OmniLytics: A Blockchain-based Secure Data Market for Decentralized Machine Learning

Jiacheng Liang[1,2] **Songze Li** [1] **Wensi Jiang** [1] **Bochuan Cao** [2] **Chaoyang He** [3]

[1] Hong Kong University of Science and Technology
[2]University of Electronic Science and Technology of China
[3] University of Southern California
jliangbb@connect.ust.hk, songzeli@ust.hk, wensi.jiang@connect.ust.hk, bochuancao@outlook.com, chaoyang.he@usc.edu

Abstract

We propose OmniLytics, a blockchain-based secure data trading marketplace for machine learning applications. Utilizing OmniLytics, many distributed data owners can contribute their private data to collectively train an ML model requested by some model owners, and receive compensation for data contribution. OmniLytics enables such model training while simultaneously providing 1) model security against curious data owners; 2) data security against the curious model and data owners; 3) resilience to malicious data owners who provide faulty results to poison model training; and 4) resilience to malicious model owners who intend to evade payment. OmniLytics is implemented as a blockchain smart contract to guarantee the atomicity of payment. In OmniLytics, a model owner splits its model into the private and public parts and publishes the public part on the contract. Through the execution of the contract, the participating data owners securely aggregate their locally trained models to update the model owner's public model and receive reimbursement through the contract. We implement a working prototype of OmniLytics on Ethereum blockchain and perform extensive experiments to measure its gas cost, execution time, and model quality under various parameter combinations. For training a CNN on the MNIST dataset, the MO is able to boost its model accuracy from 62% to 83% within 500ms in blockchain processing time.This demonstrates the effectiveness of OmniLytics for practical deployment.

Introduction

With the rapid development of sensing, processing, and storage capabilities of computing devices (e.g., smartphones and IoT devices), the collection and storage of data has been increasingly convenient and cost-effective (Sheng et al. 2013; Cornet and Holden 2018). On the other hand, crowdsourcing big data has been shown to be extremely effective in improving the performance of various fields such as healthcare, smart city, and recommender systems (Raghupathi and Raghupathi 2014; Wang, Kung, and Byrd 2018; Hashem et al. 2016; Al Nuaimi et al. 2015; Yin et al. 2013). The abundant supply of data stored locally at individual nodes and the large demands from data-intensive applications incentivise the development of a data market on which data owners can easily trade the rights of using their data with interested consumers for monetary returns.

Conventionally, a data market is often operated as a centralized service platform that collects data from data owners and sells raw or processed data to the consumers (Mišura and Žagar 2016; Krishnamachari et al. 2018; Niu et al. 2018a). This approach leaves the platform as a single point of security vulnerability for the data market, and corruption on the platform servers may lead to severe security issues including leakage of private data, faulty computation results, and manipulation of data price. A number of recent works have proposed to leverage technologies of decentralized systems like blockchains and smart contracts to tackle the weakness of the centralized implementation (see, e.g., (Özyilmaz, Doğan, and Yurdakul 2018; Duan et al. 2019; Ramachandran, Radhakrishnan, and Krishnamachari 2018; Koutsos et al. 2020)).

To further improve data security, more advanced cryptographic techniques like homomorphic and functional encryption have been utilized to generate analytics over the raw data for consumers to purchase without revealing the data themselves (Duan et al. 2019; Niu et al. 2018b,a; Koutsos et al. 2020). However, these approaches are limited in the following three aspects: 1) data owners upload the encrypted raw data on the blockchain, which leads to permanent loss of the data ownership to any adversarial party with the decryption key; 2) the available analytics are limited to simple operations like linear combinations; 3) they still require a (trusted or untrusted) third party other than the data owner and consumer acting like a broker or service provider to maintain the utility and security of trading session.

In this paper, our goal is to build a *secure* and *broker-free* data market for *general machine learning* applications. Particularly, a model owner would like to crowdsource training data from interested data owners through the data market for improving the quality of its ML models (e.g., a deep neural network for image classification). Moreover, we enforce the following privacy and security requirements: *Model privacy*: parameters of the trained ML model are (mostly) kept private to the model owner itself and are not revealed to other parties; *Data privacy*: a model owner learns nothing about data owners' private data other than the learnt model; *Byzantine resistance*: 1) robust to malicious data owners who intentionally provide incorrect computation results, and 2) robust to malicious model owners who try to evade payment.

Our main contribution is the construction of a novel blockchain-based data market named OmniLytics, which is the first implementation of an Ethereum smart con-

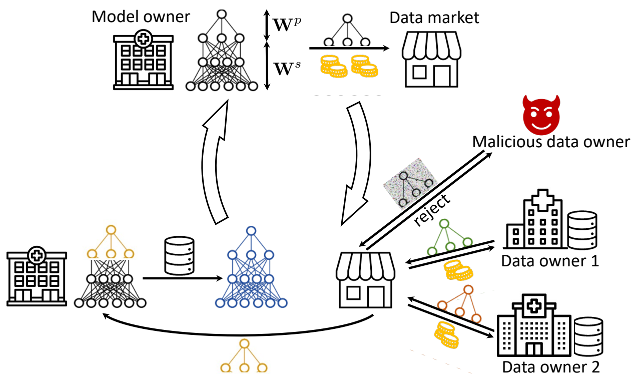

Figure 1: An overview of the operation of the proposed OmniLytics data market. The model owner splits its initial model into a private model $\boldsymbol{W}^s$ and a public model $\boldsymbol{W}^p$, updates the public model through the data market using data owners' private data, and combines the updated public model with the old private model and performs local model adaptation with its private data. The smart contract implementing the data market reimburses the honest data owners who contribute to updating public model, and rejects erroneous results from malicious data owners.

tract (ETH 2021) that simultaneously satisfies all the above security requirements. As shown in Figure 1, during a trading session, the model owner splits its initial model into a private part that contains most of the model parameters, and a public part with a much smaller size, and deploys a smart contract only containing the public model. This model splitting helps to protect the privacy of model parameters, and to reduce the computation complexity hence the gas cost of running the contract. Upon observing the contract, interested data owners retrieve the public model, update the model locally using their private data, and upload the updated models back to the contract for further aggregation. During this process, the data owners hide their local models with pair-wise masks generated according to the secure aggregation protocol (Bonawitz et al. 2017) to further protect from data leakage. Moreover, OmniLytics implements the multi-Krum algorithm (Blanchard et al. 2017a,b) to remove faulty computation results from malicious data owners. Finally, the model owner fetches the updated public model, concatenates it with its private model, and continues to train the combined model with the private data. The training reward is provided by the model owner when deploying the contract and is automatically distributed to honest data owners by the contract, preventing malicious model owners from evading payments after obtaining the trained model.

