


Tailored IoT & BigData Sandboxes and Testbeds for Smart,
Autonomous and Personalized Services in the European
Finance and Insurance Services Ecosystem

Infinitech

D4.14 – Encrypted Data Querying and Personal Data Market - II

| | |
|---|--|
| Revision Number | 3.0 |
| Task Reference | T4.5 |
| Lead Beneficiary | FBK |
| Responsible | Bruno Lepri |
| Partners | IBM, FBK, INNOV |
| Deliverable Type | Report (R) |
| Dissemination Level | Public |
| Due Date | 2021-07-30 |
| Delivered Date | 2021-08-06 |
| Internal Reviewers | Final |
| Quality Assurance | Internally Reviewed and Quality Assurance Reviewed |
| Acceptance | WP Leader Accepted and Coordinator Accepted |
| EC Project Officer | Pierre-Paul Sondag |
| Programme | HORIZON 2020 - ICT-11-2018 |
|  | This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no 856632 |

Contributing Partners

| Partner Acronym | Role ¹ | Author(s) ² |
|-----------------|-------------------|---|
| FBK | Lead Beneficiary | Bruno Lepri, Gabriele Santin, and Raman Kazhamiakin |
| IBM | Contributor | Fabiana Fournier, and Inna Skarbovsky |
| INNOV | Contributor | Nikos Kapsoulis, and Filia Filippou |
| UNIC | Internal Reviewer | Marianna Charalambous |
| GFT | Internal Reviewer | Maurizio Megliola |
| INNOV | QA | John Soldatos |

Revision History

| Version | Date | Partner(s) | Description |
|---------|------------|------------|--|
| 0.1 | 2021-04-29 | IBM | First draft of ToC |
| 0.2 | 2021-06-01 | IBM, FBK | Revised ToC |
| 0.3 | 2021-07-1 | IBM | Initial contribution to Section 2 and 4 |
| 0.4 | 2021-07-02 | INNOV | Initial contribution to Section 5 |
| 0.5 | 2021-07-05 | FBK | Initial contribution to Section 3 |
| 0.6 | 2021-07-30 | FBK | Initial contribution to Sections 1 and 5 |
| 0.7 | 2021-08-02 | FBK | Finalisation of Sections 1, 2, 3, 4, 5 and executive summary |
| 1.0 | 2021-08-03 | FBK, IBM | First Version for Internal Review |
| 1.1 | 2021-08-04 | UNIC | Internal Review |
| 1.2 | 2021-08-04 | GFT | Internal Review |
| 2.0 | 2021-08-04 | FBK | Version for Quality Assurance |
| 2.1 | 2021-08-05 | INNOV | Quality Assurance |
| 3.0 | 2021-08-06 | FBK | Version for Submission |

¹ Lead Beneficiary, Contributor, Internal Reviewer, Quality Assurance

² Can be left void

Executive Summary

This deliverable (D4.14) is the second of three deliverables planned in the scope of Task 4.5 of the INFINITECH project. The purpose of this task is to overcome the current limitations of standard data sharing paradigms, and this ambitious goal is achieved by the design and implementation of a framework for securely querying, processing, and analyzing data over the INFINITECH permissioned blockchain infrastructure. This goal will enable decentralized, federated, and secure execution of machine learning (ML) algorithms and lay the foundation for a market of insights obtained by running ML algorithms.

Within this general vision, deliverable D4.14 has the goal of documenting in detail the novel *Insights Sharing and Provenance* conceptual module jointly designed by IBM and FBK. More precisely, we here describe the five components of this module, namely (i) *Identity Manager and User/Client Authentication*, (ii) *Federated Learning Artifacts Store*, (iii) *Artifacts Usage Audit*, (iv) *Secured Execution*, and (v) *Tokenization*. It is worth noting that this module is built on top of IBM Hyperledger Fabric [1, 4], the blockchain platform selected by the INFINITECH project.

Then, we introduce the federated learning algorithm, based on Random Forests (RF) [3], implemented as the basis for the INFINITECH framework for securely sharing ML-derived insights, instead of raw data, among different organizations (e.g., banks, insurance companies, etc.). Doing this, we also provide the details of its architecture, and describe the preliminary results we have obtained in a specific application to a fraud detection task.

In the last part of the deliverable, we propose a secure execution framework, based on IBM's Vulcan project and built on top of the Hyperledger Fabric blockchain, that provides a verifiable privacy-preserving computation environment for federated learning scenarios and for enabling the sharing and trading of ML-derived insights (i.e., a marketplace of ML-derived insights). This execution framework is here applied to federated learning scenarios where the computation is decentralized and there is no inherent trust between the parties (i.e., the learning nodes).

Overall, this report provides the conceptual design and some of the implementation steps for the development of a secure environment for running and auditing federated learning approaches and for enabling a token-based marketplace of ML-derived insights among different organizations. Doing this, we are proceeding toward the building of an MVP that will permit INFINITECH partners and INFINITECH users to cooperatively solve financial and insurance challenges using machine learning and deep learning approaches but without losing the control of the data they own.

Table of Contents

| | |
|---|-----------|
| Introduction | 8 |
| Objectives of the Deliverable | 9 |
| Insights from other Tasks and Deliverables | 10 |
| Updates with respect to the Previous Version (D4.13) | 10 |
| Structure | 10 |
| Background and Motivation | 12 |
| Federated learning | 14 |
| Use case for the development of the algorithm | 14 |
| Dataset | 14 |
| Task and metrics | 15 |
| Evaluation of the existing methodologies | 16 |
| Architecture of federated learning algorithm | 16 |
| Single-node learning | 17 |
| Federated learning strategy | 17 |
| Key features | 18 |
| Additional properties | 19 |
| Preliminary evaluation of the algorithm | 19 |
| Communication networks | 19 |
| Preparation of the dataset | 20 |
| Hyperparameters setup | 21 |
| Accuracy results | 21 |
| Mixing of the estimators | 22 |
| Blockchain framework | 24 |
| Blockchain architecture for federated learning | 24 |
| Secured execution environment | 26 |
| Federated learning artifacts store and secured execution components | 27 |
| Image and execution record data model and chaincodes | 27 |
| Federated learning process orchestration chaincode | 30 |
| ML models' data and chaincode | 31 |
| Artifacts Usage Audit | 33 |
| Algorithm image auditing | 33 |
| Execution auditing | 34 |
| Model lifecycle auditing | 34 |
| Model usage trail – for insights' marketplace | 34 |

| | |
|--|-----------|
| ML insights' marketplace and trading | 34 |
| ML models' insights' trading use cases | 35 |
| Scenario 1: External organization buys a trained ML model | 35 |
| Scenario 2: External organization orders the execution of a federated learning process | 36 |
| Tokenization | 37 |
| Conclusions and next steps | 38 |
| Appendix A: Literature | 39 |

List of Figures

| | |
|--|----|
| Figure 1: Revised representation of the INFINITECH framework for securely accessing, managing and sharing data and ML insights | 13 |
| Figure 2: Communication networks for the federated learning use case | 21 |
| Figure 3: Accuracy results for the fraud detection use case | 23 |
| Figure 4: Distribution of the trees within each of the RFs trained on local nodes | 24 |
| Figure 5: Blockchain framework's building blocks and flows | 26 |
| Figure 6: Secured execution environment | 28 |
| Figure 7: Federated learning process with blockchain | 31 |
| Figure 8: Logical components of the ML models' insights marketplace | 36 |
| Figure 9: Acquisition of ML model's insights' process | 37 |

List of Tables

| | |
|---|----|
| Table 1: Dataset for the federated learning use case on fraud detection | 16 |
| Table 2: Metrics for the federated learning use case on fraud detection | 16 |
| Table 3: Distributed dataset for the federated learning use case on fraud detection | 21 |
| Table 4: Name, role, and value of the parameters used in our experiments | 22 |
| Table 5: Image data types | 29 |
| Table 6: Execution record entity schema | 30 |
| Table 7: Learning process instance entity schema | 32 |
| Table 8: ML model instance and input entities' schema | 32 |

Abbreviations/Acronyms

Abbreviation Definition

| | |
|------|------------------------------|
| Bacc | Balanced Accuracy |
| CA | Certification Authority |
| FN | False Negative |
| FP | False Positive |
| JSON | JavaScript Object Notation |
| ML | Machine Learning |
| MPC | Multi-Party Computation |
| MVP | Minimum Viable Product |
| OPAL | Open Algorithms |
| PCA | Principal Component Analysis |
| Prec | Precision |
| Rec | Recall |
| RF | Random Forest |
| RFF | Random Fourier Features |
| SVM | Support Vector Machine |
| TP | True Positive |
| TN | True Negative |

1. Introduction

The current deliverable is the second one of a series of three deliverables whose aim is to describe the activities conducted in the Task 4.5 “Secure and Encrypted Queries over Blockchain Data” of the INFINITECH project. The main objective of this task is the design and implementation of a framework for querying encrypted data over the INFINITECH permissioned blockchain infrastructure and for running Machine Learning (ML) algorithms on these data.

As already mentioned in the deliverable D4.13 “Encrypted Data Querying and Personal Data Market – I” (submitted at M14), the inspiration for this framework comes from two recent approaches, ENIGMA [14] and Open Algorithms (OPAL) [8]. Both approaches provide a mechanism for the privacy-preserving sharing of data across multiple data repositories. In particular, OPAL introduces the concept of moving the ML algorithms to the data repositories, where each data repository participating in the computation performs all its computations behind the firewalls. Additionally, ENIGMA introduces the notion of Multi-Party Computation (MPC) [9] that gives the data repositories the ability to collectively perform an algorithm computation that produces some results without revealing the raw data.

In the current deliverable, we revise our initial proposal for an INFINITECH framework for securely accessing, managing, and sharing data across financial and insurance institutions. To this end, Section 2 describes the novel *Insights Sharing and Provenance* conceptual module jointly designed by IBM and FBK. More precisely, we introduce the five components of this module, namely (i) *Identity Manager and User/Client Authentication*, (ii) *Federated Learning Artifacts Store*, (iii) *Artifacts Usage Audit*, (iv) *Secured Execution*, and (v) *Tokenization*. This module is built on top of Hyperledger Fabric (simply Fabric) [1, 4], the blockchain platform selected by the INFINITECH project.

In Section 3, we introduce the federated learning algorithm, based on Random Forests (RF) [3], implemented as the basis for the INFINITECH framework for securely sharing ML insights, instead of raw data, among different organizations. Here, we also provide the details of its architecture, and describe the results obtained in a preliminary application to a fraud detection task.

Then, Section 4 introduces a secure execution framework, based on IBM’s Vulcan project and built on top of Hyperledger Fabric blockchain, that provides a verifiable privacy-preserving computation environment for federated learning scenarios. This execution framework was originally developed to address the domain of problems where one party has some sensitive data, and another party is interested in the result of a computation on that data. Here, we extend the framework to the federated learning scenarios where the same problem exists when the computation is decentralized and there is no inherent trust between the parties. In addition, we enable the sharing and trading of insights by applying tokens on top of the Hyperledger Fabric capabilities developed within the scope of the INFINITECH project.

Finally, we draw some conclusions, and we propose as a next step to add blockchain-based consent mechanisms on top of the ML federated algorithms, in collaboration with INNOV.

1.1 Objectives of the Deliverable

The main goals of Task 4.5 “Secure and Encrypted Queries over Blockchain Data” are the design and implementation of a framework for querying encrypted data over the INFINITECH permissioned blockchain infrastructure, namely Hyperledger Fabric, and for running ML algorithms on them, as well as creating the foundation for a personal data market where individuals and organizations will be able to trade their data or insights in exchange for tokens or other assets.

