applied sciences

MDPI

Article

Towards Blockchain-Based Federated Machine Learning: Smart Contract for Model Inference

Vaidotas Drungilas *, Evaldas Vaičiukynas, Mantas Jurgelaitis, Rita Butkienė and Lina Čeponienė

Department of Information Systems, Faculty of Informatics, Kaunas University of Technology, Studentų str. 50, LT-51368 Kaunas, Lithuania; evaldas.vaiciukynas@ktu.lt (E.V.); mantas.jurgelaitis@ktu.lt (M.J.); rita.butkiene@ktu.lt (R.B.); lina.ceponiene@ktu.lt (L.Č.)

* Correspondence: vaidotas.drungilas@ktu.lt

Abstract: Federated learning is a branch of machine learning where a shared model is created in a decentralized and privacy-preserving fashion, but existing approaches using blockchain are limited by tailored models. We consider the possibility to extend a set of supported models by introducing the oracle service and exploring the usability of blockchain-based architecture. The investigated architecture combines an oracle service with a Hyperledger Fabric chaincode. We compared two logistic regression implementations in Go language—a pure chaincode and an oracle service—at various data (2–32 k instances) and network (3–13 peers) sizes. Experiments were run to assess the performance of blockchain-based model inference using 2D synthetic and EEG eye state datasets for a supervised machine learning detection task. The benchmarking results showed that the impact on performance is acceptable with the median overhead of oracle service reaching 2–4%, depending on the dimensionality of the dataset. The overhead tends to diminish at large dataset sizes with the runtime depending on the network size linearly, where additional peers increased the runtime by 6.3 and 6.6 s for 2D and EEG datasets, respectively. Demonstrated negligible difference between implementations justifies the flexible choice of model in the blockchain-based federated learning and other machine learning applications.

Keywords: distributed ledger; oracle service; chaincode; machine learning; model validation; runtime benchmarking; system architectures; microservices

Citation: Drungilas, V.; Vaičiukynas, E.; Jurgelaitis, M.; Butkienė, R.; Čeponienė, L. Towards Blockchain-Based Federated Machine Learning: Smart Contract for Model Inference. *Appl. Sci.* **2021**, *11*, 1010. https://doi.org/10.3390/app11031010

Academic Editor: Donato Cascio
Received: 30 November 2020
Accepted: 20 January 2021
Published: 23 January 2021

1. Introduction

Federated learning (FL) is a distributed machine learning (ML) approach that facilitates model training on a decentralized network of individual data sources. Unlike traditional ML, which relies on centralized data computing to develop machine learning models, FL develops and uses a model across multiple decentralized devices [1]. Rather than gathering all data samples to one server, decentralized devices hold local data samples without exchanging them [2–4]. Despite the setting, it is advantageous if the generalization abilities of the ML model (or models) are evaluated on a separate validation data, which is agreed upon and available to all participants in the learning process. If some data gets uploaded later, after model creation and sharing, these data records, when used for model inference, could be considered as unseen or testing data. Both validation and testing settings correspond to the ML model inference.

In recent years, a few novel solutions emerged which attempt to decentralize the machine learning process using various techniques. Google uses cloud infrastructure to distribute the Gboard (Google keyboard) [1] model among phone devices to improve predictive typing. A single shared model is used and developed on the user's data, then changes are summarized and pushed as a model update to the cloud. A number of these updates are integrated and used to improve the existing shared model. The basic principles of this approach show that decentralization can be used in a manner which facilitates the development of a single shared ML model. While the Google approach is quite novel, it

supports a specific type of model and suffers from scalability since the training of a model requires a substantial number of users.

Another novel solution by Nvidia proposes building robust ML algorithms, which enable collaboration of different nodes in model training while preserving data privacy [5]. Nvidia implements a server-client approach, where a centralized server works as a manager/facilitator of participating clients. The infrastructure allows the developers to share their models and components and have control over the training process. Partial models are trained on clients, then the partial model weights are shared, and the model is updated based on weights and history of contributions. Even though the infrastructure supports the usage of different models, it still is dependent on a centralized node to aggregate weights and update the shared model, creating a bottleneck in terms of resources and computational power.

FL can be useful in several aspects, including load-balancing (to distribute computations in the machine learning task) and privacy preservation (to contribute to the machine learning task without revealing training data). To adapt infrastructure for several ML models and distribute the computing power without relying on a centralized node, several solutions using distributed ledger technology (DLT) [6] or blockchain [7–9], were developed. Even though FL can be implemented without DLT, the introduction of DLT or blockchain as a specific type of DLT can partially solve interoperability and heterogeneity technical limitations, which are present in the traditional FL approaches. Data heterogeneity with respect to multivariate distribution could be alleviated by introducing the possibility to donate some data for model validation purposes. Interoperability, in terms of compatibility among network nodes, could be solved by providing a common architecture for collaborative model sharing.

A solution combining machine learning and blockchain technologies is discussed in [10], where possible integrations between blockchain and machine learning for processes such as model training, validation, testing, deployment, and storage are analyzed. Blockchain infrastructure holds potential to improve transparency and trust in the machine learning process and resulting models by logging everything into a distributed ledger permanently. As the model is public, and the history is recorded, the blockchain stores not only the current model but all the evolutions, which can be traced back. Some blockchains also support smart contracts which could expand capabilities to develop an infrastructure for hosting heterogenous ML models, as well as to customize the workflow of some parts of the machine learning process. Smart contracts not only facilitate iterative development but also provide flexibility to implement custom FL solutions. Federated learning can benefit from blockchain infrastructure since decentralization is inherent in both. While ML and blockchain can be combined, the solutions still come with few key concerns that need to be addressed such as scalability, smart contract vulnerabilities, and complicated testing. Blockchain-based ML solutions could further be customized by integrating external ML libraries implemented using blockchain oracles [11–13]. Oracles act as decentralized web services, which extend the capabilities of blockchain smart contract implementations. Oracles introduce flexibility in blockchain-based ML implementations by enabling the usage of virtually any programming language, which does not need to be compatible with the language supported by the smart contract. This way the existing solutions for ML can be reused in blockchain while simultaneously benefiting from DLT functionality, which allows storing transactions, keeping a distributed exchange of ML models, and data transparent and auditable.

The paper is related to research on the specification of blockchain-based scientific processes [14] in terms of smart contract development for federated learning decentralization. Since blockchain technology solutions are quite innovative and complex, visual representation techniques can help describe the implementation logic. Both in [14] and in this paper, unified modelling language (UML) [15] diagrams are used to outline the proposed solution details and to demonstrate the behaviour and structure of the implemented smart contract.

In our research, a blockchain architecture for the ML model inference is implemented for experimental evaluation. We use Hyperledger Fabric blockchain [16] as a platform of choice, as Hyperledger supports smart contract calls to external APIs, and features a modular infrastructure by adopting the microservice ecosystem. Hyperledger Fabric enables the implementation of smart contracts, which are referred to as chaincodes in the Hyperledger infrastructure. It also supports existing programming languages such as Go, Javascript, Java, and service-based architecture. In our experiments, we implement a chaincode dedicated to the logistic regression model inference using Go. The logistic regression was chosen as a proof of concept due to its simplicity and popularity for the ML detection task [17].

We develop two experimental implementations: A model inference chaincode, and a model inference oracle service. The execution logic implemented in the chaincode and the oracle service is identical. To test the performance of these blockchain implementations we conduct two experiments where we benchmark the ML model validation performance on generated synthetic 2D dataset [18] and real-world EEG (electroencephalogram) eye state data [19]. In the experiments, we compare the runtime of functions which were executed using either the chaincode or oracle service.

As existing approaches implementing the FL using blockchain are limited in scope with respect to the available models, the goal of our work is to expand the set of ML models which could benefit from decentralization using blockchain technologies and to facilitate the development of such solutions. Therefore, we have to determine whether there is a substantial difference in the model inference runtime between chaincode variants, where the code is encapsulated entirely into a smart contract, and a standalone microservice, which is used as a trusted oracle inside a smart contract. Various ML models are adapted for FL purposes [20,21] and attempts exist to train heterogenous local models of varying complexity for producing a single powerful global inference model [22]. Chaincode would be more suitable for basic ML models with a lightweight internal structure and low computational load, for example, various regression variants, linear discriminant analysis, naive Bayes, decision tree, etc. The calculation transfer to the oracle service from the chaincode would be beneficial for complex ML models with a complicated internal structure and high computational load, for example, support vector machine, random forest, gradient boosting ensemble, etc. The experimental results can be used to determine whether the oracle service is a viable solution to extend the possibilities of blockchain-based ML.

The rest of the paper is structured as follows. Section 2 presents related research on the combination of machine learning and blockchain technologies. Section 3 depicts the implemented blockchain-based architecture and the developed smart contract for machine learning. In Section 4, experiment setting is introduced, and experiment findings are presented. Section 5 discusses the main contributions, limitations, and possible applications of our research. Section 6 concludes the research results and outlines future work.

2. Related Work

The combination of federated learning and blockchain technologies is applied in various domains, such as Internet of things [6,23,24], healthcare [7,25], privacy and security [26,27], etc. Some researchers propose utilizing a public blockchain [23] while others believe that distributed ledger technologies [6–8], are more suited for federated learning. As proof-of-work consensus algorithms are considered redundant, since computational resources could be exploited to execute ML processes, some authors propose novel consensus algorithms [8,25]. Research employing already existing blockchain solutions suggests that improved federated learning approaches could bring an increase in transparency and model accuracy [8,28].