We implement an Ethereum smart contract SecModelUpdate and the off-chain application of the proposed OmniLytics data market. We conduct extensive experiments to measure the gas cost, execution time of SecModelUpdate, and trained model accuracy under various combinations of system and design parameters. For instance, for a training task on the MNIST dataset, the MO is able to boost its model accuracy from 62% to 83% using OmniLytics, with less than 500ms in blockchain processing time.

Related works

Secure data markets. Traditional data markets require full trust on the centralized service platform, which leaves the platform a single point of failure. To resolve this issue, implementing data market over decentralized systems like blockchains has been recently proposed (Özyilmaz, Doğan, and Yurdakul 2018; Ramachandran, Radhakrishnan, and Krishnamachari 2018; Banerjee and Ruj 2018; Duan et al. 2019; Koutsos et al. 2020). In these implementations, encrypted data are uploaded to the blockchain, on which a smart contract with a funding deposit from the consumer is executed automatically, guaranteeing the atomicity of the payment. To further enhance the data privacy and robustness against malicious behaviors, more advanced techniques like homomorphic encryption, functional encryption, and differential privacy have been utilized to securely generate simple analytics over the raw data for sale (Duan et al. 2019; Niu et al. 2018b; Koutsos et al. 2020), and zero-knowledge proofs and trusted hardware like Intel SGX have been used to guarantee the correctness of the computations (Duan et al. 2019; Niu et al. 2018a; Koutsos et al. 2020).

Federated learning on blockchains. Federated learning (FL) (McMahan et al. 2017) has recently emerged as a privacy-preserving framework for distributed ML, where a set of clients, instead of directly uploading their private data to the cloud, upload the gradients computed from the data, which are aggregated at a cloud server to update a global model. In addition, techniques of masking local gradients like secure aggregation (Bonawitz et al. 2017; Bell et al. 2020; So, Güler, and Avestimehr 2021) and differential privacy (Geyer, Klein, and Nabi 2017; Wei et al. 2020) have been developed for FL to further protect data privacy.

Recently, it has been proposed to execute FL tasks on blockchains to combat server corruption and facilitate a more fair and transparent reward distribution (see, e.g., (Zhao et al. 2020; Liu et al. 2020; Kim et al. 2019; Shayan et al. 2020; Ma et al. 2020)). In (Zhao et al. 2020), an FL on blockchain system is designed for the smart appliance manufactures to learn a ML model from customers' data. Differential privacy techniques are applied to protect data privacy. One major weakness of the design in (Zhao et al. 2020) is that the learnt model of the manufacturer is completely revealed to public, which may not be desirable for the model owner who pays for training. In (Lyu et al. 2020), an FL framework over blockchain named FPPDL was proposed to facilitate fair and privacy-preserving machine learning. While both FPPDL and the proposed OmniLytics data market protect data privacy via encrypting model updates, they differ drastically in the scope of applicability. In FPPDL, a data owner is co-located with a blockchain miner such that a learning participant has to perform local training and block verification. This restricts FPPDL to permissioned blockchains. In contrast, OmniLytics is designed as a plug-and-play smart contract on top of any blockchain platform, without needing to modify miners' operations. This allows OmniLytics to be easily deployed on permissionless public blockchains like Ethereum.

Secure Data Market for Decentralized Machine Learning

We consider a network of many compute nodes (e.g., mobile devices like smartphones, or institutions like hospitals, banks, and companies), each of which has some local data storage and processing power. Some nodes would like to obtain a machine learning model (e.g., to predict certain disease from patients' exam data). We call such node the model owner, denoted by MO. However, as the local data possessed by MO may not be sufficient to train a good model with high accuracy, the MO intends to crowdsource data from other nodes to improve the model quality. In return, the MO compensates the nodes who contribute to the model training with their local private data. We call these nodes as data owners, denoted by DOs.

We consider a threat model where neither the MO nor the DOs can be trusted, i.e., they may attempt to recover the private data of other participants, deviate from the agreed data trading protocol, or intentionally evade payments. The goal of this paper is to design a secure data market that meets the following specific security requirements:

- **Model privacy.** The MO would like to keep its (complete) model secret from the DOs and the data market, because 1) the MO pays to train the model, which may provide it with competitive advantage over other parties; and 2) the model itself may be pre-trained using MO's private data that contains confidential information about MO.
- **Data privacy.** A DO would like to keep its private data secret from the MO and other DOs.
- **Resistance to Byzantine data owners.** The data market should protect the quality of the trained model from being undermined by malicious DOs, who might arbitrarily deviate from the trading protocol, and supply faulty results.
- **Resistance to Byzantine model owner.** The data market should restrain malicious MO from escaping the payment, and enforce that honest DOs who faithfully follow the protocol are properly compensated.

Secure Federated Machine Learning

In this section we review the federated learning (FL) framework (McMahan et al. 2017) and some of its security-enhancing techniques, which constitute the starting point towards our design of a secure data market.

Federated learning framework

A FL network consists of a central server and a group of N clients. Each client k locally has a dataset $\mathcal{S}_k = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_{M_k}, \boldsymbol{y}_{M_k})\}$ of M_k data samples. Each data sample $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ consists of an input vector $\boldsymbol{x}_i \in \mathbb{R}^d$ and its label $\boldsymbol{y}_i \in \mathbb{R}^p$ for some input dimension d and output dimension p. The server aims to train a global model $\boldsymbol{W}$ (e.g., a deep neural network) to minimize the objective function $\mathcal{L}(\boldsymbol{W}) = \sum_{k=1}^{N} p_k \mathcal{L}_k(\boldsymbol{W})$. Here $\mathcal{L}_k(\boldsymbol{W}) = \frac{1}{M_k} \sum_{i=1}^{M_k} \ell(\boldsymbol{W}; (\boldsymbol{x}_i, \boldsymbol{y}_i))$ is the empirical loss at client k for some loss function ℓ. The weight $p_k \triangleq \frac{M_k}{\sum_{k=1}^{N} M_k}$.