These goals encompass, in this second deliverable, the following specific objectives:

- To describe the ongoing design, research and implementation work on the components of the INFINITECH framework for securely accessing, managing and sharing data. The first deliverable (D4.13) focuses on the description of the initial work performed on the *Data Policy Framework* component, the *Algorithm Runtime* component, and the *Data Marketplace* component. Here, as

planned, we further describe in detail the design, research, and implementation of these components as well as the *Data Usage Audit* component. In particular, we introduce the novel *Insights Sharing and Provenance* conceptual module jointly designed by IBM and FBK. More precisely, we introduce the five components of this module, namely (i) *Identity Manager and User/Client Authentication*, (ii) *Federated Learning Artifacts Store*, (iii) *Artifacts Usage Audit*, (iv) *Secured Execution*, and (v) *Tokenization*. This module is built on top of Hyperledger Fabric, the blockchain platform selected by the INFINITECH project.

- To describe the ongoing design, research and implementation work on the federated learning algorithms selected as the basis for the INFINITECH framework for securely sharing ML insights. More specifically, we introduce an approach based on Random Forests (RF), also providing the details of its architecture, and describing its results in a preliminary application to a fraud detection task.

1.2 Insights from other Tasks and Deliverables

Deliverable D4.14 “Encrypted Data Querying and Personal Data Market – II” is released in the scope of WP4 “Interoperable Data Exchange and Semantic Interoperability” activities and documents the collaborative work performed by IBM and FBK within the context of task T4.5 “Secure and Encrypted Queries over Blockchain Data”.

This document relies on previous work reported in the following deliverables:

- D4.7 “Permissioned Blockchain for Finance and Insurance - I” (submitted at M11) which revolves around the first version of blockchain applications carried out in the scope of the INFINITECH project.
- D4.8 “Permissioned Blockchain for Finance and Insurance - II” (submitted at M19) presents the second version of the blockchain activities in the INFINITECH project.
- D4.10 “Blockchain Tokenization and Smart Contracts – I” (submitted at M14) motivates the usage of tokenization in blockchain networks in the financial and insurance sectors and describes the first round of activities carried out on tokenization in the INFINITECH project.
- D4.13 “Encrypted Data Querying and Personal Data Market – I” (submitted at M14) motivates the need for a paradigm change concerning the currently dominant model of siloed data collection, management, and exploitation as well as it illustrates the first round of activities carried out in INFINITECH for designing and implementing a framework for securely accessing, managing, and sharing data or ML insights between customers, financial and insurance institutions.

1.3 Updates with respect to the Previous Version (D4.13)

It is worth highlighting that the content of the current deliverable, D4.14 “Encrypted Data Querying and Personal Data Market – II”, is entirely novel with respect to the one reported in D4.13 “Encrypted Data Querying and Personal Data Market – I”.

1.4 Structure

We organized the structure of D4.14 as follows:

- Section 2 describes the novel *Insights Sharing and Provenance* conceptual module jointly designed by IBM and FBK. More precisely, we motivate and introduce the five components of this module, namely (i) *Identity Manager and User/Client Authentication*, (ii) *Federated Learning Artifacts Store*, (iii) *Artifacts Usage Audit*, (iv) *Secured Execution*, and (v) *Tokenization*. This module is built on top of Hyperledger Fabric, the blockchain platform selected by the INFINITECH project.
- Section 3 introduces in detail the federated learning algorithm, based on Random Forests (RF), implemented as the basis for the INFINITECH framework for securely sharing insights. In this section, we

also provide the details of its architecture as well as we describe its results in a preliminary application to a fraud detection task.

- Section 4 introduces a secure execution framework, based on IBM's Vulcan and the tokenization work carried out within the INFINITECH project, that provides a verifiable privacy-preserving computation environment for federated learning scenarios and for enabling the sharing and trading of ML-derived insights.
- In Section 5 we conclude the deliverable describing the next steps toward the implementation of a minimum viable product (MVP) for the proposed framework as well as the possible addition of blockchain-based consent mechanisms on top of the ML algorithms, in collaboration with INNOV, and the extension of the federated learning approach to task and scenarios where training and testing data will be encrypted.

2 Background and Motivation

In the deliverable D4.13 “Encrypted Data Querying and Personal Data Market – I” (submitted at M14), we have introduced an initial proposal for an INFINITECH framework for securely accessing, managing and sharing data and ML insights across financial and insurance institutions. As explained in detail in D4.13, the proposed framework assumes an organization model, where different institutions (e.g., companies, public and private institutions) work together in a shared environment (i.e., sandbox) in a federated manner. The critical and sensitive data managed by individual organizations under certain circumstances cannot be shared with third parties in their raw form. Hence, in D4.14 “Encrypted Data Querying and Personal Data Market – II” we propose a federated learning framework where only insights obtained by running locally machine learning algorithms can be shared among organizations.

In particular, in the current deliverable we describe the *Insights Sharing and Provenance* conceptual module depicted in Figure 1, as well as the federated learning approach selected as the basis for the INFINITECH framework for sharing ML models’ insights. In particular, The *Insights Sharing and Provenance* module, jointly developed by IBM and FBK, is built on top of the blockchain platform selected by the INFINITECH project, namely Hyperledger Fabric. The basic elements and principles of blockchain platforms, and in particular of Hyperledger Fabric, are described in D4.7 “Permissioned Blockchain for Finance and Insurance - I” (submitted at M11). Therefore, we assume the reader is familiar with blockchain basic terminology and that the description of concepts such as peers, organizations, orderers, certificate authority (CA), channels, and chaincodes (i.e., smart contracts in the Hyperledger Fabric jargon) are out of the scope of this document. For more details on Hyperledger Fabric the reader can refer to [1] and [4].

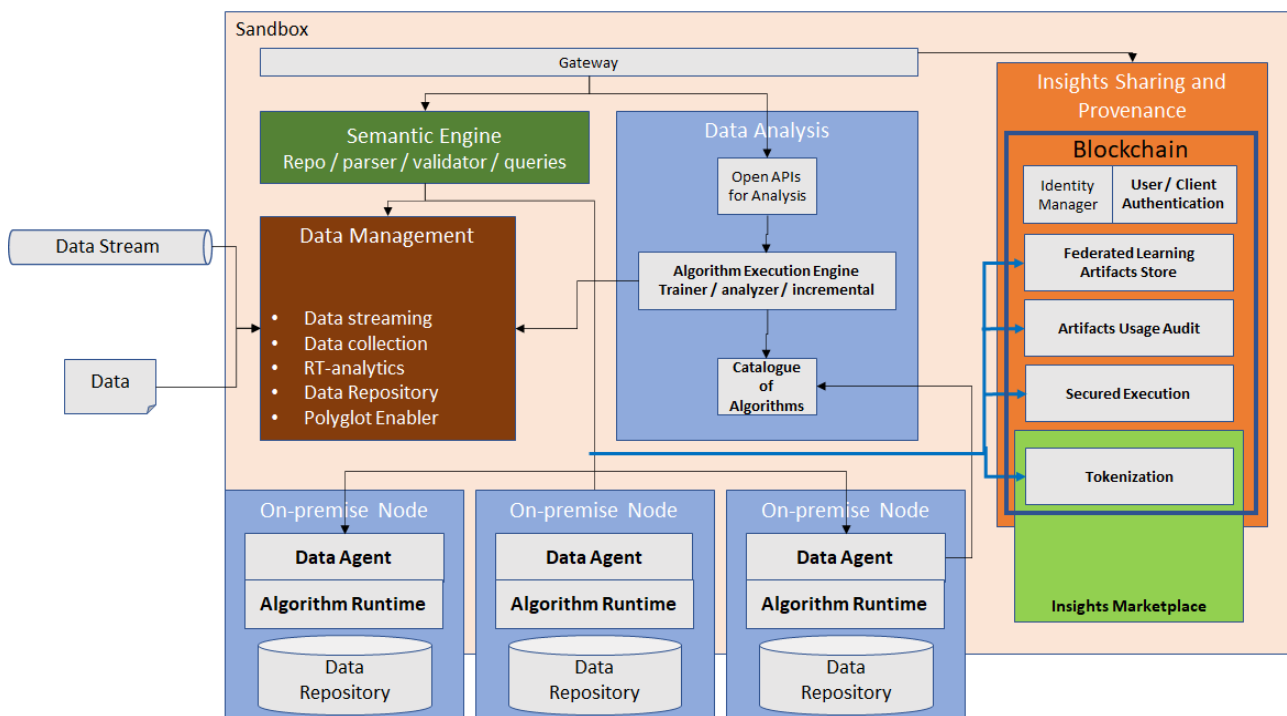


Figure 1: Revised representation of the INFINITECH framework for securely accessing, managing and sharing data and ML insights. In this figure is depicted the *Insights Sharing and Provenance* module that we describe in the current deliverable.

The choice of designing the *Insights Sharing and Provenance* module on top of a blockchain is due to the fact that it is the natural platform to support the tracking of the provenance of the federated learning

artifacts, providing an indisputable history of updates, which is immutable, tamper-proof, and transparent among all participants of the blockchain network.

More precisely, the designed module includes the following components:

- *Identity Manager and User/Client Authentication*: these components provide several mechanisms in the Hyperledger Fabric environment such as (i) the specification of Certification Authority (CA) servers (defined as part of the blockchain 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.
- *Federated Learning Artifacts Store*: these components are the chaincodes implementing the business logic for storing, updating, retrieving, and querying business artifacts related to federated learning – from algorithm images to model metadata, to the metadata of the learning process.
- *Artifacts Usage Audit*: these are the functionalities built in into each chaincode which allow to query the history of updates for each business artifact stored on the ledger, thus allowing to present a clear and complete picture of the artifact's provenance.
- *Secured Execution*: a blockchain-based framework for securely running the computational tasks of the ML algorithm, producing signed outputs (i.e., insights from the learning round), and storing these outputs on the ledger. This framework is based on Vulcan. Vulcan is a framework developed by IBM Research – Haifa and extended in the scope of INFINITECH for managing federated learning approaches. Vulcan deals with the inability of one organization to trust an execution of logic inside another organization. For example, in value-based agreements, an insurance company needs to pay based on computation done by providers on their respective data. In the case of federated learning, Vulcan provides a mechanism to decentralize the learning process while establishing the auditability and verifiability of the ML local models and improving 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 in order to participate in the learning process (see Section 4 for details).
- *Tokenization*: the chaincode that supports the trading of ML-derived insights in the INFINITECH marketplace which transfers tokens from the buying organization's account to the selling's organization account (see Section 4.5.2).

3 Federated learning

In this section, we introduce the federated learning algorithm implemented as the basis for the INFINITECH framework for securely sharing ML insights. We also provide the details of its architecture and describe the results in a preliminary application to a fraud detection task.

Federated learning has recently emerged as a new field of research to provide decentralized and secure counterparts to traditional ML algorithms [6]. In this setting both the decentralized and the cooperative aspects are of great importance. During the last couple of years, there have been attempts to apply federated learning in banking and finance problems [15].

The need to perform secure and orchestrated data-sharing or insights-sharing approaches requires algorithms that can exploit multiple sources of data to obtain knowledge that thus may be shared across multiple nodes. In this way, the single node performance can be significantly improved by cooperative actions, in a way that makes it possible to break the limitations faced by each single learning agent, user, or organization, while maintaining data integrity.

In this first phase, we concentrate on the design of an algorithm with the following characteristics:

- Several nodes should be able to interact in the federation.
- Each node has access to its own dataset described by a common set of features.
- Each node aims at solving the same task, given its own data.
- Each node has a memory constraint that limits the size of the model to be stored.
- The algorithm should support the following architectural settings:
 - Asynchronous communication;
 - Temporal varying connection network;
 - Decentralized architecture.