Research on blockchain and federated learning for privacy-preserved data sharing in an industrial IoT [24] proposed a system architecture made from two main components—permissioned blockchain component and federated learning component. The blockchain component is used to store transactions for data retrieval from IoT devices and data sharing

among peers, which facilitates the auditing of data flows. The authors also implement a proof of training quality consensus algorithm, that combines data model training with the consensus protocol. To evaluate the proposed data sharing architecture, the authors conducted experiments on two real-world text-based datasets. The proposed federated learning mechanism achieves a high diagnostic ability reaching an average AUC (area under the receiver operating curve) value of 0.918. The experiment also shows that the proposed architecture is scalable as the AUC value varies slightly depending on the number of data providers. The runtime statistics from three peers up to nine peers' averages to 880 ms using various datasets, and this runtime reaches a higher average as the number of data providers increases.

D. Preuveneers et al. [6] proposed the permissioned blockchain-based federated learning method for the detection of intrusions in IoT networks. The blockchain implementation, more specifically MultiChain, provided transparency and auditability to the federated learning process with a negligible runtime overhead of 5–15%. Authors note that the performance overhead caused by the blockchain is closely tied to the complexity of the ML model (autoencoder with three hidden layers totaling 3000 weights), which is a direct consequence of weight sharing and the averaging approach in federated learning.

Hyesung Kim et al. [23] proposed a BlockFL federated learning architecture improved by blockchain that allows participating members to exchange devices' local model updates while verifying and providing rewards for participation. The BlockFL enhances federated learning by introducing decentralization and providing a monetary incentive for members of the blockchain network. Authors evaluated the accuracy of their proposed solution and proved that as the number of iterations increases, the BlockFL achieves almost the same accuracy as traditional federated learning.

T.-T. Kuo and L. Ohno-Machado [25] presented the decentralized privacy-preserving healthcare predictive modelling framework. The framework allows model storing in a private blockchain, as well as introduces the proof-of-information consensus algorithm. Authors also describe how models should behave if new data are committed to the blockchain network, where the provided rules determine whether a model should be updated or not.

J. Weng et al. [8] presented DeepChain—a decentralized platform for a secure collaborative deep training through gradient sharing. The solution encourages blockchain peers to participate in the learning process by providing an incentive in the form of DeepCoin. Authors speculate that a well-trained model could be profitable in the future as a form of a paid service. Experiments with 4–10 peers show that the more parties participate in collaborative training the higher the training accuracy is achieved, but it is an expected result since each new party brings a fixed-sized batch of training data. Authors also experimented by increasing the number of participating parties revealing that the gradient sharing time increased with a higher number of participating parties.

G. J. Mendis et al. [28] proposed a cooperative decentralized deep learning architecture. The contributors can train deep learning models with private data and share them with the cooperative data-driven applications. Shared models are fused to produce a higher quality model. Authors proposed two approaches to performing federated learning. The first approach involves an initiator that proposes the model and contributors that train the proposed model. The second approach enables contributors to train their models which are then merged to create a complete model. The authors present that the model accuracy has increased while using the merging technique rather than just developing a model proposed by the initiators. In their research, the authors evaluate the impact of model fusion techniques such as Federated Averaging and propose alternatives to the model accuracy. Fusion strategies that authors have proposed show promising results. Another experiment that authors performed was the evaluation of the effect on model accuracy comparing their proposed solution to the solution proposed by H. B. McMahan et al. [3]. Both solutions used the MNIST database to evaluate the accuracy of their models. The model classification accuracy was tested in two scenarios, producing a lower accuracy in the first one and higher in the second one. As experiments presented a similar model

classification accuracy result, the authors favoured the blockchain solution for its increased security and lower communication cost.

Nelson Kibichii Bore et al. [7] extended the Hyperledger Fabric endorsement policy by implementing a validation mechanism for machine learning. The validation mechanism was created as a blockchain network component responsible for sending recompilation calls to selected peers called endorsers. Endorsers recompute models, and if the deviation from the initial computation is within a defined acceptable tolerance, models are endorsed and committed to the blockchain. Authors performed an analysis on the detection of faulty or anomalous workers using this validation engine. Research suggests that machine learning using blockchain technologies is relevant and achievable since the authors extended the machine learning process by using a smart contract (chaincode) to ensure a decentralized trust between the parties involved.

The smart contract used in various implementations of blockchain-based machine learning [7,9,28], is a type of software engineering artefact which could be developed using software engineering principles [29]. While blockchain usage is continuing to grow, no common tools and methods for the specification of either blockchain technology-based engineering solutions, or smart contracts, exist. Some authors use formalized specification languages, such as BPMN [30], Petri nets [31], or simple nonformalized notations, such as flowcharts [28,32]. Since UML is commonly used in traditional software development, it can be successfully employed for the specification of blockchain-based systems and smart contract behaviour [33]. In this paper, UML diagrams are used for both outlining the blockchain architecture and specifying the smart contract structure and behaviour. The smart contract is represented using UML class and sequence diagrams which are closely related to the smart contract code and could even be used for code generation purposes. The presented smart contract specification (in Section 3) extends research [14] on the specification of blockchain-based business and scientific processes.

The comparison of overviewed blockchain-based FL solutions with respect to various technological aspects is summarized in Table 1. We can notice that the fields of application are quite broad and implementation details (DLT type, consensus used, support for smart contracts) are diverse. As most existing blockchain frameworks and their application to FL are in the early stages of development, no common trends and state-of-the-art can be distinguished.

Table 1. Comparison of overviewed federated learning (FL) solutions employing blockchain. Abbreviations: PoQ: Proof of Training Quality; RR: Round-robin-based; PoW: Proof-of-Work; PoI: Proof-of-Information; PoS: Proof-of-Stake; PoA: Proof-of-Authority; BFT: Byzantine Fault Tolerance.

Ref.	Consensus Algorithm	Incentive	DLT Type	Smart Contracts	Field of Application	Access Type
[24]	PoQ	No	Custom	No	Industrial IoT	Permissioned
[6]	RR	No	MultiChain	No	Cyber security	Permissioned
[23]	PoW	Yes	Custom	No	Universal	Public
[25]	PoI	No	MultiChain	Undefined	Healthcare	Permissioned
[8]	Algorand [1]	Yes	Corda	Yes	Deep learning	Permissioned
[28]	PoW, PoS, PoA	Yes [2]	Ethereum	Yes	Universal	Public
[7]	BFT [3]	No	Hyperledger Fabric	Yes	Disease modelling	Permissioned

[1] Modified version; [2] partially; [3] assumed.

Our approach differs from the aforementioned research in a combination of several important aspects: (a) Our solution could be extended to include several different model types, and could be customized for various model or data acceptance strategies by tweaking smart contracts; (b) we measure the runtime of model inference entirely implemented as a Hyperledger Fabric chaincode solution; (c) we measure the runtime of blockchain oracle

service implementation, to determine whether it can be used to include the wide range of ML approaches on blockchain technology; (d) we compare the runtime differences between chaincode and oracle service implementations for the ML model inference. Additionally, the proposed solution is specified using UML diagrams, whereas other authors mostly use non-formalized notations, such as flowcharts.

3. Methods

The environment for experiments was implemented using the Hyperledger Fabric technology, which features modular architecture and allows developing of smart contracts in Go or Java languages. The choice of Hyperledger Fabric over other blockchain frameworks was due to its ability to perform calls to external endpoints from the inside of a smart contract. The smart contract for model inference was implemented using Go language and two variants were developed with respect to logistic regression ML model inference calculations: (1) Model inference is implemented as a pure chaincode solution; (2) the model inference part is outsourced to the local oracle service implemented as a RESTful microservice running at each peer. Logistic regression as a proof-of-concept was chosen due to the fast runtime since most other ML models are more computationally demanding. A faster inference runtime is expected to make network communication overheads more noticeable in benchmarking experiments.

ML models were prepared by training them beforehand since this stage of ML lies in the responsibility of participants and is not encompassed by our approach due to a privacy-preserving aspect. The experiments performed evaluate only the model inference stage, which in ML terms would correspond to validation (if a model is uploaded after data are shared) or testing (when new data are shared, triggering inference for existing on-chain models).

3.1. Implemented Blockchain Architecture

Using Hyperledger Fabric technology, ML solutions can be implemented in two variations: (a) By entirely relocating the training, evaluation, and usage to the blockchain, using the smart contracts (or chaincode as they are called in Hyperledger), or (b) by utilizing blockchain oracles [12,13], (decentralized web services) since Hyperledger supports a limited set of programming languages for implementing the chaincode. These oracle services can be integrated into smart contracts developed using Hyperledger Fabric technology. In this research, we implemented both (a) and (b) variations in the model inference prototype application to compare them and find differences between the two.

The architecture of the solution components is presented in Figure 1. The prototype implementation is composed of Hyperledger Fabric components and Model Inference Service, which communicates with each other through APIs. The Hyperledger Fabric chaincode was developed using the Go programming language and is responsible for model inference computations. Additionally, an oracle model inference web service was developed using Go. The oracle web service is essentially responsible for the same execution logic.

