fellowship (2012). His main research interest is in the area of Security and Privacy. He published more than 500 papers in topmost international peer-reviewed journals and conferences. He is EiC for IEEE TIFS, Area EiC for IEEE COMST, and has been AE for several journals. He was Program Chair and General Chair for several international conferences. He is Fellow of the IEEE and Senior Member of the ACM.