3.1 Use case for the development of the algorithm

Although the framework is developed to handle general federated learning problems, we use a fraud detection task as a use case to define the architecture and develop the algorithm. The use of a specific task dictates a series of features that the algorithm should have and requires specific implementation choices.

It is nevertheless important to remark that this use case serves only as a concrete instance of a federated learning task, while the algorithm will be designed to account for a more general scenario, as detailed in the following sections.

3.1.1 Dataset

We have built the algorithm to solve the *Credit Card Fraud Detection* problem published as a Kaggle challenge in November 2016 [16]. The dataset collects credit card transactions performed in Europe during two days in September 2013. The data are anonymized by publishing only 28 features resulting from a Principal Component Analysis (PCA) [5] of the original data. Additionally, each transaction contains a timestamp, the amount transferred, and a label indicating whether the transaction was fraudulent. We do not make use of the amount of the transaction and of the timestamp for this task.

The dataset is highly unbalanced, meaning that most of the transactions are legit ones. Table 1 reports the statistics of the entire dataset.

Table 1: Dataset for the federated learning use case on fraud detection.

| | | | |
|------------------------|------------------|----------------------|--------------------------|
| Number of transactions | Number of frauds | Percentage of frauds | Features per transaction |
| 284807 | 492 | 0.173% | 28 |

In the next months we are planning to apply the proposed federated learning algorithm to the fraud detection task and datasets defined by Pilot 7 “Avoiding financial crime” led by Caixa Bank, as pilot leader, and Fujitsu as tech proxy.

3.1.2 Task and metrics

The dataset is associated with the task of predicting if a new transaction is a fraudulent or a legal one, given the 28 features used to describe it.

This task can be understood in the more general framework of *anomaly detection* [2], which is a fundamental class of problems in ML and Pattern Recognition. These types of problems are binary classification problems (i.e., the data may belong either to the positive or negative class) with the additional challenge that the positive class is largely under-represented, meaning that a few examples of the anomalous behaviour (the frauds in our case) are present in the dataset. In particular, it is important to adopt metrics that are suitable to measure the performances of an algorithm in this task. We summarize the metrics that we will use in Table 2.

Table 2: Metrics for the federated learning use case on fraud detection.

| Metric | Abbreviation | Description |
|-------------------|--------------|---|
| True Positive | TP | The number of frauds that are correctly detected. |
| False Positive | FP | The number of legitimate transactions that are wrongly marked as frauds. |
| True Negative | TN | The number of legitimate transactions that are correctly detected. |
| False Negative | FN | The number of frauds that are wrongly marked as legitimate transactions. |
| Recall | Rec | The quantity $TP / (TP + FN)$, that is the ratio between the correctly identified frauds, and the total number of actual frauds (detected or not). |
| Precision | Prec | The quantity $TP / (TP + FP)$, that is the ratio between the correctly identified frauds, and the total number of transactions marked as frauds (correctly or not). |
| Balanced Accuracy | BAcc | The quantity $(TP / (TP + FN) + TN / (TN + FP)) / 2$, that is the mean between the normalized number of true positives and true negatives, each normalized by the total number of either |

| | | |
|--|--|--|
| | | positive or negative samples in the dataset. |
|--|--|--|

3.1.3 Evaluation of the existing methodologies

We tested two existing methodologies to address this task.

Distributed gradient descent [10]. This is a kernel-based regression method for federated learning. It is based on the use of Random Fourier Features (RFF) [11], which allow the design of a distributed gradient descent algorithm. The data stored in each node are never exchanged, but instead at each iteration a vector of coefficients, resulting from local gradient computations, is shared among neighbouring nodes.

The advantages of this method are as follows:

- Local data are never exchanged, in perfect compliance with the federated learning paradigm.
- Theoretical guarantees are available to prove the convergence of the distributed optimization to a unique consensus shared among the nodes.
- The use of RFF makes the computations extremely efficient.

Nevertheless, some limitations emerged during the testing of this approach on our scenario:

- The algorithm requires a centralized authority that orchestrates the sharing of the coefficients between the nodes, and that makes sure that this exchange is synchronized.
- The orchestrated sharing requires that all nodes work in a synchronous manner, i.e. each node needs to wait for all the other ones to be ready to communicate before proceeding to the next iteration.
- It is necessary that the network of connections is fixed during the entire training phase, and in particular the algorithm is not robust with respect to disconnection of the nodes, even if only temporary.

Moreover, even when executed on a single node, the algorithm has an additional limitation:

- The use of kernel regression seems unsuited to solve anomaly detection problems. It is well known that the unbalanced nature of the dataset is not captured well by pure kernel regression algorithms.

These limitations convinced us to look for other approaches in this early phase. Nevertheless, this algorithm seems to be a good candidate to solve problems with both a more balanced dataset and with a setting where a centralized orchestration is possible.

One class Support Vector Machine (SVM) [2]. This algorithm is a well-known adaptation of classical SVM to work with anomaly detection problems, and it seems thus the perfect solution to tackle the poor performances of the previous approach on highly unbalanced datasets. After preliminary testing on a single node, the following observations emerged:

- The accuracy of the algorithm (measured with respect to the various metrics introduced in Section 3.1.2) is sufficiently good.
- The computational time to train the classifier is prohibitively large.

The second observation, together with the fact that there are no known approaches to extend One class SVM to a federated setting, motivated the choice of discarding further investigations on this method.

3.2 Architecture of federated learning algorithm

To meet the specifications outlined at the beginning of Section 3, we developed a new algorithm that we are going to describe in detail.

The method is based on the use of Random Forests (RFs) [3], which are flexible and effective learning algorithms that are commonly used for classification and anomaly detection tasks. We extend this method to work in a federated approach by designing suitable rules for the sharing and merging of these learners between different nodes.

3.2.1 Single-node learning

On each single node, the base algorithm is a simple RF regressor that is used to distinguish the positive class (label +1) from the negative class (label -1). The use of a regressor, rather than a classifier, makes it easier to combine different RFs, as discussed in the next section.

The implementation is based on the implementation that is available in the ML package Scikit-Learn [17]. Each node, when working alone, implements the following operations:

- **Fit:** The node starts by training a RF containing a fixed number `n_estimators` of trees. After the first iteration, if the fit operation is executed again then the existing forest is enlarged by training an additional number `n_estimators` of trees. This is realized by the Scikit-Learn implementation, which implements a `warm_start` flag to allow one to add trees to an existing forest, rather than starting from scratch. Each newly trained tree is marked with the ID of the node, and with an incremental and unique ID of the trees in this node. In this way, at each stage of the algorithm it is possible to uniquely identify the identity and source of each tree in the federation.
- **Rank:** Once a set of trained trees is available, the single node can rank them according to their importance in the prediction. Since each node has access only to its own training data, it is important to implement a strategy that is able to rank the nodes using only these data, but on the other hand that avoids overfitting, i.e. the excessive fine tuning of the classifier on the training data that can possibly prevent a meaningful generalization on new data [2]. To meet these requirements, we implement a bootstrap aggregation of `n_rep` linear models trained on the output prediction of each tree in the RF. Each linear estimator is trained on randomly drawn subsets of the training data consisting of a fraction `sample_size` of the whole dataset. In this way, we obtain a linear estimator that assigns to each tree a coefficient that is the mean of the corresponding `n_rep` coefficients. Since none of these coefficients is estimated using the entire dataset, this procedure largely reduces the overfitting. Finally, the magnitude of these coefficients is used to rank the single trees from the most important (large-magnitude coefficient) to the less important (small-magnitude coefficient).
- **Crop:** To satisfy the memory limit of each node, a **Crop** operation is implemented. In particular, the memory constraints are met by imposing a maximum depth `max_depth` on the newly trained trees (see point 1), and additionally imposing that each node stores at most a number `max_estimators` of trees. To enforce this condition, whenever new trees are added to the node (either using the **Fit** operation of point 1, or by the federated operations described in the next section) the trees are ranked using the **Rank** operation of point 2, and only the top `max_estimators` are kept, while the other ones are discarded.

Executing these operations, each node, if not communicating with the other ones, keeps enlarging its own RF by the addition of a constant number of trees until the reaching of the maximum capacity. Once the capacity is reached, the new trees will be ranked with the existing ones, and new **Fit** steps only improve the quality of the new trees, without increasing the size of the RF.

3.2.2 Federated learning strategy

In addition to the single node learning, we have implemented a federated strategy that allows several nodes to collaborate in solving a common problem.

The network of connections between the nodes is represented by an undirected graph, so that each node can communicate only with its set of neighbours. This network is possibly time-varying, and this allows us to model temporary interactions and communication failures.

In particular, each node has access to its own labelled dataset, which contains samples from a common setting. In our use case, this means that each node has a collection of transactions represented by the same set of features, some of which are labelled as frauds. Of course, different nodes have different datasets, and in particular different levels of balance between the positive and the negative classes may be present.

Each node still aims at the training of an own classifier (in our case, a fraud detection classifier), and in particular the federated approach does not require the creation of a single centralized model. Instead, each node can communicate with its neighbours to improve its own classifier.

In particular, the three operations of **Fit**, **Rank**, and **Crop** are still the base of the learning of each node. Additionally, the following operations are added:

- **Share:** A node may decide to participate in the federation and share insights with its neighbours. In this case, it ranks its current set of trees using the **Rank** operation, and then it transmits the top `n_share` trees with the neighbours. Observe that at this stage it is crucial to have a non-overfitting mechanism to rank the trees, since they are shared with other nodes, which are working on a different dataset.
- **Get:** A node may decide to accept the trees shared by its neighbours in the previous times. In this case, it collects all the shared trees and it merges them with its own ones, creating a large RF. If this temporary RF consists of more than the maximal number `max_estimators` of trees, the node runs the **Crop** operation, which keeps the top estimators and discards the others. In this way, the RF of the node may be composed of trees coming from different nodes. Observe that also at this stage it is crucial that the **Rank** operation, used in the **Crop** operation, is not too biased towards the own dataset of the node. Indeed, in this case the **Crop** operation would largely favour its own trees, making it difficult to accept and incorporate trees coming from other nodes. This incorporation is instead fundamental to benefit from the knowledge shared on the network.

The two operations are implemented with a registry assigned to each link in the network. In this way, if a node decides to run the **Share** operation, it writes its nodes on the corresponding registry of each of its neighbours. If previously shared trees are already in the registry, they are overwritten. Similarly, when a node runs the **Get** operation it reads its own registries, where the neighbours may have put their shared trees previously. In this way, it is guaranteed that each time a node gets trees from the neighbours, they are always the latest shared ones.

3.2.3 Key features

This algorithm satisfies the given requirements and possesses some advantages:

- The interaction between the nodes is based on the communication of trees, which are weak learners trained only on a subset of the nodes' dataset and on a subset of its features. In particular, **no data transmission** is necessary.
- The algorithm is based on the **simple structure** of the RF, and each of its components are modular. In particular, it is easy to redesign the implementation of one of the key operations without affecting the other ones.
- An explicit **memory bound** is implemented by limiting the maximal depth of each tree and the maximal number of trees in a RF.
- The communication is **fully asynchronous**, since each node can run the operations **Fit**, **Share**, **Get** at any time, independently of the other nodes. In particular, no central orchestration is required.
- The architecture supports **time-varying networks**, since the list of neighbours is accessed only at the time when the **Share** or **Get** operations are executed.
- Each node is affected only by **local changes** in the network topology.

The method is **robust** to other nodes' connection issues or misbehaviours, since there is no need to wait for other nodes to be ready before proceeding.