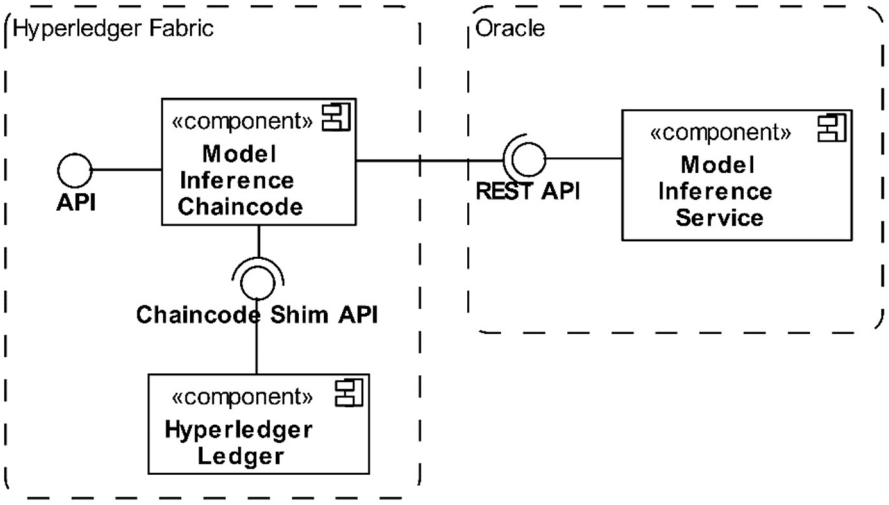

Figure 1. The utilized blockchain architecture: Unified modelling language (UML) component diagram. Chaincode Shim API provides access to data stored in the chaincode state database.

3.2. Smart Contract Implementation Details

In our model inference prototype, two types of stakeholders exist (Figure 2). The data owner (A1) can upload evaluation data (UC1) to the prototype application, which is used for new model validation and testing of already existing models. The model owner (A2) can upload the model (UC3), which once validated is committed to the blockchain. Additionally, Hyperledger Fabric (A3) participates in all use cases (UC1-4), as an external network, which is responsible for transaction recording. Either type of user acts as a participant during the data or model upload processes, and only after the majority of participants approve the transaction, it is committed to the blockchain.

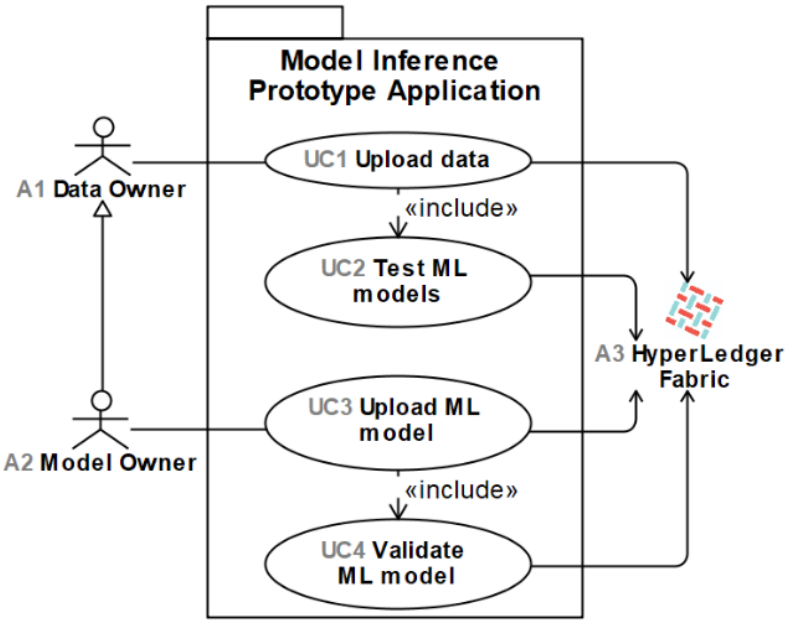

Figure 2. The main functionality of the blockchain-based model inference application: UML use case diagram.

In the model inference solution, federated learning participants can contribute their data and off-chain computed machine learning models for evaluation using the chaincode, which is responsible for the data and model storage.

The data upload procedure (UC1) starts as the user provides a data file. The data file is then checked for the .csv data file format and whether the provided data columns are of the expected format and correspond to the outlined data structure. If the provided file is valid, the data is inserted into the Hyperledger's data storage by recording each data row. After the evaluation data are uploaded, the models already committed to the blockchain are tested (UC2). Since the model is already uploaded, it is safe to assume that the model during this process gets tested, as the possibility that the model was trained on the newly uploaded data is quite low. After the testing, the ML model results are appended according to the new model inference computations for newly inserted data records.

The model upload procedure (UC3) starts in the same way, but rather than a data file, a user is asked to provide the ML model file. It is important to note, that the ML model is trained by the model owner and is not included in the scope of the model inference prototype application. The submitted file contains logistic regression coefficients, these again are checked whether the data are of an appropriate format and structure. Afterwards, an ML model validation process is initiated (UC4). We present chaincode specification fragments of the ML model validation process in more detail. The provided class and sequence diagrams (Figures 3 and 4) describe the essential parts which should be recorded in the smart contract execution code. The class diagram (Figure 3) outlines the smart contract functions, their parameters, and types. The classes that participate in the smart contract execution are presented as lifelines in the sequence diagram (Figure 4). Using the behaviour and the structure recorded in these diagrams, a source code can be produced which would facilitate the process of development. Several specification patterns are described in UML diagrams (Figures 3 and 4) and were used to produce the source code (Figure 5) for this federated learning solution.

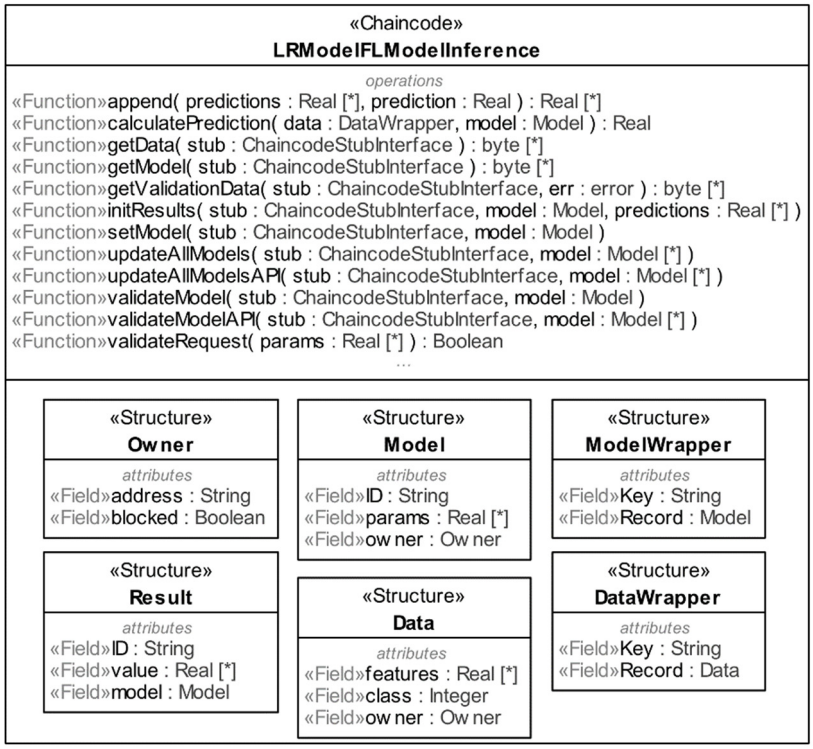

Figure 3. Smart contract structure: UML class diagram representing the main structures and functions of the class LRModelFLModelInference.