The server collaborates with the clients to train the global model using the FedAvg algorithm (McMahan et al. 2017) over multiple iterations. To start with, the server broadcasts an initial model $\boldsymbol{W}^{(0)}$ to all clients. In iteration t, each client k first splits its local dataset $\mathcal{S}_k$ into batches of size B, and starting from the global model received in last iteration $\boldsymbol{W}^{(t-1)}$, runs for E local epochs batched-gradient descent through $\mathcal{S}_k$ to obtain local model $\boldsymbol{W}_k^{(t)}$, and sends it to the server. The server aggregates the received gradients from the N clients and updates the global model as $\boldsymbol{W}^{(t)} = \sum_{k=1}^{N} p_k \boldsymbol{W}_k^{(t)}$.

Secure model aggregation

While the FL framework was designed to protect the privacy of clients' data by having them send merely the models (rather than the actual data) to the server, it was shown in (Zhu and Han 2020; Wang et al. 2019; Geiping et al. 2020) that the server can recover private data $\mathcal{S}_k$ from the local model $\boldsymbol{W}_k^{(t)}$ and the global model $\boldsymbol{W}^{(t-1)}$ via model inversion attacks. To prevent such data leakage, the secure model aggregation protocol was proposed in (Bonawitz et al. 2017) to mask local models before uploading to the server. Specifically, each client k sends a masked model $\widetilde{\boldsymbol{W}}_k = \boldsymbol{W}_k + \boldsymbol{Z}_k$ with some random mask $\boldsymbol{Z}_k$ to the server. The secure aggregation protocol guarantees 1) server cannot deduce any information about each individual $\boldsymbol{W}_k$ and hence the private data $\mathcal{S}_k$; and 2) the masks are generated such that $\sum_{k=1}^{N} \boldsymbol{Z}_k = \boldsymbol{0}$, and server can exactly recover the aggregation of the local model weights $\sum_{k=1}^{N} \widetilde{\boldsymbol{W}}_k = \sum_{k=1}^{N} \boldsymbol{W}_k$.

Byzantine-resilient federated learning

In FL, malicious clients can manipulate the models uploaded to the server to poison the global model (see, e.g., (Bhagoji et al. 2019; Fang et al. 2020)), or plant backdoors in the global model (see, e.g., (Bagdasaryan et al. 2020; Wang et al. 2020)). Current strategies to defend Byzantine clients mainly follow distributed machine learning protocols designed under adversarial settings (Blanchard et al. 2017a; Chen, Su, and Xu 2017; Yin et al. 2018; Yang et al. 2019; Li et al. 2019; Ghosh et al. 2020; Data and Diggavi 2021). The main idea is to take advantage of statistical similarities among models from honest clients, and remove or suppress the adversarial effects introduced by faulty results from Byzantine clients. For instance the Krum algorithm, proposed in (Blanchard et al. 2017a), detects the gradient vectors that have large ℓ_2-distances from the others as being from adversarial clients, and removes them from the aggregation process.

The Proposed Secure Data Market

While the above techniques can help to protect DOs' data privacy, and defend against Byzantine DOs in the federated learning framework, they are insufficient to meet our security requirements of keeping model secret from the DOs, and robustness against malicious MO, for a data market.

We propose to resolve these issues via adopting a private-public model splitting paradigm at MO, and leveraging blockchain technologies to securely collect and reimburse DOs' contributions to train the public model. Specifically, we design OmniLytics, an end-to-end solution to provide a transparent, fair, yet private and secure data market. In the rest of the section we describe in detail the OmniLytics data market, which includes the local computations of the MO and DOs, the design of a smart contract implementing secure model update and reward distribution, and the interactions between the model and data owners with the contract.

Private-public model splitting

We consider the scenario where the MO starts with some initial model $\boldsymbol{W}$. In practice, this is often obtained by the MO through adapting a pre-trained foundation model (e.g., BERT for language understanding or DALL-E for image generation) to its local task and data (Bommasani et al. 2021).

To improve the accuracy of the initial model $\boldsymbol{W}$, the MO pays the DOs to further refine the model using their local private data. Following the local-global model splitting paradigm for federated learning (Liang et al. 2020), the MO first splits the initial model $\boldsymbol{W} = (\boldsymbol{W}^s, \boldsymbol{W}^p)$ into a private model $\boldsymbol{W}^s$ and a public model $\boldsymbol{W}^p$. The splitting is performed such that the public model is much smaller than the private one. For instance, the public model could be the last few layers of a large DNN. After the splitting, the MO submits the public model $\boldsymbol{W}^p$ to the data market to be updated by the participating DOs using their private data. Once the public model is updated over the data market and returned to the MO, the MO concatenates the updated public model and the old private model, and performs another round of model adaptation using local data, improving the model accuracy. Subsequently, as shown in Figure 1, the MO can repeat this pipeline where public model is first updated via the data market, and can then be combined with the private model for local adaptation until no significant improvement is observed on model accuracy.

The approach of splitting MO's model into private and public parts provides the following salient benefits:

- *Model privacy*: most model parameters of the MO are contained in the private model $\boldsymbol{W}^s$, and kept secret from the data market and the DOs;
- *Computation and communication complexities*: as only a small number of model parameters in the public model $\boldsymbol{W}^p$ need to be updated and communicated over the data market, the computation and communication costs at the market and the DOs are significantly smaller, compared with operating over the entire model.

We next describe our design of a smart contract to execute the secure update of the public model $\boldsymbol{W}^p$. Since the contract exclusively deals with the public model, for ease of exposition, we will drop the superscript of $\boldsymbol{W}^p$, and simply refer to the public model as the model.

Smart contract design for secure model update

The proposed OmniLytics data market implements the model update and reward distribution through a smart contract named SecModelUpdate, over an underlying blockchain (e.g., Ethereum). The MO and the DOs are users with dedicated addresses who submit transactions to change the state of SecModelUpdate, while the actual execution of the contract program and the recording of the contract state are performed by blockchain miners and verifiers, who may or may not co-locate with the MO and DOs.