3.2.4 Additional properties

In addition to the core properties discussed in the previous section, the method has additional advantages, that will be explored in the third and final version of this deliverable:

- The method makes it possible to implement the collaboration between nodes which have access to **heterogeneous features**. Indeed, RF can be evaluated on datasets with missing data quite effectively. In particular, if two nodes have access to two different sets of features, the trees trained on one node can still be executed on the data possessed by the other node, provided that an intersection between the two sets of features exists. This extension would require both to implement data-integrity guarantees to make sure that different features are uniquely identified, and a more in-depth testing of the algorithm to understand its behaviour in this regime. When this analysis is completed, it will be possible to create a federation between nodes which have access to datasets described by different features.
- The nature of RFs makes it possible to obtain **explainability** results. In particular, each tree can be easily analysed to understand its learned decision path, and as a consequence RFs can sort the data features according to their importance in producing the prediction. Since each node can identify the source node of each of the trees in its RF, this capability will provide information on which features are favoured by each node.

3.3 Preliminary evaluation of the algorithm

This section gives a detailed account of the tests of the federated learning algorithm we have run on the dataset described in Section 3.1.1. Before presenting and discussing the actual results, we provide specifications of the communication network, of the preparation of the dataset, and of the setup of the algorithm's hyperparameters.

3.3.1 Communication networks

We consider ten nodes organized according to three different communication networks. We remark that, although the algorithm supports time-varying networks, we stick to a simpler static-network scenario for these preliminary testing.

The networks are a fully disconnected one (D), a pairwise connected network (P), and fully connected one (C) (see Figure 2). These three networks model an increasing level of connectivity, from non interacting nodes in D, to locally connected nodes in P, to fully connected nodes in C.

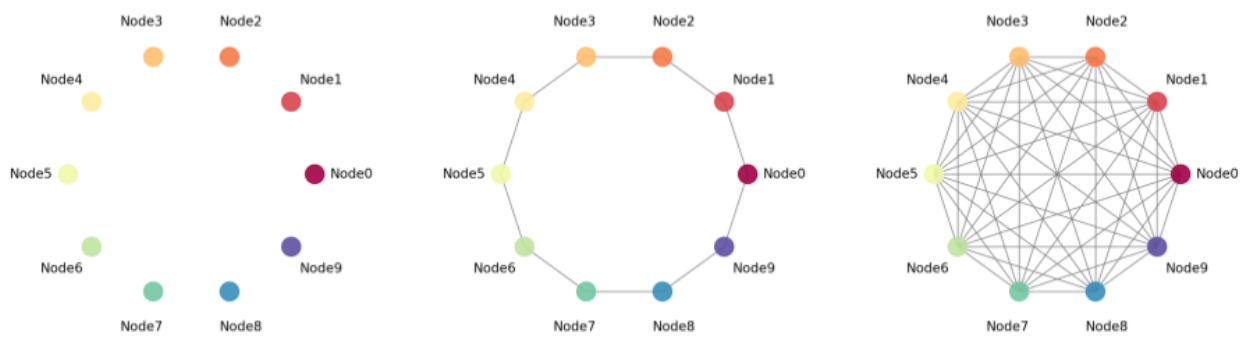


Figure 2: Communication networks for the federated learning use case.

3.3.2 Preparation of the dataset

To assign a set of data to each node, we split the dataset described in Section 3.1.1 in an unbalanced manner, in order to simulate the presence of nodes owning data of different quality. This is done by non-uniform randomized sampling of the positive and negative classes, in such a way that each transaction is assigned to a unique node. The statistics of the resulting set of ten datasets is summarized in Table 3.

For testing purposes only, we also create a shared test set that is used to assess the performance of the algorithm in an unbiased manner, using the metrics defined in Section 3.1.2. The set is obtained by collecting a random sample of the 10% of each of the ten datasets.

We remark that the existence of such a centralized test set is not required by the actual architecture, but just used here to measure the performances.

Table 3: Distributed dataset for the federated learning use case on fraud detection.

| ID | Samples | Frauds | Frauds' ratio |
|-------|---------|--------|---------------|
| Node0 | 50090 | 56 | 0.0011 |
| Node1 | 25384 | 12 | 0.0005 |
| Node2 | 429 | 106 | 0.2471 |
| Node3 | 53537 | 38 | 0.0007 |
| Node4 | 10 | 10 | 1 |
| Node5 | 27627 | 111 | 0.004 |
| Node6 | 30148 | 53 | 0.0018 |
| Node7 | 31347 | 37 | 0.0012 |
| Node8 | 60988 | 49 | 0.0008 |

| | | | |
|-----------------|--------------|-----------|---------------|
| Node9 | 16221 | 20 | 0.0012 |
| Test set | 29581 | 37 | 0.0012 |

3.3.3 Hyperparameters setup

The algorithm is flexibly parametrized by several parameters that are set to specific values in these experiments. Their name, role, and value are defined in Table 4.

We remark that at this stage there has been no particular efforts to fine tune these parameters, since our interest is in obtaining a first insight into the effectiveness of the method. For this reason, they have been set either to default values (`sample_size` and `max_depth`), or set to values that produce a relatively small model that is fast to train and test. The optimization of these parameters, and a thorough analysis of their impact on the outcomes of the algorithms, will be the object of further research and it will be documented in the final deliverable of task T4.5, namely D4.15 “Encrypted Data Querying and Personal Data Market – II”.

Table 4: Name, role, and value of the parameters used in our experiments.

| Name | Description | Value |
|-----------------------------|---|-------|
| <code>n_estimators</code> | Number of new trees that are trained at each execution of the Fit operation. | 5 |
| <code>n_rep</code> | Number of repetitions of the sampling in the bootstrap aggregation used in the Rank operation. | 10 |
| <code>sample_size</code> | Relative size of the validation set in the bootstrap aggregation used in the Rank operation. | 0.7 |
| <code>max_depth</code> | Maximal depth of each trained tree. | 10 |
| <code>max_estimators</code> | Maximal number of trees that are stored in the RF of each node. | 25 |
| <code>n_share</code> | Number of trees that are sent to the neighbours via the Share operation. | 5 |

3.3.4 Accuracy results

The accuracy values (according to the different measures of Section 3.1.2) reached with the three network configurations are depicted in Figure 3.

It is evident that for most nodes, moving from an isolated learning strategy to a federated one is largely beneficial in terms of Prec, Rec, and BAcc, even if for nodes that already performed well, a slightly

decreased performance can be observed. This is likely due to the stochastic nature of the learning process and will be addressed in the next and final version of this deliverable. In this regard, we stress in particular that the modular nature of the algorithm will allow us to test different approaches and clearly assess their performances. Nodes with poor performances in the disconnected case (Node 1, Node 2, and Node 4), on the other hand, show the largest improvement in moving to the federated approach.

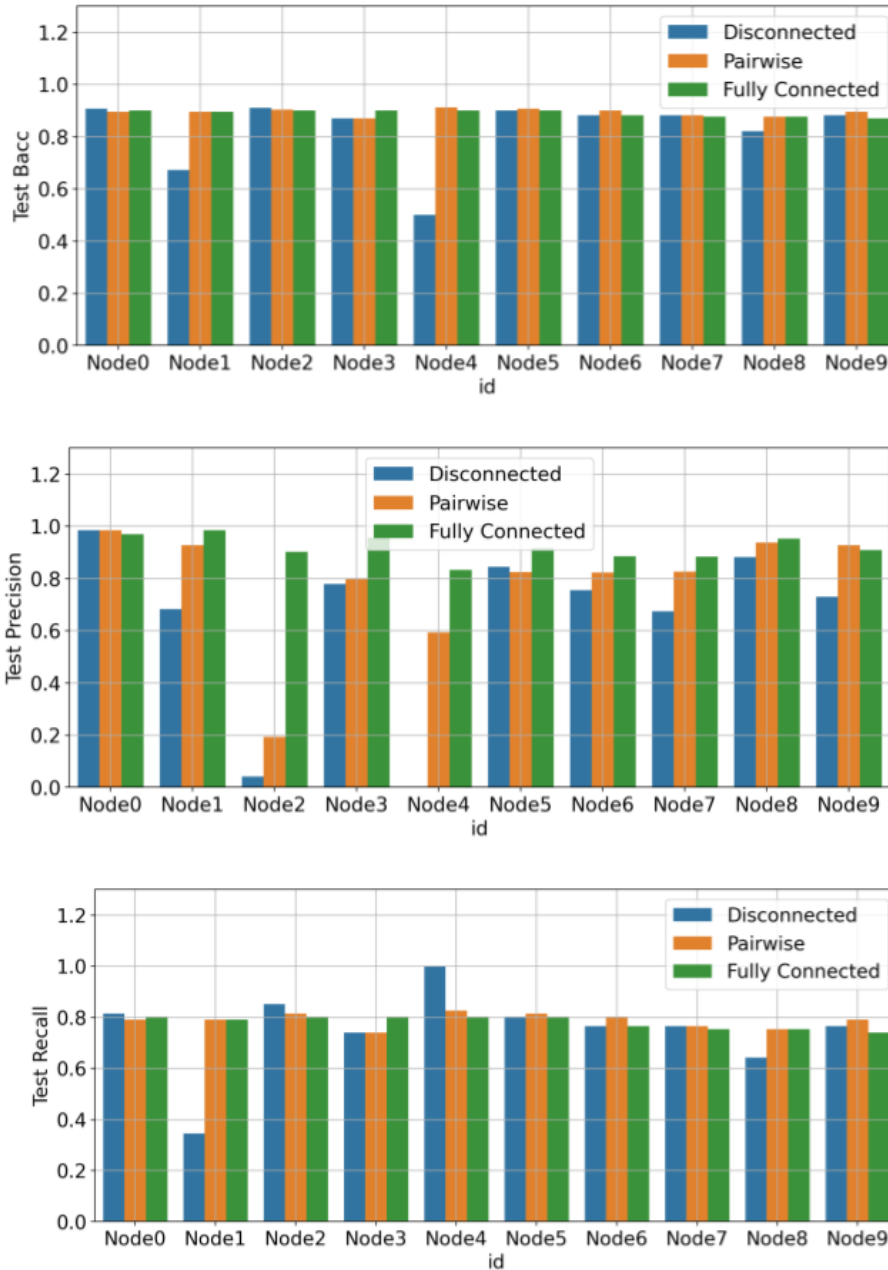


Figure 3: Accuracy results for the fraud detection use case.

3.3.5 Mixing of the estimators

It is interesting to note now how the network connection influences the propagation of the shared estimators among the different nodes. To this end, Figure 4 shows the final status of the RF of each node, and in particular the distribution of the trees within each RF according to the source node. In the first case (disconnected nodes) clearly each node can only use its own estimators, while a non-trivial mixing is visible

in the other two cases. In the loop-connection case, one can observe that each node uses mostly its own estimators, but with contributions from neighbouring nodes, also beyond direct connections. This fact demonstrates that estimators are shared also after the first-order connections. In the fully-connected case, on the other hand, a mode uniform mixing is observed.

