IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 12 July 2022, accepted 31 July 2022. Date of publication 00 xxxx 0000, date of current version 00 xxxx 0000.

Digital Object Identifier 10.1109/ACCESS.2022.3196391

A Framework for Verifiable and Auditable Collaborative Anomaly Detection

GABRIELE SANTIN[1], INNA SKARBOVSKY[2], FABIANA FOURNIER[2], AND BRUNO LEPRI[1]
[1]Digital Society Center, Bruno Kessler Foundation (FBK), 38123 Trento, Italy
[2]IBM Research, Haifa University Campus, Mount Carmel Haifa 3498825, Israel

Corresponding author: Gabriele Santin (gsantin@fbk.eu)

This work was supported in part by the H2020 INFINITECH Project under Agreement 856632.

ABSTRACT Collaborative and Federated Leaning are emerging approaches to manage cooperation between a group of agents for the solution of Machine Learning tasks, with the goal of improving each agent's performance without disclosing any data. In this paper we present a novel algorithmic architecture that tackle this problem in the particular case of Anomaly Detection (or classification of rare events), a setting where typical applications often comprise data with sensible information, but where the scarcity of anomalous examples encourages collaboration. We show how Random Forests can be used as a tool for the development of accurate classifiers with an effective insight-sharing mechanism that does not break the data integrity. Moreover, we explain how the new architecture can be readily integrated in a blockchain infrastructure to ensure the verifiable and auditable execution of the algorithm. Furthermore, we discuss how this work may set the basis for a more general approach for the design of collaborative ensemble-learning methods beyond the specific task and architecture discussed in this paper.

INDEX TERMS Algorithm auditing, anomaly detection, blockchain, collaborative learning.

I. INTRODUCTION

In a data-driven world, Machine Learning (ML) has progressively established itself as a fundamental tool that spreads across multiple fields and permeates an increasing variety of applications. After a decade of fast technological developments mainly driven by the exceptional new results achieved by Deep Learning [17], [28], a new wave of reflection is emerging about the scope, applicability, and technical limitations of these techniques. In particular, an increasing new attention is devoted to the issues of data ownership, data privacy, and data trading. In this setting, multiple related aspects are being analyzed and systematized within the frameworks of Federated Learning (FL) [23], [34], [53], [55] and Collaborative Learning (CL) [1], and several real-world problems have been approached with these techniques, e.g. in the banking [32], [54] and health [12], [44] sectors, even beyond classical domains [31], and considering privacy and fairness constraints [26], [45]. This new fields deals with the study of various scenarios where multiple agents own separate batches of data, and they are willing to cooperate for the construction of some ML models. This collaboration leverages different communication strategies to overcome the limitations of the single agents, which can be due to scarcity of data or scarcity of computational resources, but with the important constraint that data should never leave the location where it resides. This approach is in stark contrast with more traditional data-centralized methods, and it paves the way for a number of new algorithms that focus on various aspects of data ownership. For a comprehensive analysis of the key goals, applications, and challenges of FL we refer to the recent overviews [25], [52], [53]. To put our approach into context, we just recall that there is an important distinction between centralized and decentralized FL, and we recall in the following some important concepts discussed in [2], [25] and [14]. In the first case, a central orchestrator coordinates a set of distributed agents (or nodes) and their computational resources to improve the fitting of a central model. In the second case, instead, the entire process is collectively directed by the distributed agents. Heterogeneous cases are also of interest, where the central controller acts, or is queried, only when needed. In all cases, the focus of current research are the issues related to

The associate editor coordinating the review of this manuscript and approving it for publication was Amin Zehtabian.

communication efficiency, to the influence of the topology of the connections in the agents' network, and the quality of the learned model. Additionally, in the decentralized approach the absence of an omniscient orchestrator opens the way for new possibilities for privacy preservation and flexibility, but it poses new challenges for the security of the communications, the integrity of the system, and the accuracy of the algorithmic procedure.

In this paper we present a fully decentralized distributed and collaborative learning framework (see [25, Sec 2.1]). In particular, these kinds of solutions relax one of the core assumptions of FL systems, which is the presence of a central orchestration that maintains a global state of the model. Indeed, in a typical federated learning scenario a federation server is expected which collects local models and federates a generic model to be pushed back to all local agents, while in our approach the role of federation is only centered on initialisation and on enabling the running of the blockchain (BC). For this reason, we will refer to *Collaborative Learning* rather than *Federated Learning* in the following when discussing our method.

We focus on Anomaly Detection (AD) [3], [9] as a use case for CL. The scenario is motivated by AD systems that are common in the financial industry, such as fraud detectors or default predictors. The peculiar characteristic of these applications is that a classifier has to be trained to identify anomalous cases, i.e., events that are unusual compared to the most frequent patterns observed in the data. In particular, anomalous examples are scarce by definition. As a consequence, different agents such as banks, financial institutions, insurance companies may foresee a benefit in collaborating with their peers in order to trade knowledge and improve their individual models. On the other hand, the data that is used to train these systems is usually shared with caution, since it typically comprises sensitive personal information regarding the financial position or the individual characteristics of the clients. Moreover, the possession of these data is often an important asset for the single agents, which are possibly not willing to give them away once for all, but would rather like to develop an on-purpose sharing. This option is inherently difficult with easily copyable digital data.

With these constraints and goals in mind, we present in this paper a fully decentralized CL system where multiple agents collaborate for the training of one model per agent, and which is privacy preserving by design, robust to changes in the network topology and to asynchronous communications, and resistant to malicious intrusions and adversarial attacks, in terms that will be discussed in more details in Section IV-C.

The system is designed so that each agent trains an ensemble classifier [40], i.e., a ML model that is made of multiple simple estimators that are combined as atomic building blocks. This structure makes it easy to iteratively improve local models as well as exchanging knowledge between agents by sharing the top performing blocks. We use in particular Random Forests (RFs) [7] as ensemble models, as they are well-suited for anomaly detection problems and robust to missing data, but we comment along the paper how this is not a restrictive choice and other ensembles could be adopted. Moreover, the chosen design of the ML algorithm permits to integrate the system in a BC infrastructure that guarantees trustable and verifiable execution of the algorithm, and certifies the communication between the nodes.

Other works have proposed solutions for the integration of FL and CL in a BC environment [29], [30], [33], [38], [47], [48], [51]. In this work, we introduce two main novelties over existing approaches: (i) The framework supports collaborations where the agents are connected by means of a time-varying network in a fully decentralized scenario. This includes the case of single agents joining or leaving the group at different times, or exploiting the collaboration in an on-demand fashion. This permits to treat the participation in the collaboration as a tradable utility (see Section III-B and Section IV-C), and leverages the BC as a verification tool; (ii) the solution is algorithm-agnostic in its main components, meaning that it can be applied on top of a large class of ML models, provided that some atomic operations can be defined (see Section III-A). In particular, the algorithm is not bound to specific architectures or optimization methods.

The paper is organized as follows. We start by recalling the necessary background on RF and BC in Section II, and with these tools we introduce the novel algorithm in Section III, discuss the full BC solution in Section IV, and comment on the overall computational cost in Section V. We validate our new system through a number of experiments in Section VI, and conclude by discussing some perspectives and open problems in Section VII.

II. BACKGROUND

We start by recalling some background details in order to facilitate the reading of the paper by researchers from both the ML and BC communities.

A. SETTING OF THE ML ALGORITHM

In the following we assume that each agent has a labeled dataset of examples, where each data point (e.g., a transaction) is represented by a d-dimensional vector $x := (x^1, \dots, x^d) \in \mathbb{R}^d$, collecting d features x_i (e.g., the ID of the user performing the transaction, its timestamp, the amount transferred, etc.). Each example is associated to a label $y_i \in \{0, 1\}$ indicating whether the i-th example is normal ($y_i = 0$) or anomalous ($y_i = 1$). These examples are collected in a dataset $(\mathcal{X}, \mathcal{Y})$ of $m \in \mathbb{N}$ data points $\mathcal{X} := \{x_1, \dots, x_m\}$ with labels $\mathcal{Y} := \{y_1, \dots, y_m\}$. In this paper we work with tabular data, but this is not required in general and other data types may be supported, such as images or texts.

For the detection of anomalous examples each agent trains its own classifier, i.e., a map $\Phi : \mathcal{X} \to [0, 1]$ that is optimized on the training set, and that can be used to approximately predict the class of an unseen data point x, with the usual convention that the example is classified as normal if $\Phi(x) \leq 0.5$ and as anomalous if $\Phi(x) > 0.5$. We consider ensemble classifiers, which means that we actually train a set of $n \in \mathbb{N}$

simpler classifiers (or estimators) $\phi^i : \mathcal{X} \to [0,1]$, $1 \leq i \leq n$, each trained on the same classification task, and define the global prediction of Φ either by averaging, i.e., $\Phi(x) := \frac{1}{n}\sum_{i=1}^{n} \phi^i(x)$, or by majority voting among the n predictions $\{\phi^i(x)\}_{i=1}^n$. To explicitly denote the transformation from the estimators to the ensemble and vice-versa, we use the notation $\Phi := \text{Ens}\left(\{\phi^i\}_{i=1}^n\right)$ and $\{\phi^i\}_{i=1}^n := \text{Estim}(\Phi)$. This kind of classifiers will be instrumental for our construction, since they are quite straightforward to improve by enlarging the ensemble size n and adding new simple learners, and it is possible to mix different classifiers Φ and Φ' by mixing their simple learners.