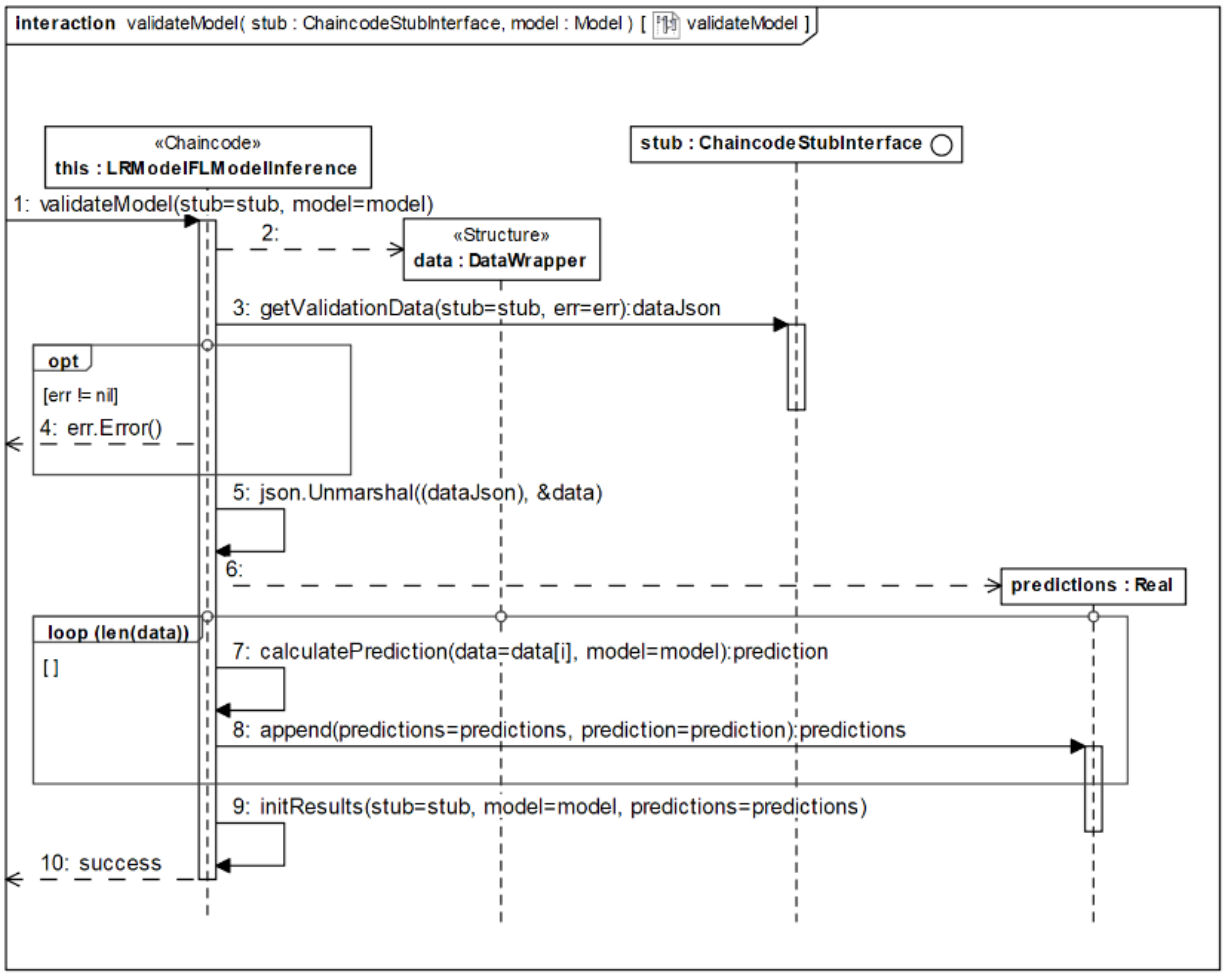

Figure 4. Smart contract behaviour during model validation: UML sequence diagram.

```
1  func validateModel(stub shim.ChaincodeStubInterface, model Model) pb.Response {
2      var data []DataWrapper
3      dataJson, err := getValidationData(stub)
4      if err != nil {
5          return shim.Error(err.Error())
6      }
7      json.Unmarshal((dataJson), &data)
8      var predictions []float64
9      for i := 0; i < len(data); i++ {
10         prediction := calculatePrediction(data[i], model)
11         predictions = append(predictions, prediction)
12     }
13     initResults(stub, model, predictions)
14     return shim.Success(nil)
15 }
```

Figure 5. Validate model function: Chaincode fragment of the smart contract.

Model validation (Figure 2 UC4) begins after the model is uploaded to the blockchain (Figure 2—UC3). Once the *validateModel* function is called, a new instance of the DataWrapper array is created. Afterwards, the JSON data (in the form of a byte array) from the ledger are selected (Figure 4—messages 2–3, Figure 5—lines 2–3), and if the selection is successful the validation continues. Otherwise, once the error object is populated, the *validateModel*

function returns a failure message (Figure 4—opt fragment, message 4, Figure 5—lines 4–6). Afterwards, the JSON data and structure are mapped using the JSON library package function (Figure 4—message 5, Figure 5—line 7). Then, one more instance is instantiated—an array of float type (Figure 4—message 6, Figure 5—line 8). Once the required variable instance is constructed, a prediction is calculated for each validation data record and the model parameters. Considering that the prediction is calculated for each data validation record, a loop fragment is used. Inside a loop, a separate method for the prediction calculation is used and each calculated prediction is appended to the previously instantiated float type array (Figure 4—loop fragment, messages 7–8, Figure 5—lines 9–12). Once all the predictions for all DataWrapper records are calculated, the calculated predictions for a specific model are saved in the blockchain ledger using the *shim* library [34], and finally, the success message is returned (Figure 4—message 9–10, Figure 5—lines 13–14).

The test models function (Figure 2—UC2), which is called during the data upload process, is based on the same principles as the model validation function. The test function differs from the model validation in several aspects. During the testing, the models are selected from the ledger, and new prediction results are calculated (similar as in Figure 4—loop fragment, message 8) and updated accordingly. During the testing, the model and the data records remain unchanged, only the results are appended to the blockchain ledger. Both testing and validation functions calculate the prediction results, and the only difference is the number of models that passed to the function. The following experiment sections cover the runtime benchmarking results of the *validateModel* function.

The execution of the *validateModel* function relies heavily on the *calculatePrediction* function, where the logic of model inference resides. The running example of the logistic regression inference is demonstrated in Figure 6, which depicts how prediction is obtained for a single data row using a pre-trained model. As it can be noticed, the prediction got very close to the ground truth.

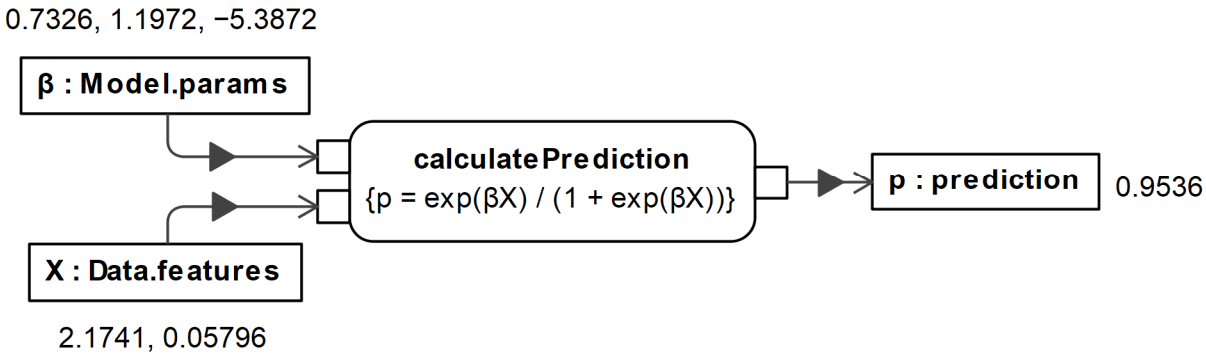

Figure 6. Running example of the *calculatePrediction* function featuring the logistic regression inference. The shown data instance corresponds to the ground-truth class label, in this case, the target class instance (Data.class = 1), which is not used in the calculations.

4. Benchmarking Experiments

The goal of the experiments is to evaluate the impact of the introduction of oracle service on the performance of blockchain-based model inference. We have developed the oracle service for the model inference, which was integrated into the blockchain-based prototype application to enable a comparison of model validation runtimes between the chaincode and oracle service.

4.1. Experimental Setup

The experiments were performed using the architecture described in Section 3.1 on a server powered by an 8 Intel Xeon Silver 4114 CPU running at 2.20 GHz and featuring 16 GB of RAM with data stored on the SSD storage. The server was running an Ubuntu 18.04

operating system and using Docker 19.03.6 containers as an environment for Hyperledger Fabric 1.4.9 with CouchDB 2.3.1 as the state database. All peers existing in the blockchain network were required to execute the chaincode to reach a consensus. Experiments were conducted with peers configured as a single organization.

Chaincode was developed using Go, and the oracle was implemented as a RESTful service API endpoint using the Go *net/http* library [35], encapsulating the benchmarked function, with the code content identical to the chaincode implementation.

The deployment architecture of the model inference prototype is presented in Figure 7. The main components were hosted on a Linux Ubuntu server using Docker [36] containers. Hyperledger Fabric CLI (command-line interface) was deployed into a node, which was used to experiment by calling chaincode functions. The CLI was linked to the Orderer service and Certificate authority, which issues certificates. The Orderer regulates the transaction, commits among peers, and controls the chaincode execution order. Each of the peer nodes had a Hyperledger solution deployed, which consisted of a model inference chaincode and local ledger copy. Alongside the Hyperledger Fabric solution, a model inference web service was deployed to each node to accurately judge the performance of the oracle web service against the entirely blockchain-based solution.

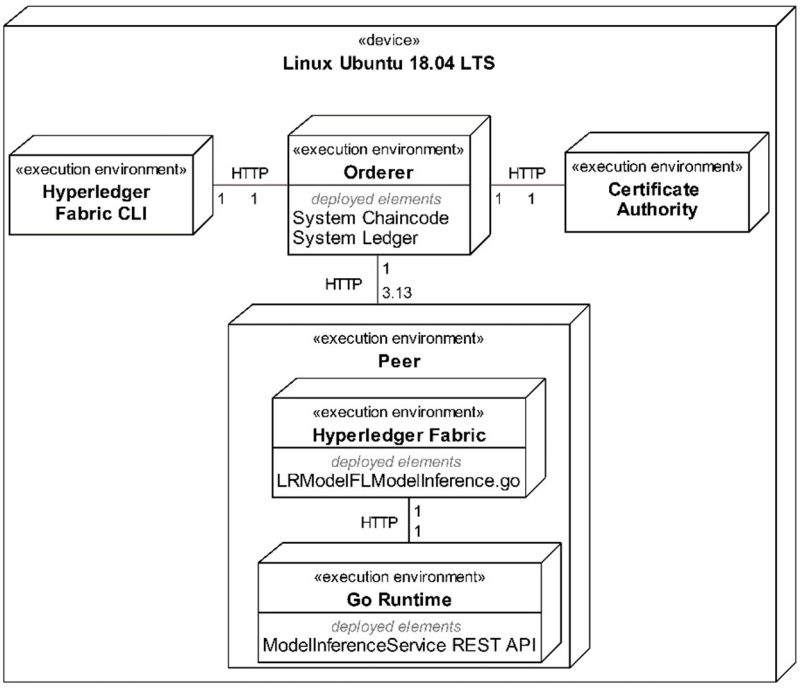

Figure 7. Physical view of the utilized blockchain architecture: UML deployment diagram. All components were hosted on the same physical server, using Docker [36] containers.