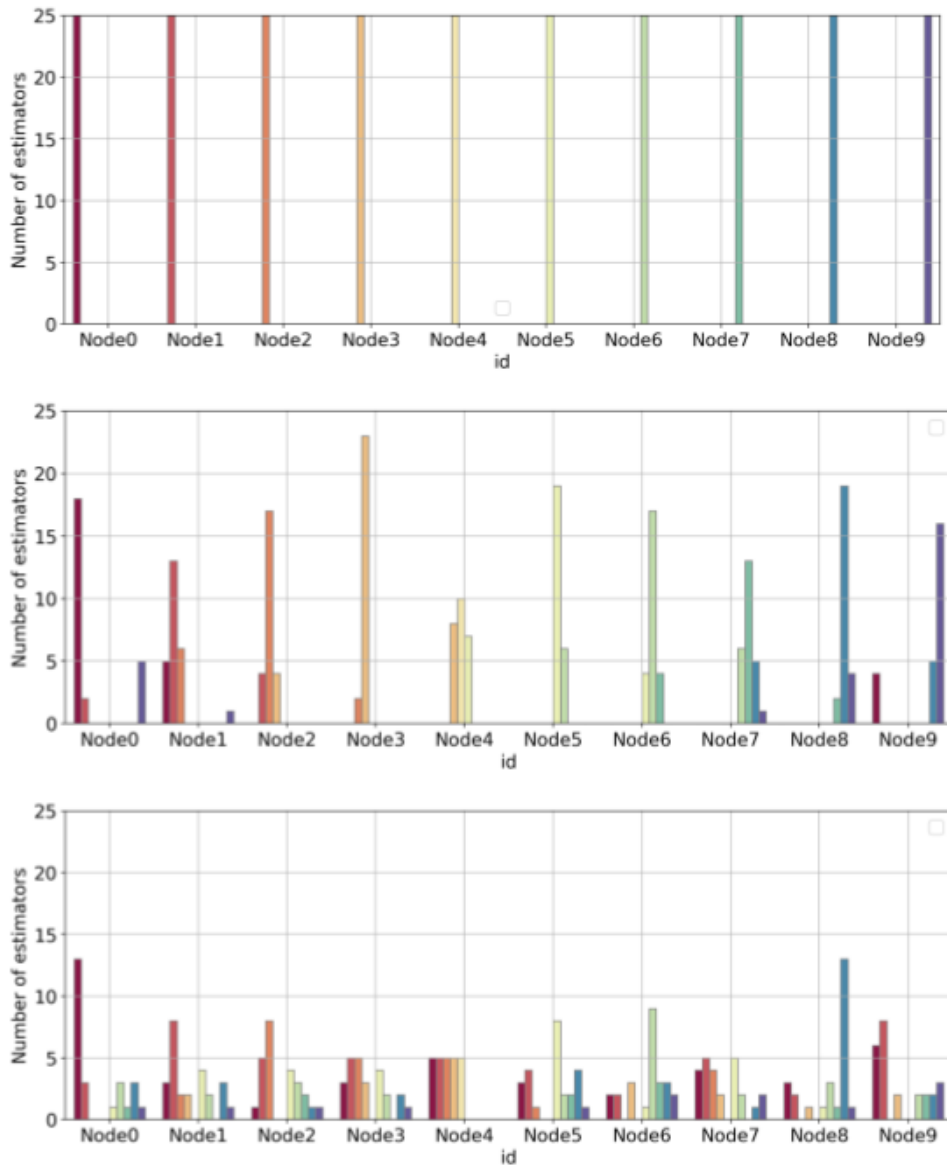


Figure 4: Distribution of the trees within each of the RFs trained on local nodes.

4 Blockchain framework

In the case of federated learning as described in detail in Section 3, it is necessary for an organization to trust another organization with the local execution of the ML algorithm on the data residing in the premises of the latter. Since the computation is defined by one party (the algorithm's owner) and the data is owned by another party, there exists an inherent trust problem. Today, the most common solution is to rely on a third party that is trusted by both organizations (a "trusted third party"), to perform the computation and share the outcomes. This solution is inherently more complex, as data needs to be transferred outside of the owning organization, more costly (due to the transfers), and requires the two organizations to put their trust in a third entity. The proposed approach in Vulcan allows delegating the computation over sensitive data to the data owner, while achieving the following key features:

- The computation workload is portable so that it is possible to deploy 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.

The execution framework in Vulcan was originally developed to address the domain of problems where one party has some sensitive data, and another party is interested in the result of a computation on that data. We extend the current framework to the federated learning scenarios where the same problem exists when the computation is decentralized and there is no inherent trust between the parties.

In this section we introduce a secure execution framework built on top of a blockchain (in our case Hyperledger Fabric) providing a verifiable privacy-preserving computation environment for federated learning scenarios.

4.1 Blockchain architecture for federated learning

Figure 5 depicts the different building blocks and the flows comprising our proposed framework. The chaincodes shown in the figure are the implementation of the logical components shown in Figure 1: the *image and execution record*, the *model*, and the *learning process orchestration* chaincodes implement the *federated learning artifacts store* component, while the *image and execution record* implements the *secured execution* component, and the *tokens* chaincode implement the *tokenization* component.

Figure 5 also shows some additional external artifacts related to the complete business flow of creating a ML algorithm image, executing this image securely on learning nodes, and sharing the insights: (i) the *container registry*, where the actual images of the ML algorithms are stored, (ii) the *Vulcan libraries* supporting the creation of computational tasks on the learning node from the ML algorithm image, (iii) the *outcome artifacts* produced by the learning process (the model) which is either stored in the ledger (using the *model* chaincode) or in an external object store (if the model size is too large to be stored in the ledger).

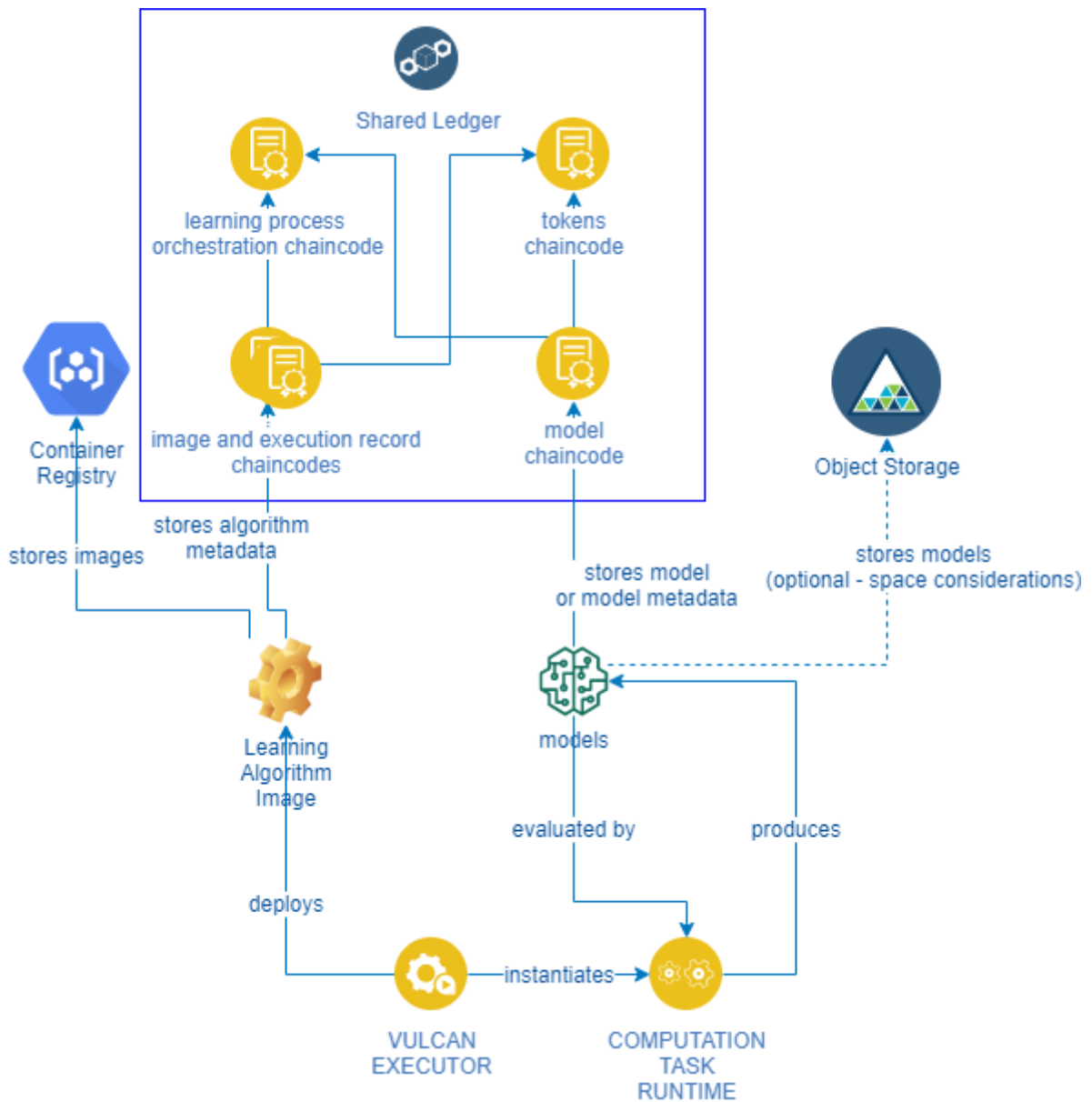


Figure 5: Blockchain framework’s building blocks and flows.

The following flow describes the interactions among the components. The image for the ML algorithm, intended to be run as a particular instance of a federated learning process, is stored in the *container registry* (which can be either shared or a private repository). The metadata describing the ML algorithm image along with additional information required for the auditing and execution of the federated learning process, such as access rules depicting the consortium of nodes participating in the process and therefore who are entitled to access the relevant artifacts, are stored on the blockchain ledger using the *image and execution record chaincode* (Section 4.3.1).

During the instantiation of the execution phase, the ML algorithm image is pulled from the registry and instantiated as computation task runtime on each learning node using the Vulcan execution framework, as described in Section 4.2. The task is instantiated with the ML algorithm runtime, the parameters for the run and the initial state model. The metadata representing the particular learning task (the algorithm image to run and the consortium of nodes to run it) is stored on chain using the *learning process orchestration chaincode* as described in Section 4.3.2.

During the algorithm execution phase, the learning node reads the relevant models from previous rounds created by itself and by the other learning nodes, it runs the algorithm image on the relevant inputs (the

models from previous rounds, the input parameters, and the organizational datasets), and it publishes the resulting model on the ledger using the *model* chaincode (Section 4.3.3) Once the learning process is completed, the status of the learning task on chain is updated to *completed* and the final ML model can be shared with other organizations as described in the data marketplace and trading scenarios (Section 4.5).

To support such data or insights' sharing and trading scenarios, the *tokens* chaincode (Section 4.5.2) is invoked once the trading process on the asset to share (completed model of a particular learning instance and its insights) is finalized between the selling organization and the buying organization. At the same time, the acquiring organization is granted access so it can use the model chaincode to read the model.

The framework aims to provide auditing and verifiability capabilities for different scenarios and use cases, from managing the ML algorithm image lifecycle (e.g., publishing a new algorithm image and usage of the algorithm image in different learning processes), to execution verification and auditing using Vulcan framework (ensuring, for example, that the correct algorithm image was used in each execution of computation task), to model updates (allowing to answer questions, such as, which models were created at the end of each run, or which models a particular organization published for a particular learning task), and finally to model usage (which models were acquired by which external organization). The auditing capabilities are described in detail in Section 4.4.

4.2 Secured execution environment

The Vulcan framework enables delegation of a computation over sensitive data to the entity that is authorized to access the data, while establishing trust of the rest of the stakeholders in the computation result. This is achieved via implementation of the following core characteristics:

2. The computation workload (i.e., the ML algorithm image) is portable so that it is possible to deploy in the environment of the party that has access to the sensitive data.
3. Integrity of the computation workload is verifiable, i.e. computation stakeholders have guarantees that the actual computation was performed on the respective data.
4. Provenance over the input data, the output of the computation, and the computation logic are tracked.

We propose to 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 would 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 workload's metadata on blockchain. 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. In case of artifacts (ML algorithm images) that are stored in systems external to blockchain (e.g., docker image repositories), their metadata will be stored on blockchain (via the image chaincode in Figure 5), specifically an identifier of the respective artifact in the external repository and a cryptographic fingerprint (e.g., a hash value) that can be used to verify the integrity of the artifact. The certified computation version (i.e., the execution record) is determined by the consensus of the participating parties in the blockchain network. Its life cycle is controlled and automated with the smart contract mechanism provided by the underlying blockchain platform and implemented as a specific smart contract code (i.e., the image and execution record chaincode).