As a prototype of ensemble classifiers, in this paper we focus on RFs [22], which use decision trees as their simple learners. Decision trees [8] are maps $\phi^i : \mathcal{X} \to [0,1]$ that compute their prediction according to a binary tree: Once the tree is trained on the data, at prediction time an input enters the tree from its root, and it follows a sequence of binary tests until it reaches a leaf node. Each of these leaf nodes is associated to a unique label, which is the prediction assigned by the tree to each input that falls into this leaf. At each non-leaf node, instead, the splitting is decided by the value of a single feature of the input, and thus a decision tree can be understood as a sequence of binary splits of the input space according to a subset of features at given splitting values. The training of this structure requires to select the sequence of features and the threshold values to define the splitting, and this is usually realized by guaranteeing that the examples in the training set are distributed in a balanced manner among the leaf nodes, and adopting criteria for the growth of the tree in depth and width. We refer to [7] for a detailed treatment of this topic.

In addition to their basic ensemble structure, RFs perform two randomization operations to improve their accuracy and robustness. Namely, RFs are trained by bootstrap aggregation, i.e., each tree in the ensemble is trained on a random subset of the full dataset, extracted by a sampling with replacement. Furthermore, the single trees are trained with feature bagging, i.e., each splitting of each tree is constructed by considering only a uniformly randomly selected subset of the features of the data.

RFs are particularly suited for tabular data and they can deal quite effectively with missing entries thanks to their structure that do not require the knowledge of each single feature. Moreover, their training is quite simple and thus suitable to be performed repeatedly, as will be the case in our algorithm.

B. SETTING OF THE BC SOLUTION

A BC is essentially a digital ledger of transactions that is duplicated and distributed across the participants in the BC network. Transactions are recorded in a final and immutable manner by the BC, providing all network members with an identical and trustworthy real-time view of the state. Due to its inherent characteristics, BC is the natural platform to support privacy and trust as well as a secure execution environment [10], [18], [35]. Our proposed BC solution ensures a secure, auditable, and verifiable framework for execution of collaborative and federated learning algorithms.

The idea is that each learning node in the BC network publishes intermediate results at the end of each iteration. These results can be consumed by other learning nodes to improve the accuracy of their next computations. Our solution is generic and can support any ML algorithm having the following properties: The algorithm can be represented as a portable computation workload (e.g., a docker image which can be instantiated to a container running the algorithm's computation); the algorithm can be iterative or single-step; and it can either be centralized and require orchestration and synchronization between iterations or be distributed and thus self-orchestrating.

For our proposed framework, as underlying BC technology we leverage Hyperledger Fabric (or simply Fabric) [4], [24], which is one of the most promising BC platforms for enterprises (see e.g., [19] for a comprehensive and foundational analysis of the BC solutions and services for enterprises).

III. COLLABORATIVE TRAINING OF ENSEMBLE CLASSIFIERS

With these tools in hand we now introduce the collaborative learning algorithm. We first formulate the algorithm under as general assumptions as possible, and then we provide some specifications in the case of RFs. We will anyhow comment on how these can be generalized to different scenarios.

A. AGENTS AND ATOMIC OPERATIONS

We assume to have a number $N \in \mathbb{N}$ of agents (or nodes) participating in the collaboration, and denote them as $V := \{v_1, \ldots, v_N\}$. Each node v_j has an own dataset $(\mathcal{X}_j, \mathcal{Y}_j)$ of size m_j of the form described in Section II-A, and its goal is to obtain an ensemble classifier Φ_j for the detection of anomalies, working possibly beyond its own data.

We consider three atomic operations to modify an ensemble: one enlarges the ensemble, one keeps its size bounded, and one selects the top performing estimators. Assuming that Φ is an existing ensemble with n estimators, $\{\phi^i\}_{i=1}^{n'}$ is another set of estimators, and $k \in \mathbb{N}$ is an integer parameter, the three operations are formally defined as follows:

- `ADD`$(\Phi, \{\phi^i\}_{i=1}^{n'})$ returns the enlarged ensemble $\Phi' := \text{Ens}\left(\text{Estim}(\Phi) \cup \{\phi^i\}_{i=1}^{n'}\right)$.
- `GET_TOP`(Φ, k) sorts the n estimators of Φ according to some order that needs to be specified, and returns the top k. If $n \leq k$ all the n estimators are returned.
- `CROP`(Φ, k) keeps only the k best estimators of an ensemble Φ, i.e., it sets $\Phi = \text{Ens}(\texttt{GET\_TOP}(\Phi, k))$.

B. COLLABORATIVE LEARNING

The group of agents is partially connected according to a network represented by an undirected graph $G = (V, E)$, where there is an edge $(v_i, v_j) \in E$ if and only if a connection is active between the i-th and j-th node.

The assumption that the connection graph G is fixed is only made for simplicity of exposition, but it is straightforward to deal with time-varying graphs that may represent e.g., agents entering and leaving the collaboration, or temporary failures in the connection system, and in fact an example of a time-varying graph will be tested in Section VI. Indeed, for the algorithm to run it is sufficient to assume that each node v_j, whenever it is interested in a communication, is able to get the list of its first order neighbors, i.e., the set of all agents v_i such that there is a link $(v_j, v_i) \in E$. Moreover, each node in practice has no need to know the entire graph, and has no option to modify it. More advanced scenarios could be envisioned and investigated, for example by assigning to the agents a certain budget that can be used to establish optimized connections to certain nodes, or by using the knowledge of the entire connection graph to take some decisions on the learning mechanism. We leave these extensions for future work.

To manage the communication, each node v_j has a registry R_j with a slot $R_j(v_i)$ for each of the other nodes v_i. We assume that each node v_i can write a message to the slot $R_j(v_i)$ in the registry of the node v_j if this is one of its first order neighbors.

Using the registry and the atomic operations on the ensemble, we are in the position to define the three fundamental operations that each agent v_j can perform to change its status at each iteration. They are controlled by three parameters $n_{\text{new}}, n_{\max}, n_{\text{share}} \in \mathbb{N}$ that we assume to be globally set, even if local parameters (i.e., node-dependent) may be used without significant modifications. The three operations are the following:

1) `FIT`: A number $n_{\text{new}} \in \mathbb{N}$ of simple estimators $\{\phi^i\}_{i=1}^{n_{\text{new}}}$ are trained by the agent on its own dataset $(\mathcal{X}_j, \mathcal{Y}_j)$, and the ensemble Φ_j is enlarged as $\Phi_j := \texttt{ADD}(\Phi_j, \{\phi^i\}_{i=1}^{n_{\text{new}}})$. If the resulting number of estimators is larger than $n_{\max}$, then the method $\texttt{CROP}(\Phi_j, n_{\max})$ is used to keep only the best ones.
2) `SHARE`: The agent identifies its top n_{share} estimators with the `GET_TOP` method, and writes them to the registry of each of its first order neighbors. If a registry slot contains already some estimators from previous communications, they are overwritten.
3) `GET`: The agent reads its registry slot to collect all the estimators received in the previous iterations (if any), and adds them to its current ensemble by using the `ADD` method. If this operation makes the ensemble larger than $n_{\max}$, excess estimators are removed by a call to the `CROP` method.

Finally, the algorithm requires initialization and termination conditions. For simplicity we assume that each agent v_j starts with an empty ensemble $\Phi_j := \text{Ens}(\emptyset)$ and runs `FIT` as its first operation. Moreover, each agent terminates its execution when the prescribed iterations are executed.

C. PROPERTIES OF THE ALGORITHM

The entire algorithm is completely decentralized, since it only requires the existence of a communication network and the agreement on a set of initial parameters. The model supports time-varying networks, and it allows for completely asynchronous communication, including the option for different nodes to join or leave the collaboration at different times.

Observe that all the operations except for `GET_TOP` are well defined for any type of ensemble classifier, and do not require further specification to be implementable. The only method-specific operation is thus `GET_TOP`, that requires to define a way to rank the estimators within an ensemble. We discuss our solution in the case of RFs in the next section, but we remark that this choice is not unique, and that similar design principles could be adopted to work with more general ensembles. In this sense, the present algorithm may be understood as a family of algorithms, parametrized by the method that is used to promote some estimators with respect to other ones.

The importance of this ranking system is reflected in the fact that we are employing a registry with slots that stores only the last written information. In this way, when a node reads its registry via the `GET` method, it only reads the result of the most recent call of `GET_TOP` transmitted by its neighbors.

This solution is used also to guarantee that the registry has bounded memory footprint, since in this way it needs to store at most $n_{\text{share}} \cdot (N-1)$ estimators at each time. Similarly, the bound $n_{\max}$ on the number of estimators held by each single node controls the size of each ensemble classifier. These two requirements can be translated to memory bounds if we assume that each estimator has a maximal memory size.

Moreover, the only operation that can create new estimators is `FIT`. Whenever this method is called, the newly constructed estimators are labeled with identifiers (v_j, i), where v_j is the identifier of the creator node, and i is a progressive counter maintained by v_j. In this way each estimator in the collaboration is uniquely identified, and it is always possible to know which nodes trained it. Moreover, communication between different nodes amounts only at the exchange of estimators via the `SHARE` and `GET` methods. Both the operations of creation and sharing are thus easily secured by means of the BC integration that we are discussing in detail in Section IV, so that the collaboration is protected against anomalous agents and malicious injections of information.