To evaluate the impact on the model inference runtime performance, we conducted model validation experiments on two datasets:

1. The synthetic dataset contained points in the 2D feature space of two non-linearly separated banana-like shaped classes and were generated using the *make_moons* function from the *scikit-learn* Python library [18].
2. The EEG eye state dataset [19] contained 14 features for detecting if the participant's eyes were open or closed during a 117 s recording of signal through 14 channels of EMOTIV EEG Neuroheadset.

The number of rows for both datasets was determined according to the following grid: {1024; 2048; 4096; 8192; 16,348; 32,768}. Since the EEG eye state dataset contained only 14,980 instances it had to be expanded to contain 16,348 or 32,768 rows through bootstrapping of the original dataset. Data were stored using indexing, to speed up data

reading during the experiments. The sizes of synthetic and EEG eye state datasets in kB are presented in Table 2. The number of peers participating in the network was set according to the following grid: {3; 5; 7; 9; 11; 13}. These specific network sizes were chosen to represent a gradual increase and the maximum network size was capped according to the known limitations of the Hyperledger Fabric platform [37]. For each combination of data and network size, the blockchain infrastructure was initialized from scratch.

Table 2. Amount of data records and size (in kB) of the corresponding JSON (JavaScript Object Notation) structure.

Data Records	Synthetic Dataset (kB)	EEG Eye State Dataset (kB)
1024	114	510
2048	227	1023
4096	453	2049
8192	905	4097
16,348	1809	14,446
32,768	3650	16,384

For both datasets, we measured model inference runtimes and summarized a comparison between different implementations using the overhead derived by the following ratio:

$$Overhead = 100 \times \left(median \left(\frac{T_{OracleAPI}}{T_{Chaincode}} \right) - 1 \right) \quad (1)$$

where T is a set of runtimes, obtained while repeating the experiment over 100 logistic regression models, which were prepared ahead of the experiments using the *GoML* library [38] on a random subsample of the dataset. The runtime was measured from the moment the model validation function was called until the function returned the results to the smart contract.

4.2. Synthetic 2D Dataset Results

The preparation step for the model validation use case experiment was uploading validation data to Hyperledger CouchDB through the chaincode. During the experiment, the peers uploaded their models. The uploaded models were validated using chaincode and the runtime of the validation function was measured. Blockchain was restarted and then the function that uses the oracle component rather than the chaincode was called, measuring the oracle service runtime. The median runtimes for all the tested network and data sizes are presented in Figure 8, where the bar plots indicate that the model inference time substantially increases after the dataset size reaches more than 8192 rows. Surprisingly, the linear increase in runtime with respect to the network size can only be noticed when testing on large datasets containing 16,348 or 32,768 rows.

For the largest dataset scenario, we can observe a linear relationship between the network size and model inference runtime (see Figures 8 and 9) except for the outliers consistently resulting from the first run of the experiment. Interestingly, the outliers for the chaincode were fixed at almost a constant low value, while for the oracle service the outlier values fluctuated, especially when the network exceeded eight peers.

The largest runtimes at the extreme scenario of large network and data settings are shown in Figure 10. We speculate that overheads are mainly due to the transferring of data from the blockchain state database to the oracle component and receiving back the model output, required for peers to arrive at a consensus to confirm the transaction. Most of the runtime results are clustered around 1 min and 19 s for both model inference implementations. Figure 10 also displays that both types of validation methods displayed the lowest runtime values during benchmarking initialization runs, as the transaction count between peers were still low and peers were not busy computing new blockchain blocks.

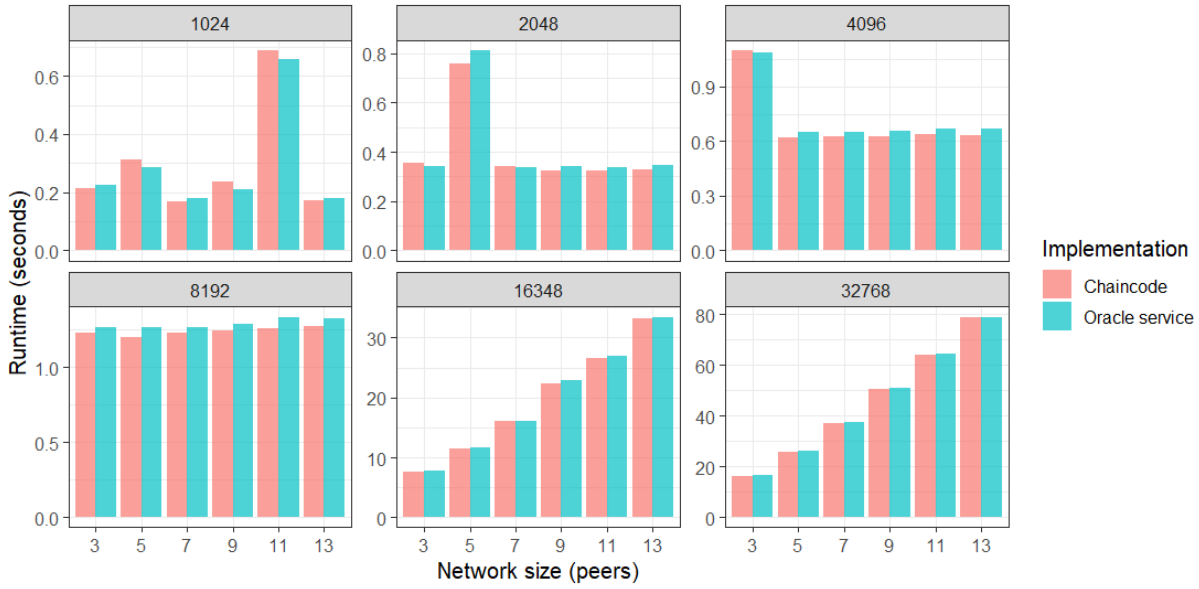

Figure 8. Model inference runtime (in seconds) depending on the network size for different amounts of the synthetic dataset.

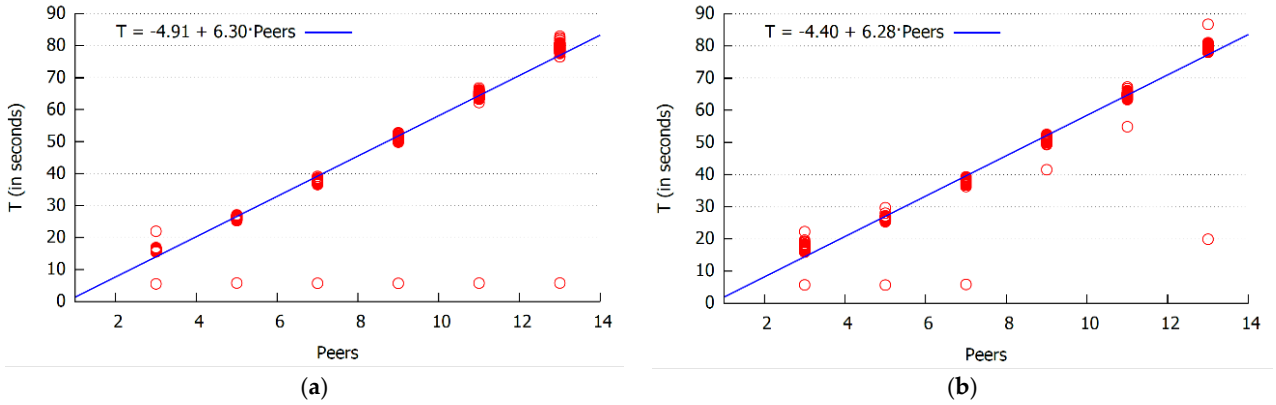

Figure 9. Model inference runtime (in seconds) in relation to the network size for the largest variant of the synthetic dataset: Chaincode (**a**) and oracle service (**b**) results. The linear relationship is evident where an additional peer in the network increases runtime T by 6.3 s for the chaincode and 6.28 s for the oracle service.

Table 3 demonstrates the runtime overhead in percentages of the oracle service component. Results indicate that the synthetic dataset, surprisingly, had both negative and positive overheads with an overall median overhead of 1.99%. The results could be summarized as follows: (a) On all network sizes (3–13 peers) and data records not exceeding 8192 (1024–8192 records) the overhead ranges from −4.31 to 6.59%. As validation with these configurations takes a relatively short time, the variability in inference overhead is quite high; (b) as data increase to 16,348 records variability in overhead drops for all peers (3–13 peers) with the overhead ranging from 0.73 to 4.28%; (c) using the highest amount of data records tested (32,768 records), the variability in validation times drops significantly only producing at most 2.79% overhead on the least number of peers (three peers) tested and less than 1% (0.03–0.73%) on the remaining peer configurations (5–13 peers). Results indicate that the overhead diminishes once the number of participants and data records increase. The higher variability of overheads for smaller dataset sizes irrespective of the network configuration was observed.