4.3 Federated learning artifacts store and secured execution components

This section describes the chaincodes and data models required for supporting the process of federated learning. It contains the chaincodes for management of ML algorithm’s image metadata (*image and execution record chaincode*), learning process metadata (*learning process orchestrating chaincode*), and ML models’ metadata (*model chaincode*) as well as metadata of execution tasks (*image and execution record chaincode*). The *tokens* chaincode is discussed in 4.5.2.

4.3.1 Image and execution record data model and chaincodes

As aforementioned, the image and execution record chaincode provides the functionalities for storing and retrieving the ML algorithm’s image metadata. This chaincode also provides queries helpful in determining the provenance of the image, e.g., who is the creating organization and when the image was created.

Figure 6 depicts the process of registration of the image metadata using the Vulcan framework. Once the learning image is created, it is published to a container registry, while the image metadata is published via *Vulcan API* and stored on the ledger using the image and execution record chaincode. During the learning process initialization, the Vulcan executor pulls the image from the container registry. Then, it uses this image to build and run a computational task (algorithm image container) on the learning nodes, which access the private datastore to read the datasets used to build the ML models. The execution task metadata is also stored on the ledger through the *Vulcan API* using the image and execution record chaincode. Once the outcome of each learning round is completed at the node, it publishes the insights on the chain using the model chaincode.

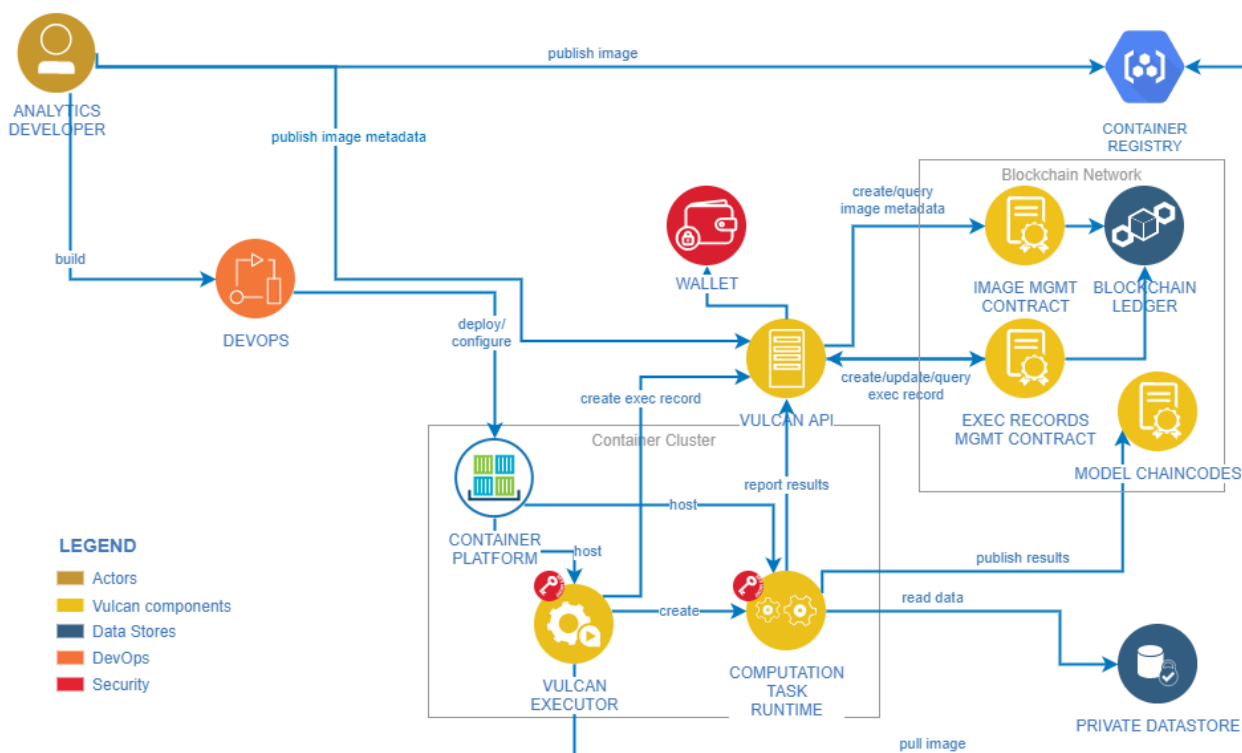


Figure 6: Secured execution environment.

We henceforth present the schema of the two main data entities we manage in the image and execution record chaincode: image and algorithm metadata (Table 5) followed by the chaincodes functions' descriptions.

Table 5: Image data types.

| Image | | |
|---------------------------------------|-------------------|---|
| Represent a learning algorithm image. | | |
| Field | Type | Description |
| imageTag | String | Image tag. |
| imageId | String | Image Identifier - a digest, which contains an SHA256 hash of the image's JSON configuration object. |
| creationTimestamp | Date | Image creation timestamp. |
| creatorOrg | String | Identifier of the organization which is the image creator. |
| status | String | Status of the image approval cycle (NEW/PROCESSING/ACCEPTED). |
| algorithmMetadata | AlgorithmMetadata | Object describing the learning algorithm details (name/id/version/framework/type), etc. |

| AlgorithmMetadata | | |
|--|--------|-----------------------|
| Represent a learning algorithm metadata. | | |
| Field | Type | Description |
| algorithmID | String | Algorithm identifier. |
| algorithmName | String | Algorithm name. |
| version | String | Version number. |

Chaincode functions: CreateImage, DeleteImage, ReadImage. No update image function is provided as a new version of the algorithm would represent a new image (with different algorithm metadata)

- `CreateImage (imageId string, imageData JSON)`: The specific function is performing the necessary actions in order to create a new image metadata entry into the ledger by updating the world state with a new record and appending new blocks at the end of the ledger. The function receives the data containing the image information.
- `ReadImage (imageId string)`: This specific function reads the required image metadata entry from world state and returns the information in JSON format to the invoking client.
- `DeleteImage (imageId string)`: It deletes existing images. Operation allowed only for image creator org.
- `getImagesPerAlgorithm (algorithmId)`: It queries the images state to return all the images of a particular algorithm (different versions might imply multiple images).

Regarding the image and execution record chaincode, the following steps take place at the execution time to authenticate the certified image and deploy the correct workload for the requested computation:

1. Query the shared ledger to fetch the required computation asset record. As mentioned above, that asset may consist of a single image’s metadata or a composite record incorporating several components.
2. Load the particular image artifacts from external repositories (if needed, e.g. a docker repository) and verify their authenticity using the fingerprint from the blockchain record (in case of docker images the verification is done as part of the docker pull mechanism, using the docker ID as the fingerprint).
3. Generate a key pair for the execution – a signing key (private), and a verification key (public).
4. Create a new execution record asset on blockchain, with the following fields:
 - a. Execution identifier;
 - b. Computation workload identifier;
 - c. Verification key.
5. Deploy the computation workload in the runtime environment (the data owner’s controlled environment), including instantiating containers and corresponding orchestration and choreography components. The signing key is incorporated in the container(s) and is used to sign the execution results.
6. Update the execution record in the blockchain with the execution result (or its digest) signed with the private signing key. The corresponding transaction proposal is verified by the smart contract by the endorsing peers.
7. When the computation is completed, destroy the containers and the signing key.

The provenance information for each execution is stored in the corresponding execution record on blockchain. It’s composed of the following attributes (Table 6).

Table 6: Execution record entity schema.

| Execution record | | |
|---|---------------|---|
| Represent an execution record of the ML image task run. | | |
| Field | Type | Description |
| executionIdentifier | String | Identifies the execution (the computation task instance). |
| computationWorkloadIdentifier | String | Identifies the “image”. |
| verificationKey | String | The public key used for verification of the results. |
| executionResult (or result digest/hash) | String (JSON) | Serialized result object, or its hash (if the result is stored off-chain). |
| executionResultSignature | String | Result signature (signed with the private key). |
| sourceDataFingerprints | String (JSON) | Optional. Signed digest of data items that the computation is performed on. |

In case of a dispute, the information in the execution record along with the computation workload assets (such as images) can be used to re-run the computation and verify the result.

4.3.2 Federated learning process orchestration chaincode

Once a consortium of nodes participating in the learning process is established, and the ML algorithm is published on chain as described earlier, the learning process instance is registered on chain, and the learning process is initiated. The process is described in Figure 7.

Each of the learning nodes pulls the learning algorithm image from the image repository, instantiates it with an initial state (e.g., parameters and initial model), instantiates the Vulcan computation task container, and runs the algorithm on its private datasets, until it produces an output. The output model is stored on chain as explained in the next section. The process defined above is identified by an appropriate data entity stored on chain using the appropriate chaincode; this data entity describes, among others, the consortium running the processes, the algorithm being executed, and the current state of the process (i.e., the *round*). The consortium definition basically defines the organizational access rules for all the learning process produced artifacts. This means that all intermediate and complete models derived as outcome of the learning process contain a list of organizations which are allowed to access those models, and this list is derived from the consortium definition.

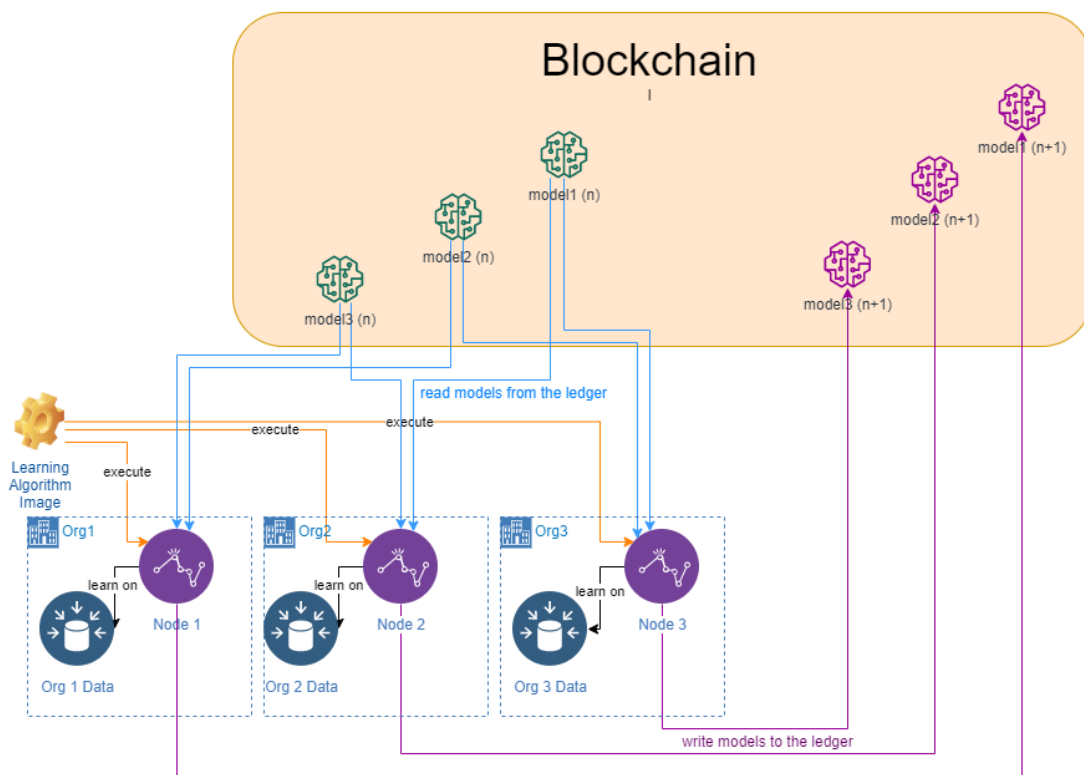


Figure 7: Federated learning process with blockchain.