SecModelUpdate executes a single iteration of the model update, with the initial model from the MO, and the model updates collected up to as many as R rounds and from up to N DOs in each round. The use of smart contract enforces automatic payment towards participating DOs whose computation results are considered valid, via some verification mechanism implemented on the contract.

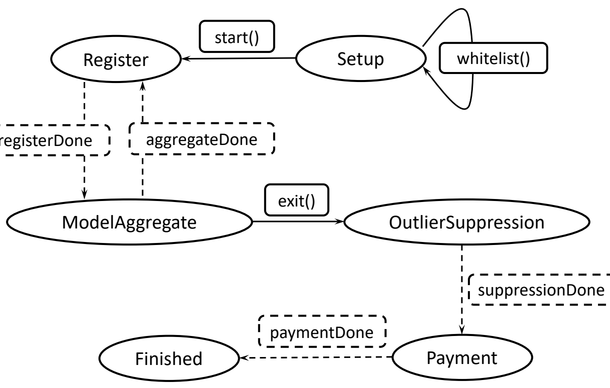

Figure 2: State transition of the smart contract SecModelUpdate. The six states are represented by ovals. State transitions are triggered by either applying a method (in a solid box), or occurrence of an event (in a dashed box).

Model collection and aggregation The MO initializes the data trading session by deploying a smart contract SecModelUpdate with a training reward deposit on the blockchain. As shown in Figure 2, SecModelUpdate transitions between six states, i.e., Setup, Register, ModelAggregate, OutlierSuppression, Payment, and Finished. Upon deployment, SecModelUpdate is in the Setup state with a set of DOs the MO would like to purchase data from specified by a whitelist() method. The MO issues a transaction with the start() method specifying the following public parameters:

- The initial model parameters $\boldsymbol{W}$;
- Minimum number of data points required for each participating DO to compute its local model, denoted by M_0;
- Minimum number of local epochs E;
- Local batch size B;
- Number of rounds to collect models, denoted by R;
- Maximum number of distinct DOs to collect gradients from within each round, denoted by N.

Executing start() moves SecModelUpdate into the Register state, and the contract starts to register for the DOs who intend to participate in the model aggregation for the first round. In each round r, $r = 1, \ldots, R$, after N DOs have registered for this round, SecModelUpdate moves to the ModelAggregate state and starts to collect local computation results from the DOs. For each $k = 1, \ldots, N$, the kth DO registered for round r reads $\boldsymbol{W}$ from the contract, and performs local training using M_0 private data points, as specified in the previous section, obtaining the local model $\boldsymbol{W}_{r,k}$. Next, the DOs in round r execute the secure aggregation protocol to generate the random masks $\boldsymbol{Z}_{r,1}, \ldots, \boldsymbol{Z}_{r,N}$, and the kth DO sends the masked model $\widetilde{\boldsymbol{W}}_{r,k} = \boldsymbol{W}_{r,k} + \boldsymbol{Z}_{r,k}$ to the contract, for $k = 1, \ldots, N$.

After receiving the masked models from all registered DOs in round r, the contract aggregates them to obtain

$$\boldsymbol{A}_r = \frac{1}{N} \sum_{k=1}^{N} \widetilde{\boldsymbol{W}}_{r,k} = \frac{1}{N} \sum_{k=1}^{N} \boldsymbol{W}_{r,k}. \tag{1}$$

If the results of some DOs are still missing after certain amount of time, the aggregation in round r fails and we have $\boldsymbol{A}_r = \emptyset$.

After the aggregation process of round r, SecModelUpdate moves back to the Register state for the next round $r + 1$. Only DOs whose local computation results have not been incorporated in the aggregation results of any previous rounds are eligible to register. By the end of the aggregation process the contract SecModelUpdate obtains a set of results $\{\boldsymbol{A}_r\}_{r \in \mathcal{P}}$, where $\mathcal{P} \subseteq \{1, \ldots, R\}$ denotes the indices of the rounds in which the secure aggregation had been successfully executed. The contract transits to the OutlierSuppression state if R rounds of gradient aggregation have been executed or is manually triggered by the exit() method.

Outlier Suppression During the model collection and aggregation process, Byzantine DOs may upload maliciously generated models which corrupt the aggregation results in $\{\boldsymbol{A}_r\}_{r \in \mathcal{P}}$. SecModelUpdate adopts Byzantine resistance techniques to remove outliers in the OutlierSuppression state. Specifically, we assume that at most $\mu < \frac{1}{2}$ fraction of the aggregation results $\{\boldsymbol{A}_r\}_{r \in \mathcal{P}}$ may be corrupted by malicious DOs. The contract runs a distance-based outlier suppression algorithm m-Krum (Blanchard et al. 2017a,b) (see Algorithm 1 in Appendix A) on $\{\boldsymbol{A}_r\}_{r \in \mathcal{P}}$ to select a subset $\mathcal{P}' \subset \mathcal{P}$ of $|\mathcal{P}'| = m$ aggregation results that are considered computed correctly, for some $m < (1 - 2\mu)|\mathcal{P}| - 2$.

Reward distribution The contract enters the Payment state after the outlier suppression and the set $\mathcal{P}'$ is obtained. The training reward deposited by the MO on the contract is evenly distributed into the accounts of the DOs, whose computation results from rounds in $\mathcal{P}'$ have been accepted.

Public model update After the execution of the SecModelUpdate contract, the MO obtains from the contract the selected aggregation results $\{\boldsymbol{A}_r\}_{r \in \mathcal{P}'}$, and updates its public model to

$$\boldsymbol{W} = \frac{1}{|\mathcal{P}'|} \sum_{r \in \mathcal{P}'} \boldsymbol{A}_r. \tag{2}$$

Security Analysis

In this section, we analyze the security properties of OmniLytics. Particularly, our analysis includes the following four aspects: 1) the confidentiality of the MO's model parameters; 2) the privacy of each DO's data; 3) the security of the model update; and 4) the correct execution of the SecModelUpdate contract.

Confidentiality of model parameters

During the execution of the SecModelUpdate contract, only MO's public model containing a small number of parameters is revealed to the data market and the DOs, while the majority of the model parameters in the private model are kept secret to the MO itself.