D. RANKING OF THE ESTIMATORS FOR RF

To obtain a fully functioning algorithm, it remains to specify the mechanism used to rank the estimators within each ensemble, i.e., to define the `GET_TOP` operation. We define it for RFs, which are the method of choice of this paper.

As discussed before, the sorting of the estimators is the most delicate operation and the one that have the largest potential to affect the result of the algorithm. In general terms, we aim at using unsupervised methods for this task, namely, we do not use the labels of the data to sort the estimators. The reason for this choice is that any supervised operation must rely on the data available to each node, and using the same local data that are used for training to rank the estimators is

very likely to lead to a downplay of the importance of the estimators received from the other nodes. For this reason, we decided to analyze only methods that rely on the structure of the estimators.

Although different RF pruning schemes have been introduced [16], [27], [36], we use here a mechanism that allows us to obtain a full sorting of the set of trees, and not only a reduction of its number. To this end, we recall that each estimator is a decision tree, and thus it can be represented by a tree where each non terminal node v is associated with the index $s(v) \in \{1, \ldots, d\}$ of the splitting feature, and the corresponding splitting value $x(v) \in \mathbb{R}$ (see section II-A). We use the splitting index $s(v)$ to identify the type of a node, and we regard $x(v)$ as node feature, so that each decision tree can be identified as $D := (T, X)$, where T is a tree with labeled nodes, and X is a vector of node features associated to the non-terminal nodes.

Given a pair of decision trees $D := (T, X)$, $D' := (T', X')$, we define a similarity measure that is used to compute the estimators' ranking in a structure-dependent way, i.e., one that takes into account the definition of each single estimator. To this end we define a positive definite and symmetric kernel $k(D, D')$ over pairs of decision trees. The kernel is a modification of the tree kernel of [21], and we provide its explicit construction in Section VII. We refer to [43], [49] for a detailed treatment of the topic of kernel methods, and we recall here that k can be used to encode general data (decision trees in this case) in a possibly high dimensional Hilbert space where standard numerical techniques are available. Moreover, the same method can be extended to other ensembles as soon as a kernel can be defined on its building blocks, and thus the present method has the potential to be applied in more general settings.

In particular, it is possible to define a Gaussian Process [39] with covariance function k over the space of decision trees. Given the process, one may select a subset of the set of trees so that, conditioning the process on the labels associated to these trees, the maximal standard deviation of the posterior process is minimized. In this sense, this subset of trees may be regarded as the one that controls the maximal variation in the ensemble. This problem may be efficiently approximated by a greedy algorithm [13] that selects this set in an iterative way, and this gives the ordering of the estimators that we are looking after. It can be shown that this process is quasi-optimal [41], [50], meaning that the greedy selection is as effective as a global optimization, up to a constant. Running this algorithm until it selects k elements, we obtain an ordered sequence $D_1, \ldots, D_k$ representing the n most important estimators, thus implementing the `GET_TOP` operation.

IV. A BC FRAMEWORK FOR SECURE AND TRUSTWORTHY CL

We describe now in detail our proposed framework, after recalling the necessary background. We refer to [24] and [6] for more details on Hyperledger Fabric.

A. BLOCKCHAIN BACKGROUND

Blockchain (BC) is a peer-to-peer network and distributed ledger technology that allows any participant in a business network to see the system of record (ledger). At the heart of any BC network is a distributed decentralized ledger, replicated across many network participants, that records all the transactions that take place on the network. A transaction is essentially an asset transfer onto or off the ledger. In addition to being decentralized and collaborative, the ledger is append-only, using cryptographic techniques that guarantee that once a transaction has been added to the ledger it cannot be modified. This property of ''immutability'' makes it simple to determine the provenance of information, allowing network participants to be sure information has not been changed after the fact. Each peer on the network (a network participant) keeps a copy of the transaction ledger and world state database, which reflects the current state of all the assets in the network. The process of keeping the ledger transactions synchronized across the network is called consensus. A BC network uses smart contracts to support the consistent update and controlled access of information, and to enable ledger functions such as transacting and querying. The main goal of smart contracts is to automatically execute the terms of an agreement once certain conditions are met. For each transaction, the flow of value and transaction state must be defined [15].

Hyperledger Fabric (or simple Fabric) is a collaborative effort created to advance cross-industry BC technologies for business [6], [24]. It provides an open source, industrial-grade implementation of a private or permissioned BC under the Linux Foundation umbrella. Fabric provides a modular architecture with a delineation of roles between the nodes in the BC network, execution of smart contracts, and configurable consensus and membership services. Chaincodes are the mechanisms through which smart contracts are defined in the Fabric BC implementation. At a high level, the system is comprised of (i) peer servers, potentially belonging to different organizations, which replicate and validate the blocks creating the transactions comprising the ledger; (ii) an ordering service which determines the total order of the transactions and publishes the corresponding blocks to be picked up by the peer processes; and (iii) a client that interacts with the system programmatically for invoking transactions or queries. A configurable sub-set of the peers is involved also in endorsing transactions submitted to the system, supporting consensus for inserted transactions. All entities hold verifiable security certificates issued by a Certification Authority (CA) component. Transactions among members in a consortium is performed in the context of a channel. A channel creates a separate ledger visible only to the organizations included in the channel. Once the chaincode is in place, users can start invoking transactions and queries on the BC channel. Fabric provides several mechanisms allowing for data sovereignty and permissioned access to data, including the privacy mechanisms inherent in the network itself (permissioned access only), the concept of channels, encryption of data, and user

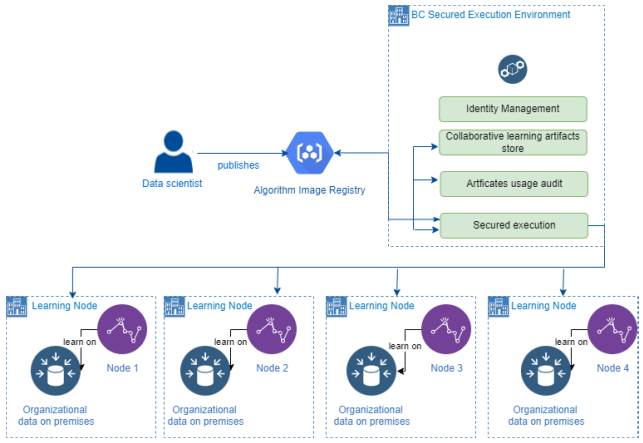

FIGURE 1. Verifiable and auditable collaborative machine learning framework.

roles allowing partial access to data to different network participants.

B. STRUCTURE OF THE BC SOLUTION

The framework consists of different conceptual elements (see Figure 1): (i) a data scientist, responsible for creating and pushing the CL algorithm image to the algorithm image registry after the training phase of the algorithm is over; (ii) the algorithm image registry, which is any kind of local or hosted image registry for storage of docker images representing the ML algorithms; and (iii) the learning nodes, which are the organizational nodes in the BC network participating in the CL process. Here, production data is stored in premises and only intermediate and final results of the algorithm execution are stored in the BC ledger.

Additionally, the system comprises a BC execution environment, i.e., a secure execution environment that provides a verifiable privacy-preserving computation environment for CL scenarios. The environment comprises the following modules:

- Identity Management: A built-in service in Hyperledger Fabric that provides a membership identity that manages user IDs and authenticates all participants in the network including (i) the specification of Certification Authority (CA) servers (defined as part of the BC network configuration); (ii) the certification of users and applications using these CA servers; and (iii) mechanisms to sign and validate the signatures of all transactions and messages submitted to the network.
- Collaborative Learning Artifacts Store: The chaincodes implementing the business logic for storing, updating, retrieving, and querying business artifacts related to CL, i.e., algorithm images' metadata, metadata of the learning process, and intermediate results and models.
- Artifacts Usage Audit: The inherent functionalities in chaincodes which allow to query the history of updates for each artifact stored in the ledger, thus allowing to present a clear and complete picture of the artifact's provenance.
- Secure Execution: This module securely runs the computation tasks of the ML algorithm (we refer to computation task or workload as the ML algorithm instance or iteration), producing signed outputs (i.e., the insights from the learning round), and storing these outputs in the ledger. In the case of CL, it helps to establish the auditability and verifiability of the execution of local ML models and to improve the trust among the participants. Moreover, in the case of updates to the learning algorithm, it is guaranteed that all the parties are aware of the correct image version and are enforced to use the correct one to participate in the learning process.

C. VERIFIABILITY OF THE EXECUTION

Our proposed approach allows delegating the computation over sensitive data to the data owner, while establishing trust of the rest of the stakeholders in the computation result. This is achieved via implementation of the following core characteristics:

- The computation workload is portable so that it is possible to deploy it in the data owner's environment.
- The integrity of the computation workload is verifiable, i.e., computation stakeholders have guarantees that the actual computation was performed on the respective data.
- The provenance over the input data, the output of the computation, and the computation logic is tracked.

We implement the portability characteristic by packaging the computation logic in a portable artifact. A docker image is an example of such a portable artifact, which is suitable for relatively simple computations that allow incorporating the entire logic into a single image. In cases where the computation involves multiple steps and components, it can be packaged as a composite asset, consisting of a set of images (each incorporating a relevant phase or function in the computation) and an artifact (or a set of artifacts) that define the orchestration and the choreography of the composite computation.