We henceforth present the schema of the learning process instance entity (see Table 7) followed by the chaincodes functions' descriptions.

Table 7: Learning process instance entity schema.

| Learning process instance Represent a learning process instance. | | |
|--|-----------------|---|
| Field | Type | Description |
| processId | String | Identifier of the learning process instance. |
| imageId | String | The identifier of the algorithm learning image used by this learning process. |
| consortiumDefinition | Array of String | A list of nodes' identifiers (orgs) participating in the federated learning. |
| processStatus | String | An enumeration value identifying the current status of the learning process (INITIALIZING, ACCEPTED, INPROCESS, COMPLETED). |

Chaincode functions: (i) CreateLearningProcesses, (ii) UpdateLearningProcess, (iii) GetLearningProcessInformation, and (iv) GetProcessesPerImage.

- CreateLearningProcesses (processId string, processData JSON): This specific function is performing the necessary actions in order to create a new learning process' metadata entry into the ledger by updating the world state with a new record and appending new blocks at the end of the ledger. The function receives the data containing the process information.
- UpdateLearningProcesses (processId string, updatedProcessesData JSON): This specific function updates existing process entities on the ledger with new information. For example, it updates them with the new round of learning execution, or with the state of the processes.
- GetLearningProcessInformation (processId string): It retrieves information about the current state of the learning process.
- GetProcessesPerImage (imageId string): It retrieves all processes for a particular algorithm image. This, combined with query allowing to retrieve all images per a particular algorithm, allows to search for learning processes running specific algorithm's versions and to determine their state.

4.3.3 ML models' data and chaincode

As described in previous sections, models from previous rounds, consumed by the learning nodes as inputs for next learning rounds, and the outcomes of the run are published on chain as input for the next rounds or as a completed model. This is done by utilizing the *model chaincode*. This section depicts the data entities (see Table 8) and the chaincode functions providing such functionality.

Table 8: ML model instance and input entities' schema

| Model instance Represent a model instance. | | |
|--|-------------|--|
| Field | Type | Description |
| modelId | String | Identifier of the model instance. Created by combining the |

| | | |
|---------------------|-----------------|--|
| | | information from learningProcessesId + creatorNodeId + roundIdentifier. |
| imageId | String | The identifier of the algorithm learning image used by this learning process. |
| learningProcessesId | String | The identifier of the learning process instance during which this model was created. |
| creatorNodeId | String | Identifier of the learning node which is the model creator. |
| creatorOrg | String | Identifier of the organization which is the model creator. |
| creationTimestamp | Date | Model creation timestamp. |
| model | Object | The model binary. |
| modelInputs | Object | Inputs to the model: parameters, previous models' identifiers used as inputs, datasets' identifiers. |
| modelCompletion | Boolean | A value indicating whether the model represents the final step in the learning process (if completion criteria is reached) or an intermediate step. |
| modelAccessRules | Array of String | List of valid organizations' identifiers allowed to access the model (READ access). Those include the consortium organizations participating in the learning process, and external organizations which have been granted explicit access to the model. |

Chaincode functions: (i) CreateModelInstance, (ii) DeleteModelInstance, (iii) ReadModelInstance, (iv) GetModelsForAlgorithm, (v) GetModelsForAlgorithmAndOrg, (vi) GrantAccess, (vii) GetModelAccessHistory, (viii) GetOrganizationAccessHistory, (ix) GetLatestModelForNodeAndAlgorithm.

- CreateModelInstance (modelId string ,modelData JSON): This function stores on the ledger the model metadata.
- ReadModelInstance (modelId string): This function reads the model metadata entry from world state and returns the information in JSON format to the invoking client. The invoking user organization affiliation (provided by the transaction context as each user invoking the transaction signs it with his/her credentials which include organization affiliation) is read and checked versus the access rules specified in the model metadata entry. Only if the user's organization appears in the list the user is granted read access.
- DeleteModelInstance (modelId string): It deletes the specified model instance.

- `GetModelsForAlgorithm (imageId string)`: It retrieves all the models created by running the specific ML algorithm image.
- `GetModelsForAlgorithmAndOrg (imageId string, orgName string)`: It retrieves all models for a particular ML algorithm image which were created by the specified organization.
- `GetLatestModelForNodeAndAlgorithm (nodeId string, imageId string)`: It retrieves the last model created by a particular learning node for the particular ML algorithm.
- `GrantAccess (modelId string, requestingOrg string)`: It adds the requesting organization as an organization with READ access to the model. Useful for auditing purposes and as part of granting access to completed models in a trading process.
- `GetModelAccessHistory (modelId string)`: It returns the history of READ access to the specified completed model by an external organization (not by one of the organizations in the course of learning processes).
- `GetOrganizationAccessHistory (organizationId string)`: It returns a list of all completed models which were accessed by a particular external organization.

4.4 Artifacts Usage Audit

One of the reasons behind storing metadata of the federated learning process (including data on the ML algorithms being used and their execution, datasets' versions used for model training, and provenance of the model artifacts) is to provide provenance into the artifacts' creation and usage. This is useful for audit trails and data governance along many other use cases. Understanding the provenance of deployed models, their origin (which learning algorithm and which version of the learning algorithm they are built from) are important when we would like to know, for example, which models were derived from a particular dataset (for example, allowing us to identify models which might have been built on incorrect data). Proven, verifiable, and immutable audit trail of execution tasks producing such models can help establish without doubt that the models are derived from the desired ML algorithm, the desired version of that algorithm, and no mistakes were made in the process.

One of the built-in core functionalities of blockchain platforms is an immutable chain of blocks of transaction, establishing verifiable and transparent history of updates for each artifact stored on the chain. In this section, we demonstrate how we leverage this functionality to provide a solution for provenance and auditing trails of different artifacts and stages of the federated learning process.

The functionalities we detail in subsequent subsections correspond to the Artifacts Usage Audit logical component depicted in Figure 1.

4.4.1 Algorithm image auditing

Provenance of the algorithm images is an important step in validating the outcome of the federated learning execution. Knowing attributes such as the origin of the algorithm (the creator) and the version of the algorithm used can help in establishing the provenance of the whole learning process and understanding and ensuring the correctness of the results.

Section 4.3.1 describes the chaincode for storing and retrieving algorithm's image metadata on the chain. As part of the provided chaincode functionality, in particular, two queries ensure the provenance of the algorithm image:

- `ReadImage` - it allows to get the image metadata, including the creator organization and creation timestamp.
- `GetImagesPerAlgorithm` - It allows to get a list of all the metadata entries of all the images of a particular ML algorithm (for example, different versions of the same ML algorithm will result in

a list of different images) along with information on who the creator of the image is and the creation timestamp.

4.4.2 Execution auditing

As described in Section 4.3.1, the execution record stores the information about each computational task which was run as a part of the learning process. Such execution records provide the provenance and the verifiability of the computational task which was executed, and the outcomes of this task, and provide the possibility to re-run the task and verify the results.

4.4.3 Model lifecycle auditing

As described in Section 4.3.3, the model chaincode is responsible for providing functionalities for model metadata lifecycle management on the chain. Part of these functionalities is to provide different views into the model history, whether it is a model history created by a particular organization or fetching all models of a certain round of a particular learning process.

The functions providing such provenance in the model chaincode are:

- `GetModelsForAlgorithm` - retrieving all the models logged on the chain and created by running the specific algorithm image.
- `GetModelsForAlgorithmAndOrg` - retrieving all models for a particular algorithm image which were created by the specified organization.

4.4.4 Model usage trail – for insights' marketplace

One of the purposes and functionalities of our proposed solution is the ability to trade the completed ML models on a data marketplace. Both for this purpose, and for the purpose of the model usage auditing, an ability to record usage of “ready for use” completed ML models is required. The *model* chaincode provides functions allowing to fetch the history of access to a particular model and/or history of accesses for a particular organization.

- `GetModelAccessHistory` - It returns the history of READ access to the specified completed ML model by an external organization. Such access would first require granting of permission for the access to the particular model from a specific external organization, which is done as part of the data/insights' trading processes.
- `GetOrganizationAccessHistory` - It returns the list of all completed ML models which were accessed by a particular external organization.

4.5 ML insights' marketplace and trading

Data marketplaces for ML models are an emerging trend. They provide the opportunity to decentralize AI 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.

Such a marketplace (see Figure 8) should provide all the functionalities required to publish the model metadata (including the algorithm, the number of runs, the participating companies, and the pricing information) to a metadata catalogue component (i.e., *Broker*), (i) to search for such published models given different search parameters (via the *Broker*), (ii) to provide the model publisher with the ability to use different pricing models for trading its models or insights, (iii) to issue bids and come to an agreement on a price through a negotiation process (via the *Trading framework*), and, finally, (iv) to connect to the model repository (via the *Model data store* implemented as part of the Federated Learning Artifacts Store logical component described in Figure 1) to grant the acquiring company access to the particular model and to the tokenization chaincode, via the *Payment service*, to transfer tokens as payment.

The *Insights Marketplace* will make use of the model chaincode to fetch information regarding a particular ML model and grant access to the model to an external company (at the end of the trading process). The *Insights Marketplace* component will also utilize the tokens’ chaincode, described in Section 4.5.2, to provide functionalities like transferring tokens between accounts, checking the balance of accounts, and allowing a third party to transfer tokens on behalf of an account owner.

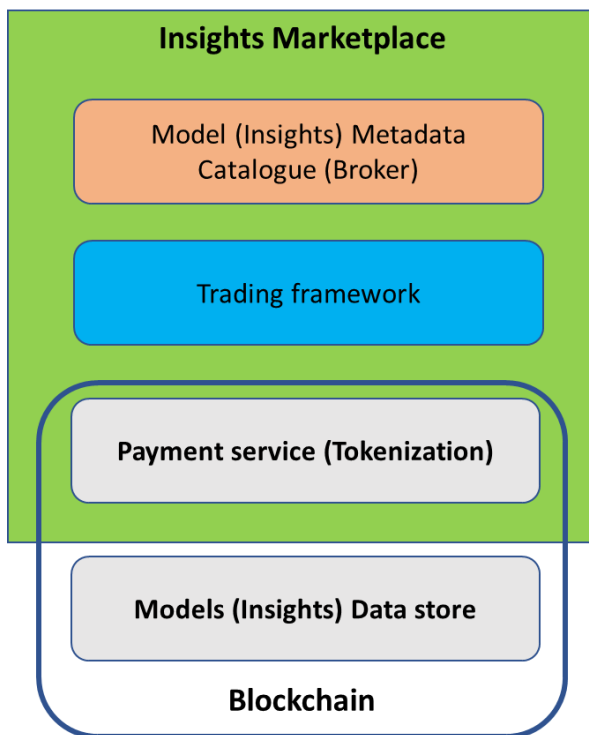


Figure 8: Logical components of the ML models’ insights marketplace.

We next summarize two trading scenarios we propose to leverage the developed framework to support models’ insights’ trading which might be of interest in such a marketplace and provide a brief introduction to the tokens’ chaincode implementation that realizes the popular ERC20 standard for tokenization (for more details the reader should refer to D4.10 “Blockchain Tokenization and Smart Contracts – I”).

4.5.1 ML models’ insights’ trading use cases

Our framework supports two possible scenarios for training ML models’ insights: (1) an external organization intends to buy the insights obtained by a completed learning model (see Section 4.5.1.1), and (2) an external organization orders the execution of a learning process (see Section 4.5.1.2).

Transactions’ history provides the full provenance of all the reads to the ML models and therefore serve as basis for auditing purposes.

4.5.1.1 Scenario 1: External organization buys a trained ML model

The process steps of this scenario are shown in Figure 9 and they are organized as follows.

Step 1: The selling organization or a network of organizations (the consortium which participated in the completed ML model generation) submits information regarding the insights they are willing to trade to the insights’ catalogue (also called “the broker”).