Confidentiality of local data

While each data owner can participate in model aggregation in at most one round in the contract SecModelUpdate, its private data is only related to the aggregation result $\boldsymbol{A}_r$ for a single round r. We consider the situation where a subset $\mathcal{C} \subset \{1, \ldots, N\}$ of DOs in round r may collude to infer the private data of some DO in the same round. Based on the privacy guarantee of the secure aggregation protocol (Bonawitz et al. 2017) employed by the contract, we argue that as long as the number of colluding DOs $|\mathcal{C}|$ is less than some secure parameter T[1], no information about other DOs' private data other than the aggregation result $\boldsymbol{A}_r$ can be inferred.

Security of model update

To combat malicious data owners uploading faulty computation results to the contract, we employ the m-Krum algorithm from (Blanchard et al. 2017b) to select the aggregation results from a subset of $\mathcal{P}' \subset \mathcal{P}$, which are considered to be close to the expected value with respect to the underlying data distribution.

We consider the scenario where all DOs have i.i.d. data. In this case, all the aggregation results $\{\boldsymbol{A}_r\}_{r \in \mathcal{P}}$ from the successful rounds of SecModelUpdate are independently and identically distributed. This is because each honest data owner follows the same local training operation of FedAvg (McMahan et al. 2017) with M_0 data points, and each round aggregates results from N DOs. Therefore, according to Proposition 2 and Proposition 3 in (Blanchard et al. 2017b), as long as $|\mathcal{P}'| < (1 - 2\mu)|\mathcal{P}| - 2$, where μ is the maximum fraction of the aggregation results that may be corrupted, the estimated overall gradient in (2) provides a close approximation of the true gradient, which leads to the convergence of the model training.

Ethereum Implementation and Experimental Results

We implement a working prototype of OmniLytics over Ethereum, using Solidity (Sol 2021) to develop the contracts, Python for the off-chain applications and PyTorch (pyt 2021) for the neural network training. We deploy the smart contract SecModelUpdate via the Remix IDE (Rem 2021) on the local Geth Ethereum Testnet (get 2021). We connect the off-chain applications to the smart contracts using the Web3py library (Web 2021) and monitor the created transactions using Etherscan. Each data owner is connected to the Geth network with a unique Ethereum address. We conduct experiments on a machine with AMD R5-5600X CPU @3.70

[1]Each DO's secret keys to generate random masks are secret shared with other DOs such that any T colluding DOs can reveal the secret keys of any data owner.

GHz, Nvidia RTX3070 GPU, 32 GB of Memory and 1 TB SSD.

SecModelUpdate consists of four major operations: Register, PubKeyInteract, ModelAggregate, and OutlierSuppression. Once a SecModelUpdate contract is deployed on Ethereum by a model owner, data owners greedily register with the contract to upload their local computation results until the results are incorporated in the final model aggregation. Data owners within the same aggregation group runs PubKeyInteract to exchange public keys for secure aggregation as done in (Bonawitz et al. 2017). Data owners pay for the transaction fee to upload the results and secure aggregation, which will be reimbursed by the model owner in the Payment phase.

During the secure aggregation process in each round, while the default Pytorch data type is $float32$, we scale each value by 10^8 and aggregate the integer part to improve the precision of the aggregation result. We turn on automatic mining mode and set the mining time to generate a new block to 1 second.

Parallel group aggregation. We perform a system-level optimization such that each round of secure aggregation is carried out in parallel to speed up contract processing. This means that if all the data owners in one round have submitted their local results, the process of secure aggregation would be performed. There is no need to wait until previous rounds are completed. When the last round is completed, model owner can initiate the multi-Krum process to obtain the final result.

Aside from experimental results reported in this section, we provide additional results in Appendix B.

Smart contract measurements

We first measure the gas cost and the execution time of running the SecModelUpdate smart contract, for training a five-layer CNN for MNIST Dataset (mni 2021). We split the CNN model such that the public model contains 5701 parameters, which are 15% of the parameters in the entire model. Since an Ethereum block can accommodate a maximum of 250 parameters, we distribute the task of aggregating the public model into multiple contracts. The following measurement results are taken from one of the contracts, and we expect similar numbers in all other contracts.

Gas consumption We fix the number of data owners in each round $N = 4$ and an estimated fraction of adversarial data owners $\mu = 25\%$, and evaluate the impact of number of groups R on the gas consumption. As shown in Figure 3, for fixed N, the gas costs of Register, PubKeyInteract, and ModelAggregate increase linearly with R. The computational complexity of multi-Krum scales quadratically with R, which leads to a faster increase in the gas cost of OutlierSuppression.

Execution time From the breakdown of the execution time in Tables 1, we observe that the overall contract execution time is dominated by the ModelAggregate and the OutlierSuppression steps. As we increase the number of rounds R for model aggregation, the execution time of the OutlierSuppression step increases rapidly, leading to a significant increase in the overall execution time.

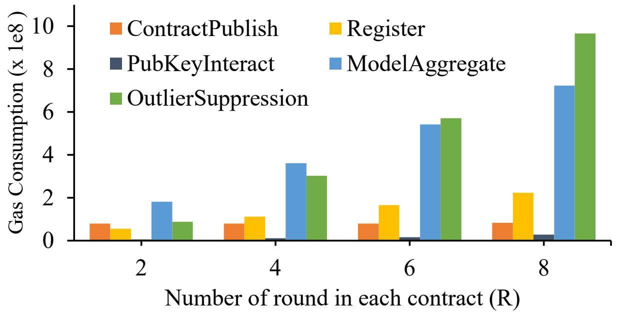

Figure 3: Gas consumption of SecModelUpdate contract for $N = 4$ data owners in each aggregation round and $\mu = 25\%$ adversarial data owners, with different number of rounds.

Table 1: Breakdown of the SecModelUpdate running time (ms) for different number of aggregation rounds.

R	2	4	6	8
Register	1.75	1.85	1.77	1.3
PubKeyInteract	0.53	0.58	0.68	0.71
GradientAggregate	3.69	3.93	3.63	3.94
OutlierSuppression	16.65	43.57	69.18	124.02
Total	22.62	49.93	75.26	130.37

Model performance evaluations

We now evaluate the model accuracy achieved by OmniLytics under various parameter choices. We assume that each DO has the same but a small amount of IID data. MO has an even smaller amount of IID data. We consider the task of training a five-layer CNN network for the MNIST dataset (mni 2021). Parameters for the following experiments are show in Table 2.