To establish correctness and integrity guarantees over the computation logic, we propose to manage computation workloads metadata in BC. Having the metadata record in a shared distributed ledger ensures that all the parties have joint understanding of how to verify that a given portable deployable artifact is of the correct version and its contents have not been tampered with. For the algorithm images, we store a SHA256 hash of the docker image on the ledger. At the time of computation task creation from an image, when pulling the image from the algorithm image registry, we can verify image authenticity by calculating and comparing the image's hash to the one stored in the ledger. For the computation task results we use public/private key verification. When creating a computation task, we use a crypto library to generate private/public key pairs. A public key of the pair is stored in the ledger in the execution task record, while the private key

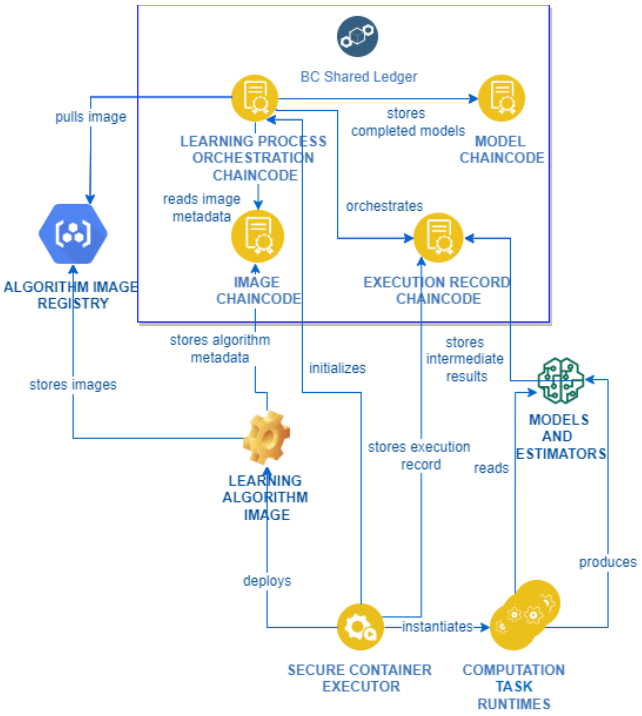

FIGURE 2. BC solution building blocks and flows.

is passed to the computation task runtime. Once it finalizes, the computation task updates its record in the chain with the results of the execution signed with the private key. The updating chaincode then verifies the signed result element with the public key of the computation task to ensure that the results are being updated by the entity with the correct private key.

Trackability and provenance is gained by providing auditing and verifiability capabilities for managing the ML algorithm image lifecycle (e.g., publishing a new algorithm image and usage of the algorithm image), for secure execution (ensuring, for example, that the correct algorithm image is used in each execution of computation task), and for recording of intermediate results (allowing to answer questions, such as which artifacts were published at the end of each run for a particular learning process, or which artifacts a particular organization published for a particular learning task).

Figure 2 depicts the interactions between the chaincodes in our BC network and the other building blocks comprising our secure execution environment. The secure execution environment components are indicated as round yellow circles in the diagram. They consist of the chaincodes, which are part of the BC runtime and the external (to BC) secure execution environment building blocks written in Python. The latter include the Secure Container Executor script, which allows the user to deploy an algorithm image, initialize the learning process and instantiate the CL computational tasks on learning nodes; and the Computational Task Runtime script which executes the learning algorithm image. Additional artifacts appearing in the diagram include the Algorithm Image Registry, where the algorithm images are stored; the Learning Algorithm Image, which is stored in the Algorithm Image Registry and is used in the learning process; and the intermediate and complete models (designated in the picture as Models for completed models and Estimators for intermediate results) which are the outcome of the learning tasks.

The following flow describes the interactions among the different building blocks and relate to the cycle of creating a ML algorithm image, executing this image securely on learning nodes, and sharing the insights.

First of all, the image for the ML algorithm, intended to be run as a particular instance of a CL process, is stored in the algorithm image registry (which can be either a shared or a private image repository). The metadata describing the ML algorithm image, specifically an identifier of the respective artifact in the external repository and a cryptographic fingerprint (i.e., a hash value) that can be used to verify the integrity of the artifact, are stored in the BC ledger using the image chaincode.

During the instantiation of the execution phase, learning process metadata is created using the learning process orchestration chaincode. This metadata includes the unique ID for the learning process, the algorithm image this process is intended to execute, the consortium of organizational nodes participating in the CL process, indicators of the current state of the learning process (e.g., current iteration in case of iterative learning process), and the current execution status. After the learning process metadata record is created on the chain, the ML algorithm image is pulled from the algorithm image registry and instantiated as computation task runtime on each learning node by the Secure Container Executor. The task is instantiated with the ML algorithm runtime, the parameters for the run, and the initial state model.

During the algorithm execution phase, the learning node reads the relevant insights from previous rounds by the other learning nodes, runs the algorithm image on the relevant inputs (the insights from previous rounds, the input parameters, and the organizational datasets), and publishes the resulting insights or completed model to the ledger using the execution record and the model chaincodes. Once the learning process is completed, the status of the learning task on chain is updated to completed.

As shown in Figure 2, our BC solution comprises four chaincodes: the image and execution record chaincodes which form the secure execution module (Figure 1), and the learning process orchestration and model chaincodes which form the CL artifacts store module (Figure 1). The image chaincode provides the functionality for storing and retrieving the ML algorithm image metadata. This chaincode also provides queries helpful in determining the provenance of the image, e.g., who is the creating organization or when the image was created. The learning process orchestration chaincode records the information about the CL task, including the definition of the algorithm image the learning task is about to execute; the consortium of organizations participating in

the learning process and the nodes which will run the computation tasks; and the current status of the learning process (current iteration, completion status). The execution record chaincode stores the execution task metadata in the ledger. Once the outcome of a single-step computation task, or of the particular learning round task (for iterative learning algorithms) is completed at a node, it publishes the insights to the chain (the estimators in our case of a fraud detection algorithm) updating the execution record. It also updates the model chaincode in case the learning process is completed. The model chaincode is responsible for publishing the completed model to the ledger once the learning task finishes. The complete model record contains a list of organizations allowed to access these models which initially equals the consortium members.

As stressed before, one of the built-in core properties of BC platforms is an immutable chain of blocks of transactions, establishing verifiable and transparent history of updates for each artifact stored in the chain. This is of fundamental importance when striving for trust and transparency of the execution of CL scenarios. Proven, verifiable, and immutable audit trail of execution tasks producing CL models can help establish without doubt, for example, that the models are derived from the desired ML algorithm, the specific version of the algorithm, and the executing organizations. To this end, the artifacts usage audit logical module in Figure 1 supports provenance for algorithm images, computation tasks executed, and model metadata.

We would further remark that the proposed BC solution is resistant to malicious intrusion attempts. Indeed, the BC network is composed only from the nodes participating in the CL and belonging to a blockchain organization, meaning all CL nodes have a digital identity encapsulated in a verifiable X.509 digital certificate issued by Hyperledger Fabric Certificate Authority (CA). In other words, a node that is not part of the learning process and has no CA-issued X509 verifiable certificate, cannot even read or write to the BC. Additionally, although we do not explore this option in our work, it could be possible to encrypt the results of the algorithm execution and write on the BC only the encoded result, so that the information will be available only to a node with a suitable key.

V. COMPUTATIONAL COST OF THE CL ALGORITHM

When compared with a single-node execution of the ML algorithm, the participation in the collaboration requires an additional computational effort to the nodes. Even if this participation is clearly beneficial, as we quantitatively demonstrate in Section VI, it is important to assess the required computational overhead.

The CL algorithm itself (Section III) includes the `FIT` operation, which would be executed also in an isolated-nodes setting, and it introduces the `SHARE` and `GET` operations. Both require the execution of a `GET_TOP` atomic operation, that in the current implementation defined in Section III-D has a cost $\mathcal{O}(n_0 k^2)$, where n_0 is the number of parsed trees and k is the number of selected trees [42]. When executing the `SHARE` operation we have $n_0 \leq n_{\max}$ and $k \leq n_{\text{share}}$, and thus the cost is controlled by the algorithm parameters. When executing a `GET`, instead, we have $k \leq n_{\max}$ and $n_0 \leq \deg \cdot n_{\text{share}} + n_{\max}$, where deg is the cumulative number of neighbors of the node at the previous times, i.e., those who wrote a message in the node's registry, and in particular $\deg \leq N-1$. This second operation has thus a cost which is strongly dependent on the network's connection pattern, where more sparse networks provide a faster execution. Similarly, the exchange of information required by both the `SHARE` and `GET` operations may be assumed to scale linearly in the number of connections.

The BC based secure execution environment provides security, verifiable execution, and provenance of results. Naturally, it comes with a performance cost (communication and computation) as the results are stored in the shared ledger after undergoing a round of endorsements to achieve consensus between all BC nodes. The solution is intended for algorithms for which security, transparency, and trust are required, but where the number of transactions and iterations are relatively small and latency is not a key factor, such as the case with our fraud detection CL algorithm.

VI. EXPERIMENTS

The implementation of the algorithm and the code to replicate the experiments presented in this section are available on GitHub.[1]

A. EXPERIMENT SETUP

We test the algorithm on a benchmark dataset for fraud detection.[2] This dataset collects electronic credit card transactions that have been executed in some European banks during September 2013. Each transaction is represented by 28 numerical features which are obtained after applying a Principal Component Analysis (PCA) on the original features, in order to hide any sensitive information, and it is labeled either as normal or as a fraud. The dataset contains 284807 transactions, of which 492 (the 0.17%) are frauds, making the dataset highly unbalanced.

We simulate a scenario with $N := 20$ agents, each holding its own private data. To create a suitable setup, we split the given dataset into N disjoint subsets by random sampling. To make the problem more challenging and interesting for the testing of a collaborative scenario, we perform an unbalanced sampling: instead of splitting the positive and negative examples into N groups of $284807/N$ agents, we allow for each group to contain up to 70% more or less elements than the average. Additionally, each of the resulting datasets is split into a train dataset (90% of the samples) and test dataset. The actual number of samples for each node and the corresponding statistics are reported in Table 1. To simplify the measurement of the performances of the algorithm,

[1] https://github.com/GabrieleSantin/federated_fraud_detection
[2] https://www.kaggle.com/mlg-ulb/creditcardfraud

we artificially create a unique and centralized test set obtained by joining the N test sets of the single nodes, so that all the test metrics are computed on the same test set. This breaks the absence of centralized orchestration in the design of the algorithm, but it is only a convenience choice made for the purpose of exposition.

TABLE 1. Size of the datasets for the 20-nodes simulation, and corresponding numbers and ratio of frauds.

ID	Type	Samples	Frauds	Fraud ratio
Node00	Train	14733	16	0.0011
	Test	28489	57	0.0020
Node01	Train	9570	37	0.0039
	Test	28489	57	0.0020
Node02	Train	12992	0	0.0000
	Test	28489	57	0.0020
Node03	Train	15544	49	0.0032
	Test	28489	57	0.0020
Node04	Train	13064	21	0.0016
	Test	28489	57	0.0020
Node05	Train	16036	13	0.0008
	Test	28489	57	0.0020
Node06	Train	11149	8	0.0007
	Test	28489	57	0.0020
Node07	Train	17571	32	0.0018
	Test	28489	57	0.0020
Node08	Train	3297	11	0.0033
	Test	28489	57	0.0020
Node09	Train	10820	34	0.0031
	Test	28489	57	0.0020
Node10	Train	18365	17	0.0009
	Test	28489	57	0.0020
Node11	Train	7875	33	0.0042
	Test	28489	57	0.0020
Node12	Train	8298	9	0.0011
	Test	28489	57	0.0020
Node13	Train	28249	28	0.0010
	Test	28489	57	0.0020
Node14	Train	11155	32	0.0029
	Test	28489	57	0.0020
Node15	Train	3363	12	0.0036
	Test	28489	57	0.0020
Node16	Train	7894	25	0.0032
	Test	28489	57	0.0020
Node17	Train	13798	9	0.0007
	Test	28489	57	0.0020
Node18	Train	14098	41	0.0029
	Test	28489	57	0.0020
Node19	Train	18450	11	0.0006
	Test	28489	57	0.0020

To analyze the effect of different configurations of the collaboration, we analyze four different connection scenarios (see Figure 3): (i) a fully disconnected setting, (ii) a pairwise connected setting (i.e., each node is connected to exactly two nodes), (iii) a random and time dependent setting, and (iv) a fully connected setting. In more details, the setting (iii) is a time-dependent network, where at each iteration a link per node is drawn uniformly at random among the 19 possible ones. Observe in particular that this network has, at each iteration, the same number of connections as the setting (ii). The static network resulting from time aggregation (Figure 3c) has an average degree 7, with minimal degree 6 and maximal degree 8. The disconnected case serves as a baseline, since it represents the case where no collaboration takes place and each node can only rely on its own dataset. Moreover, we consider the fully centralized scenario where a single agent has access to the entire dataset that is obtained by merging the 20 datasets. This configuration is not representative of the setting considered in this paper, where we assume that the data ownership should not be broken, but it offers a possibility to investigate the maximal payoff that the agents would obtain in trading their data security for a larger accuracy.

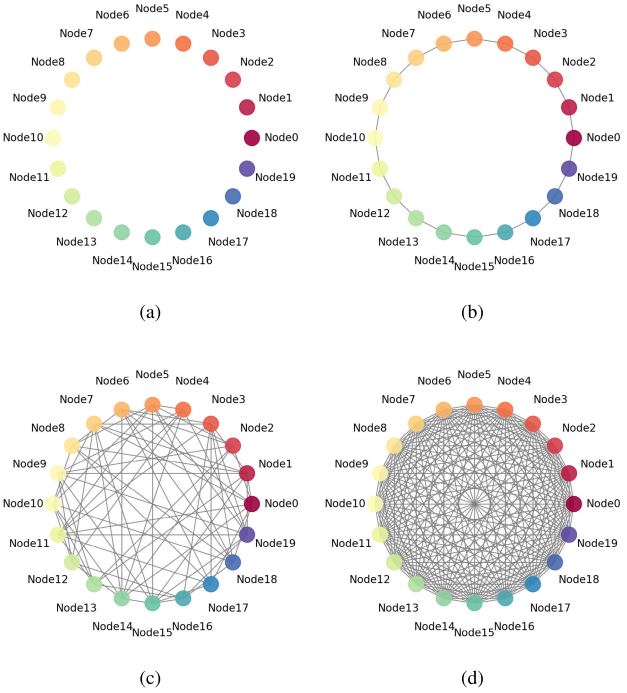

FIGURE 3. Connection configurations tested in the experiments: disconnected (Figure 3a), pairwise connected (Figure 3b), random time-varying (Figure 3c, which shows the aggregation of the networks over time), fully connected (Figure 3d).

For each configuration, we train the CL algorithm by letting each node executes the same sequence of operations (see Section III-B). Namely, in the base case of the disconnected topology we run four repetitions of `FIT`, i.e., each node creates its own model and refines it three times. In the connected cases, instead, we add a `SHARE` and a `GET` operation after each fit. In this way, after each training on the local dataset each node shares its insights to its neighbors, and subsequently reads and incorporates the knowledge received by the neighbors themselves. The algorithm is run with values $n_{\text{new}} := 10$, $n_{\text{share}} := 10$, $n_{\max} := 50$ for the parameters defined in Section III-B.

To measure the efficacy of the models we use three metrics, namely the balanced accuracy BAcc, the precision Prec, and the recall Rec. Given the true test labels and the predicted test labels, we may count the number of false positive FP, true positive TP, false negative FN, false positive FP. With these

TABLE 2. Minimal and maximal improvement with respect to the disconnected case for the three collaborative scenarios (Fully connected, Pairwise, Random), as measured by the three test metrics.

	BAcc			
	min		max	
Fully connected	Node18	-1.75e-02	Node2	3.95e-01
Pairwise	Node7	3.52e-05	Node2	3.95e-01
Random	Node18	-1.75e-02	Node2	3.33e-01
	Prec			
	min		max	
Fully connected	Node10	-5.22e-02	Node2	9.18e-01
Pairwise	Node10	-7.93e-02	Node2	9.38e-01
Random	Node6	-3.66e-02	Node2	9.74e-01
	Rec			
	min		max	
Fully connected	Node9	-3.51e-02	Node2	7.89e-01
Pairwise	Node3	0	Node2	7.89e-01
Random	Node 9	-3.51e-02	Node 2	6.67e-01

numbers, the three metrics are defined as

$$\text{Prec} := \frac{TP}{TP+FP}, \quad \text{Rec} := \frac{TP}{TP+FN},$$
$$\text{BAcc} := \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right).$$

It should be noted that all the metrics have value in [0, 1].

B. RESULTS AND DISCUSSION

We use these metrics to assess the improvement of the collaborative models over the scenario where each node is isolated.

To this end, for each node we compute on the test set the metrics in the three collaborative cases (pairwise connected, fully connected, and random) and their difference with the corresponding value in the disconnected case. We report in Table 2 the nodes for which these differences are maximal and minimal, and the corresponding values. It should be noted that for some nodes there is indeed a negative improvement, which means that the participation in the collaboration has a negative effect, but the corresponding values are of order at most 10^{-2}. This is expected since the algorithm has a randomization component, and a change of this order of magnitude may be considered as a reasonable fluctuation. On the other hand, the maximal improvement is of order 10^{-1}. In all connection scenarios and for all metrics, the node of maximal improvement is Node2: looking at Table 1, it appears that this node has no frauds in the training set, and it is thus not capable of learning any meaningful classifier when isolated. On the other hand, participating in the collaboration it receives insights from its neighbors, and it is able to improve its model in a very significant way, up to an improvement of 0.9 for the Prec metric.

Apart from these extreme values, we compute the mean and median of these differences over the 20 nodes. These values are reported in Figure 4b, and the absolute value used to compute these differences can be found in Table 3. It can be observed that overall there is a significant increase $(0.1 - 0.2)$ both in the mean and the median, and for all the three metrics. This confirms that, apart from the case of single nodes, the collaboration is very effective to improve the classifiers.

To offer an additional insight into the functioning of the sharing mechanism, we visualize in Figure 4a the same metrics, but computed over the train sets of each single node. In this case, it is remarkable to observe that both the mean and median are negative, meaning that the accuracy is decreasing on the train set when entering the collaboration. Since the test metrics are instead increasing, this is a good sign that the CL algorithm is able to equip each node with a model that has an accuracy that goes far beyond the own dataset, and is effectively able to share insights not present in each single node.