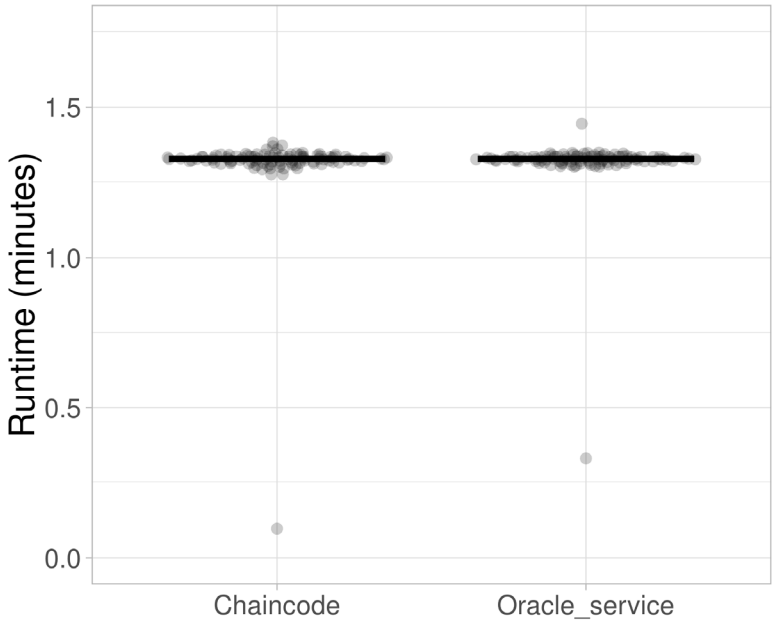

Figure 10. Distribution of runtime results using a network of 13 peers with the dataset of 32,768 records in the model validation experiment. The median runtime for model validation was 1 min 19 s for the chaincode (**left**) and 1 min 19 s for the oracle service (**right**).

Table 3. Model inference overhead (in%) for the synthetic dataset, calculated as a median of 100 runtime ratios. The overall median from 3600 runs ($6 \times 6 \times 100$ runtime ratios) was 1.99% and the overhead was above zero for 69.15% of all runs.

Data Records	Number of Peers in the Blockchain Network					
	3	**5**	**7**	**9**	**11**	**13**
1024	2.66	2.56	6.28	−4.31	−1.51	6.59
2048	2.18	4.48	−0.63	3.72	4.99	5.31
4096	1.17	5.61	3.93	5.37	4.27	4.89
8192	3.16	5.01	2.85	3.79	2.96	3.59
16,348	4.28	1.20	0.32	1.36	1.02	0.77
32,768	2.79	0.73	0.63	0.40	0.05	0.03

As network and data sizes increase, the performance overhead approaches zero since the time required to transfer data to the oracle service gets relatively smaller in comparison to the model inference time. We speculate that the performance of the implementations compared would reach nearly identical runtime results in a real-world scenario.

4.3. EEG Eye State Dataset Results

The median runtimes for all the tested network and data sizes are presented in Figure 11, median values presented similar results to the first experiment performed with the synthetic 2D dataset. Similar to the synthetic 2D dataset results, the biggest increase in the model inference time occurred when the data size reached more than 8192 records. The bigger number of features and dataset size increased the amount of time required to validate the model, thus displaying higher median values for the EEG eye state dataset. Identically to the results of the previous experiment, a linear increase in runtime with respect to the network size can only be noticed when testing on large datasets containing 16,348 or 32,768 rows.

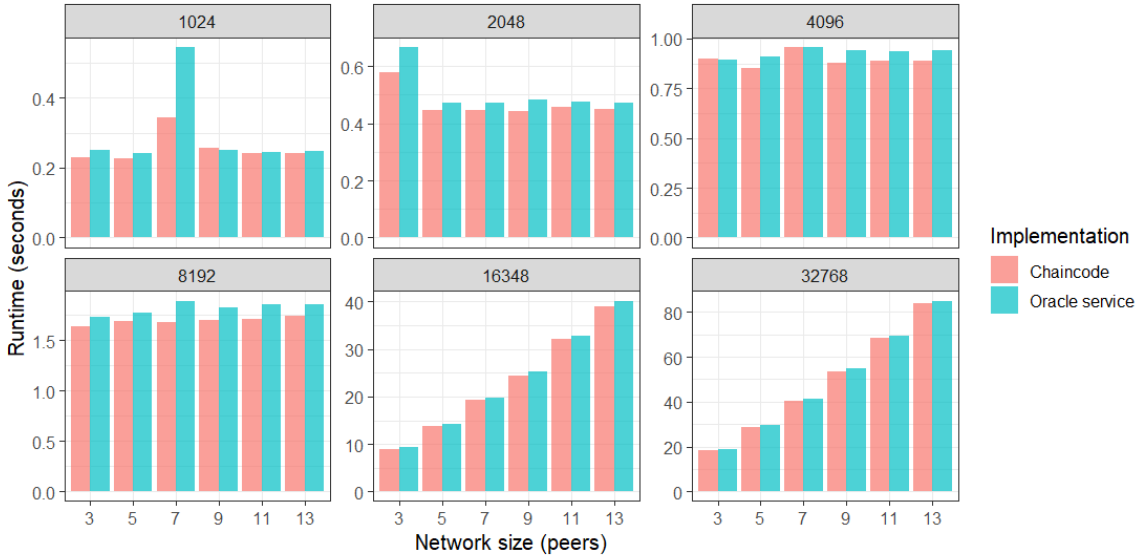

Figure 11. Model inference runtime (in seconds) depending on the network size for different amounts of EEG (electroencephalogram) eye state dataset.

For the largest dataset scenario, we can observe a linear relationship between the network size and model inference runtime (see Figures 11 and 12) except for outliers consistently resulting from the first run of the experiment. Similarly, to the first experiment, outliers for the chaincode were fixed at almost a constant low value, while the oracle service outlier values started fluctuating, especially when the network exceeded eight peers.

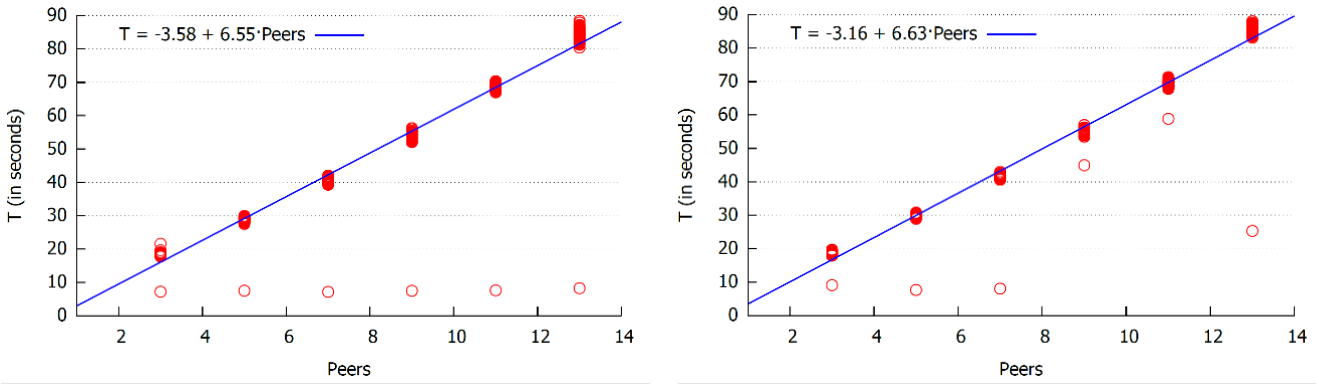

Figure 12. Model inference runtime (in seconds) in relation to the network size for the largest variant of EEG eye state dataset: Chaincode (**a**) and oracle service (**b**) results. The linear relationship is evident, where an additional peer in the network increases runtime T by 6.55 s for the chaincode and 6.63 s for the oracle service.

The largest runtimes at the extreme scenario of large network and data settings are shown in Figure 13. The initial runs for both validation types provided the least amount of validation time clearly describing the overhead introduced by the oracle service. All sequential validation run results clustered around the validation time of 1 min and 24 s for the chaincode validation and around 1 min 25 s for the oracle service validation.

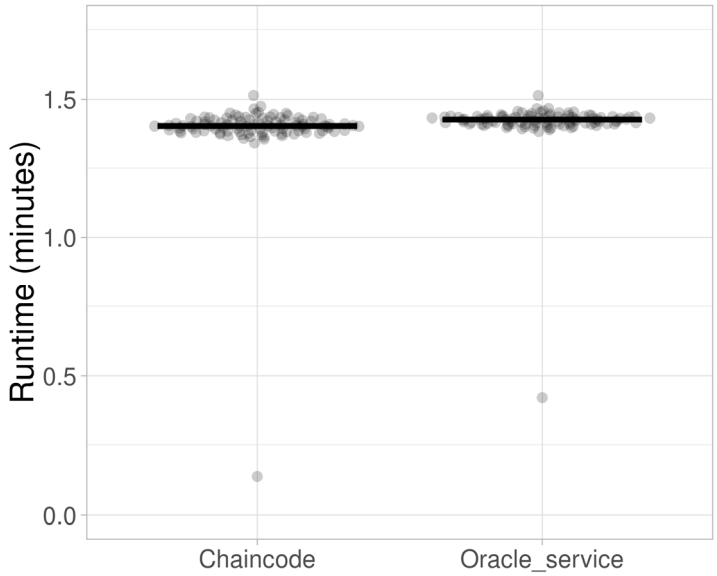

Figure 13. Distribution of runtime results using a network of 13 peers with the dataset of 32,768 records in the model testing experiment. The median runtime for model validation was 1 min 24 s for the chaincode (**left**) and 1 min 26 s for the oracle service (**right**).