Table 2: Design parameters for a image classification task on OmniLytics

Parameter	Explanation	Default
MO$PreEp$	Number of MO pre-train epochs	20
MOEp	Number of MO local training epochs	2
DOEp	Number of DO local training epochs	2
DONum	Number of DOs	64
N	Number of DOs in each round of SecModelUpdate	4
R	Number of rounds of SecModelUpdate	8
PL	Number of layers in public model	3
$Frag$	Fragmentation of DO sparticipating in each iteration	0.5

To understand the impact of DONum on the contract execution time and model accuracy, we fix $N = 4$ and vary DONum. Consequently, the number rounds in SecModelUpdate R changes accordingly. When DONum increases, the accuracy of the model can be improved faster, in terms of number of times the SecModelUpdate contract is invoked. However, when the accuracy reaches a certain threshold, introducing more DOs does not help any more. We plot in Figure 4 the model accuracy as a function of the total amount of blockchain processing time. While having more DOs contributing to the model training allows the MO to invoke less number of SecModelUpdate contract to reach certain model accuracy, the execution time of each contract is also longer.

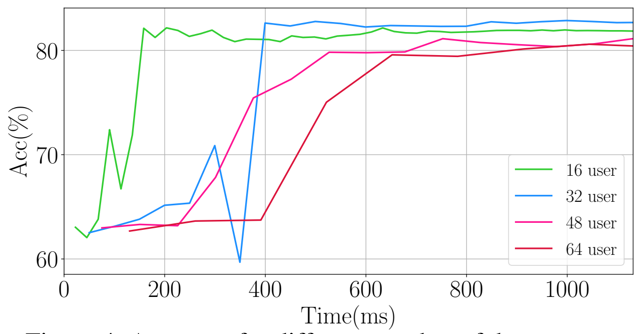

Figure 4: Accuracy for different number of data owners

Via changing PL, we examine the impact of the size and the structure of the public model on the training outcome. We consider different model splitting and public model configurations in Table 3. When $PL = 0$, it means that MO only uses its own small dataset for local training. We take it as our baseline. After many iterations of training, it can only achieve 62% accuracy. When $PL = 100\%$, it means that MO publishes all its model parameters on the blockchain. We observe in Figure 5 a sizable improvement for each increased layer in public model, demonstrating the effectiveness of OmniLytics on data crowdsourcing to help MO's learning task.

Table 3: Considered public model configurations

PL	Public layer	Public/Total
0	/	0%
1	Linear 3	0.71%
2	Linear 3+Linear 2	7.29%
3	Linear 3+Linear 2+ Conv 2	15.54%
4	Linear 3+Linear 2+ Conv 2+ Conv 1	15.96%

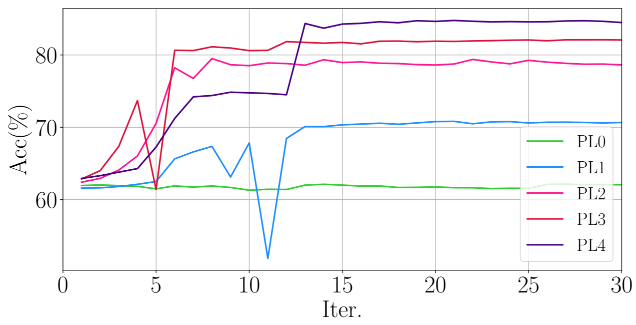

Figure 5: Accuracy for different number of public layers

We also evaluate the influence of MO$PreEp$, DOEp, and MOEp on the model accuracy, and present the results in Table 4. We observe that when MO$PreEp$ is 80, the initial model is easy to be overfit, resulting in a lower accuracy of the final model. Choosing 40 epochs of for pre-training gives the best model accuracy, over various combinations of local training epochs at the MO and DOs.

Table 4: Impact of $Epochs$ (the 1st column is MO$PreEp$).

	DO$Ep = 1$			DO$Ep = 2$		
	MO$Ep = 1$	2	4	1	2	4
20	80.68	81.37	82.46	82.33	70.74	82.46
40	82.7	82.68	82.55	83.0	82.83	82.36
80	72.12	71.86	80.99	82.19	73.28	73.1

Resilience on Byzantine data owners

We simulate Byzantine data owners who instead of uploading local computing results to the contract, simply upload randomly generated data of the same dimension. We set the parameter μ in our outlier suppression mechanism to 0.25 and 0.5, then vary the actual number of Byzantine DOs to observe the ability of OmniLytics to defend Byzantine attacks. Through repeated experiments, we simulate different scenarios where different number of Byzantine DOs join different rounds, and present the average accuracy of the final model in Table 5. We note that setting μ to be a higher value allows OmniLytics to be better resilient to attacks from Byzantine DOs, but it will incur higher complexity in the contract execution, and increased running time and gas cost.

Table 5: Average model accuracy achieved by OmniLytics under Byzantine data owners.

$AttackRate$	$\mu = 0.5$	$\mu = 0.25$
2%	83.12%	76.62%
4%	82.89%	79.72%
6%	81.77%	78.15%
8%	79.97%	75.31%
12%	72.72%	77.91%
16%	56.86%	25.41%

Conclusion

We develop OmniLytics, the first Ethereum smart contract implementation of a secure data market for decentralized machine learning. OmniLytics simultaneously achieves 1) model privacy against curious data owners; 2) data privacy against curious model and data owners; 3) resilience against Byzantine data owners who intend to corrupt model training; and 4) resilience to Byzantine model owner who tries to evade payment. We develop and deploy an Ethereum smart contract SecModelUpdate, and measure its gas cost, execution time, and training performance over various system and design parameters. Through extensive experiments we observe high computation and cost efficiency of the contract, and high accuracy of the trained model, which demonstrate the applicability of OmniLytics as a practical secure data market.