Moreover, it is of interest to compare the performances achieved by the collaborative algorithm with the hypothetical case where the data are centralized in a single node. The values obtained by running the algorithm in this scenario are reported in boldface in Table 3. As it is reasonable to expect, for all the three metrics this scenario provides by far the best results, in particular obtaining on the test set an improvement over the disconnected case of 0.08 in BAcc, $0.15 - 0.19$ in Prec, $0.16 - 0.19$ in Rec. Nevertheless, in all these cases the activation of a collaboration is able to significantly close this gap, by reducing these values, in the case of the fully connected configuration, to 0.03 in BAcc, $0.05 - 0.06$ in Prec, 0.05 in Rec. Similar values are obtained in the other connection configurations.

All these results make it clear that the benefit of the collaboration is increased for the fully connected scenario, as one may reasonably expect. On the other hand, the pairwise connected and random and time-varying settings are almost as effective, and the random setting is even the most accurate in terms of the Prec metric when considering the mean improvement, and essentially equivalent to the other two settings when considering the median.

This fact is interesting in possible real applications since one may foresee that establishing and utilizing a connection may be expensive in different terms, and thus the nodes should be interested in establishing the minimal set of connections that are sufficient to obtain the desired improvement in the model.

In more general terms, the effect of the topology of the connections on the outcome of the algorithm is an interesting aspect to explore. As a first element to explain the quite good effectiveness of the pairwise interaction, we show in Figure 5 the distribution of the estimators over the $N = 20$ nodes at the end of the iteration. Namely, since each estimator is uniquely identified, it is possible at each moment to check where the estimators of each node have been fitted. In the figure, we show in each row the origin of the estimators of each node. In the disconnected case (left panel) there is no mix, and indeed each node owns only estimators that it

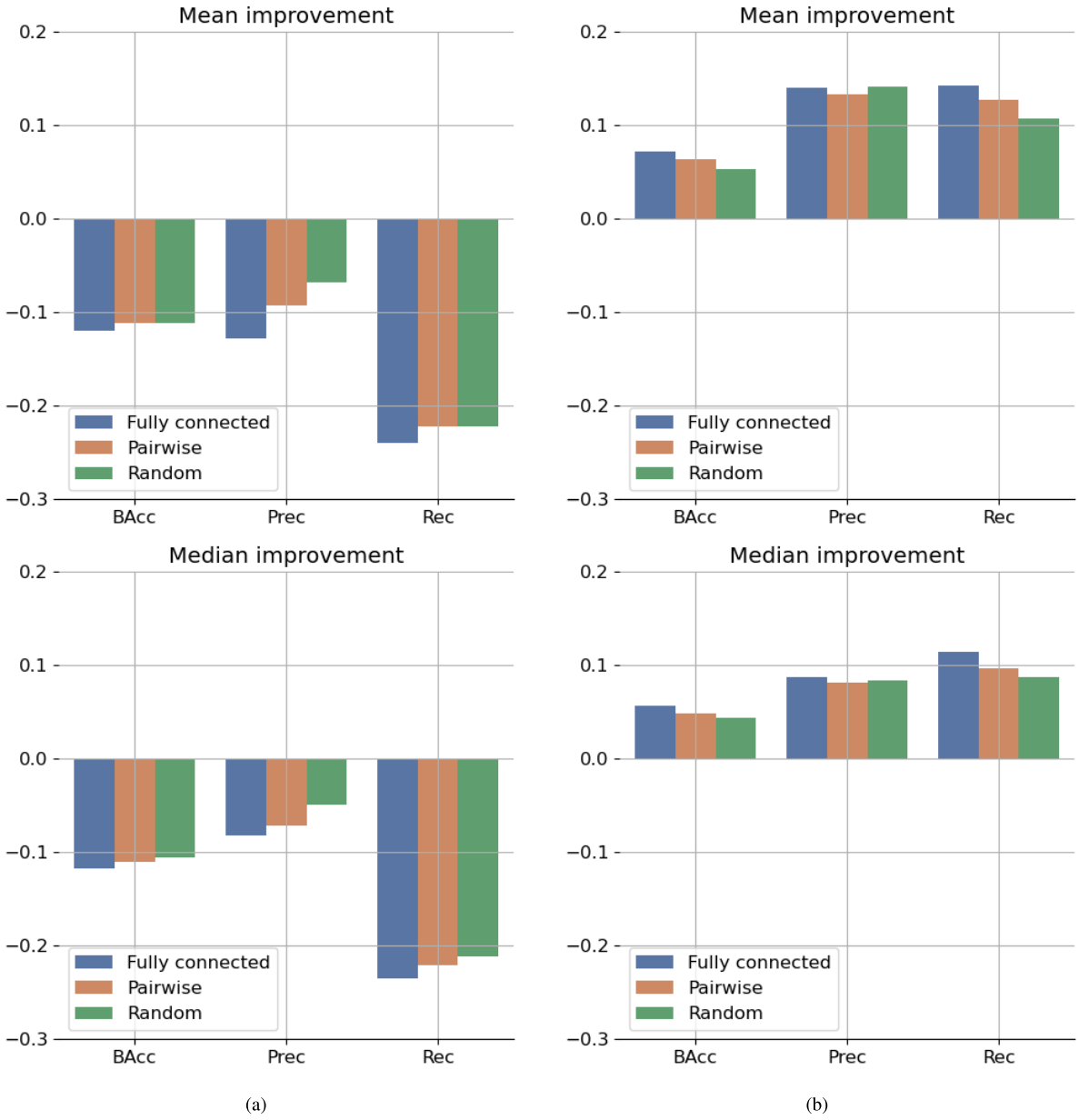

FIGURE 4. **Mean and median improvement in the three metrics over the disconnected case for the three collaborative scenarios (Fully connected, Pairwise, and Random). The metrics are computed over the train set (Figure 4a) and the test set (Figure 4b).**

fitted itself. In the fully connected case (right panel) a quite uniform mixing can instead be observed, with the addition that some nodes (Node0, Node2, Node5, Node6, Node8, Node10) produce almost no estimators that are used by the other ones. The fact that the mixing is quite stable among the nodes is an indication of the effectiveness of the sharing and ranking mechanism. In the intermediate case of the pairwise connected nodes (second from left panel) the mixing reflects the connection pattern, since each node holds estimators from its direct neighbors. In this case it is worth remarking that the estimators are effectively transmitted beyond the first order neighbors of a node, and this suggests that even a not fully connected network may be effective for the collaboration to work. The random and time varying case (second from right panel) is remarkable because it shows that changing the network at each iteration, even if the number of links is still as low as in the pairwise-connected setting, results in a significantly larger mixing. Moreover, this level of connection is sufficient to observe the emergence of the same mixing pattern as in the fully-connected case, with the same set of nodes (Node0, Node2, Node5, Node6, Node8, Node10) not being trusted.

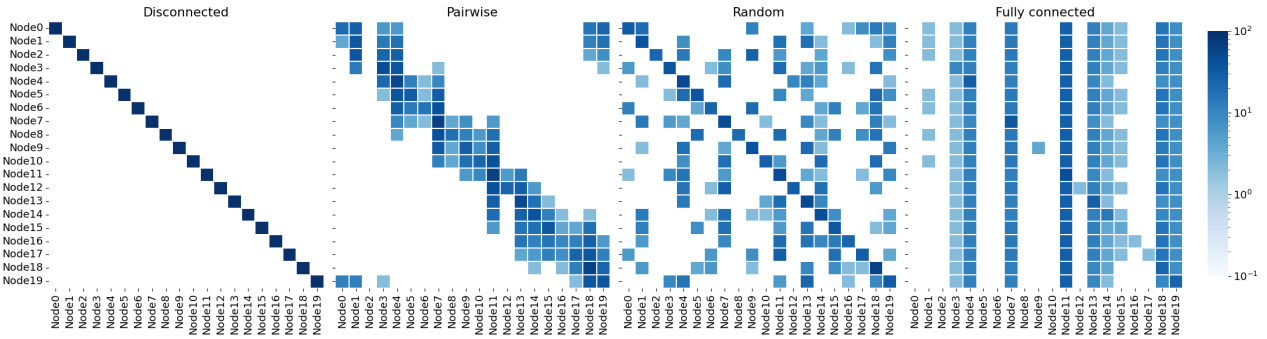

FIGURE 5. Origin of the estimators selected by each node at the end of the iteration for the four connection settings. Each row represents a node, and the columns indicate the origin of its estimators. The values of each row are normalized as percentages which sum to 100%.

TABLE 3. Mean and median absolute values of the three metrics for the three collaborative scenarios (Fully connected, Pairwise, and Random), and for the fully disconnected and fully centralized cases. The metrics are computed over the train set (Figure 4a) and the test set (Figure 4b). Only one value per metric is reported in the centralized scenario since in this case there is only one agent.

	BAcc			
	Train		Test	
	mean	median	mean	median
Fully connected	0.87	0.87	0.89	0.89
Pairwise	0.88	0.88	0.89	0.89
Random	0.88	0.89	0.88	0.89
Disconnected	0.99	1.00	0.82	0.84
Centralized	**0.92**		**0.92**	
	Prec			
	Train		Test	
	mean	median	mean	median
Fully connected	0.82	0.88	0.93	0.92
Pairwise	0.86	0.91	0.92	0.92
Random	0.88	0.93	0.93	0.93
Disconnected	0.95	1.00	0.79	0.83
Centralized	**1.00**		**0.98**	
	Rec			
	Train		Test	
	mean	median	mean	median
Fully connected	0.69	0.73	0.79	0.79
Pairwise	0.71	0.76	0.77	0.79
Random	0.71	0.77	0.75	0.77
Disconnected	0.93	1.00	0.65	0.68
Centralized	**0.84**		**0.84**	

VII. CONCLUSION

In this paper we developed and presented a collaborative anomaly detection algorithm that can leverage the communication between collaborating nodes in order to improve the models' performances. The algorithm is designed according to a fully decentralized structure, and it allows the sharing of algorithmic insights without the movement of any data. Although the classifiers are defined on top of Random Forests, we discussed how the same structure can be adapted to more general scenarios. Remarkably, the collaborative learning algorithm is developed in order to be fully integrated into a blockchain solution that ensures privacy-preserving guarantees on the execution and on its results. This component make it possible to verify the identity of the participating nodes, and to audit the execution of the algorithms and the correct functioning of the collaboration. Also in this case, the BC solution is not bounded to the specific algorithm of choice, and we discussed how more general ML models may be secured within the same framework.

In this work we have focused on a single ML ensemble method, namely RF, since it provides the necessary level of complexity for the development of the algorithm, without causing an excessive complication of the method. However, this solution is rather elementary, and extensions of the basic algorithmic structure will be analyzed in future work, where more complex building blocks can be exploited in place of Random Forests and Decision Trees. To this end, the algorithm and its integration in the BC have been designed to be as model-agnostic as possible, with the exception of the ranking system of Section III-D. Moreover, the effect of the connection topology on the behavior of the algorithm has been only partially explored in this work, and interesting options for its optimization remain open. In particular, we tested only one time-varying setting, and this showed already some promising features. A more in-depth analysis of the role of the network and methods for its optimization will be the focus of future research. Ultimately, we may foresee the application of these techniques to leverage the models stored in the BC for data sharing and trading in a data marketplace. Data marketplaces for ML models are an emerging trend [20], [37], [46], [56], which provide the opportunity to decentralize model development and lower the entry barrier into ML usage for companies which do not have either the skills, the capacity, or the access to learning data to develop the algorithms and train the models. Chaincodes in the BC network could control the access and permissions to the different models stored in chain applying governance rules defined by the consortium organizations.

APPENDIX
CONSTRUCTION OF THE TREE KERNEL

We consider a set $\mathcal{D} := \{D_i\}_{i=1}^{n_D}$ of $n_D \in \mathbb{N}$ decision trees, where $D_i := (T_i, X_i)$, T_i is a tree where each non-terminal node v has a label $s(v) \in \{1, \dots, d\}$, $d \in \mathbb{N}$, and a node feature $x(v) \in \mathbb{R}$.

We define a positive definite and symmetric kernel over $\mathcal{D}$ by a modification of the convolutional kernel of [11], [21]. Namely, we first enumerate the set $t_1, \dots, t_M$ of all subtrees of the trees in $\mathcal{D}$. We remark that the trees here are labeled,

meaning that the trees are equal only if the corresponding nodes have the same label. Given a tree T and any node $v \in T$, we then define a feature map

$$h(v) := [I_1(v), \ldots, I_M(v)]^T \in \{0, 1\}^M,$$

where $I_i(v) = 1$ if and only if the subtree t_i is rooted in v. This allow us to define the kernel k_{node} of [11] between two nodes $v \in T$, $v' \in T'$, as

$$k_{\text{node}}(v, v') := h(v)^T h(v') = \sum_{i=1}^{M} h_i(v) h_i(v').$$

It can be proven that $k_{\text{node}}(v, v')$ can be efficiently computed in polynomial time, and it simply counts the number of common subtrees rooted at both v and v' (see [11]). This kernel can be used to define a tree kernel k between T, T' simply by aggregation over all pairs of nodes, i.e.,

$$k_{\text{tree}}(T, T') := \sum_{v \in T, v' \in T'} k_{\text{node}}(v, v').$$

We extend this definition to a kernel k on our Decision Trees $D := (T, X), D' := (T', X') \in \mathcal{D}$ simply by adding a second kernel that takes into account the values of the node features, namely we sets

$$k(T, T') := \sum_{v \in T, v' \in T'} k_{\text{feat}}(x(v), x(v')) k_{\text{node}}(v, v'),$$

where $k_{\text{feat}} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is any positive definite kernel. Observe that k is positive definite because it is obtained by sums and products of positive definite kernels [5]. Moreover, for simplicity we use the linear kernel $k_{\text{feat}}(x(v), x(v')) := x(v)x(v')$, and this make it possible to write also k as an aggregation over node kernels via

$$\begin{aligned} k(T, T') &= \sum_{v \in T, v' \in T'} k_{\text{feat}}(x(v), x(v')) k_{\text{node}}(v, v') \\ &= \sum_{v \in T, v' \in T'} x(v)x(v')h(v)^T h(v') \\ &= \sum_{v \in T, v' \in T'} h_x(v)^T h_x(v'), \end{aligned}$$

where $h_x(v) := [x(v)I_1(v), \ldots, x(v)I_M(v)]^T \in \mathbb{R}^M$.

REFERENCES

[1] "Collaborative learning without sharing data," *Nature Mach. Intell.*, vol. 3, no. 6, p. 459, Jun. 2021. [Online]. Available: https://www.nature.com/articles/s42256-021-00364-5, doi: 10.1038/s42256-021-00364-5.

[2] S. A. Rahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, and M. Guizani, "A survey on federated learning: The journey from centralized to distributed on-site learning and beyond," *IEEE Internet Things J.*, vol. 8, no. 7, pp. 5476–5497, Apr. 2021.

[3] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *J. Netw. Comput. Appl.*, vol. 60, pp. 19–31, Jan. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804515002891

[4] E. Androulaki *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.* New York, NY, USA: Association for Computing Machinery, Apr. 2018, pp. 1–15, doi: 10.1145/3190508.3190538.

[5] N. Aronszajn, "Theory of reproducing kernels," *Trans. Amer. Math. Soc.*, vol. 68, no. 3, pp. 337–404, 1950.

[6] S. A. Baset, L. Desrosiers, N. Gaur, P. Novotny, A. O'Dowd, and V. Ramakrishna, *Hands-On Blockchain With Hyperledger: Building Decentralized Applications With Hyperledger Fabric and Composer*. Birmingham, U.K.: Packt, 2018.

[7] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[8] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification Regression Trees*. Evanston, IL, USA: Routledge, 2017.

[9] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009, doi: 10.1145/1541880.1541882.

[10] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[11] M. Collins and N. Duffy, "Convolution kernels for natural language," in *Proc. 14th Int. Conf. Neural Inf. Process. Syst., Natural Synthetic*. Cambridge, MA, USA: MIT Press, 2001, pp. 625–632.

[12] I. Dayan *et al.*, "Federated learning for predicting clinical outcomes in patients with COVID-19," *Nature Med.*, vol. 27, no. 10, pp. 1735–1743, Oct. 2021, doi: 10.1038/s41591-021-01506-3.

[13] S. De Marchi, R. Schaback, and H. Wendland, "Near-optimal data-independent point locations for radial basis function interpolation," *Adv. Comput. Math.*, vol. 23, no. 3, pp. 317–330, Oct. 2005.

[14] E. Diao, J. Ding, and V. Tarokh, "HeteroFL: Computation and communication efficient federated learning for heterogeneous clients," 2020, *arXiv:2010.01264*.

[15] F. Fournier and I. Skarbovsky, "Enriching smart contracts with temporal aspects," in *Proc. Blockchain-ICBC, 2nd Int. Conf., Held Services Conf. Fed. (SCF)*, in Lecture Notes in Computer Science, San Diego, CA, USA, vol. 11521, J. Joshi, S. Nepal, Q. Zhang, and L.-J. Zhang, Eds. Springer, Jun. 2019, pp. 126–141, doi: 10.1007/978-3-030-23404-1_9.

[16] L. Giffon, C. Lamothe, L. Bouscarrat, P. Milanesi, F. Cherfaoui, and S. Koço, "Pruning random forest with orthogonal matching trees," Vannes, France, Tech. Rep. hal-02534421, V 1, Jun. 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02534421 and https://cap-rfiap2020.sciencesconf.org/

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[18] H. Guo and X. Yu, "A survey on blockchain technology and its security," *Blockchain, Res. Appl.*, vol. 3, no. 2, Jun. 2022, Art. no. 100067. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2096720922000070

[19] S. Gupta and M. Mayank, "HFS top 10 enterprise blockchain services 2018," Cambridge, MA, USA, 2018. [Online]. Available: https://us.nttdata.com/en/-/media/assets/reports/digitalblockchain-hfs-top-10-enterprise-blockchain-services-report.pdf

[20] M. Ha, S. Kwon, Y. J. Lee, Y. Shim, and J. Kim, "Where WTS meets WTB: A blockchain-based marketplace for digital me to trade users' private data," *Pervas. Mobile Comput.*, vol. 59, Oct. 2019, Art. no. 101078. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574119218305947

[21] D. Haussler, "Convolution kernels on discrete structures," Dept. Comput. Sci., Univ. California Santa Cruz, Tech. Rep., 1999, vol. 646.

[22] T. K. Ho, "Random decision forests," in *Proc. 3rd Int. Conf. Document Anal. Recognit.*, vol. 1, Aug. 1995, pp. 278–282.

[23] S. Horvath, S. Laskaridis, M. Almeida, I. Leontiadis, S. I. Venieris, and N. D. Lane, "FjORD: Fair and accurate federated learning under heterogeneous targets with ordered dropout," 2021, *arXiv:2102.13451*.