In Table 4, oracle service overheads in percentages are reported, revealing the suitability of the oracle service for the larger dataset and network sizes. Results indicate that the EEG eye state dataset had only positive overheads with the overall median overhead of 4.06%. The results could be summarized as follows: (a) On smaller network sizes (3–7 peers) and all dataset sizes (1024–32,768 records), the overhead was in the range of 2.16 to 9.67%, using these network and data configurations the highest amount of overhead was recorded on the smallest dataset (1024 records) and the least amount of overhead 2.16% was recorded using the highest amount of data records (32,768 records); (b) using the larger network sizes (9–13 peers), the validation overhead was in the range of 0.80–6.91%, for these peer network configurations the overhead peaked at medium-sized datasets (2048–8192 records) 6.91% for nine peers, 6.53% for 11 peers, and 6.76% for 13 peers. The least overhead 0.80% was recorded using the largest amount of data (32,768 peers) and network composed of 11 peers. Overall, the maximum overhead for various network sizes was in the range of 0.80–9.67%.

Table 4. Model inference overhead (in%) for the EEG eye state dataset, calculated from a median of 100 runtime ratios. The overall median from 3600 runs ($6 \times 6 \times 100$ runtime ratios) was 4.06% and the overhead was above zero for 85.51% of all runs.

Data Records	Number of Peers in the Blockchain Network					
	3	**5**	**7**	**9**	**11**	**13**
1024	9.32	7.69	9.67	3.52	1.93	4.61
2048	4.83	4.36	2.80	6.91	5.89	4.95
4096	2.13	6.88	5.67	6.19	6.53	5.41
8192	5.73	5.05	7.01	6.65	6.09	6.76
16,348	4.06	2.97	2.35	2.34	1.54	2.69
32,768	2.17	3.63	2.16	1.76	0.80	1.82

4.4. Comparison and Summary of Results

Although the size for the EEG dataset was more than four times larger than for the synthetic dataset, the median performance overhead only increased by 2.07%. Both experimental results demonstrate that the model validation runtime is more sensitive to the network size increases rather than increases in the dataset size. Moreover, the linear

relationship between the network size and runtime was evident at the largest dataset sizes benchmarked. We speculate that at small amounts of data for inference the communication-related overhead outweighs the inference-related overhead. Therefore, linear dependence is revealed only when data amounts for the inference become extensively large.

5. Discussion

Our implementation is partially related to [7] as they use similar deployment principles and the Hyperledger Fabric network. Authors of [7] implement the model validation using chaincode and do not employ the oracle service. Even though ML can be completed without the oracle services, we believe that these services hold potential to improve the ecosystem of ML on private blockchains, since they unlock the flexibility to use programming languages which are not supported by Hyperledger Fabric. Communication with real-world API from the chaincode is a well-known source of failure in decentralized blockchain applications, and recently some solutions emerged, such as Band Protocol [39] or Hyperledger Avalon [40]. Hyperledger Avalon differs from our proposed solution by allowing many different blockchain technologies, such as Ethereum, Hyperledger Fabric, or Hyperledger Sawtooth, to use various oracles. These oracle components are trusted and centralized, whereas our solution deploys an oracle service instance for each peer existing on the blockchain network. Having oracle service implementation on each participating node eliminates the need to access trusted real-world information outside the network. Expecting identical computation results (in our case, predictions from the ML model) regardless of the node is sufficient. These node-based oracle instances do not degrade the network security level [41] and allow reducing the communication latency. The services for model inference in the Hyperledger Fabric infrastructure could also be deployed by some orchestration entity existing in each organization. The entity would manage updates and would ensure that the organization's peers use the same version of the oracle service. Any mismatch in versions could lead to a failure state in which peers would not be able to reach consensus. Some standard procedures should be provided for new peers joining the network and peers rejoining the network, such as updating and setting up of the oracle service.

During the development process, we explored more than one development environment and discovered that the performance of the model inference depends on the peer's storage type, as it impacts the state database performance. Consequently, the hard disk drive storage used during the development performed way slower compared to the solid-state drive used in the reported experiments. A drawback of our approach is that benchmarking was executed in a virtual environment and not on a network of physical computers. Even though experiments were conducted on a single virtual server using Docker containers for network simulation, the explored architecture is mostly restricted to communication between the smart contract and oracle service over a local connection (the oracle service is running on a localhost). Thus, in a more realistic scenario, additional network latency would be introduced only to maintain that the blockchain network is synchronized, irrespective of the choice between oracle service and chaincode implementations.

We assume performance to be the most important factor for choosing between the oracle and chaincode solution. The blockchain architecture provides additional properties, such as scalability, privacy, auditability, decentralization, etc. In addition to performance, the implemented solution's scalability was partially assessed by benchmarking network configurations of various sizes. The other properties, such as privacy, auditability, or decentralization, were not experimentally assessed since these properties are inherent to the Hyperledger Fabric platform and are not directly affected by the alternatives compared.

The performed experiments demonstrated that an increase in the dataset size results in diminishing differences between implementation alternatives, and we assume that a similar tendency would hold when switching from a simple to a more complex ML model. For expanding experiments to cover more models, slight modifications of model data structure and smart contract execution logic would be needed. One potential solution would be

to use a compact text-based representation of the ML model file rather than adapting the structure for each specific model. The introduction of a more complex ML model would incur longer model inference runtimes with a relatively smaller communication overhead. Therefore, the marginal differences between alternatives would diminish, even more, thus justifying the choice of oracle for the implementation of the ML model inference logic.

In the context of this research, we consider federated learning in a broader perspective than traditional approaches concerned with collecting the ML model updates and integrating them into a global shared model. The blockchain-based network in our case would be used for collective sharing of a small sample of data and models trained on the remaining privately held data. The upload of new unseen data would trigger inference for all the previously uploaded ML models (testing use case) and the upload of the new ML model would trigger its inference for all the previously uploaded data (validation use case). Results of the model inference are then agreed upon by the consensus of network peers and saved permanently in the ledger as a triple of *{ML model ID; data row ID; prediction}*, ensuring that for each possible combination of a shared model and data instance, the recorded model output exists as a prediction verified by the network. Various scenarios to achieve federated learning could co-exist and we could: (a) Upload the traditional ML model (i.e., created using the Python library *scikit-learn*); (b) upload the online ML model (i.e., created using the Python library *creme*); (c) take the existing online ML model from the ledger, train it on individual data, and upload it as a model's new version; (d) propose the meta-model or ML ensemble as a set of weights for the fusion of pre-selected shared models (usual candidates being a few best traditional ML models) through weighted averaging of model-wise predictions. The blockchain network being a back-end, all other tasks of monitoring shared models through tracking of their accuracy by various metrics (log-loss, Kappa, AUC, RMSE, etc.) depending on the supervised ML task (usually, classification or regression) would be fundamental front-end use cases. The envisaged solution encompasses all possible FL scenarios (a–d), supports a wide range of ML models implemented in various libraries and languages (Python, R, MATLAB, Java, Rust, Go, etc.), and features an intuitive graphical user interface in the form of a client application, which would help monitor how the ledger evolves by tracking all the uploaded data and models (performance metrics need to be recalculated after the new data are uploaded). Additionally, the solution could help peers create any chosen model offline on private data without many hurdles. However, this would need additional API endpoints for model training, bundled alongside the model inference oracle service.

Concerns over user data privacy are becoming increasingly important all over the world. This shift in data security also challenges the way ML models are trained and demands techniques to train ML models without the need to share user data or disclose it to a centralized server for processing. Concerns regarding private data protection are serious, especially in such sectors as healthcare or finance. We speculate that the need for privacy-preserving scenarios of ML will continue to grow in demand. While we concentrate on security and immutability aspects of ML, the blockchain technology already provides reliable solutions to log transaction history into the immutable ledger and trace data and model evolution with consistent and verified performance evaluation reached through network consensus.

6. Conclusions

The developed Hyperledger Fabric application incorporated two different implementation variants—a chaincode-based solution, and a combination of chaincode and oracle web service component. Although the introduction of oracle service enables the reusability of existing ML libraries and a more flexible development of FL solutions, it introduces a runtime overhead. To determine the impact on the performance, we conducted experiments assessing runtime differences between the standalone chaincode and the combination of chaincode with the oracle service. Furthermore, the performance overhead was evaluated

using two different logistic regression models validation—one of three coefficients with synthetic data, and the other one of 15 coefficients with the EEG eye state dataset.

The benchmarking experiment using the synthetic dataset has shown promising results with the median runtime overhead of ~2% overall and individual medians not exceeding 6.59% irrespective of the experimental setup. Larger overheads were introduced at smaller data sizes mostly due to the data upload step to the oracle service. As the network and data increase, the impact on the overhead tends to diminish due to the longer inference runtimes compared to the data transfer delay. The benchmarking experiment using the EEG eye state dataset had a median runtime overhead of ~4% and individual medians were in the range from 0.8 to 9.7%. Although the size of the EEG eye state dataset was bigger more than four times (and data record dimensionality bigger than seven times) than the synthetic dataset, the overall median overhead increased by two percentage points, suggesting that the oracle runtime is mostly affected by the model inference rather than the dataset transfer step. Therefore, to summarize, the performance is not as sensitive to the dataset size changes, as opposed to the network size changes, where an additional peer increases the runtime linearly by over 6 s at the largest amount of data records benchmarked.