References

2021. Ethereum smart contracts. https://ethereum.org/en/developers/docs/smart-contracts/. Accessed: 2021-06-05.

2021. Official Go implementation of the Ethereum protocol. https://geth.ethereum.org/. Accessed: 2021-06-05.

2021. Pytorch. https://pytorch.org/. Accessed: 2021-06-05.

2021. Remix - Ethereum IDE. https://remix.ethereum.org/. Accessed: 2021-06-05.

2021. Solidity. https://docs.soliditylang.org/. Accessed: 2021-06-05.

2021. THE MNIST DATABASE. http://yann.lecun.com/exdb/mnist/. Accessed: 2021-09-07.

2021. Web3py. https://web3py.readthedocs.io/en/stable/. Accessed: 2021-06-05.

Al Nuaimi, E.; Al Neyadi, H.; Mohamed, N.; and Al-Jaroodi, J. 2015. Applications of big data to smart cities. *Journal of Internet Services and Applications*, 6(1): 1–15.

Bagdasaryan, E.; Veit, A.; Hua, Y.; Estrin, D.; and Shmatikov, V. 2020. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, 2938–2948. PMLR.

Banerjee, P.; and Ruj, S. 2018. Blockchain Enabled Data Marketplace–Design and Challenges. *arXiv preprint arXiv:1811.11462*.

Bell, J. H.; Bonawitz, K. A.; Gascón, A.; Lepoint, T.; and Raykova, M. 2020. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 1253–1269.

Bhagoji, A. N.; Chakraborty, S.; Mittal, P.; and Calo, S. 2019. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, 634–643. PMLR.

Blanchard, P.; El Mhamdi, E. M.; Guerraoui, R.; and Stainer, J. 2017a. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 118–128.

Blanchard, P.; Mhamdi, E. M. E.; Guerraoui, R.; and Stainer, J. 2017b. Byzantine-tolerant machine learning. *arXiv preprint arXiv:1703.02757*.

Bommasani, R.; Hudson, D. A.; Adeli, E.; Altman, R.; Arora, S.; von Arx, S.; Bernstein, M. S.; Bohg, J.; Bosselut, A.; Brunskill, E.; et al. 2021. On the Opportunities and Risks of Foundation Models. *arXiv preprint arXiv:2108.07258*.

Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H. B.; Patel, S.; Ramage, D.; Segal, A.; and Seth, K. 2017. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 1175–1191.

Chen, Y.; Su, L.; and Xu, J. 2017. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2): 1–25.

Cornet, V. P.; and Holden, R. J. 2018. Systematic review of smartphone-based passive sensing for health and wellbeing. *Journal of biomedical informatics*, 77: 120–132.

Data, D.; and Diggavi, S. 2021. Byzantine-resilient high-dimensional SGD with local iterations on heterogeneous data. In *International Conference on Machine Learning*, 2478–2488. PMLR.

Duan, H.; Zheng, Y.; Du, Y.; Zhou, A.; Wang, C.; and Au, M. H. 2019. Aggregating crowd wisdom via blockchain: A private, correct, and robust realization. In *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 1–10. IEEE.

Fang, M.; Cao, X.; Jia, J.; and Gong, N. 2020. Local model poisoning attacks to byzantine-robust federated learning. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 1605–1622.

Geiping, J.; Bauermeister, H.; Dröge, H.; and Moeller, M. 2020. Inverting Gradients–How easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*.

Geyer, R. C.; Klein, T.; and Nabi, M. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*.

Ghosh, A.; Maity, R. K.; Kadhe, S.; Mazumdar, A.; and Ramachandran, K. 2020. Communication efficient and byzantine tolerant distributed learning. In *2020 IEEE International Symposium on Information Theory (ISIT)*, 2545–2550. IEEE.

Hashem, I. A. T.; Chang, V.; Anuar, N. B.; Adewole, K.; Yaqoob, I.; Gani, A.; Ahmed, E.; and Chiroma, H. 2016. The role of big data in smart city. *International Journal of Information Management*, 36(5): 748–758.

Kim, H.; Park, J.; Bennis, M.; and Kim, S.-L. 2019. Blockchained on-device federated learning. *IEEE Communications Letters*, 24(6): 1279–1283.

Koutsos, V.; Papadopoulos, D.; Chatzopoulos, D.; Tarkoma, S.; and Hui, P. 2020. Agora: A privacy-aware data marketplace. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 1211–1212. IEEE.

Krishnamachari, B.; Power, J.; Kim, S. H.; and Shahabi, C. 2018. I3: An iot marketplace for smart communities. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, 498–499.

Li, L.; Xu, W.; Chen, T.; Giannakis, G. B.; and Ling, Q. 2019. RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1544–1551.

Liang, P. P.; Liu, T.; Ziyin, L.; Allen, N. B.; Auerbach, R. P.; Brent, D.; Salakhutdinov, R.; and Morency, L.-P. 2020. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523*.

Liu, Y.; Ai, Z.; Sun, S.; Zhang, S.; Liu, Z.; and Yu, H. 2020. Fedcoin: A peer-to-peer payment system for federated learning. In *Federated Learning*, 125–138. Springer.

Lyu, L.; Yu, J.; Nandakumar, K.; Li, Y.; Ma, X.; Jin, J.; Yu, H.; and Ng, K. S. 2020. Towards fair and privacy-preserving federated deep models. *IEEE Transactions on Parallel and Distributed Systems*, 31(11): 2524–2541.

Ma, C.; Li, J.; Ding, M.; Shi, L.; Wang, T.; Han, Z.; and Poor, H. V. 2020. When Federated Learning Meets Blockchain: A New Distributed Learning Paradigm. *arXiv preprint arXiv:2009.09338*.

McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; and y Arcas, B. A. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, 1273–1282. PMLR.

Mišura, K.; and Žagar, M. 2016. Data marketplace for Internet of Things. In *2016 International Conference on Smart Systems and Technologies (SST)*, 255–260. IEEE.

Niu, C.; Zheng, Z.; Wu, F.; Gao, X.; and Chen, G. 2018a. Achieving data truthfulness and privacy preservation in data markets. *IEEE Transactions on Knowledge and Data Engineering*, 31(1): 105–119.

Niu, C.; Zheng, Z.; Wu, F.; Tang, S.; Gao, X.; and Chen, G. 2018b. Unlocking the value of privacy: Trading aggregate statistics over private correlated data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2031–2040.

Özyilmaz, K. R.; Doğan, M.; and Yurdakul, A. 2018. IDMoB: IoT data marketplace on blockchain. In *2018 crypto valley conference on blockchain technology (CVCBT)*, 11–19. IEEE.

Raghupathi, W.; and Raghupathi, V. 2014. Big data analytics in healthcare: promise and potential. *Health information science and systems*, 2(1): 1–10.

Ramachandran, G. S.; Radhakrishnan, R.; and Krishnamachari, B. 2018. Towards a decentralized data marketplace for smart cities. In *2018 IEEE International Smart Cities Conference (ISC2)*, 1–8. IEEE.

Shayan, M.; Fung, C.; Yoon, C. J.; and Beschastnikh, I. 2020. Biscotti: A Blockchain System for Private and Secure Federated Learning. *IEEE Transactions on Parallel and Distributed Systems*.

Sheng, X.; Tang, J.; Xiao, X.; and Xue, G. 2013. Sensing as a service: Challenges, solutions and future directions. *IEEE Sensors journal*, 13(10): 3733–3741.

So, J.; Güler, B.; and Avestimehr, A. S. 2021. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1): 479–489.

Wang, H.; Sreenivasan, K.; Rajput, S.; Vishwakarma, H.; Agarwal, S.; Sohn, J.-y.; Lee, K.; and Papailiopoulos, D. 2020. Attack of the Tails: Yes, You Really Can Backdoor Federated Learning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 16070–16084.

Wang, Y.; Kung, L.; and Byrd, T. A. 2018. Big data analytics: Understanding its capabilities and potential benefits for healthcare organizations. *Technological Forecasting and Social Change*, 126: 3–13.

Wang, Z.; Song, M.; Zhang, Z.; Song, Y.; Wang, Q.; and Qi, H. 2019. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, 2512–2520. IEEE.

Wei, K.; Li, J.; Ding, M.; Ma, C.; Yang, H. H.; Farokhi, F.; Jin, S.; Quek, T. Q.; and Poor, H. V. 2020. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15: 3454–3469.

Yang, H.; Zhang, X.; Fang, M.; and Liu, J. 2019. Byzantine-resilient stochastic gradient descent for distributed learning: A lipschitz-inspired coordinate-wise median approach. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, 5832–5837. IEEE.

Yin, D.; Chen, Y.; Kannan, R.; and Bartlett, P. 2018. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, 5650–5659. PMLR.

Yin, H.; Sun, Y.; Cui, B.; Hu, Z.; and Chen, L. 2013. Lcars: a location-content-aware recommender system. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 221–229.

Zhao, Y.; Zhao, J.; Jiang, L.; Tan, R.; Niyato, D.; Li, Z.; Lyu, L.; and Liu, Y. 2020. Privacy-preserving blockchain-based federated learning for IoT devices. *IEEE Internet of Things Journal*.

Zhu, L.; and Han, S. 2020. Deep leakage from gradients. In *Federated learning*, 17–31. Springer.

Appendix A
Pseudocode of m-Krum

Algorithm 1: m-Krum

```
Input: $\{\boldsymbol{A}_r\}_{r\in\mathcal{P}}$: aggregation results of successfully executed rounds, $\mu$: fraction of rounds whose result are corrupted
Output: $\{\boldsymbol{A}_r\}_{r\in\mathcal{P}'}$ with $|\mathcal{P}'| = m$
1:  $\mathcal{T} = \mathcal{P}, \mathcal{P}' = \emptyset$
2:  for $i = 1, \ldots, m$ do
3:      for $r \in \mathcal{T}$ do
4:          neighbors $= |\mathcal{T}| - \mu|\mathcal{P}| - 2$ closest ($\ell_2$ distance) vectors to $\boldsymbol{A}_r$
5:          $S(r) = \sum_{\boldsymbol{A}\in\text{neighbors}} ||\boldsymbol{A}_r - \boldsymbol{A}||^2$
6:      end for
7:      $r^* = \arg\min_r S(r)$
8:      $\mathcal{T}$.remove($r^*$)
9:      $\mathcal{P}'$.add($r^*$)
10: end for
11: return $\{\boldsymbol{A}_r\}_{r\in\mathcal{P}'}$
```

Appendix B
Additional Experiment Results

Smart contract measurements

Gas consumption We fix the number of ModelAggregate rounds $R = 6$ and an estimated fraction of adversarial data owners $\mu = 25\%$, and measure the gas cost of the SecModelUpdate contract for different number of DOs in each round (N). We observe in Figure 6 that as N varies from 2 to 8, the total gas consumption of Register, PubKeyInteract, and ModelAggregate increases linearly N. In contrast, since the number of rounds R has not changed, the computation complexity of the multi-Krum algorithm does not change, and the gas cost of OutlierSuppression stays almost constant.

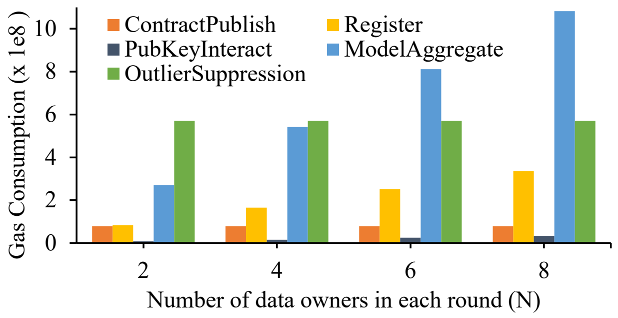

Figure 6: Gas consumption of the SecModelUpdate contract for $R = 6$ aggregation rounds and $\mu = 25\%$ adversarial data owners, for different number of data owners in each round.

Execution time We present additional measurement results on the execution time of the SecModelUpdate contract in Table 6. We note that as we increase the number of data owners in each round, the total execution time increases mildly as the execution time of the bottleneck operation OutlierSuppression is not significantly affected by N.

Table 6: Breakdown of the SecModelUpdate running time (ms) for different number of data owners in each round.

N	2	4	6	8
Register	1.73	1.77	1.9	1.85
PubKeyInteract	0.65	0.68	0.67	0.69
GradientAggregate	3.43	3.63	3.65	3.85
OutlierSuppression	67.78	69.18	72.53	73.43
Total	73.59	75.26	78.76	79.81