[24] (2021). *Hyperledger Fabric*. Hyperledger Foundation. [Online]. Available: https://www.hyperledger.org/use/fabric

[25] P. Kairouz *et al.*, "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, nos. 1–2, pp. 1–210, Jun. 2021, doi: 10.1561/2200000083.

[26] Y. Kang, Y. Liu, Y. Wu, G. Ma, and Q. Yang, "Privacy-preserving federated adversarial domain adaption over feature groups for interpretability," 2021, *arXiv:2111.10934*.

[27] V. Y. Kulkarni and P. K. Sinha, "Pruning of random forest classifiers: A survey and future directions," in *Proc. Int. Conf. Data Sci. Eng. (ICDSE)*, Jul. 2012, pp. 64–68.

[28] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, Dec. 2015, doi: 10.1038/nature14539.

[29] C. Li, Y. Yuan, and F.-Y. Wang, "Blockchain-enabled federated learning: A survey," in *Proc. IEEE 1st Int. Conf. Digit. Twins Parallel Intell. (DTPI)*, Jul. 2021, pp. 286–289.

[30] D. Li, D. Han, T.-H. Weng, Z. Zheng, H. Li, H. Liu, A. Castiglione, and K.-C. Li, "Blockchain for federated learning toward secure distributed machine learning systems: A systemic survey," *Soft Comput.*, vol. 26, pp. 4423–4440, Nov. 2021, doi: 10.1007/s00500-021-06496-5.

[31] R. Liu and H. Yu, "Federated graph neural networks: Overview, techniques and challenges," 2022, *arXiv:2202.07256*.
[32] G. Long, Y. Tan, J. Jiang, and C. Zhang, "Federated learning for open banking," in *Federated Learning: Privacy and Incentive*, Q. Yang, L. Fan, and H. Yu, Eds. Cham, Switzerland: Springer, 2020, pp. 240–254, doi: 10.1007/978-3-030-63076-8_17.
[33] C. Ma, J. Li, M. Ding, L. Shi, T. Wang, Z. Han, and H. V. Poor, "When federated learning meets blockchain: A new distributed learning paradigm," 2020, *arXiv:2009.09338*.
[34] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AISTATS*, 2017, pp. 1–10.
[35] W. Meng, E. W. Tischhauser, Q. Wang, Y. Wang, and J. Han, "When intrusion detection meets blockchain technology: A review," *IEEE Access*, vol. 6, pp. 10179–10188, 2018.
[36] F. Nan, J. Wang, and V. Saligrama, "Pruning random forests for prediction on a budget," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2016, pp. 2342–2350.
[37] D. Nasonov, A. A. Visheratin, and A. Boukhanovsky, "Blockchain-based transaction integrity in distributed big data marketplace," in *Computational Science—ICCS*, Y. Shi, H. Fu, Y. Tian, V. V. Krzhizhanovskaya, M. H. Lees, J. Dongarra, and P. M. A. Sloot, Eds. Cham, Switzerland: Springer, 2018, pp. 569–577.
[38] S. R. Pokhrel and J. Choi, "Federated learning with blockchain for autonomous vehicles: Analysis and design challenges," *IEEE Trans. Commun.*, vol. 68, no. 8, pp. 4734–4746, Aug. 2020.
[39] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.
[40] O. Sagi and L. Rokach, "Ensemble learning: A survey," *WIREs Data Mining Knowl. Discovery*, vol. 8, no. 4, p. e1249, Jul. 2018, doi: 10.1002/widm.1249.
[41] G. Santin and B. Haasdonk, "Convergence rate of the data-independent P-greedy algorithm in kernel-based approximation," *Dolomites Res. Notes Approx.*, vol. 10, pp. 68–78, Dec. 2017.
[42] G. Santin and B. Haasdonk, "Kernel methods for surrogate modelin," in *System- and Data-Driven Methods and Algorithms*, vol. 1, P. Benner, S. Grivet-Talocia, A. Quarteroni, G. Rozza, W. Schilders and L. M. Silveira, Eds. Berlin, Germany: De Gruyter, 2021, pp. 311–354, doi: 10.1515/9783110498967-009.
[43] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[44] M. J. Sheller, B. Edwards, G. A. Reina, J. Martin, S. Pati, A. Kotrotsou, M. Milchenko, W. Xu, D. Marcus, R. R. Colen, and S. Bakas, "Federated learning in medicine: Facilitating multi-institutional collaborations without sharing patient data," *Sci. Rep.*, vol. 10, no. 1, p. 12598, Jul. 2020, doi: 10.1038/s41598-020-69250-1.
[45] Y. Shi, H. Yu, and C. Leung, "Towards fairness-aware federated learning," 2021, *arXiv:2111.01872*.
[46] M. Travizano *et al.*, *Wibson: A Case Study a Decentralized, Privacy-Preserving Data Marketplace*. Cham, Switzerland: Springer, 2020, pp. 149–170, doi: 10.1007/978-3-030-44337-5_8.
[47] M. H. ur Rehman, K. Salah, E. Damiani, and D. Svetinovic, "Towards blockchain-based reputation-aware federated learning," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 183–188.
[48] Z. Wang and Q. Hu, "Blockchain-based federated learning: A comprehensive survey," 2021, *arXiv:2110.02182*.
[49] H. Wendland, *Scattered Data Approximation* (Cambridge Monographs on Applied and Computational Mathematics). Cambridge, U.K.: Cambridge Univ. Press, 2005, vol. 17.
[50] T. Wenzel, G. Santin, and B. Haasdonk, "A novel class of stabilized greedy kernel approximation algorithms: Convergence, stability and uniform point distribution," *J. Approximation Theory*, vol. 262, Feb. 2021, Art. no. 105508.
[51] S. Xu, S. Liu, and G. He, "A method of federated learning based on blockchain," in *Proc. 5th Int. Conf. Comput. Sci. Appl. Eng.*, New York, NY, USA, Oct. 2021, pp. 1–8, doi: 10.1145/3487075.3487143.
[52] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, Mar. 2019, doi: 10.1145/3298981.
[53] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," *Synth. Lectures Artif. Intell. Mach. Learn.*, vol. 13, no. 3, pp. 1–207, 2019.
[54] W. Yang, Y. Zhang, K. Ye, L. Li, and C.-Z. Xu, "FFD: A federated learning based method for credit card fraud detection," in *Big Data*, K. Chen, S. Seshadri, and L.-J. Zhang, Eds. Cham, Switzerland: Springer, 2019, pp. 18–32.
[55] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowl.-Based Syst.*, vol. 216, Mar. 2021, Art. no. 106775. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705121000381, doi: 10.1016/j.knosys.2021.106775.
[56] G. Zyskind, O. Nathan, and A. S. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *Proc. IEEE Secur. Privacy Workshops*, May 2015, pp. 180–184.

GABRIELE SANTIN received the B.Sc., M.Sc., and Ph.D. degrees in applied mathematics from the University of Padova, Italy. He was a Postdoctoral Researcher at the University of Stuttgart, Germany. He is currently a Researcher at the Mobile and Social Computing Laboratory (MobS), Digital Society Center, Bruno Kessler Foundation, Trento, Italy. He is also an Associate Member of the Cluster of Excellence Data-Integrated Simulation Science, Stuttgart. His research interest includes the intersection of machine learning and computational sciences.

INNA SKARBOVSKY received the B.A. degree in computer science and the M.B.A. degree from the Technion—Israel Institute of Technology. She is currently a Research Staff Member with the Business Automation and Optimization Group, IBM Research—Haifa. She is also the Principal Architect and a Developer of the IBM Proactive Technology Online (Proton) complex event processing research tool. Her research interests include event-based technologies, distributed applications, complex event processing, streaming data, big data technologies, and blockchain. She has received the IBM Open Source Accomplishment Award for the work on PROTON, in 2016, and the IBM Outstanding Accomplishment Award for her work on event processing, in 2018.

FABIANA FOURNIER received the B.Sc. degree in information systems, and the M.Sc. and Ph.D. degrees in industrial engineering from the Technion—Israel Institute of Technology. She is currently a Senior Researcher with the Business Automation and Optimization Group, IBM Research—Haifa. She has over 20 years of research in the areas of business processes, event processing, organizational and process modeling, business transformation, and blockchain. Many of the projects in which she has been involved in received prestigious awards, including the IBM Technical and Science Accomplishment Awards in the areas of component business modeling, systems engineering in the aerospace and defense, and artifact-centric processes; the IBM Open-Source Accomplishment Award for the work on the complex event processing tool PROTON; and the IBM Technical and Science Outstanding Accomplishment Award for her work in the area of event processing.

BRUNO LEPRI received the Ph.D. degree in computer science from the University of Trento. He leads the Mobile and Social Computing Laboratory (MobS), Bruno Kessler Foundation, Trento, Italy. He has recently launched the Center for Computational Social Science and Human Dynamics, a joint initiative between Fondazione Bruno Kessler and the University of Trento. In 2010, he has won a Marie Curie Cofund Postdoctoral Fellowship and he has held a postdoctoral position at the MIT Media Laboratory. His research interests include computational social science, machine learning, network science, and personal data management. His research has received attention from several international press outlets and obtained the ten year impact award at MUM, in 2021, the James Chen Annual Award for the Best 2016 UMUAI Paper, and the Best Paper Award at ACM Ubicomp, in 2014. His work on personal data management was one of the case studies discussed at the World Economic Forum.