Our research suggests that the increases in runtime, due to the transfer of computations to oracles running on individual peers, seem to be negligible if compared to the possibilities that such an implementation has the potential to unlock. The implemented prototype and Hyperledger Fabric need to be further developed to achieve federated learning for real-world applications. Organizations that require more participating peers should evaluate if the time required to validate the model using the pure chaincode is sufficiently lower than the more universal approach using the oracle service component.

As future work, we plan to introduce the capability to combine the uploaded ML models into the ensemble or sequentially evolve a single online ML model. Furthermore, since the proof-of-concept demonstrated that the impact of oracle service on runtime performance is marginal, we plan to support a wider variety of models through third-party ML libraries and experiment with model merging approaches.

Author Contributions: Conceptualization, E.V., V.D., M.J., L.Č.; methodology, L.Č., M.J., V.D.; software, V.D.; validation, E.V., L.Č.; formal analysis, V.D., M.J.; resources, L.Č., R.B.; data curation, E.V.; writing—original draft preparation, V.D., M.J.; writing—review and editing, E.V., L.Č., M.J., V.D.; visualization, E.V., M.J., L.Č.; supervision, R.B.; funding acquisition, L.Č. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were used: Synthetic 2D [18], EEG-eye-state [19].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hard, A.; Rao, K.; Mathews, R.; Ramaswamy, S.; Beaufays, F.; Augenstein, S.; Ramage, D. Federated learning for mobile keyboard prediction. *arXiv* **2019**, arXiv:1811.03604v2.
2. Konečný, J.; McMahan, H.B.; Yu, F.X.; Richtárik, P.; Suresh, A.T.; Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv* **2017**, arXiv:1610.05492v2.
3. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. *Artif. Intell. Stat.* **2017**, *54*, 1273–1282.
4. Mothukuri, V.; Parizi, R.M.; Pouriyeh, S.; Huang, Y.; Dehghantanha, A.; Srivastava, G. A survey on security and privacy of federated learning. *Future Gener. Comput. Syst.* **2020**, *115*, 619–640. [CrossRef]
5. Wen, Y.; Li, W.; Roth, H.; Dogra, P. Federated Learning Powered by NVIDIA Clara. 2019. Available online: https://developer.nvidia.com/blog/federated-learning-clara/ (accessed on 26 November 2020).

6. Preuveneers, D.; Rimmer, V.; Tsingenopoulos, I.; Spooren, J.; Joosen, W.; Ilie-Zudor, E. Chained anomaly detection models for federated learning: An intrusion detection case study. *Appl. Sci.* **2018**, *8*, 663. [CrossRef]
7. Bore, N.K.; Raman, R.K.; Markus, I.M.; Remy, S.L.; Bent, O.; Hind, M.; Weldemariam, K. Promoting distributed trust in machine learning and computational simulation. In Proceedings of the IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Seoul, Korea, 14–17 May 2019; pp. 311–319. [CrossRef]
8. Weng, J.; Weng, J.; Zhang, J.; Li, M.; Zhang, Y.; Luo, W. Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. *IEEE Trans. Dependable Secure Comput.* **2019**. [CrossRef]
9. Li, Z.; Liu, J.; Hao, J.; Wang, H.; Xian, M. CrowdSFL: A secure crowd computing framework based on blockchain and federated learning. *Electronics* **2020**, *9*, 773. [CrossRef]
10. Salah, K.; Rehman, M.H.; Nizamuddin, N.; Al-Fuqaha, A. Blockchain for AI: Review and open research challenges. *IEEE Access* **2019**, *7*, 10127–10149. [CrossRef]
11. Beniiche, A. A study of blockchain oracles. *arXiv* **2020**, arXiv:2004.07140.
12. Lo, S.K.; Xu, X.; Staples, M.; Yao, L. Reliability analysis for blockchain oracles. *Comput. Electr. Eng.* **2020**, *83*, 106582. [CrossRef]
13. Kochovski, P.; Gec, S.; Stankovski, V.; Bajec, M.; Drobintsev, P.D. Trust management in a blockchain based fog computing platform with trustless smart oracles. *Future Gener. Comput. Syst.* **2019**, *101*, 747–759. [CrossRef]
14. Jurgelaitis, M.; Drungilas, V.; Butkienė, R.; Vaičiukynas, E.; Čeponienė, L. Modelling principles for blockchain-based implementation of business or scientific processes. *CEUR Workshop Proc. IVUS* **2019**, *2470*, 43–47.
15. Object Management Group. UML 2.5 Specification. 2015. Available online: http://www.omg.org/spec/UML/2.5/PDF (accessed on 26 November 2020).
16. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; Caro, A.D.; Sethi, M. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1–15. [CrossRef]
17. Lessmann, S.; Baesens, B.; Seow, H.-V.; Thomas, L.C. Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *Eur. J. Oper. Res.* **2015**, *247*, 124–136. [CrossRef]
18. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Perrot, M. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830. Available online: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html (accessed on 26 November 2020).
19. Dua, D.; Graff, C. UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences. 2017. Available online: http://archive.ics.uci.edu/ml (accessed on 26 November 2020).
20. Li, L.; Fan, Y.; Tse, M.; Lin, K.-Y. A review of applications in federated learning. *Comput. Ind. Eng.* **2020**, *149*. [CrossRef]
21. Li, Q.; Wen, Z.; Wu, Z.; Hu, S.; Wang, N.; He, B. Federated learning systems: Vision, hype and reality for data privacy and protection. *arXiv* **2021**, arXiv:1907.09693v4.
22. Diao, E.; Ding, J.; Tarokh, V. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. *arXiv* **2020**, arXiv:2010.01264v1.
23. Kim, H.; Park, J.; Bennis, M.; Kim, S.-L. Blockchained on-device federated learning. *IEEE Commun. Lett.* **2019**, *24*, 1279–1283. [CrossRef]
24. Lu, Y.; Huang, X.; Dai, Y.; Maharjan, S.; Zhang, Y. Blockchain and federated learning for privacy-preserved data sharing in industrial IoT. *IEEE Trans. Ind. Inform.* **2019**, *16*, 4177–4186. [CrossRef]
25. Kuo, T.-T.; Ohno-Machado, L. ModelChain: Decentralized privacy-preserving healthcare predictive modeling framework on private blockchain networks. *arXiv* **2018**, arXiv:1802.01746.
26. Kang, J.; Xiong, Z.; Niyato, D.; Zou, Y.; Zhang, Y.; Guizani, M. Reliable federated learning for mobile networks. *IEEE Wirel. Commun.* **2020**, *27*. [CrossRef]
27. Lu, Y.; Huang, X.; Zhang, K.; Maharjan, S.; Zhang, Y. Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles. *IEEE Trans. Veh. Technol.* **2020**, *69*. [CrossRef]
28. Mendis, G.J.; Wu, Y.; Wei, J.; Sabounchi, M.; Roche, R. Blockchain as a service: A decentralized and secure computing paradigm. *arXiv* **2018**, arXiv:1807.02515.
29. Zou, W.; Lo, D.; Kochhar, P.S.; Le, X.-B.D.; Xia, X.; Feng, Y.; Xu, B. Smart contract development: Challenges and opportunities. *IEEE Trans. Softw. Eng.* **2019**. [CrossRef]
30. Rocha, H.; Ducasse, S. Preliminary steps towards modeling blockchain oriented software. In Proceedings of the IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, Gothenburg, Sweden, 27 May–3 June 2018; pp. 52–57.
31. García-Bañuelos, L.; Ponomarev, A.; Dumas, M.; Weber, I. Optimized execution of business processes on blockchain. In Proceedings of the International Conference on Business Process Management, Barcelona, Spain, 10–15 September 2017; pp. 130–146.
32. Lugan, S.; Desbordes, P.; Brion, E.; Tormo, L.X.; Legay, A.; Macq, B. Secure architectures implementing trusted coalitions for blockchained distributed learning (TCLearn). *IEEE Access* **2019**, *7*, 181789–181799. [CrossRef]
33. Marchesi, M.; Marchesi, L.; Tonelli, R. An agile software engineering method to design blockchain applications. In *Proceedings of the 14th Central and Eastern European Software Engineering Conference*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1–8. [CrossRef]

34. Package Shim—GoDoc. Available online: https://godoc.org/github.com/hyperledger/fabric-chaincode-go/shim (accessed on 26 November 2020).
35. Package http—GoDoc. Available online: https://godoc.org/net/http/ (accessed on 26 November 2020).
36. Merkel, D. Docker: Lightweight Linux containers for consistent development and deployment. *Linux J.* **2014**, *2014*, 2.
37. Nasir, Q.; Qasse, I.A.; Talib, M.A.; Nassif, A.B. Performance analysis of hyperledger fabric platforms. *Secur. Commun. Netw.* **2018**. [CrossRef]
38. DiPaolo, C. Cdipaolo/goml: On-line Machine Learning in Go (And So Much More). Available online: https://github.com/cdipaolo/goml (accessed on 26 November 2020).
39. Bandprotocol. Available online: https://bandprotocol.com/ (accessed on 26 November 2020).
40. Linux Foundation. Hyperledger Avalon Proposal. Available online: https://wiki.hyperledger.org/display/TSC/Hyperledger+Avalon+Proposal (accessed on 26 November 2020).
41. Yamashita, K.; Nomura, Y.; Zhou, E.; Pi, B.; Jun, S. Potential risks of hyperledger fabric smart contracts. In Proceedings of the IEEE International Workshop on Blockchain Oriented Software Engineering, Hangzhou, China, 24 February 2019; pp. 1–10. [CrossRef]