Step 2: An external organization searches the metadata broker for suitable models matching its requirements (which can be the organizations participating in the model creation, the type of algorithm

used, and the creation time among the model metadata), and assuming a model answering the searching organization’s criteria is found, the model metadata is returned to the invoker organization.

Step 3: The invoker organization can submit a bid/request to buy the models’ insights to the owning company or companies. This step can iterate during the negotiation phase until an agreement is reached.

Step 4 and Step 5: Once the bid/buy request is accepted, then two operations should be performed as one atomic transaction in the blockchain. The trading component invokes the model chaincode to grant access to the model to the buying organization (this organization is added to the list of access allowed organizations in the model metadata). Then, the model chaincode performs the following actions as part of the granting transaction:

- o It checks the requesting organization account to ensure the account holds at least the amount of tokens required for the payment of the model’s insights. This is a pre-condition for enabling access.
- o It transfers the specified amount of tokens from the requesting organization account to the owner/owners’ organization account (via the tokenization chaincode).
- o Only when this operation is completed, the model access rules are updated with a rule allowing READ access to the buying organization.

Step 6: The buying organization invokes the model chaincode to read the model metadata from the chain.

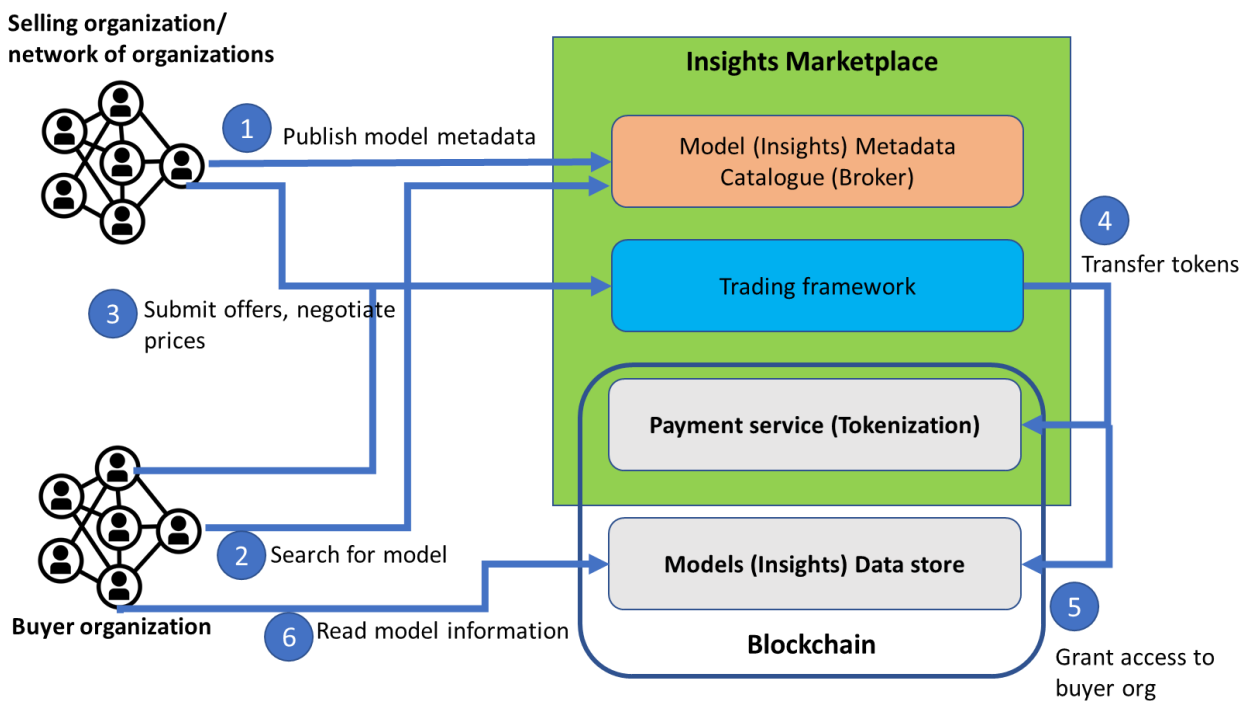


Figure 9: Acquisition of a ML model’s insights’ process

4.5.1.2 Scenario 2: External organization orders the execution of a federated learning process

An external organization “orders” the execution of a learning process. The organization specifies the algorithm’s image to use and the consortium of organizations to execute the learning process with additional parameters (see Section 4.3.2), thus specifying details of the execution. Then, the consortium of learning nodes instantiates and executes the learning process. Once the completion criteria for the model is reached, the requesting organization receives notification on completion.

Since the requesting organization already has access permission to the completed model, it reads the completed model. During this step, as part of the read transaction, the model chaincode interacts with the token chaincode to:

- Check the requesting organization account to ensure the account holds at least the number of tokens required for the payment of the model. This is a pre-condition for granting access.
- Transfer the specified number of tokens from the requesting organization account to the owner/owners' organization account.

4.5.2 Tokenization

The deliverable D4.10 “Blockchain Tokenization and Smart Contracts – I” provides comprehensive information regarding the tokenization process. Specifically, Section 3 of D4.10 provides an extensive explanation on the tokens' chaincode implementing the ERC20 standard carried out in the scope of the INFINITECH project. Particularly, Section 3.1.2 of D4.10 specifies the APIs provided by the tokenization chaincode for checking the account balance, invoking tokens' transfer and receiving notifications on the completion of transfer processes.

Regarding the trading scenarios described above, the following ERC20 functions are in use:

- `BalanceOf (address)` - It returns the token balance of an address.
- `Transfer (address, amount)` - It transfers the amount of tokens: these tokens are taken from the balance of the address that executed the transaction.
- `Transfer event` - This event is triggered upon successful transfer (call to `transfer` or `transferFrom`), even for 0 value transfers.
- `Mint (amount)` - It initializes the overall accounts' balance with a specified number of tokens. Each organizational account participating in the data marketplace needs to “buy in” into this amount – transfer a particular amount of initial assets representing a particular number of tokens to the minting organization to be assigned this particular number of tokens to the buying-in organization account.

5 Conclusions and next steps

The current deliverable has been developed around the two main objectives that have been outlined in Section 1.1. As the first objective, this deliverable has provided a more detailed description of the design, research, and implementation of the *Data Policy Framework*, the *Algorithm Runtime*, and the *Data Marketplace* components as well as a description of the *Data Usage Audit* component. In particular, we have introduced the novel *Insights Sharing and Provenance* conceptual module, jointly designed by IBM and FBK. More precisely, we have detailed the five components of this module, namely (i) *Identity Manager and User/Client Authentication*, (ii) *Federated Learning Artifacts Store*, (iii) *Artifacts Usage Audit*, (iv) *Secured Execution*, and (v) *Tokenization*. This module is built on top of Hyperledger Fabric, the blockchain platform selected and used by the INFINITECH project for the implementation of digital finance use cases.

The second objective has been pursued by designing, implementing, and evaluating the federated learning algorithms selected as the basis for the INFINITECH framework for securely sharing ML insights. More specifically, we have proposed an innovative approach based on Random Forests (RF), also providing the details of its architecture, and describing its results in a preliminary application to a fraud detection task.

As mentioned in the Introduction Section, the final objectives of the task T4.5 and of its three associated deliverables (D4.13, D4.14, and D4.15) are the design and implementation of a framework for querying encrypted data over the INFINITECH permissioned blockchain infrastructure and for running ML algorithms on them, as well as the creation of the foundation for a personal data market where individuals and organizations will be able to trade their data in exchange for tokens. To this end, the third and final deliverable of T4.5 will document the implementation of a minimum viable product (MVP) for the proposed framework as well as the addition of novel functionalities, such as a blockchain-based consent mechanism on top of the federated ML algorithms. In particular, there is a need for consent techniques: This is because the individuals and the organizations (i.e., banks, insurance companies) adopting the proposed blockchain-framework for running federated learning algorithms and sharing learning models' insights, must be capable of distinguishing and ultimately choosing the parties that they trust in order to share their insights. This additional consent mechanism will be developed by INNOV and will be integrated in the MVP in collaboration with IBM and FBK. In the scope of the MVP implementation, the solution prescribed in this deliverable will be also presented to relevant stakeholders (e.g., blockchain experts, financial organizations) as part of the INFINITECH series of stakeholders' workshops. In this way, we will also attempt to collect and incorporate stakeholders' feedback in the MVP implementation.

6 Appendix A: Literature

- [1] “Hyperledger Fabric – Hyperledger,” 2020. [Online]. Available: <https://www.hyperledger.org/use/fabric>. [Accessed 5-October-2020].
- [2] Bishop, C.M. (2006). Pattern recognition. *Machine learning* 128, no. 9.
- [3] Breiman, L. (2001). Random Forests. *Machine Learning*, 45 (1), pp. 5-32.
- [4] Gaur, N., Desrosiers, L., Ramakrishna, V., Novotny, P., Baset, S.A., and O'Dowd, A. (2018). *Hands-on Blockchain with Hyperledger: Building decentralized applications with Hyperledger Fabric and Composer*. Packt Publishing.
- [5] Jolliffe, I.T. (2002). *Principal Component Analysis*, second edition, New York: Springer-Verlag New York, Inc.
- [6] Kairouz, P., McMahan, H.B., Avenet, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R.G.L., El Rouayheb, S., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P.B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konečný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint, T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Raykova, M., Qi, H., Ramage, D., Raskar, R., Song, D., Song, W., Stich, S.U., Sun, Z., Suresh, A.T., Tramér, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F.X., Yu, H., and Zhao, S. (2021). Advances and open problems in federated learning. *Foundations and trends in machine learning*, 14 (1-2), pp. 1-210.
- [7] Mahendraratnam, N., Sorenson, C., Richardson, E., Daniel, G.W., Buelte, L., Westrich, K., Qian, J., Campbell, H., McClellan, M., and Dubois, R.W. (2019). Value-based arrangements may be more prevalent than assumed. *The American Journal of Managed Care*, 25(2), pp. 70--76.
- [8] Oehmichen, A., Jain, S., Gadotti, A., and de Montjoye, Y.-A. (2019). OPAL: High performance platform for large-scale privacy-preserving location data analytics. In *Proceedings of BigData 2019*: pp. 1332-1342.
- [9] Pinkas, B., and Lindell, Y. (2009). A proof of security of Yao's for two-party computation. *J Cryptol* 22, 161-188.
- [10] Richards, D., Rebeschini, P., and Rosasco, L. (2020). Decentralised learning with random features and distributed gradient descent. In *Proceedings of the International Conference on Machine Learning (PMLR 2020)*.
- [11] Rahimi, A., and Recht, B. (2008). Random features for large-scale kernel machines. In *Proceedings of Advances in Neural Information Processing Systems* 20.
- [12] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson, Estimating the Support of a High-Dimensional Distribution. *Neural Comput.* 13, 7, 1443–1471, 2001.
- [13] MetaCX for Healthcare. [Online]. Available: <https://metacx.com/healthcare>
- [14] Zyskind, G., Nathan, O., and Pentland, A. (2015). Decentralizing privacy: Using blockchain to protect personal data. In *Proceedings of IEEE Symposium on Security and Privacy Workshops*: 180-184.
- [15] Long G., Tan Y., Jiang J., Zhang C. (2020) Federated Learning for Open Banking. In: Yang Q., Fan L., Yu H. (eds) *Federated Learning. Lecture Notes in Computer Science*, vol 12500. Springer, Cham. https://doi.org/10.1007/978-3-030-63076-8_17.
- [16] “Credit Card Fraud Detection”, 2016. [Online]. Available: <https://www.kaggle.com/mlg-ulb/creditcardfraud>

[17] “Random Forest Regressor”, 2021 [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor>