PrivFL: Practical Privacy-preserving Federated Regressions on High-dimensional Data over Mobile Networks

Kalikinkar Mandal
University of Waterloo
Waterloo, Ontario
kmandal@uwaterloo.ca

Guang Gong
University of Waterloo
Waterloo, Ontario
ggong@uwaterloo.ca

ABSTRACT

Federated Learning (FL) enables a large number of users to jointly learn a shared machine learning (ML) model, coordinated by a centralized server, where the data is distributed across multiple devices. This approach enables the server or users to train and learn an ML model using gradient descent, while keeping all the training data on users' devices. We consider training an ML model over a mobile network where *user dropout* is a common phenomenon. Although federated learning was aimed at reducing data privacy risks, the ML model privacy has not received much attention.

In this work, we present PrivFL, a privacy-preserving system for training (predictive) linear and logistic regression models and oblivious predictions in the federated setting, while guaranteeing data and model privacy as well as ensuring robustness to users dropping out in the network. We design two privacy-preserving protocols for training linear and logistic regression models based on an additive homomorphic encryption (HE) scheme and an aggregation protocol. Exploiting the training algorithm of federated learning, at the core of our training protocols is a secure multiparty global gradient computation on alive users' data. We analyze the security of our training protocols against semi-honest adversaries. As long as the aggregation protocol is secure under the aggregation privacy game and the additive HE scheme is semantically secure, PrivFL guarantees the users' data privacy against the server, and the server's regression model privacy against the users. We demonstrate the performance of PrivFL on real-world datasets and show its applicability in the federated learning system.

KEYWORDS

Privacy-preserving computations, Predictive analysis, Federated learning, Machine learning

1 INTRODUCTION

Due to their powerful capabilities, machine learning (ML) algorithms are deployed in various applications, from mobile applications to massive-scale data centers for serving various tasks such as predictive analysis, classification, and clustering. Companies such as browser, telecom, web services, edge computing, and advertisement collect a huge amount of data from users to learn ML models for improving quality of services and user experiences, and for providing intelligent services. Predictive analytics is a fundamental task in machine learning, which has many applications ranging from advertisement analytics to financial modeling, supply chain analysis, to health analytics. A common approach to the data analytics is that users send their data to a centralized server where the server executes machine learning algorithms on collected data. A major drawback of this approach is that the transparency of data processing at the server is not clear, and such centralized servers (or Artificial Intelligence (AI) platforms) are easy targets for hackers (e.g., AI.type [1],). Recently the European Union's General Data Protection Regulation (GDPR) mandates companies to exhibit transparency in handling personal data, data processing, etc.

Federated learning [6, 46] is a promising distributed machine learning approach, and its goal is to collaboratively learn a shared model, coordinated by a (centralized) server, while the training data remains on user devices. In federated learning, an ML model on users' data is learnt in an iterative way, which has four steps: 1) a set of users is chosen by the server to compute an updated model; 2) each user computes an updated model on its local data; 3) the updated local models are sent to the server; and 4) the server aggregates these local models to construct a global model. With the advancement of 5G, the federated learning approach will be an attractive solution for machine leaning in edge computing.

With the growing concern of data privacy, federated learning aimed at reducing data privacy risks by avoiding storing data at a centralized server. In privacy-preserving machine learning, two major privacy concerns are the user input privacy and the ML model privacy [23, 56]. However, the model privacy in the federated learning setting has not received much attention. Besides the model leaking sensitive information about training data, the model privacy must also be protected against unauthorized use or misuse. For instance, a *covert user* belonging to one organization participating in a federated training process may disclose the trained model to another organization, which may lead to obtaining a better model by further training it on their dataset or may use it as a prediction service to do business. Although the training data remains on the device, we have found certain scenarios in which the local gradient computation *leaks private information* about locally stored data (see Section 3.2).

In this work, we consider a federated machine learning setting in which a server that coordinates the machine learning process wishes to learn an ML model on a joint dataset belonging to a set of mobile users. We consider both the mobile users' data privacy against the server, and the model privacy against the users in the federated setting. A *user dropout* is an important consideration in mobile applications, recently considered in [7, 44]. Especially, during the training phase which is a computationally intensive and time consuming task, it can happen any time from the network. Our goal is to design secure and dropout-robust training protocols for predictive analytics for mobile applications where, in this work, we consider two fundamental regression models, namely linear/ridge regression and logistic regression.

In a recent work, the authors of [7] considered a general problem of secure aggregation for privacy-preserving machine learning in the federated setting. In a follow-up work, the authors of [44] presented an improved protocol for secure aggregation. No complete training protocol for any machine learning algorithm is developed in [7, 44]. Privacy-preserving training for linear and logistic regressions is not a new problem. Secure training protocols for linear regression have been proposed, e.g., in [24, 49] where, unlike ours, a set of users participating in the training process is connected on a stable network. Secure training protocols for Machine-Learning-as-a-Service (MLaaS) for logistic regression are proposed in [2, 8, 39, 63]. Although several solutions have been proposed for linear and logistic regressions using somewhat or fully homomorphic encryption (SWHE or FHE) [26], garbled circuit [61], or hybrid techniques e.g., in [47], until now, the regression model training over mobile networks has not been considered, under the scenario of users dropping out. Moreover, the suitability of expensive cryptographic primitives such as SWHE or FHE on mobile devices has not been studied. Our goal is to design protocols using lightweight crypto-primitives and schemes suitable for mobile devices in both training and prediction phases.

1.1 Our Contributions

We design, analyze, and evaluate PrivFL, a system for privacy-preserving training and oblivious prediction of regression models, namely linear/ridge regression and logistic regression in the federated setting.

Dropout-robust regression training protocols. We design two privacy-preserving protocols – one is for a linear regression and another is for a logistic regression – for training a regression model over a mobile network while providing robustness in the event of users dropping out. In a nutshell, our privacy-preserving protocol for multiparty regression training consists of multiple (parallel) two-party shared local gradient computation protocols, followed by a global gradient share-reconstruction protocol (see Section 4.2). In our protocol, the users and the server execute the following three steps: 1) a shared local gradient computation protocol is run between the server and a user to securely compute two (additive) shares of the local gradient on the user's data to prevent the input leakage, even if the user has a single data point; 2) the server and all alive users execute an aggregation protocol to construct one share of the global gradient; and 3) the server computes the second share of the global gradient from its local gradient shares. This offers a great flexibility for computing the global gradients robustly. Our regression training protocols are developed using an additive homomorphic encryption scheme and a secure aggregation protocol built using practical crypto-primitives. We also show how to obliviously compute regression models for prediction services for the future use of the trained models by the users.

Security. We prove the security of the training protocol, in three different threat models, against semi-honest adversaries in the simulation paradigm. As our privacy-preserving training protocol is built upon several subprotocols, we first prove the security of the shared local gradient computation protocol for linear and logistic regression models. We formally show that the security of the training protocol is based on the semantic security of the additive HE scheme that guarantees the server's model privacy and the aggregation privacy game, as defined in [11], which ensures users' data privacy.

Experimental evaluation. We implement and evaluate the efficiency of PrivFL for training linear and logistic regression models on eleven real-world datasets from the UCI ML repository [18]. In our experiment, we use the Joye-Libert (JL) cryptosystem [33] to realize the additive HE scheme as its ciphertext expansion is 2× lesser and its ciphertext operations faster, when compared to Paillier's [51]. We implement an aggregation protocol that is a compilation of the protocols of [7] and [44]. As machine learning algorithms work on floating-point numbers, we show how to encode floating-point numbers for cryptographic operations, and how to decode results obtained after applying crypto-operations. For training linear (resp. logistic) regression models, we perform experiments on 6 (resp. 5) different datasets of various sizes and dimensions. The accuracy of the trained models obtained by our protocols is very close to that of the model trained using `sklearn` (no security) [52]. We present the benchmark results for the execution time, data transfer, and storage overhead for cryptographic operations to train regression models. We provide a comparison of our scheme with other approaches in Section 6.

2 PRELIMINARIES

Here, we briefly describe the regression algorithms, namely linear, and logistic regressions, federated learning, and the cryptographic schemes and protocols that we use to build our new protocols.

Basic notations. We denote the message space by $\mathbb{Z}_{2^k}$, a ring of 2^k elements, and E by an additive HE encryption scheme. $\mathbf{x} = (x_1, \cdots, x_n), x_i \in \mathbb{Z}_{2^k}$ denotes an n dimensional vector over $\mathbb{Z}_{2^k}$.

- For $\mathbf{x} = (x_1, \cdots, x_n)$, $\mathrm{E}(\mathbf{x}) = (\mathrm{E}(x_1), \cdots, \mathrm{E}(x_n))$.
- $\mathrm{E}(\mathbf{x}) \cdot \mathrm{E}(\mathbf{y}) = (\mathrm{E}(x_1) \cdot \mathrm{E}(y_1), \cdots, \mathrm{E}(x_n) \cdot \mathrm{E}(y_n))$
- $\mathrm{E}(\mathbf{x})^{\mathbf{y}} = (\mathrm{E}(x_1 y_1), \cdots, \mathrm{E}(x_n y_n))$
- For $\boldsymbol{\theta} = (\theta_0, \theta_1, \cdots, \theta_n)$, $\boldsymbol{\theta} \cdot \mathbf{x} = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$.

2.1 Overview of Regression Algorithms

We now describe three regression algorithms and the gradient descent algorithm for the training process. Let $\mathcal{D} = \left\{(\mathbf{x}^{(i)}, y^{(i)})\right\}_{i=1}^{d}$ be a training dataset with labeled output $y^{(i)} = h(\boldsymbol{\theta}, \mathbf{x}^{(i)})$ and input $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)})$, where h is the regression algorithm, $\boldsymbol{\theta}$ is the model, and n is the dimension of $\mathbf{x}$. The task in the training process is to learn the model $\boldsymbol{\theta}$, given $\mathcal{D}$ and h.

Gradient Descent Algorithm. Gradient descent (GD) is an iterative optimization algorithm for minimizing a cost function. Given a dataset $\mathcal{D}$, the cost function is defined as $J(\boldsymbol{\theta}) = \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x},y) \in \mathcal{B}} C(\boldsymbol{\theta}, (\mathbf{x}, y))$ where $C(, \cdot,)$ is a cost function and $\mathcal{B} \subseteq \mathcal{D}$. An optimal $\boldsymbol{\theta}$ is computed iteratively as

$$\theta^{i+1} \leftarrow \theta^{i} - \eta \nabla J(\theta^{i}), i \geq 0$$

where $\boldsymbol{\theta}^0$ is initialized with a random value or all-zero, η is the learning rate, and ∇J is the gradient of J over $\mathcal{B}$.

Federated Learning. Federated learning enables a large number of users to collaboratively learn a model while keeping all training data on their devices where the model is trained under the coordination of a central server [41, 45]. We assume that there are m users, denoted by P_i, with a set of data points, denoted by $\mathcal{D}_i$. The cost function for the joint dataset $\mathcal{D} = \cup_{i=1}^{m} \mathcal{D}_i$ is given by $J(\boldsymbol{\theta}) = \sum_{i=1}^{m} \frac{d_i}{d} F_i(\boldsymbol{\theta})$ where $F_i(\boldsymbol{\theta}) = \frac{1}{d_i} \sum_{(\mathbf{x},y)\in\mathcal{D}_i} C(\boldsymbol{\theta}, (\mathbf{x}, y))$ and $d = \sum_{i=1}^{m} d_i$. The model update on the total training set $\mathcal{D}$ is performed as [45]

$$\boldsymbol{\theta}^{i+1} \leftarrow \boldsymbol{\theta}^{i} - \eta \sum_{j=1}^{m} \frac{d_j}{d} \nabla F_j(\boldsymbol{\theta}^i), i \geq 0$$

where ∇F_j is the *local* gradient value on $\mathcal{D}_j$. In federated averaging [45], the users are chosen randomly.

Linear Regression. For a linear regression model, the function $h(\cdot,\cdot)$ is an affine function, which is defined as $h(\boldsymbol{\theta}, \mathbf{x}^{(i)}) = \boldsymbol{\theta} \cdot \mathbf{x}^{(i)} = \theta_0 + \sum_{j=1}^{n} \theta_j x_j^{(i)}$. The model parameter $\boldsymbol{\theta}$ is obtained by optimizing the following cost function

$$J(\boldsymbol{\theta}) = \frac{1}{d} \sum_{i=1}^{d} (h(\boldsymbol{\theta}, \mathbf{x}^{(i)}) - y^{(i)})^2.$$

Ridge Regression. For the ridge regression, the function h is the same as the function of linear regression, but the model parameter is obtained by optimizing the following cost function

$$J(\boldsymbol{\theta}) = \frac{1}{d} \sum_{i=1}^{d} (h(\boldsymbol{\theta}, \mathbf{x}^{(i)}) - y^{(i)})^2 + \lambda ||\boldsymbol{\theta}||^2$$

where λ is the regularization parameter.

Logistic Regression. The logistic regression is a binary linear classifier, which maps an input $\mathbf{x}^{(i)}$ from the feature space to a value in $[0, 1]$ as follows:

$$h(\boldsymbol{\theta}, \mathbf{x}^{(i)}) = \sigma(\boldsymbol{\theta} \cdot \mathbf{x}^{(i)}) = \frac{1}{1 + e^{-(\sum_{i=j}^{n} \theta_i x_j^{(i)} + \theta_0)}}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. The binary class is decided based on a threshold value.

For the logistic regression, the model parameter $\boldsymbol{\theta}$ is obtained by optimizing the following log-likelihood cost function

$$J(\boldsymbol{\theta}) = -\frac{1}{d} \sum_{i=1}^{d} y^{(i)} \log(h(\boldsymbol{\theta}, \mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h(\boldsymbol{\theta}, \mathbf{x}^{(i)})).$$

2.2 Multiparty Federated Regression Training and Oblivious Prediction

Multiparty Gradient Computation. Assume that there are m users, and each user has a dataset $\mathcal{D}_i, 1 \leq i \leq m$. A server wishes to learn a regression model $\boldsymbol{\theta}$ on the joint dataset $\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_m = \left\{(\mathbf{x}^{(i)}, y^{(i)})\right\}$ (horizontally distributed). To train a model based on the gradient descent algorithm, the server jointly computes the gradient $\boldsymbol{\omega}$ on $\mathcal{D}$ as

$$\boldsymbol{\omega} = (\omega_0, \omega_1, \cdots, \omega_n) = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathtt{GD}(\boldsymbol{\theta}, \mathcal{D}, d)$$

where $d = |\mathcal{D}|$, $\omega_0 = \sum_{i=1}^{d} e^{(i)}$, $\omega_j = \sum_{i=1}^{d} e^{(i)} x_j^{(i)}, 1 \leq j \leq n$, $e^{(i)} = h(\boldsymbol{\theta}, \mathbf{x}^{(i)}) - y^{(i)}$ and h is a (linear, ridge, or logistic) regression function. We denote the gradient computation on a single data point $(\mathbf{x}^{(i)}, y^{(i)})$ by $\mathbf{err}^{(i)} = f(\boldsymbol{\theta}, (\mathbf{x}^{(i)}, y^{(i)})) = (e^{(i)}, e^{(i)} x_1^{(i)}, \cdots, e^{(i)} x_n^{(i)})$ with $e^{(i)} = h(\boldsymbol{\theta}, \mathbf{x}^{(i)}) - y^{(i)}$. Knowing the model $\boldsymbol{\theta}$, each user can compute the local gradient $\boldsymbol{\omega}_i$ on $\mathcal{D}_i$ as $\boldsymbol{\omega}_i = \sum_{(\mathbf{x},y)\in\mathcal{D}_i} \mathbf{err}^{(i)} = \sum_{(\mathbf{x},y)\in\mathcal{D}_i} f(\boldsymbol{\theta}, (\mathbf{x}, y))$. Thus, the global gradient can be written as $\boldsymbol{\omega} = \sum_{i=1}^{m} \boldsymbol{\omega}_i = \sum_{j=1}^{d} \mathbf{err}^{(j)}$. Using the gradient $\boldsymbol{\omega}$ over $\mathcal{D}$, the server can update the model as $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\eta}{d} \cdot \boldsymbol{\omega}$ for linear and logistic regressions, and $\boldsymbol{\theta} \leftarrow (1 - 2\lambda\eta)\boldsymbol{\theta} - 2\eta \cdot \boldsymbol{\omega}$ for ridge regression for some regularization parameter λ. Secure gradient computation enables the server to learn only $\boldsymbol{\omega}$, while ensuring users' input ($\mathcal{D}_i$) privacy against the server, and the server's model privacy against the users.

Oblivious Regression Prediction. Oblivious prediction for neural networks was introduced in [42]. An oblivious prediction for regression models is defined as follows. Assume that the server holds a (private) regression model $\boldsymbol{\theta}$, and a user has a private input $\mathbf{x}$ and wishes to learn the predicted value $h(\boldsymbol{\theta}, \mathbf{x})$ for a regression function h with model $\boldsymbol{\theta}$. In an oblivious prediction computation, the user will learn only the predicted value $h(\boldsymbol{\theta}, \mathbf{x})$ without leaking any information about $\mathbf{x}$ to the server, and the user should not learn any information about $\boldsymbol{\theta}$, except what can be learnt from the output.

2.3 Cryptographic Tools

We now provide a description of the cryptographic schemes that we will use to construct our protocols.

Additive Homomorphic Encryption. An additive homomorphic encryption (HE) scheme consists of a tuple of four algorithms, denoted as $\mathsf{HE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ where

- KeyGen: Given a security parameter ν, it generates a pair of private and public keys $(pk, sk) \leftarrow \mathsf{KeyGen}(1^{\nu})$.
- Enc: It is a randomize algorithm, denoted by E, takes the public key pk, a random coin r and the message $m \in \mathbb{Z}_{2^k}$ as input, and outputs a ciphertext $c = \mathrm{E}(pk; m, r)$.
- Eval: It supports the following operations on ciphertexts: 1) Addition: $\mathsf{Add}(\mathrm{E}(x), \mathrm{E}(y)) = \mathrm{E}(x + y)$; and 2) Constant multiplication: $\mathsf{ConstMul}(\mathrm{E}(x), z) = \mathrm{E}(xz)$.
- Dec: The algorithm, denoted by D, takes the secret key sk and a ciphertext c as input, and outputs a plaintext message $m = \mathrm{D}(sk; c)$.

We use an additive HE scheme that is semantically secure, and $\mathsf{Add}(,)$ and $\mathsf{ConstMul}(,)$ are ciphertext multiplication and exponentiation operations, respectively.

Secure Aggregation Protocol. We will use a secure aggregation protocol that can handle the dropout scenario as well as no dropout in an aggregated-sum computation. We call this protocol a *dropout-enabled aggregation* protocol, denoted by π_{DEA}. A dropout-enabled aggregation protocol accepts as input a set of users $\mathcal{U}$, their private inputs $\{x_u\}_{u\in\mathcal{U}}$, the total number of users m, and the threshold security parameter t, and outputs x_{sum}, and a set of *alive* users $\mathcal{U}_a \subseteq \mathcal{U}$ participated in the sum computation, i.e.,

$$(\mathcal{U}_a, x_{sum}) \leftarrow \pi_{\mathrm{DEA}}\Big(\mathcal{U}, \{x_u\}_{u\in\mathcal{U}}, m, t\Big)$$

where $x_{sum} = \sum_{u\in\mathcal{U}_a} x_u$ if $|\mathcal{U}_a| \geq t$, and $\perp$ otherwise. Note that the aggregation scheme is secure under a coalition of up to $(t-1)$ users in the system. Such protocols with the dropout-enabled property have been investigate recently in [7, 44]. For more details, see [7, 44]. These aggregation protocols need to establish a pair of pairwise keys: one key will be used for realizing an authenticated channel, and another key will be used to encrypt private inputs. We provide a brief background on the cryptographic primitives that are needed to construct the π_{DEA} protocol in Appendix A.

3 OUR SYSTEM MODEL AND GOALS

In this section, we describe the system model, its goals, and possible privacy leakage in the existing multiparty gradient computation.

3.1 System Model and Trust

System model. We consider a system in the federated learning setting introduced in [6, 41]. In this model, the system consists of two types of parties: a server and a set of mobile users or parties connected in mobile network, where the server conducts the regression training process on mobile users' data. The users may *dropout* any time from the system, as recently considered in [7, 44].

Assume that there are m mobile users in the system, and each user has a unique identity in $[m] = \{1, 2, \cdots, m\}$. We denote the server by S, and a user by $P_i, i \in [m]$. Each user P_i holds a dataset with high-dimensional points, denoted by $\mathcal{D}_i = \{(\mathbf{x}^{(j)}, y^{(j)})\}$ where $|\mathcal{D}_i| \geq 1$. We have the following system goals:

- **Dropout-robust secure regression training:** The first goal of our system is to enable the server training a regression model $\boldsymbol{\theta}$ (e.g., linear, ridge or logistic) over the combined dataset $\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_m$, i.e., $\boldsymbol{\theta} \leftarrow \mathsf{Traning}(h, \mathcal{D}, m)$, while allowing user dropout any time in the system, and ensuring mobile users' input privacy and the servers' model privacy where h is the regression algorithm (see Section 2.2).
- **Oblivious prediction:** The second goal is to enable a mobile user learn the predicted output $y = h(\boldsymbol{\theta}, \mathbf{x})$ on its private input $\mathbf{x}$, corresponding to a regression model $\boldsymbol{\theta}$, without leaking any information about $\mathbf{x}$ to the server and about $\boldsymbol{\theta}$ to the user.

Figure 1 provides an overview of our system and approach to achieve system goals. The main challenges in the training phase are:

- *Correctness:* For correct inputs of the users, the protocol for the regression training should output the correct model. We claim no correctness of the regression model if any user uses an incorrect input or the server manipulates the model in the training phase.
- *Privacy:* Our system has aimed at protecting users' inputs privacy and the server's model privacy. The server should learn no information about mobile users' private inputs in $\mathcal{D}_i$. Similarly, the users should not learn anything about the model $\boldsymbol{\theta}$, except what they can infer from the output.
- *Efficiency:* As mobile users do not send the private inputs out of the devices, they should perform a minimal work, and the server should perform the majority of work in the training phase. The computational and communication costs of the training protocol should be minimal.

Threat model. In our system, we consider semi-honest adversaries (inside adversary) where a group of mobile users and/or the server compromised by an adversary follow and observe the prescribed actions of the protocol and aim at learning unintended information about $\boldsymbol{\theta}$ or honest users' $\mathcal{D}_i$ from the execution of the protocol. We consider three different threat models: 1) users-only adversary; 2) server-only adversary; and 3) users-server adversary. We assume the training is conducted in a synchronous way (each iteration within a time interval of length Δ), meaning all users use the correct model to compute the local gradient and the server updates the model consistently at every round of the model update.

3.2 Privacy Leakage and Model Privacy

Input privacy leakage from local gradient. When a user holds a single data point $(\mathbf{x}, y)$, i.e., $\mathcal{D}_j = \left\{(\mathbf{x}, y)\right\}$, the gradient computation on $\mathcal{D}_j$, denoted by $\boldsymbol{\omega} = (\omega_0, \omega_1, \cdots, \omega_n)$, leaks information about the private input $\mathbf{x}$ as $\omega_0 = (h(\boldsymbol{\theta}, \mathbf{x}) - y)$ and $\omega_j = (h(\mathbf{x}, \boldsymbol{\theta}) - y)x_j = \omega_0 \cdot x_j, 1 \leq j \leq n$, where h is a regression function. In the cases of stochastic gradient descent and federated averaging algorithm with $|\mathcal{D}_i| = 1$ [45], this directly allows the server to recover x_j from $\boldsymbol{\omega}$ for $\boldsymbol{\omega} \neq \mathbf{0}$.

In the application of smart grid data aggregation, an analysis on aggregation is performed in [11], which experimentally finds privacy leakage in the aggregate-sum when it has a small number of inputs. According to the analysis of [11], $\omega_j = \sum_{l=1}^{d_i}(h(\boldsymbol{\theta}, \mathbf{x}^{(l)}) - y^{(i)})x_j^{(l)}$ and $\omega_0 = \sum_{l=1}^{d_i}(h(\boldsymbol{\theta}, \mathbf{x}^{(l)}) - y^{(l)})$ may leak information about $\mathbf{x}^{(i)}$ when d_i is small. In mobile applications, when mobile devices do not have enough data, it may compromise the privacy of the input dataset $\mathcal{D}_i$. In this work, PrivFL enables mobile users' to train a regression model without leaking their privacy, even when $|\mathcal{D}_i| = 1$ and $d = \sum d_i$ is large enough to protect input privacy.

Model privacy of federated learning. In federated learning, a substantial focus has been put on users' data privacy. The authors of [45] mentioned about achieving the model privacy using differential privacy techniques [20]. However, there is no such concrete proposal, and it is also not known the accuracy of the training process after applying such techniques. PrivFL takes the secure MPC approach to provide the ML model privacy as well as users' data privacy.

4 OUR REGRESSION PROTOCOLS

The key objective of this work is to develop secure training protocols for regression models based on cryptographic primitives suitable for mobile devices as well as robust against the user dropout scenario. As the training process involves a pre-processing of the training data, called the *scaling or normalization* of the dataset, we show how to securely perform this using an aggregation protocol while ensuring the dropout scenario. We start by presenting the secure scaling operation.

4.1 Dropout-robust Secure Scaling Operation

Scaling a dataset is performed by computing the mean and standard deviation of the dataset. Given the set of numbers $\{x_i\}$ of size m, the scaling is performed as $x_i \leftarrow \frac{x_i - \mu}{\sigma}$ where $\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$ and $\sigma = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(x_i - \mu)^2}$. For a high-dimensional dataset $\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_m$ from users $\mathcal{U}$, the scaling operation is performed by scaling individual components of each data point. Note that scaling is performed only on $\mathbf{x}^{(i)}$ in $\mathcal{D}$. Let $\mathbf{x}^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)}\right)$ be an n-dimensional data point. The mean of the $\mathbf{x}^{(i)}$ component of $\mathcal{D}$ of size d is given by $\mu = (\mu_1, \cdots, \mu_n)$ where $\mu_j = \frac{1}{d}\sum_{i=1}^{d} x_j^{(i)}$, and the standard deviation is given by $\sigma = (\sigma_1, \cdots, \sigma_n)$ where $\sigma_j = \sqrt{\frac{1}{d-1}(\sum_{i=1}^{d} {x_j^{(i)}}^2 - (2d-1)\mu_j^2)}, j \in [n]$. Secure computation

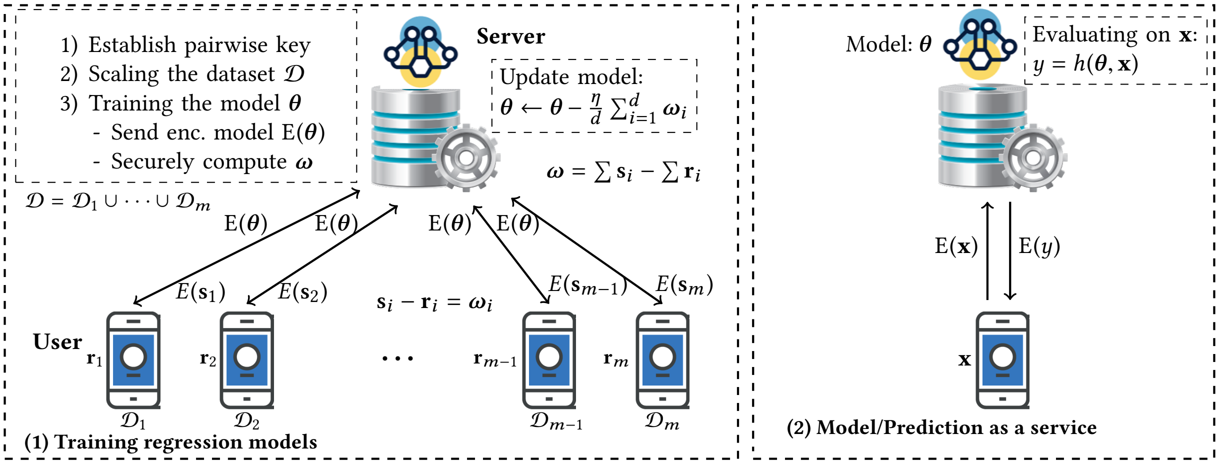

1) Establish pairwise key
2) Scaling the dataset $\mathcal{D}$
3) Training the model $\boldsymbol{\theta}$
- Send enc. model $\mathrm{E}(\boldsymbol{\theta})$
- Securely compute $\boldsymbol{\omega}$
Server
Update model:
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\eta}{d}\sum_{i=1}^{d}\boldsymbol{\omega}_i$
$\boldsymbol{\omega} = \sum \mathbf{s}_i - \sum \mathbf{r}_i$
$\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_m$
$\mathrm{E}(\boldsymbol{\theta})$
$\mathrm{E}(\boldsymbol{\theta})$
$\mathrm{E}(\boldsymbol{\theta})$
$\mathrm{E}(\boldsymbol{\theta})$
$E(\mathbf{s}_1)$
$E(\mathbf{s}_2)$
$\mathbf{s}_i - \mathbf{r}_i = \boldsymbol{\omega}_i$
$E(\mathbf{s}_{m-1})$
$E(\mathbf{s}_m)$
User
$\mathbf{r}_1$
$\mathbf{r}_2$
$\cdots$
$\mathbf{r}_{m-1}$
$\mathbf{r}_m$
$\mathcal{D}_1$
$\mathcal{D}_2$
$\mathcal{D}_{m-1}$
$\mathcal{D}_m$
(1) Training regression models
Model: $\boldsymbol{\theta}$
Evaluating on $\mathbf{x}$:
$y = h(\boldsymbol{\theta}, \mathbf{x})$
$\mathrm{E}(\mathbf{x})$
$\mathrm{E}(y)$
$\mathbf{x}$
(2) Model/Prediction as a service

Figure 1: An overview of the regression training process and prediction as a service over a mobile network in the federated setting. For the training phase, each user holds a dataset $\mathcal{D}_i$, and the server holds an ML model θ. Users and the server jointly compute the global gradient ω. For the prediction service, the user holds a data point and the server holds a trained model.

of basis statistics such as mean and standard deviation has been widely investigated using secret sharing, (labeled) homomorphic encryption, and aggregation protocols, e.g., [4, 5, 13, 37].

As we have considered a scenario of users dropping out, we perform the scaling process using a dropout-enabled aggregation protocol (π_{DEA}), coordinated by the server. Instead of running π_{DEA} twice (once for computing μ and another for σ), our idea is to run π_{DEA} only once on $2n$ dimensional inputs $X^{(i)}$, which we construct from n-dimensional inputs $\mathbf{x}^{(i)}$ as follows. Each user P_i locally computes a single input $X^{(i)}$ on its dataset $\mathcal{D}_i$ as

$$X^{(i)} = \Big(\sum_{\mathbf{x}\in\mathcal{D}_i} x_1, \cdots, \sum_{\mathbf{x}\in\mathcal{D}_i} x_n, \sum_{\mathbf{x}\in\mathcal{D}_i} x_1^2, \cdots, \sum_{\mathbf{x}\in\mathcal{D}_i} x_n^2\Big)$$

where $\mathbf{x} = (x_1, \cdots, x_n)$. This can be viewed as processing multiple-data using a single execution of the protocol. On inputs $\{X^{(i)}, d_i\}$ from users in $\mathcal{U}$ and a security parameter t, the server receives $(\mathcal{U}_a, X) \leftarrow \pi_{\mathrm{DEA}}(\mathcal{U}, \{X^{(i)}, d_i\}, |\mathcal{U}|, t)$ with $|\mathcal{U}| \geq t$ where $X = (X_1, \cdots, X_n, X_{n+1}, \cdots, X_{2n}) = \sum_{u\in\mathcal{U}_a} X^{(u)}$ if $|\mathcal{U}_a| \geq t$, and $\mathcal{U}_a$ are the alive users who participated in the scaling process. If $|\mathcal{U}_a| < t$, abort the protocol. The server computes the mean $\boldsymbol{\mu} = (\mu_1, \cdots, \mu_n)$ and standard deviation $\boldsymbol{\sigma} = (\sigma_1, \cdots, \sigma_n)$ as $\mu_j = \frac{X_j}{d_a}$, and $\sigma_j = \sqrt{\frac{1}{d_a - 1}(X_{n+j} - (2d_a - 1)\mu_j^2)}, j \in [n]$ and $d_a = \sum_{u\in\mathcal{U}_a} d_u$. The server then sends $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ to all users in $\mathcal{U}_a$ through an authenticated channel. Using $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, the users scale their local datasets.

4.2 A High-level Overview of Our Regression Training Protocols

Basis idea. To train a regression model, the users and the server execute a multiparty global gradient computation protocol where the server gives the model and the users provide their local datasets as inputs, and the server obtains an updated model as an output. The novelty of PrivFL's multiparty gradient computation is that the global gradient computation is performed by executing multiple two-party shared local gradient protocols in parallel, followed by executing a global gradient share-reconstruction protocol realized using an aggregation protocol (see Figure 2). As shown in Section 3, sending the local gradient directly to the server may leak users' datasets $\mathcal{D}_i$. Our idea is to prevent such leakage by additively secret-sharing the local gradient $\boldsymbol{\omega}_i$ of a user P_i between the server and the user such that $\mathbf{s}_i - \mathbf{r}_i = \boldsymbol{\omega}_i$, where the server holds the share $\mathbf{s}_i$, and the user P_i holds the share $\mathbf{r}_i$. To prevent leaking the model to the users, the server encrypts the model $\boldsymbol{\theta}$ using an additive homomorphic encryption scheme. In the first phase, each user computes an encrypted share $\mathrm{E}(\mathbf{s}_i)$ of $\boldsymbol{\omega}_i$ from $\mathrm{E}(\boldsymbol{\theta})$ and $\mathcal{D}_i$. This is computed by a protocol called, the *shared local gradient (SLG)* computation protocol. Instead of sending the individual users' shares to the server, the users send their shares in aggregate. For this, the server conducts a share reconstruction process to obtain an aggregate-sum of the shares. This phase is called the *share reconstruction phase.* After computing $\boldsymbol{\omega}$ on alive users' $(\mathcal{U}_a)$ datasets from $\{\mathbf{s}_i\}$ and $\{\mathbf{r}_i\}$ as $\boldsymbol{\omega} = \sum_{u\in\mathcal{U}_a} \mathbf{s}_u - \sum_{u\in\mathcal{U}_a} \mathbf{r}_u$, the server updates the model as $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\eta}{d_a}\boldsymbol{\omega}$ where $d_a = \sum_{u\in\mathcal{U}_a} d_u$. Figure 2 depicts an overview of secure multiparty global gradient computation on the joint dataset $\mathcal{D}$.

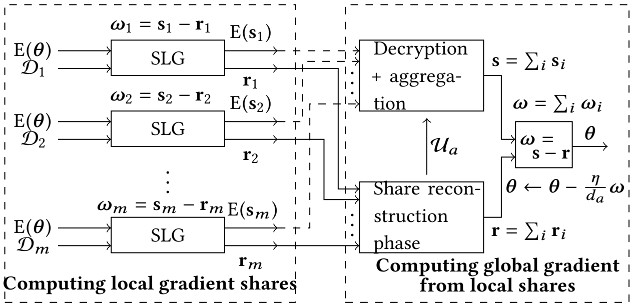

$\mathrm{E}(\boldsymbol{\theta})$
$\mathcal{D}_1$
$\boldsymbol{\omega}_1 = \mathbf{s}_1 - \mathbf{r}_1$
$\mathrm{E}(\mathbf{s}_1)$
SLG
$\mathbf{r}_1$
$\mathrm{E}(\boldsymbol{\theta})$
$\mathcal{D}_2$
$\boldsymbol{\omega}_2 = \mathbf{s}_2 - \mathbf{r}_2$
$\mathrm{E}(\mathbf{s}_2)$
SLG
$\mathbf{r}_2$
$\vdots$
$\mathrm{E}(\boldsymbol{\theta})$
$\mathcal{D}_m$
$\boldsymbol{\omega}_m = \mathbf{s}_m - \mathbf{r}_m$
$\mathrm{E}(\mathbf{s}_m)$
SLG
$\mathbf{r}_m$
Computing local gradient shares
Decryption + aggregation
$\mathcal{U}_a$
Share reconstruction phase
$\mathbf{s} = \sum_i \mathbf{s}_i$
$\boldsymbol{\omega} = \sum_i \boldsymbol{\omega}_i$
$\boldsymbol{\omega} = \mathbf{s} - \mathbf{r}$
$\boldsymbol{\theta}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\eta}{d_a}\boldsymbol{\omega}$
$\mathbf{r} = \sum_i \mathbf{r}_i$
Computing global gradient from local shares

Figure 2: PrivFL's protocol flow of privacy-preserving global gradient computation for regression models.

Advantage. The advantages of the above flow of the protocol are as follows. At the i-th round of the training process, the users participating in the global gradient computation process do not need to know the other participating users ahead of the time. The information about the other participating users will be enough to know only in the share reconstruction phase. This provides a great flexibility in the global gradient computation phase for handling the dropout scenario in mobile applications. Moreover, the server can choose the participating users randomly based on their aliveness in the global gradient computation phase. Note that the server can choose the participating users and let all users know ahead of time, but this will be computationally expensive for both users and the server for handling the dropout scenario. Notice that any two-party protocol can be used to realize the SLG computation. However, we use an additive-HE based technique for regression models to make it communication efficient because of our application scenario.

4.3 Secure Shared Local Gradient Computation Protocols

A secure shared local gradient computation involves a user and the server. We present two protocols for computing the shared local gradient computation: one protocol is for the linear regression (see Figure 3), and another is for the logistic regression (see Figure 4).

Computing the SLG for linear regression. To minimize the number of rounds, the server encrypts the model using an additive HE scheme and sends it to the user P_i so that it can compute two shares of the local gradient $\boldsymbol{\omega}_i$ on its dataset $\mathcal{D}_i$ in one round of communication. Note that only the server holds the private key of the HE scheme. An inner product computation of two vectors is a common task in training and evaluating a linear or logistic regression model. Secure inner product computation has been studied in many settings, e.g., [27, 60]. For the sake of completeness, we provide a homomorphic inner product computation protocol from [27] in Algorithm 1. Figure 3 presents the shared local gradient computation protocol for a linear/ridge regression model.

Algorithm 1 Encrypted Inner Product

```
INPUT: $\mathbf{x}$ and $\mathrm{E}(\boldsymbol{\theta})$
OUTPUT: $\mathrm{E}(\boldsymbol{\theta} \cdot \mathbf{x})$
procedure IP_USER
    $\mathrm{E}(\boldsymbol{\theta} \cdot \mathbf{x}) = \prod_{j=1}^{n} \mathrm{E}(\theta_j)^{x_j} \cdot \mathrm{E}(\theta_0)$
    return $\mathrm{E}(\boldsymbol{\theta} \cdot \mathbf{x})$
end procedure
```

Algorithm 2 Computing Shares of Local Gradient for Linear Regression

```
INPUT: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{p}$ and $\mathrm{E}(\boldsymbol{\theta})$
OUTPUT: $(\mathrm{E}(\mathbf{s}), \mathbf{r})$ s.t. $\mathbf{s} - \mathbf{r} = \sum_{j=1}^{p} \mathbf{err}^{(j)}$
procedure COMP_LINSLG_SHARE
    Randomly generate $\mathbf{r} = (r_0, r_1, \cdots, r_n)$
    Set $\mathbf{t} \leftarrow (0, 0, \cdots, 0)$
    for $i = 1$ to $p$ do
        $\mathrm{E}(e^{(i)}) = \mathrm{IP\_USER}(\mathbf{x}^{(i)}, \mathrm{E}(\boldsymbol{\theta})) \cdot \mathrm{E}(-y^{(i)})$
        $\mathrm{E}(t_0) \leftarrow \mathrm{E}(t_0) \cdot \mathrm{E}(e^{(i)}) = \mathrm{E}(t_0 + e^{(i)})$
        for $j = 1$ to $n$ do
            $\mathrm{E}(t_j) \leftarrow \mathrm{E}(t_j) \cdot \mathrm{E}(e^{(i)})^{x_j^{(i)}} = \mathrm{E}(t_j + e^{(i)} x_j^{(i)})$
        end for
    end for
    Compute $\mathrm{E}(\mathbf{s}) = \mathrm{E}(\mathbf{t}) \cdot \mathrm{E}(\mathbf{r}) = \mathrm{E}(\mathbf{t} + \mathbf{r})$
    return $(\mathrm{E}(\mathbf{s}), \mathbf{r})$
end procedure
```

Secure SLG Protocol: Linear regression (π_{LinSLG})

Server: Regression model $\boldsymbol{\theta}$, **User** (P_i)**:** $\mathcal{D}_i$

Output: Server receives $\mathrm{E}(\mathbf{s}_i)$, P_i receives $\mathbf{r}_i$.

1. Server encrypts the model $\boldsymbol{\theta}$ and sends $\mathrm{E}(\boldsymbol{\theta})$ **to** the user P_i
2. User P_i runs Algorithm 2 on inputs $\mathrm{E}(\boldsymbol{\theta})$ and $\mathcal{D}_i$, and obtains $(\mathrm{E}(\mathbf{s}^i), \mathbf{r}^i)$ such that $\mathbf{s}_i - \mathbf{r}_i = \boldsymbol{\omega}_i = \mathtt{GD}(\boldsymbol{\theta}, \mathcal{D}_i, d_i)$.
3. P_i stores $\mathbf{r}_i$ and sends $\mathrm{E}(\mathbf{s}_i)$ **to** the server.

Figure 3: Secure shared local gradient computation protocol for a linear regression model.

Computing the SLG for logistic regression. The computation of the local gradient involves an evaluation of the sigmoid function on $\boldsymbol{\theta} \cdot \mathbf{x}^{(j)}$. As shown for linear regression, the server and the user can compute $\mathrm{E}(\boldsymbol{\theta} \cdot \mathbf{x}^{(j)})$ in one round of communication. As we used an additive HE scheme, the user and the server need one more round of communication to compute $\mathrm{E}(\sigma_3(\boldsymbol{\theta} \cdot \mathbf{x}^{(j)})), \mathbf{x}^{(j)} \in \mathcal{D}_i$ from $\mathrm{E}(\boldsymbol{\theta} \cdot \mathbf{x}^{(j)})$ where the sigmoid function is approximated by a cubic polynomial, which provides a good tradeoff between the accuracy and the efficiency. Let $\sigma_3(x) = q_0 + q_1 x + q_2 x^2 + q_3 x^3$ be a cubic approximation of $\sigma(x)$ where the coefficients q_i are public. To protect an input $\mathbf{x}$ against the server, the user masks $y = \boldsymbol{\theta} \cdot \mathbf{x}$ as $z = y + r$ by choosing a random value r, and then sends it to the server. The computation of $\sigma_3(y)$ can be expressed as

$$\sigma_3(y) = \sigma_3(z) - \sigma_3(r) - (q_0 + 3q_3 r^3) - 3q_3 r z^2 - (2q_2 r - 6q_3 r^2) y.$$

To reduce computational cost for the users, the server computes $\mathrm{E}(z^2)$ and $\mathrm{E}(\sigma_3(z))$ from $\mathrm{E}(z)$ and sends $(\mathrm{E}(z^2), \mathrm{E}(\sigma_3(z)))$ to the user. Given $\mathrm{E}(y), \mathrm{E}(z^2), \mathrm{E}(\sigma_3(z))$ and r, the user can compute $\mathrm{E}(\sigma_3(y))$ using the homomorphic property of the encryption scheme as

$$\mathrm{E}(z^2)^{-3q_3 r} \mathrm{E}(y)^{-(2q_2 r - 6q_3 r^2)} \mathrm{E}(-(\sigma_3(r) + (q_0 + 3q_3 r^3))) \mathrm{E}(\sigma_3(z)).$$

Once the user has $\{\mathrm{E}(\sigma_3(\boldsymbol{\theta} \cdot \mathbf{x}^{(j)}))\}$, it can compute the local gradient $\boldsymbol{\omega}_i$ on $\mathcal{D}_i$, using the steps described in Algorithm 3. Figure 4 summarizes the shared local gradient computation protocol for a logistic regression model.

Algorithm 3 Computing Shares of Local Gradient for Logistic Regression

```
INPUT: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y_i)\}_{i=1}^{p}$ and $\{\mathrm{E}(\sigma_3(\boldsymbol{\theta} \cdot \mathbf{x}^{(i)}))\}$
OUTPUT: $(\mathrm{E}(\mathbf{s}), \mathbf{r})$ s.t. $\mathbf{s} - \mathbf{r} = \sum_{j=1}^{p} \mathbf{err}^{(j)}$
procedure COMP_LOGSLG_SHARE
    Randomly generate $\mathbf{r} = (r_0, r_1, \cdots, r_n)$
    Set $\mathbf{t} \leftarrow (0, 0, \cdots, 0)$
    for $i = 1$ to $p$ do
        $\mathrm{E}(e^{(i)}) = \mathrm{E}(\sigma_3(\boldsymbol{\theta} \cdot \mathbf{x}^{(i)})) \cdot \mathrm{E}(-y^{(i)})$
        $\mathrm{E}(t_0) \leftarrow \mathrm{E}(t_0) \cdot \mathrm{E}(e^{(i)}) = \mathrm{E}(t_0 + e^{(i)})$
        for $j = 1$ to $n$ do
            $\mathrm{E}(t_j) \leftarrow \mathrm{E}(t_j) \cdot \mathrm{E}(e^{(i)})^{x_j^{(i)}} = \mathrm{E}(t_j + e^{(i)} x_j^{(i)})$
        end for
    end for
    Compute $\mathrm{E}(\mathbf{s}) = \mathrm{E}(\mathbf{t}) \cdot \mathrm{E}(\mathbf{r}) = \mathrm{E}(\mathbf{t} + \mathbf{r})$
    return $(\mathrm{E}(\mathbf{s}), \mathbf{r})$
end procedure
```

Complexity. We measure the computational complexity of π_{LinSLG} and π_{LogSLG} for computing the local gradient in terms of the number of ciphertext multiplications, the number of encryptions, and

```
Secure SLG Protocol: Logistic regression ($\pi_{\text{LogSLG}}$ )
Server: Regression model $\boldsymbol{\theta}$, User ($P_i$): $\mathcal{D}_i$
Output: Server receives $\text{E}(\mathbf{s}_i)$, $P_i$ receives $\mathbf{r}_i$.

1. Server encrypts the model $\boldsymbol{\theta}$ and sends $\text{E}(\boldsymbol{\theta})$ to the user $P_i$
2. $P_i$ randomly generates $\{c_j\}_{j=1}^{d_i}$.
3. $P_i$ computes: for j = 1 to $d_i$ do
     Compute $\text{E}(z_j) \leftarrow \text{IP\_User}(\mathbf{x}^{(j)}, \text{E}(\boldsymbol{\theta})) \cdot \text{E}(c_j)$, where $z_j = \boldsymbol{\theta} \cdot \mathbf{x}^{(j)} + c_j$
   endfor
4. $P_i$ sends $\{\text{E}(z_j)\}$ to the server
5. Server decrypts $\{\text{E}(z_j)\}_{j=1}^{d_i}\}$ and computes
   $\{\text{E}(z_j^2), \text{E}(h_q(z_j))\}_{j=1}^{d_i}$ and sends to $P_i$
6. $P_i$ computes: for j = 1 to $d_i$ do
     $t = \text{E}(-(h_q(c_j) + (q_0 + 3q_3c_j^3))) \cdot \text{E}(\boldsymbol{\theta} \cdot \mathbf{x}^{(j)})^{-(2q_2c_j - 6q_3c_j^2)}$
     $\text{E}(\sigma_3(\boldsymbol{\theta} \cdot \mathbf{x}^{(j)})) = \text{E}(\sigma_3(z_j))\text{E}(z_j^2)^{-3q_3c_j} \cdot t$
   endfor
7. $P_i$ runs Algorithm 3 on inputs $\{\text{E}(\sigma_3(\boldsymbol{\theta} \cdot \mathbf{x}^{(j)}))\}$ and $\mathcal{D}_i$, and obtains $(\text{E}(\mathbf{s}_i), \mathbf{r}_i)$ such that $\mathbf{s}_i - \mathbf{r}_i = \boldsymbol{\omega}_i = \texttt{GD}(\boldsymbol{\theta}, \mathcal{D}_i, d_i)$.
8. $P_i$ stores $\mathbf{r}_i$ and sends $\text{E}(\mathbf{s}_i)$ to the server.
```

Figure 4: Secure shared local gradient computation protocol for a logistic regression model.

the number of homomorphic constant multiplications. Table 1 summarizes their exact numbers. Asymptotically, the overall computational complexity for both π_{LinSLG} and π_{LogSLG} is $O(nd_i)$.

Let λ be the bit length of a ciphertext. Then, the communication cost involved in computing $\boldsymbol{\omega}_i$ for the linear regression is $2(n+1)\lambda$, which is for receiving the encrypted model and sending an encrypted share of $\boldsymbol{\omega}_i$ to the server. For the logistic regression, the communication cost is $(2(n+1) + 3d_i)\lambda$ where $d_i = |\mathcal{D}_i|$ is the size of the dataset, which is due to receiving the encrypted model, sending an encrypted share of $\boldsymbol{\omega}_i$ and information exchange for homomorphically computing the approximated sigmoid function.

Table 1: Number of unit operations for each user in the gradient computation protocols

Protocol	#CT MUL	#Const MUL	#Enc
Linear regression (π_{LinSLG})	$2(n+1)d_i - (n+1)$	$2nd_i$	$d_i + (n+1)$
Logistic regression (π_{LogSLG})	$(2n+5)d_i - (n+1)$	$2(n+2)d_i$	$3d_i + (n+1)$

Security. We prove the security of two SLG computation protocols in the simulation paradigm. Theorems 1 and 2 summarize the security of the SLG computation protocols for linear and logistic regressions. We show that an adversary $\mathcal{A}$ controlling a user P_i (resp. the server) does not learn anything about $\boldsymbol{\theta}$ (resp. $\boldsymbol{\omega}_i$) during the execution of π_{LinSLG} and similarly for π_{LogSLG}. Due to the space limit, we present the security proofs in Appendix B.1.

Theorem 1. *Assume that the additive homomorphic encryption scheme E() is semantically secure. The protocol π_{LinSLG} between P_i and S securely computes two shares of the local gradient on the dataset $\mathcal{D}_i$ in the presence of semi-honest adversaries.*

Theorem 2. *Assume that the additive homomorphic encryption scheme E() is semantically secure. The protocol π_{LogSLG} between P_i and S securely computes two shares of the local gradient $\boldsymbol{\omega}_i$ on the dataset $\mathcal{D}_i$ in the presence of semi-honest adversaries.*

4.4 Privacy-preserving Regression Model Training Protocol

The training protocol is executed between the server, and a set of m users in a synchronous network and is divided into three phases: 1) a pairwise key establishment phase, 2) a dataset scaling phase and 3) computing the regression model by iteratively computing the global gradient over a subset of users' data. A pair of pairwise keys is established using a Diffie-Hellman (DH) key agreement protocol to realize an authenticated channel between a pair of users and to use in the aggregation protocol. Before starting the training process, the users need to learn the mean and standard deviation on the entire dataset $\mathcal{D}$, which can be easily computed by running the aggregation protocol as shown in Section 4.1.

The process of the global gradient computation [1] consists of four main steps:

- the server randomly chooses a set of M users and broadcasts the current encrypted model to these users,
- the server and each user privately compute two shares of the local gradient on the chosen user's dataset in parallel,
- the server and alive users compute a single (aggregated) share from the alive users' shares of the local gradients,
- the server computes the second share of the global gradient from its local shares and then recovers the global gradient for updating the model.

The parameter M is chosen based on the security parameter of the aggregation protocol and the ϵ-privacy of the aggregate-sum (see Appendix A.5). For simplicity, we assume that each user has an equal number of data points, i.e., $|\mathcal{D}_i| = d_i = \ell, \forall i$. For a coalition of c users and/or the server in the ϵ-private aggregation scheme, the number of dropouts, denoted by δ, must satisfy the following constraint: $\ell(M - \delta - c) \geq \epsilon$, which implies $\delta \leq (M - \rho - c)$ where $\rho = \lceil \frac{\epsilon}{\ell} \rceil$. To resist a coalition of up to t parties including users and the server, M must be at least $2t$ where $t = \lceil \frac{m}{3} \rceil$. For a coalition of size up to t and $M = 2t$, $\delta \leq (t - \rho)$. When $\ell = 1$, for such a coalition of size up to t and $\delta = t$ dropouts, the number of users must participate in the training process is at least $2t + \epsilon$. The threshold value of π_{DeA} in the scaling phase must be at least $2t$. Figure 5 summarizes the complete protocol for training a linear or logistic regression model, given the parameters δ, t, ϵ and ℓ, while handling the dropout scenario.

Efficiency. It is easy to verify that the training protocol is correct if the users provide their true inputs and the server does not alter the model. Since the aggregation protocol is executed $(R+1)$ times, the users establish the pairwise keys using a Diffie-Hellman protocol, coordinated by the server, as shown in [7], and then in each execution of π_{DeA} the users derive a pair of one-time pairwise keys from the master pairwise keys using a hash-chain, as shown in [44]. Thus, the key establishment phase is executed only once. The computational complexity of the aggregation protocol π_{DeA}

[1] In the training phase, the batch size in the gradient computation varies in the range of $(t + \rho - 1)\ell$ and $2t\ell$, and $|\mathcal{D}_i| = \ell$.

```
Privacy-preserving Regression Training Protocol ($\pi_{\text{LinTrain}}$ or $\pi_{\text{LogTrain}}$)
//Pairwise key establishment phase
1. Mobile users and the server establish pairwise key among themselves using a DH key agreement protocol. Let $\mathcal{U}_0 \subset \mathcal{U}$ be set of users that established pairwise keys and agreed to participate in the training phase. If $|\mathcal{U}_0| < M$, abort the protocol.
//Data scaling phase
2. Each user in $\mathcal{U}_0$ locally computes the mean of its dataset $\mathcal{D}_i$ $\mathbf{X}_i = \left(\sum_{j=1}^{|\mathcal{D}_i|} x_1^{(j)}, \cdots, \sum_{j=1}^{|\mathcal{D}_i|} x_n^{(j)}, \sum_{j=1}^{|\mathcal{D}_i|} {x_1^{(j)}}^2, \cdots, \sum_{j=1}^{|\mathcal{D}_i|} {x_n^{(j)}}^2\right)$.
3. Server and the users in $\mathcal{U}_0$ execute the aggregation protocol $\pi_{\text{DeA}}$ with inputs $\{\mathbf{X}_u\}_{u \in \mathcal{U}_0}$ and obtains $(\mathcal{U}_1, \mathbf{X}_{sum}) \leftarrow \pi_{\text{DeA}}\left(\mathcal{U}_0, \{\mathbf{X}_u\}_{u \in \mathcal{U}_0}, |\mathcal{U}_0|, 2t\right)$ where $\mathbf{X}_{sum} = \sum_{u \in \mathcal{U}_1} \mathbf{X}_u = (X_1, \cdots, X_n, X_{n+1}, \cdots, X_{2n})$. If $|\mathcal{U}_1| \geq 2t$, the server computes the mean $\boldsymbol{\mu}$ over $\mathcal{D} = \cup_{u \in \mathcal{U}_1} \mathcal{D}_u$ as $\boldsymbol{\mu} = (\mu_1, \cdots, \mu_n)$ with $\mu_j = \frac{1}{d_1} X_j$ and $d_1 = |\mathcal{U}_1| \cdot \ell$, and the standard deviation $\boldsymbol{\sigma}$ vector as $\boldsymbol{\sigma} = (\sigma_1, \cdots \sigma_n)$ with $\sigma_j = \sqrt{\frac{1}{d_1} X_{n+j} - (2d_1 - 1)\mu_j^2}$, $1 \leq j \leq n$. Server sends $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ to all users in $\mathcal{U}_1$.
4. After receiving $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, each user $P_i$ in $\mathcal{U}_1$ scales its dataset as $\mathbf{x} \leftarrow \frac{\mathbf{x} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$, $\mathbf{x} \in \mathcal{D}_i$.
// Iterative training process
5. Server randomly initializes the model $\boldsymbol{\theta}$
6. for $j = 1 \ldots R$ do
  - Let $\mathcal{V} \subseteq \mathcal{U}_1$ be the set of alive users at the current time. If $|\mathcal{V}| < M$, abort the protocol.
  - Server chooses $\mathcal{V}_j \leftarrow$ (randomly choosing $M$ users from $\mathcal{V}$) at time $T_j$. // Gradient computation at time interval $[T_j, T_j + \Delta]$
  - Server encrypts the regression model $\boldsymbol{\theta}$ and then broadcasts the encrypted model $\mathrm{E}(\boldsymbol{\theta})$ to the users in $\mathcal{V}_j$.
  - for each user $P_i \in \mathcal{V}_j$ in parallel do
      Server runs the SLG computation protocol $\pi$ with $P_i$. Server receives $\mathrm{E}(\mathbf{s}_i)$ and $P_i$ receives $\mathbf{r}_i$ as output where $(\mathrm{E}(\mathbf{s}_i), \mathbf{r}_i) \leftarrow \pi(\boldsymbol{\theta}, \mathcal{D}_i)$ and $\pi \in \{\pi_{\text{LinSLG}}, \pi_{\text{LogSLG}}\}$.
    endfor
  - Let $\mathcal{V}'_j \subseteq \mathcal{V}_j$ be the set of users who successfully completed the SLG computation protocol and sent $\mathrm{E}(\mathbf{s}_i)$ to the server. If $|\mathcal{V}'_j| < (t + \rho)$, abort the protocol.
  - Server and users in $\mathcal{V}'_j$ run the aggregation protocol $\pi_{\text{DeA}}$ and complete the protocol execution in time $(T_j + \Delta)$. Server receives $(\mathcal{V}''_j, \mathbf{r}_{sum}) \leftarrow \pi_{\text{DeA}}\left(\mathcal{V}'_j, \{\mathbf{r}_u\}_{u \in \mathcal{V}'_j}, |\mathcal{V}'_j|, t\right)$. If $|\mathcal{V}''_j| < (t + \rho)$, abort the protocol.
  - Server decrypts $\mathrm{E}(\mathbf{s}_u)$ for each $u \in \mathcal{V}''_j$, and then computes $\mathbf{s}_{sum} = \sum_{u \in \mathcal{V}''_j} \mathbf{s}_u$.
  - Server computes $\boldsymbol{\omega}$ as $\boldsymbol{\omega} = \mathbf{s}_{sum} - \mathbf{r}_{sum}$ and updates the model as $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\eta}{d_a} \cdot \boldsymbol{\omega}$ with $d_a = \sum_{v \in \mathcal{V}''_j} d_v = |\mathcal{V}''_j| \cdot \ell$.
endfor
7. Server stores the regression model $\boldsymbol{\theta}$.
```

Figure 5: Privacy-preserving training protocol for a linear or logistic regression model over a mobile network.

with m users and n dimensional inputs for a user is $O(m^2 + mn)$, and for the server is $O(m^2 n)$. The computational complexity for one execution of the global gradient computation phase involves the computational complexities of π_{LinSLG} or π_{LogSLG} and π_{DeA}, which is $O(4t^2 + 2tn + nd_i) = O(m^2 + mn + nd_i)$ when $t = \lceil \frac{m}{3} \rceil$. If the training phase takes R iterations to obtain a model, the overall computational complexity of a user is $O(R(m^2 + mn + nd_i))$. The computational complexity of the server for the training phase is $O(R(m^2 n + mnd_i))$ which includes the computational complexities of the scaling phase and R executions of the privacy-preserving gradient computation phase.

The communication complexity of each user includes the costs of the DH key exchange protocol coordinated by the server, the scaling phase and the R iterations of π_{LinSLG} or π_{LogSLG} and π_{DeA}. When we choose $t = \lceil \frac{m}{3} \rceil$, the communication complexity of π_{DeA}, π_{LinSLG} and π_{LogSLG} is $O(m+n)$, $O(n)$ and $O(2n+3d_i)$, respectively. Therefore, the overall communication complexity of each user is $O(R(m + n))$ for linear regression and $O(R(m + n + d_i))$ for logistic regression, respectively. The server's communication complexity is $O(R(m^2 + mn))$ for linear regression, and $O(m^2 + Rm(n + d_i)))$ for logistic regression, including the pairwise communication costs of protocols π_{LinSLG} or π_{LogSLG} and π_{DeA}.

The storage overhead of the protocol is dominated by that of the aggregation protocol π_{DeA}. The storage overhead for each user is $O(m + n)$ for linear regression, and $O(m + n + d_i)$ for logistic regression, which is for storing secret-shares sent by the other users in π_{DeA}, the encrypted model and intermediate results for logistic regression. As the server needs to store the model, shares of the local gradients, the secret-shares of π_{DeA}, and intermediate results of π_{LogSLG}, the storage complexity of the server is $O(m^2 + mn)$ for linear regression, and $O(m^2 + mn + md_i)$ for logistic regression.

Security. We consider the security of the training protocols against semi-honest adversaries where three different threat models, namely users-only threat model, server-only threat model and users-server threat model are considered. The security of the training protocols are proved in the simulation paradigm using the hybrid arguments. In Theorem 3, we present that a semi-honest adversary corrupting at most t parties including the server and a set of at most $(t - 1)$ users can learn no information about the honest users' datasets. The security of the regression training protocols is achieved by that of the SLG protocols, the aggregation protocol and the aggregation privacy game defined in Appendix A.5. Due to the space limit, a formal security proof is provided in Appendix B.2.

Theorem 3 (Privacy in Users-Server Threat Model). *The protocols π_{LinTrain} and π_{LogTrain} are secure in the presence of semi-honest adversaries, meaning they leak no information about the honest users' inputs $\mathcal{D}_i$ with $|\mathcal{D}_i| \geq 1$ to the adversary corrupting the server and a set of users of size up to $(t-1)$.*

In Theorem 4, we show that a semi-honest adversary corrupting at most $(t-1)$ users in the training phase learns no information about the honest server's model. Intuitively, since the model is encrypted using a semantically secure additive HE scheme, the model privacy is protected against semi-honest users.

Theorem 4 (Privacy in Users-only Threat Model). *The protocols π_{LinTrain} and π_{LogTrain} are secure in the presence of semi-honest adversaries, meaning they leak no information about the honest server's model $\boldsymbol{\theta}$ to the adversary corrupting a set of users of size up to $(t-1)$.*

4.5 Oblivious Regression Prediction Protocols

We consider a scenario where, after training the model, users wish to use the trained regression model as predictions as a service, which is quite natural because the model was trained on their datasets, or the server wishes to offer regression predictions as a service to other clients. An oblivious regression prediction protocol is run between the server and a user. In this computation, the server holds a secret regression model $\boldsymbol{\theta}$, and a user has a private input $\mathbf{x}$. An oblivious regression prediction protocol allows the user only to learn $h(\boldsymbol{\theta}, \mathbf{x})$ without revealing $\mathbf{x}$ to the server and $\boldsymbol{\theta}$ to the users.

We assume that each user has a public and private key pair, denoted by (pk_{P_i}, sk_{P_i}), of an additive homomorphic encryption scheme and its public key is known to the server. For oblivious predictions, the user sends an encrypted input to the server. For linear regression, the server computes $\mathrm{E}_{pk_{P_i}}(\boldsymbol{\theta} \cdot \mathbf{x})$ from $\mathrm{E}_{pk_{P_i}}(\mathbf{x})$ and $\boldsymbol{\theta}$. For logistic regression, our oblivious prediction protocol uses the same idea on a single input as used in the SLG computation protocol. Figures 6 and 7 present the oblivious regression prediction protocols for linear and logistic regressions, respectively.

Efficiency. For the linear regression prediction, a user needs to perform n encryptions to encrypt the input $\mathbf{x}$ of dimension n and one decryption to decrypt the result. The server needs to perform n ciphertext multiplications, n homomorphic constant multiplications and one encryption. On the other hand, for the logistic regression prediction, the user need to perform $(n+2)$ encryptions, and two decryptions. The server needs to perform $(n+3)$ ciphertext multiplications, $(n+2)$ constant multiplications and three encryptions.

The communication cost of a user for the oblivious prediction is measured in terms of the amounts of bits the user needs to exchange. The amount of bits a user needs to exchange for linear regression is $(n+1)\lambda$, and for the logistic regression is $(n+4)\lambda$.

Security. It is shown that the attacks on the model can be performed when used as prediction services e.g., in [23, 56] due to querying the model multiple times to access the prediction APIs. We assume that no information is leaked about the model through the prediction APIs under a bounded number of queries. As countermeasures proposed in [23, 56] to prevent leakage, which is out of the scope of this work. Theorem 5 summarizes the security of regression prediction protocols.

Oblivious Predicion: Linear Regression (π_{LinOP})

Server: Private model $\boldsymbol{\theta}$, **User (P_i):** Private input $\mathbf{x}$

Output: P_i receives $h(\boldsymbol{\theta}, \mathbf{x})$.

1. User P_i encrypts its input $\mathbf{x}$ and sends $\mathrm{E}_{pk_{P_i}}(\mathbf{x})$ to the server
2. Server computes $\mathrm{E}_{pk_{P_i}}(h(\boldsymbol{\theta}, \mathbf{x})) \leftarrow \prod_{j=1}^{n} \mathrm{E}_{pk_{P_i}}(x_j)^{\theta_j} \cdot \mathrm{E}_{pk_{P_i}}(\theta_0)$ where $h(\boldsymbol{\theta}, \mathbf{x}) = \mathbf{x} \cdot \boldsymbol{\theta}$. It sends $\mathrm{E}_{pk_{P_i}}(h(\boldsymbol{\theta}, \mathbf{x}))$ to P_i.
3. P_i decrypts $\mathrm{E}_{pk_{P_i}}(\mathbf{x} \cdot \boldsymbol{\theta})$ and obtains $h(\boldsymbol{\theta}, \mathbf{x})$.

Figure 6: Oblivious linear regression prediction protocol.

Oblivious Prediction: Logistic Regression (π_{LogOP})

Server: Private model $\boldsymbol{\theta}$, **User (P_i):** Private input $\mathbf{x}$

Output: P_i receives $\sigma_3(\boldsymbol{\theta} \cdot \mathbf{x})$.

1. User P_i encrypts $\mathbf{x}$ and sends $\mathrm{E}(\mathbf{x})$ to the server
2. Server randomly generates r, and computes $\mathrm{E}(z) \leftarrow \prod_{j=1}^{n} \mathrm{E}_{pk_{P_i}}(x_j)^{\theta_j} \cdot \mathrm{E}_{pk_{P_i}}(\theta_0) \cdot \mathrm{E}_{pk_{P_i}}(r)$, where $z = \boldsymbol{\theta} \cdot \mathbf{x} + r$ and sends $\{\mathrm{E}_{pk_{P_i}}(z)\}$ **to** the user
3. P_i decrypts $\mathrm{E}_{pk_{P_i}}(z)$ and computes $\mathrm{E}_{pk_{P_i}}(z^2)$, $\mathrm{E}_{pk_{P_i}}(\sigma_3(z))$ and sends to the server
4. Server computes the following:

$$t = \mathrm{E}_{pk_{P_i}}(-(h_q(r)+(q_0+3q_3r^3))) \cdot \mathrm{E}_{pk_{P_i}}(\boldsymbol{\theta} \cdot \mathbf{x})^{-(2q_2r-6q_3r^2)}$$

$$\mathrm{E}_{pk_{P_i}}(\sigma_3(\boldsymbol{\theta} \cdot \mathbf{x})) = \mathrm{E}_{pk_{P_i}}(\sigma_3(z))\mathrm{E}_{pk_{P_i}}(z^2)^{-3q_3r} \cdot t$$

5. P_i decrypts $\mathrm{E}_{pk_{P_i}}(\sigma_3(\boldsymbol{\theta} \cdot \mathbf{x}))$ and obtains $\sigma_3(\boldsymbol{\theta} \cdot \mathbf{x})$.

Figure 7: Oblivious logistic regression prediction protocol.

Theorem 5. *The protocols π_{LinOP} and π_{LogOP} are secure in the presence of semi-honest adversaries, meaning they leak no information about the model $\boldsymbol{\theta}$ to a semi-honest user, and no information $\mathbf{x}$ to a semi-honest server, except what can be learnt obviously.*

Proof. It is easy to see that both protocols output the correct result if the parties are honest. We will first consider the privacy of the user input. In both protocols, the server observes an n-dimensional ciphertext vector, which is an encryption of $\mathbf{x}$ under a semantically secure HE scheme, and hence it learns nothing about the input. Thus, the input privacy of the user is protected.

We now consider the privacy of the model against a semi-honest user. For π_{LinOP}, the user observes the output $h(\boldsymbol{\theta}, \mathbf{x})$ after encryption, thus it learns no information about the model after one query. In π_{LogOP}, the user observes z after decrypting $\mathrm{E}(z)$ and learns nothing about z as $z = \boldsymbol{\theta} \cdot \mathbf{x} + r$ is random where r is chosen uniformly at random by the server. Thus the user learns only $\sigma_3(\boldsymbol{\theta} \cdot \mathbf{x})$. □

5 EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of PrivFL under semi-honest adversaries, providing 128-bit security. We first provide implementation details and then show how to deal with floating-point numbers when conjunct the regression algorithms with cryptographic primitives. Finally, we present experimental results of PrivFL on real-world datasets from the UCI ML repository.

5.1 Implementation and Dataset Details

Implementation details. We have implemented PrivFL in C using GMP [29] for large number operation, the FLINT library [32] for efficient secret sharing implementations and OpenSSL [50] for implementing an authenticated channel. We choose the Joye-Libert (JL) cryptosystem [33] to instantiate the additive homomorphic encryption scheme, over Paillier [51], and Bresson et al. [10] cryptosystems as its ciphertext size is 2× smaller. If the server is a cloud provider (e.g., Google, Microsoft, Apple), with the computation ability of the server, we can save the communication cost by a factor of two, which is quite reasonable for mobile applications. We implement the Joye-Libert cryptosystem using GMP. We implement an aggregation protocol that is a compilation of Bonawitz et al.'s [7] and Mandal et al.'s [44] protocols that handle user dropouts. The following crypto-primitives are used to implement the aggregation protocol under semi-honest adversaries.

- We use the NIST P-256 curve from OpenSSL for the Elliptic-curve Diffie-Hellman (ECDH) pairwise key establishment.
- We use AES in counter mode implemented using AES-NI instructions to implement PRG.
- We use AES-GCM from OpenSSL to implement a secure channel.
- We use SHA256 to derive one-time pairwise keys.
- We implement Shamir's threshold secret sharing (tss) schemes, namely $(2,3)$-tss and (t,m)-tss operations using the FLINT polynomial operation library, with NIST's 256-bit prime q.

We have chosen the message space $\mathbb{Z}_{2^k}$ with $k = 256$ where $q < 2^k$, which is enough for handling high-dimension data used in our experiment. Our experiments were conducted on a desktop with a 3.40GHz Intel i7 and 12 GB RAM. The codes were compiled using gcc 5.4.0 with -std=c99 -O1 -fomit-frame-pointer flag.

Datasets. We evaluate the performance of PrivFL on 11 different real-world datasets from the UCI ML repository [19]. We use 6 different datasets with various sizes and dimensions, summarized in Table 2, for evaluating the linear/ridge regression training protocols, and 5 different datasets, summarized in Table 3, for logistic regression training protocols. The dimension of the dataset ranges from 8 to 22 for linear regression and from 9 to 32 for logistic regression. For linear regression, we use the root mean squared error (RMSE) to measure the predictive accuracy of the model. For logistic regression, we measure the accuracy of the model using the standard method by computing a confusion matrix. Note that no privacy-preserving mechanism is used to compute the accuracy of the model.

Table 2: Dataset used in our experiment for linear/ridge regression models.

Id	Name	n	d	Reference
1	Auto MPG	8	392	[3]
2	Boston Housing Dataset	14	506	[9]
3	Energy Efficiency	9	768	[59]
4	Wine Quality	12	1,599	[14]
5	Parkinsons Telemonitoring	22	5,875	[58]
6	Bike Sharing Dataset	12	17,379	[21]

Table 3: Dataset used in our experiment for logistic regression model. † 20 out of 90 features were chosen.

Id	Name	n	d	Reference
7	Breast Cancer Dataset	32	454	[19]
8	Credit Approval Dataset	14	652	[18]
9	Diabetes Dataset	9	768	[15]
10	Credit Card Clients	24	30,000	[62]
11	US Census Income Dataset	20†	48,842	[12, 40]

Table 4: Number of the users and their training dataset size. Accuracy of the trained regression models achieved by PrivFL.

Linear regression				Logistic regression			
ID	m	d_i	**RMSE**	ID	m	d_i	**Score** (%)
1	28	10	3.16	7	32	10	96.00
2	36	10	4.91	8	46	10	87.60
3	54	10	3.85	9	54	10	76.48
4	112	10	0.68	10	700	30	80.72
5	206	20	3.45	11	1140	30	89.60
6	609	20	147.80				

5.2 Dealing with Floating Point Numbers

In regression algorithms, the datasets and the model parameters are floating point numbers, both positive and negative, but the cryptographic techniques namely additive HE and secret sharing work over the finite ring of integers. We provide the details about encoding floating point numbers to elements of $\mathbb{Z}_{2^k}$ and vice-versa via decoding.

Encoding and decoding. The encoding operation is applied on floating point numbers before performing cryptographic operations. As the message space is $\mathbb{Z}_{2^k}$, we divide the message space into two halves: the positive numbers are in $[0, 2^{k-1}-1]$, and the negative numbers are in $[2^{k-1}, 2^k - 1] \equiv [-2^{k-1}, -1]$. We convert each floating point number to an element of $\mathbb{Z}_{2^k}$ while maintaining its precision. Given an absolute floating point number x in $\mathbf{x}^{(i)}$, the corresponding ring element, denoted as $\tilde{x}$ in $\mathbb{Z}_{2^k}$, is computed as $\tilde{x} = \mathsf{FE}(x,\tau) = \mathsf{round}(x \cdot 2^\tau)$, and each floating point number y in $\{y^{(i)}\}$ is converted to a ring element as $\tilde{y} = \mathsf{FE}(y, 2\tau)$. If x is negative, the corresponding ring element is $\tilde{x} = 2^k - \mathsf{FE}(x,\tau)$, and similarly for y in $\{y^{(i)}\}$.

Given $\tilde{x} \in \mathbb{Z}_{2^k}$, the decoding of $\tilde{x}$ is given by $x = \mathsf{FD}(\tilde{x},\tau) = -\frac{2^k - z}{2^\tau}$ if $\tilde{x} \geq 2^{k-1}$, otherwise $\frac{z}{2^\tau}$.

Evaluating inner product. Given $\boldsymbol{\theta} \in \mathbb{R}^{n+1}$ and $\mathbf{x} \in \mathbb{R}^n$ and the corresponding vectors in $\mathbb{Z}_{2^k}$ are $\tilde{\boldsymbol{\theta}} \in \mathbb{Z}_{2^k}^{n+1}$ and $\tilde{\mathbf{x}} \in \mathbb{Z}_{2^k}^n$ where $\tilde{\theta_0} = \mathsf{FE}(\theta_0, 2\tau)$, $\tilde{\theta_i} = \mathsf{FE}(\theta_i, \tau)$ and $\tilde{x_i} = \mathsf{FE}(x_i, \tau)$. Then, $\tilde{\boldsymbol{\theta}} \cdot \tilde{\mathbf{x}} = 2^{2\tau} \boldsymbol{\theta} \cdot \mathbf{x}$. Thus, $\boldsymbol{\theta} \cdot \mathbf{x} = \mathsf{FD}(\tilde{\boldsymbol{\theta}} \cdot \tilde{\mathbf{x}})$. If $\tilde{y} = \mathsf{FE}(-y, 2\tau)$, then $\boldsymbol{\theta} \cdot \mathbf{x} - y = \mathsf{FD}(\tilde{\boldsymbol{\theta}} \cdot \tilde{\mathbf{x}} + \tilde{y})$.

Evaluating sigmoid. We approximate the sigmoid function $\sigma(x)$ over $[-l, l]$ for some l to a cubic polynomial as $\sigma_3(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3$. Note that the coefficients of the polynomial are public. To evaluate $\sigma_3(z)$ over $\mathbb{Z}_{2^k}$, we convert the coefficients of $\sigma_3(x)$ as $q_0 = \mathsf{FE}(c_0, 7\tau)$, $q_1 = \mathsf{FE}(c_1, 5\tau)$, $q_2 = \mathsf{FE}(c_2, 3\tau)$, and $q_3 = \mathsf{FE}(c_3, \tau)$, i.e., $q_i = \mathsf{FE}\Big(c_i, (7 - 2 * i)\tau\Big)$ and $z \in \mathbb{R}$ as $\tilde{z} = \mathsf{FE}(x, 2\tau)$. Then $\widetilde{\sigma_3(z)} = 2^{7\tau}\sigma_3(z)$, this implies $\sigma_3(z) = \mathsf{FD}(\widetilde{\sigma_3(z)}, 7\tau)$, where τ is chosen so that there is no overflow in the message space $\mathbb{Z}_{2^k}$.

5.3 Experimental Results

This section presents the performance of PrivFL where we report the timings, communication costs, and storage overhead for training and oblivious evaluation of linear and logistic regression algorithms for each user and the server. As the model privacy and data privacy while considering users dropping out in mobile applications have not been considered in previous work, e.g., [7, 44], we do not compare the performance of PrivFL with others in numerical values. However, we provide a system goalwise comparison in Section 6.

Micro-benchmarking. The additive homomorphic encryption and the aggregation protocol are two main operations that are frequently performed in the protocol. We perform micro-benchmarks that measure the timings of the basic operations, namely vector encryption, decryption and constant multiplication operations for the JL cryptosystem including floating point encoding and decoding operations, and the aggregation protocol for a user and the server to understand the deeper insight about the performance of the overall protocol. Table 5 presents the timings for HE operations and the aggregation protocol.

Table 5: Time in milliseconds (ms) (using a single CPU) for vector encryption, decryption and const. multiplication of the JL cryptosystem and the aggregation protocol.

	Vector dimension (n)				
Operations	10	20	30	40	50
Encryption (E($\mathbf{x}$))	8.39	16.05	24.05	32.35	39.83
Decryption (D($\mathbf{x}$))	4,027.02	8,131.76	12,134.78	16,178.63	20,107.34
Const. multiplication (E($\mathbf{x}$)$^{\mathbf{z}}$)	60.56	128.78	181.83	248.58	301.96
Aggregation protocol (π_{DEA})	**Number of users** (m)				
Dropout = 25%, $n = 30$	50	100	150	200	250
User time	20.38	41.45	63.43	85.86	107.10
Server time	34.79	161.36	420.90	823.35	1437.02

Timing for training regression models. For each dataset, we shuffle the dataset and use approximately 70% for training and 30% for testing to calculate the accuracy of the model. The training dataset is then distributed into a set of m users and each user holds an equal number of data points, given in Table 4. We randomly choose $2t$ users out of the m users for their participations in computing the global gradient in the training phase where $t = \lceil \frac{m}{3} \rceil$ and m is the total number of users. In our experiment, we set the number of dropout users in the gradient computation to $\delta = \lceil \frac{t}{2} \rceil$, and choose the aggregation size large enough to protect users' inputs privacy. For each dataset, the experiments were repeated five times, except the datasets Credit Card Clients and US Census Income datasets.

We compare the accuracy of the trained model obtained using the privacy-preserving training protocol with the trained model obtained using the sklearn tool (in clear, no security) for each dataset. Our achieved accuracy is very closed to the no security one (which is close to the state-of-the-art accuracy). The number of iteration needed to achieve such accuracy is $R = 350$ iterations for linear/ridge regression and $R = 300$ iterations for logistic regression, respectively. Figures 8a and 8c present the timings for training the linear and logistic regression models per user. Figure 8b and 8b show the server's timings for training the linear and logistic regression models. All experiment results were obtained using a single CPU. In our experiment, no communication time is considered.

For instance, to train the parkinsons telemonitoring data for a linear regression model, a user elapses about 105 milliseconds (ms) to compute the global gradient, and in the worst case, it elapses about 170 seconds (sec) for the entire training process. The "worst case" is because of the fact that a user may be chosen a maximum of R times. On the other hand, the server's computation time is about 10 sec for each global gradient computation, and in total about 56.57 mins. To train the credit card clients data using a logistic regression model, a user elapses about 1066 ms to compute the global gradient, and in the worst case, a user elapses about 320 sec for the entire training process. The server's computation time is about 99 mins for each global gradient computation, and in total about 495.3 hours, which is because of performing a total of $4,219,200$ JL decryptions and 300 executions of π_{DEA} with $2t$ users and $t/2$ dropouts where $t = 234$. *By exploiting parallelism using 24 CPUs, the training time for the server can be reduced to approximately 20 hours (estimated using the timings of unit operations).*

For instance, according to our experimental settings, to train a linear model on the bike sharing dataset, PrivFL's computational overheads for the server and each user, compared to a normal system, are 2.7×10^5 and 1.2×10^6, resp., where the normal system (naive implementation) provides no model and data privacy. Similarly, for training a logistic regression model on the credit card clients dataset, PrivFL's computational overheads for the server and each user are 1.4×10^8 and 3.2×10^5, resp.. The server's high computational overhead is due to the JL decryption algorithm.

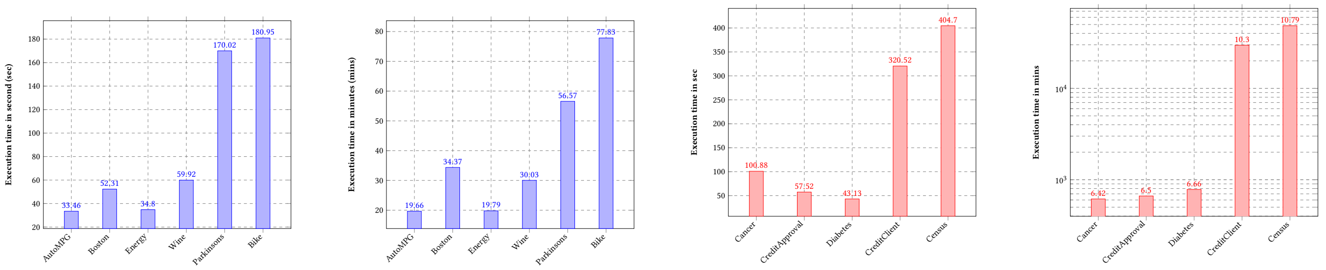
Note that the communication latencies among users or the server interactions during the protocol execution are not included in our experiment. However, the total execution time of the training protocol will be the computation times plus the communication latencies.

Timing for oblivious regression predictions. We also measure the performance of the oblivious evaluations of linear and regression models on the aforementioned datasets. The performance of the oblivious predictions depends on the dimension of the dataset. Table 6 summarizes the timings of a user and the server for obliviously evaluating linear and logistic regression models. The experiments are repeated ten times for each point. From Table 6, we can observe that the time for the user is larger than that of the server, which is due to the cost of the JL decryption operation.

Table 6: Time in milliseconds for oblivious regression prediction for the server and the user.

Linear regression			Logistic regression		
ID	**User**	**Server**	ID	**User**	**Server**
1	430.15	4.12	7	935.04	11.07
2	437.35	6.06	8	891.69	10.51
3	436.55	3.04	9	883.78	6.52
4	443.94	4.32	10	893.12	15.85
5	446.65	8.21	11	950.56	36.95
6	422.17	3.22			

Communication and storage cost. For training a linear regression model, the communication cost in the training phase involves receiving the encrypted model, transmitting the encrypted share of the local gradient and information exchange in the share reconstruction phase. For training a logistic regression model, in addition to the linear regression's communication cost, one more round of information exchange is required in the shared local gradient computation phase. Figure 9a presents the maximum amount of



(a) Time per user to train linear regression models

(b) Time elapsed by the server to train linear regression models

(c) Time per user to train logistic regression models

(d) Time elapsed by the server to train logistic regression models

Figure 8: Time in seconds for a user and in minutes for the server to train linear and logistic regression models.

bits a user needs to transfer to accomplish the training phase for the choice of the parameters and the achieved accuracy in Table 4. Note that the communication cost for a user depends on how many times the user is chosen in the training phase. For instance, for training the parkinsons telemonitoring data, in the worst case, a user need to exchange 12.05 Megabyte (MB) of data to train the linear regression model. On the other hand, for the credit card clients data, in the worst case, a user need to exchange 36.10 Megabyte (MB) of data to train the logistic regression model. The communication cost for the server for each dataset is provided in Figure 9b. The storage overheads for each user and the server is presented in Figures 9c and 9d, respectively.

Optimizations. It can be observed from the protocol that there are many scopes of paralleling the server's computation (e.g., JL vector decryption, encrypted vector multiplication, secret reconstruction and PRG evaluation, etc). Exploiting such parallelism using multiple CPUs, the server's computational time will be reduced to several magnitudes. While generating the random vector $\mathbf{r}$, users can generate it using a PRG.

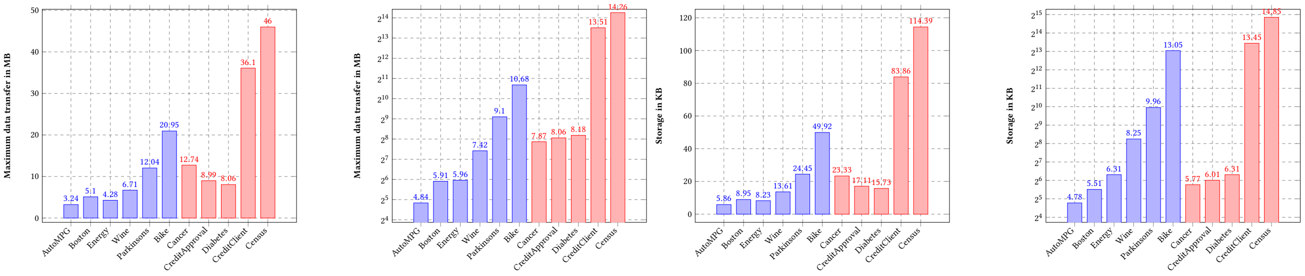
Discussions. In PrivFL, one can take a garbled circuit or purely secret-sharing based approach to implement the shared local gradient protocol. As the local gradient computation for linear regression can be expressed as a circuit of multiplicative depth two, a secret-sharing based approach will require at least two rounds of communications plus an additional cost of generating multiplication triplets in an offline phase. If the multiplication of two b-bit floating point numbers needs approximately $b^{1.58}$ gates (according to the Karatsuba algorithm), the computation of the local gradient between the server and a user has a transmission cost of at least $8 \times 128 \times b^{1.58} \times n = 2^{10} \times n \times b^{1.58}$ plus the transmission cost of the OT protocol using a garbled circuit approach for an n-dimensional vector. For an accuracy of six decimal places, this cost is much larger than $2n\log_2(N)$ bits for transmitting the encrypted model as well as the local gradient using the **JL** cryptosystem where N is a modulus of size 3072 bits. Similarly, for the logistic regression, the transmission cost is roughly $2^9 \times (2n+1)b^{1.58}$ using the garbled circuit approach, which is larger than $3n\log_2(N)$ for each data point. As mobile applications may have low bandwidth and slow connections, a garbled circuit or purely secret-sharing based approach will incur a higher communication overhead compared to an additive HE-based approach.

6 RELATED WORK

Privacy-preserving Linear Regression. Privacy-preserving computation of linear regression has received considerable attention. Early works [17, 34–36, 53] have considered learning linear regression model on either horizontally or vertically distributed datasets. Hall et al. [30] proposed protocols for linear regression based on homomorphic encryption techniques. Nikolaenko et al. [49] proposed a system for privacy-preserving computation of the ridge regression model by combining homomorphic encryption and Yao garbled circuits. In a follow-up work, Gascón et al. [25] proposed protocols for linear regression models based on hybrid-MPC techniques and Yao garbled circuits and using the conjugate gradient descent algorithm. Bogdanov et al. [5] developed tools for privacy-preserving linear regression based on secret sharing. Other approach for privacy-preserving linear regression is based on fully homomorphic encryption (FHE) scheme [26], which may not be suitable for mobile applications. The work of [28] can be applied to linear regression for the settings of MLaaS.

Privacy-preserving Logistic Regression. Logistic regression is an essential technique to classify data. Aono et al. [8] proposed a system for both training and predicting data using logistic regression relying on additive homomorphic encryption where they considered the computation outsourcing scenario in which a server computes the logistic regression model and sends an encrypted model to the user. They also showed how to make the system differential privacy enabled. Bonte and Vercauteren [2] explored secure training for logistic regression using somewhat homomorphic encryption in the computation outsourcing scenario where the training is done using Newton-Raphson method. Kim et al. [39] also investigated the training phase of the logistic regression model, using somewhat homomorphic encryption scheme, based on gradient descent algorithm and the Taylor series polynomial approximation. Zhu et al. [63] presented a secure outsourcing protocol for training and evaluating logistic regression classifier in cloud. Kim et al. in [38] proposed a method to train a logistic regression model based on the approximate homomorphic encryption. There is no satisfactory solution for training an ML model using SWHE or FHE when data come from multiple sources. All these techniques cannot handle the dropout scenario and are not suitable for mobile applications.

Privacy-preserving Federated Learning. Shokri and Shmatikov [55] presented a scheme for privacy-preserving deep learning. Hardy et al. [31] presented a three-party protocol for logistic regression using additive homomorphic encryption where the protocol consists of privacy-preserving entity resolution and federated logistic regression and the data is vertically partitioned. Their protocol cannot handle the dropout scenario, and privacy is not ensured when the dataset contains only one point. Note that their work have not



(a) Maximum data transfer per user

(b) Maximum data transfer for server

(c) Storage per user

(d) Storage for the server

Figure 9: Total data exchange in Megabyte (MB) and storage required in Kilobyte (KB) per user and the server to train linear and logistic regression models.

considered the model privacy. Our solution is more general compared to theirs. Fioretto and Hentenryck in [22] proposed a protocol for federated data sharing to use under the framework of differential privacy, which allows the users to release a privacy-preserving version of the dataset. This dataset can be used to train various predictors for linear regression, logistic regression, and support vector machines. Liu et al. [43] proposed a technique for privacy-preserving federated transfer learning. Truex et al. [57] presented an approach for private federated learning, decision tree, neural networks that provides data privacy guarantees using differential privacy and threshold homomorphic encryption schemes.

Generic Secure ML Systems. Systems, namely SecureML [48], Prio [13] and ABY^3 [47] can be used to perform privacy-preserving linear and logistic regression, but such systems need existence of addition servers (other than the server used for coordination) where users secret share their data among a set of servers. These systems provide stronger security compared to ours because in PrivFL, the server has access to the model in each iteration of the model update, which is quite suitable in the federated setting. However, these approaches are orthogonal to the federated learning setting where users send (using secret sharing) their private data to the servers, but in federated learning, users' data never go out of the devices.

7 CONCLUDING REMARKS AND FUTURE WORK

This paper presented PrivFL, a privacy-preserving system for training and oblivious predictions of the predictive models such as linear and logistic regressions in the federated setting, while ensuring dropout robustness and the data as well as the model privacy. PrivFL enables a robust and secure training process by iteratively executing a secure multiparty global gradient protocol built using lightweight cryptographic primitives suitable for mobile applications. The security of PrivFL is analyzed against semi-honest adversaries. Our experimental results on several real-world datasets demonstrate the practicality of PrivFL to incorporate in the federated learning system.

We intend to implement PrivFL on smartphones to evaluate its efficiency. As today's server platforms are equipped with multiple CPUs, we are working on designing an easy-and-parallel implementation interface at the server-side to reduce the computation time. While keeping the general protocol flow in Section 4.2 same, we are extending our work to neural networks in a separate paper. As a future work, it would be interesting to investigate the case of users asynchronously participating in the training phase.

Acknowledgement. This work is supported by the NSERC Discovery grant. The authors would like to thank the anonymous reviewers of CCSW2019 for their insightful comments and suggestions to improve the quality of the paper.

REFERENCES

[1] Ai.type. https://www.androidauthority.com/ai-type-data-exposed-820431/.

[2] Aono, Y., Hayashi, T., Trieu Phong, L., and Wang, L. Scalable and secure logistic regression via homomorphic encryption. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy* (2016), ACM, pp. 142–144.

[3] Auto MPG data set. https://archive.ics.uci.edu/ml/datasets/auto+mpg, 1993. Online; accessed 29 July 2019.

[4] Barbosa, M., Catalano, D., and Fiore, D. Labeled homomorphic encryption. In *Computer Security – ESORICS 2017* (Cham, 2017), S. N. Foley, D. Gollmann, and E. Snekkenes, Eds., Springer International Publishing, pp. 146–166.

[5] Bogdanov, D., Kamm, L., Laur, S., and Sokk, V. Rmind: A tool for cryptographically secure statistical analysis. *IEEE Transactions on Dependable and Secure Computing 15*, 3 (May 2018), 481–495.

[6] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konecný, J., Mazzocchi, S., McMahan, H. B., Overveldt, T. V., Petrou, D., Ramage, D., and Roselander, J. Towards federated learning at scale: System design. *CoRR abs/1902.01046* (2019).

[7] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2017), CCS '17, ACM, pp. 1175–1191.

[8] Bonte, C., and Vercauteren, F. Privacy-preserving logistic regression training. Tech. rep., IACR Cryptology ePrint Archive 233, 2018.

[9] Boston Housing Dataset. https://archive.ics.uci.edu/ml/machine-learning-databases/housing/, 2019. Online; accessed 29 July 2019.

[10] Bresson, E., Catalano, D., and Pointcheval, D. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In *Advances in Cryptology - ASIACRYPT 2003* (Berlin, Heidelberg, 2003), C.-S. Laih, Ed., Springer Berlin Heidelberg, pp. 37–54.

[11] Buescher, N., Boukoros, S., Bauregger, S., and Katzenbeisser, S. Two is not enough: Privacy assessment of aggregation schemes in smart metering. *Proceedings on Privacy Enhancing Technologies 2017*, 4 (2017), 198–214.

[12] Census Income Data Set. https://archive.ics.uci.edu/ml/datasets/census+income, 1996. Online; accessed 29 July 2019.

[13] Corrigan-Gibbs, H., and Boneh, D. Prio: Private, robust, and scalable computation of aggregate statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2017), NSDI'17, USENIX Association, pp. 259–282.

[14] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems 47*, 4 (2009), 547 – 553. Smart Business Networks: Concepts and Empirical Evidence.

[15] Diabetes Data Set. https://archive.ics.uci.edu/ml/datasets/diabetes, 1994.

[16] Diffie, W., and Hellman, M. New directions in cryptography. *IEEE Trans. Inf. Theor. 22*, 6 (Sept. 2006), 644–654.

[17] Du, W., Han, Y. S., and Chen, S. *Privacy-Preserving Multivariate Statistical Analysis: Linear Regression and Classification.* pp. 222–233.

[18] Dua, D., and Graff, C. UCI machine learning repository. https://archive.ics.uci.edu/ml/datasets/credit+approval, 2017.

[19] Dua, D., and Graff, C. UCI machine learning repository. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic), 2017.

[20] Dwork, C., and Roth, A. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci. 9*, 3&#8211;4 (Aug. 2014), 211–407.

[21] Fanaee-T, H., and Gama, J. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence* (2013), 1–15.

[22] Fioretto, F., and Van Hentenryck, P. Privacy-preserving federated data sharing. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems* (Richland, SC, 2019), AAMAS '19, International Foundation for Autonomous Agents and Multiagent Systems, pp. 638–646.

[23] Fredrikson, M., Jha, S., and Ristenpart, T. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2015), CCS '15, ACM, pp. 1322–1333.

[24] Gascón, A., Schoppmann, P., Balle, B., Raykova, M., Doerner, J., Zahur, S., and Evans, D. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies 2017*, 4 (2017), 345–364.

[25] Gascón, A., Schoppmann, P., Balle, B., Raykova, M., Doerner, J., Zahur, S., and Evans, D. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies 2017*, 4 (2017), 345–364.

[26] Gentry, C. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2009), STOC '09, ACM, pp. 169–178.

[27] Goethals, B., Laur, S., Lipmaa, H., and Mielikäinen, T. On private scalar product computation for privacy-preserving data mining. In *Information Security and Cryptology – ICISC 2004* (Berlin, Heidelberg, 2005), C.-s. Park and S. Chee, Eds., Springer Berlin Heidelberg, pp. 104–120.

[28] Graepel, T., Lauter, K., and Naehrig, M. Ml confidential: Machine learning on encrypted data. In *Information Security and Cryptology – ICISC 2012* (Berlin, Heidelberg, 2013), T. Kwon, M.-K. Lee, and D. Kwon, Eds., Springer Berlin Heidelberg, pp. 1–21.

[29] Granlund, T., et al. GMP: the GNU multiple precision arithmetic library, 1991.

[30] Hall, R., Fienberg, S. E., and Nardi, Y. Secure multiple linear regression based on homomorphic encryption. *Journal of Official Statistics 27*, 4 (2011), 669.

[31] Hardy, S., Henecka, W., Ivey-Law, H., Nock, R., Patrini, G., Smith, G., and Thorne, B. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *CoRR abs/1711.10677* (2017).

[32] Hart, W., Johansson, F., and Pancratz, S. FLINT: Fast Library for Number Theory, 2013. Version 2.4.0, http://flintlib.org.

[33] Joye, M., and Libert, B. Efficient cryptosystems from 2k-th power residue symbols. In *Advances in Cryptology – EUROCRYPT 2013* (Berlin, Heidelberg, 2013), T. Johansson and P. Q. Nguyen, Eds., Springer Berlin Heidelberg, pp. 76–92.

[34] Karr, A. F., Lin, X., Sanil, A. P., and Reiter, J. P. Regression on distributed databases via secure multi-party computation. In *Proceedings of the 2004 Annual National Conference on Digital Government Research* (2004), dg.o '04, Digital Government Society of North America, pp. 108:1–108:2.

[35] Karr, A. F., Lin, X., Sanil, A. P., and Reiter, J. P. Secure regression on distributed databases. *Journal of Computational and Graphical Statistics 14*, 2 (2005), 263–279.

[36] Karr, A. F., Lin, X., Sanil, A. P., and Reiter, J. P. Privacy-preserving analysis of vertically partitioned data using secure matrix products. *J. Official Statistics* (2009).

[37] Kiltz, E., Leander, G., and Malone-Lee, J. Secure computation of the mean and related statistics. In *Theory of Cryptography* (Berlin, Heidelberg, 2005), J. Kilian, Ed., Springer Berlin Heidelberg, pp. 283–302.

[38] Kim, A., Song, Y., Kim, M., Lee, K., and Cheon, J. H. Logistic regression model training based on the approximate homomorphic encryption. *BMC Medical Genomics 11*, 4 (Oct 2018), 83.

[39] Kim, M., Song, Y., Wang, S., Xia, Y., and Jiang, X. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR medical informatics 6*, 2 (2018).

[40] Kohavi, R. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (1996), KDD'96, AAAI Press, pp. 202–207.

[41] KoneÄŊnÃ¡, J., McMahan, H. B., Yu, F. X., Richtarik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning* (2016).

[42] Liu, J., Juuti, M., Lu, Y., and Asokan, N. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2017), CCS '17, ACM, pp. 619–631.

[43] Liu, Y., Chen, T., and Yang, Q. Secure federated transfer learning. *CoRR abs/1812.03337* (2018).

[44] Mandal, K., Gong, G., and Liu, C. Nike-based fast privacy-preserving high-dimensional data aggregation for mobile devices. CACR Technical Report, CACR 2018-10, University of Waterloo, Canada, 2018.

[45] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).

[46] McMahan, H. B., Moore, E., Ramage, D., and y Arcas, B. A. Federated learning of deep networks using model averaging. *CoRR abs/1602.05629* (2016).

[47] Mohassel, P., and Rindal, P. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2018), CCS '18, ACM, pp. 35–52.

[48] Mohassel, P., and Zhang, Y. Secureml: A system for scalable privacy-preserving machine learning. Cryptology ePrint Archive, Report 2017/396, 2017. http://eprint.iacr.org/2017/396.

[49] Nikolaenko, V., Weinsberg, U., Ioannidis, S., Joye, M., Boneh, D., and Taft, N. Privacy-preserving ridge regression on hundreds of millions of records. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2013), SP '13, IEEE Computer Society, pp. 334–348.

[50] OpenSSL. The openssl library. https://www.openssl.org/.

[51] Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques* (Berlin, Heidelberg, 1999), EUROCRYPT'99, Springer-Verlag, pp. 223–238.

[52] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res. 12* (Nov. 2011), 2825–2830.

[53] Sanil, A. P., Karr, A. F., Lin, X., and Reiter, J. P. Privacy preserving regression modelling via distributed computation. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2004), KDD '04, ACM, pp. 677–682.

[54] Shamir, A. How to share a secret. *Commun. ACM 22*, 11 (Nov. 1979), 612–613.

[55] Shokri, R., and Shmatikov, V. Privacy-preserving deep learning. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2015), CCS '15, ACM, pp. 1310–1321.

[56] Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., and Ristenpart, T. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, Aug. 2016), USENIX Association, pp. 601–618.

[57] Truex, S., Baracaldo, N., Anwar, A., Steinke, T., Ludwig, H., and Zhang, R. A hybrid approach to privacy-preserving federated learning. *CoRR abs/1812.03224* (2018).

[58] Tsanas, A., Little, M. A., McSharry, P. E., and Ramig, L. O. Accurate telemonitoring of parkinson's disease progression by noninvasive speech tests. *IEEE Transactions on Biomedical Engineering 57*, 4 (April 2010), 884–893.

[59] Tsanas, A., and Xifara, A. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings 49* (2012), 560 – 567.

[60] Wenliang Du, and Atallah, M. J. Privacy-preserving cooperative statistical analysis. In *Seventeenth Annual Computer Security Applications Conference* (Dec 2001), pp. 102–110.

[61] Yao, A. C.-C. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1986), SFCS '86, IEEE Computer Society, pp. 162–167.

[62] Yeh, I.-C., and hui Lien, C. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications 36*, 2, Part 1 (2009), 2473 – 2480.

[63] Zhu, X. D., Li, H., and Li, F. H. Privacy-preserving logistic regression outsourcing in cloud computing. *Int. J. Grid Util. Comput. 4*, 2/3 (Sept. 2013), 144–150.

A CRYPTOGRAPHIC BACKGROUND

In this section, we provide a background on the crypto-primitives and schemes that are needed for the aggregation protocol. We also present the aggregation privacy game of Buescher et al. [11].

A.1 PKI and DH Key Agreement

PrivFL requires an existence of a public key infrastructure (PKI) due to the use of the secure aggregation protocol in which the users need to establish two pairwise keys, and each user and the server need to establish a pairwise key, which is done using the Diffie-Hellman (DH) protocol [16]. For the users, one pairwise key is used to derive one-time pairwise keys and the other one is used to realize an authenticated channel with other users to exchange information in the aggregation protocol. The users register their identities when they join first time and publish the identities and public keys in the setup phase, as used in [7, 44].

A key agreement protocol consists of a tuple of three algorithms (ParamGen, KeyGen, KeyAgree). We use an Elliptic curve variant of the DH key agreement protocol. Given a security parameter κ, the parameter generation algorithm $(\mathbb{G}, q, G) \leftarrow \text{ParamGen}(1^\kappa)$ samples a group of points on an elliptic curve over a field $\mathbb{F}_q$ of order q and a generator G. The key generation algorithm $(x_A, x_AG) \leftarrow \text{KeyGen}(\mathbb{G}, q, P)$ generates a secret key $x_A \xleftarrow{\$} \mathbb{Z}_q$ and a public key $G_A = x_AG$. The key agreement function KeyAgree computes a pairwise key $K_{AB} = H(G_{AB})$ where $G_{AB} \leftarrow \text{KeyAgree}(x_A, G_B) = x_Ax_BG$ and H is a hash function. The security of the key agreement protocol follows from the Decisional Diffie-Hellman (DDH) assumption.

A.2 Authenticated Encryption

An authenticated encryption scheme consists of three algorithms $(\mathcal{K}, \text{Enc}, \text{Dec})$ where $\mathcal{K}$ is a symmetric key generation algorithm, Enc is a symmetric-key encryption algorithm that accepts a key and a message as input and outputs a ciphertext and a tag, and Dec is a decryption algorithm that takes a key, a ciphertext and a tag, and outputs the original message or $\perp$. We use the AES-GCM that offers the IND-CPA and INT-CTXT securities to realize an authenticated channel.

A.3 Pseudorandom Generator and One-way Function

A pseudorandom generator PRG : $\{0,1\}^l \rightarrow \{0,1\}^b$, is a deterministic function which accepts an l-bit binary string as input and outputs an b-bit binary string with $b >> l$. The input to a PRG is chosen uniformly at random. The security of a cryptographically strong PRG is defined as its output bit stream is indistinguishable from a truly random one. We used AES in counter mode as a PRG in our experiment. A one-way function $F : \{0,1\}^m \rightarrow \{0,1\}^n$ is a function that is easy to compute and hard to invert where $n \leq m$. Security of the one-way function F is the amount of work needed to invert F on an input x. We use the SHA256 hash function to realize a one-way function.

A.4 Threshold Secret Sharing

Let $\mathbb{F}_q$ be a finite field where q is a prime. A t-out-of-m threshold secret sharing scheme ((t,m)-tss) over $\mathbb{F}_q$ is a tuple of two algorithms $(\mathbf{Share}_m^t, \mathbf{Rec}_m^t)$ where the secret sharing algorithm $\mathbf{Share}_m^t$ is a randomized algorithm that takes a secret input s over $\mathbb{F}_q$ and outputs m shares of s, i.e., $(s_1, s_2, \ldots, s_m) \leftarrow \mathbf{Share}_m^t(s)$ and the reconstruction algorithm $\mathbf{Rec}_m^t$ accepts any t shares $(s_{i_1}, s_{i_2}, \cdots, s_{i_t})$ as input and outputs the original s. The correctness of the scheme is defined as for any s, $\Pr_{(s_1,\cdots,s_m)\leftarrow\mathbf{Share}_m^t(s)}[\mathbf{Rec}_m^t(s_{i_1}, s_{i_2}, \cdots, s_{i_t}) = s] = 1$. The security of the scheme is defined as any set of less than t users learns nothing about the secret s. We use Shamir's threshold secret sharing scheme [54] and a (2, 3)-tss in the secure aggregation protocol.

A.5 Aggregation Privacy Game

Buescher et al. [11] formalized the privacy of an aggregation scheme where the privacy of an aggregate is measured by the chances of an adversary in winning the game in Figure 10. We use an aggregation protocol that satisfies the aggregation privacy game.

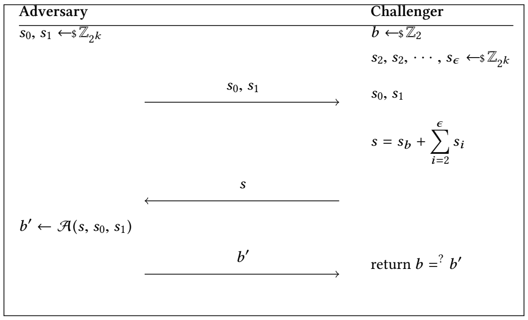

Adversary
Challenger
$s_0, s_1 \leftarrow_\$ \mathbb{Z}_{2^k}$
$b \leftarrow_\$ \mathbb{Z}_2$
$s_2, s_2, \cdots, s_\epsilon \leftarrow_\$ \mathbb{Z}_{2^k}$
s_0, s_1
s_0, s_1
$s = s_b + \sum_{i=2}^{\epsilon} s_i$
s
$b' \leftarrow \mathcal{A}(s, s_0, s_1)$
b'
return $b \overset{?}{=} b'$

Figure 10: Privacy game for an aggregation scheme

Definition 1. *We say an aggregate-sum is* ϵ-private *if the aggregate* $s = \sum_{i=1}^{\epsilon} s_i$ *is secure under the privacy game defined in Figure 10, meaning it leaks no information about any individual* s_i.

B SECURITY ANALYSIS OF PRIVFL

This section presents the security proofs of the shared local gradient computation protocols in Section 4.3, and the training protocols in Section 4.4 for linear and logistic regression models.

B.1 Security of Shared Local Gradient Computation Protocols

Before providing the proofs, we first define the ideal functionality of the shared local gradient computation protocol in Figure 11. We denote by $\mathcal{F}_{\text{LinSLG}}$ and $\mathcal{F}_{\text{LogSLG}}$ the ideal functionalities for the SLG for linear and logistic regression models, respectively.

Inputs

Server (S): Model $\boldsymbol{\theta}$

User (P_i): Dataset $\mathcal{D}_i$

Outputs

User (P_i): a random vector $\mathbf{r}_i$

Server (S): $\mathbf{s}_i$ such that $\mathbf{s}_i = \texttt{GD}(\boldsymbol{\theta}, (\mathcal{D}_i, |\mathcal{D}_i|)) + \mathbf{r}_i$.

Figure 11: Ideal functionality $\mathcal{F}_{\text{SLG}}$ for an SLG computation.

Proofs of Theorems 1 and 2. We prove the security of two SLG protocols in the simulation paradigm.

Proof of Theorem 1. To prove π_{LinSLG} is secure, we show that the adversary's ($\mathcal{A}$) view in the real-world execution of the protocol is indistinguishable from that of the ideal-world execution. We assume that a trusted party computes the ideal functionality $\mathcal{F}_{\text{LinSLG}}$ in the ideal-world execution of the protocol. We construct two different simulators, denoted by $\mathcal{S}_S$ for the server S, and $\mathcal{S}_{P_i}$ for the user P_i. We assume that the server receives the model $\boldsymbol{\theta}$ and the user P_i receives $\mathcal{D}_i$ with size d_i from the environment, and the simulator has access to $\mathcal{F}_{\text{LinSLG}}$.

Server S is semi-honest. When $\mathcal{A}$ controls the server, the simulator $\mathcal{S}_S$ works as follows:

- $\mathcal{S}_S$ receives $\boldsymbol{\theta}$ from the environment and sends it to $\mathcal{F}_{\text{LinSLG}}$ and receives $\mathbf{s}' = \boldsymbol{\omega}_i + \mathbf{r}' = (s'_0, s'_1, \cdots, s'_n)$.
- $\mathcal{S}_S$ runs S on input $\boldsymbol{\theta}$ and receives $\text{E}(\boldsymbol{\theta}) = (\text{E}(\theta_0), \cdots, \text{E}(\theta_n))$.
- $\mathcal{S}_S$ encrypts $\mathbf{s}$ using S's public key and outputs whatever S outputs.

S's view in the real-world execution of the protocol is $\mathbf{s} = (s_0, s_1, \ldots, s_n)$ after decrypting $\mathrm{E}(\mathbf{s})$ where

$$\mathbf{s} = \boldsymbol{\omega}_i + \mathbf{r} = \Big(\sum_{j=1}^{d_i} e_j + r_0, \cdots, \sum_{j=1}^{d_i} e_j x_n^{(j)} + r_n\Big)$$

where $e_i = h(\boldsymbol{\theta}, \mathbf{x}^{(j)})$. As $\mathbf{r}$ is chosen uniformly at random, any coordinate of $\mathbf{s}$, i.e., $s_l = \sum_{j=1}^{d_i} e_j x_l^{(j)} + r_l, 0 \leq l \leq n$ is indistinguishable from random s'_l. Therefore, the server's view in the real-execution is indistinguishable from the view of the ideal-execution of the protocol.

User P_i is semi-honest. When $\mathcal{A}$ controls the user P_i, the simulator $\mathcal{S}_{P_i}$ works as follows.

- $\mathcal{S}_{P_i}$ receives $\mathcal{D}_i$ from the environment and sends it to $\mathcal{F}_{\text{LinSLG}}$ and receives $\mathbf{r}$.
- $\mathcal{S}_{P_i}$ runs P_i on the input $\mathcal{D}_i$.
- $\mathcal{S}_{P_i}$ constructs $\mathrm{E}(\mathbf{0})$ and sends it to P_i and outputs whatever P_i outputs.

P_i's view in the real-world execution of the protocol is $\mathrm{E}(\boldsymbol{\theta}) = (\mathrm{E}(\theta_0), \cdots, \mathrm{E}(\theta_n))$ where each $\mathrm{E}(\theta_i)$ is computationally indistinguishable from $\mathrm{E}(0)$, resulting in $\mathrm{E}(\boldsymbol{\theta})$ is indistinguishable from $\mathrm{E}(\mathbf{0})$ due to the semantic security of E. Thus, the view of the adversary in the real-world execution of the protocol is indistinguishable from that of the ideal-world execution of the protocol. □

Proof of Theorem 2. We provide the constructions of two different simulators, denoted by $\mathcal{S}_S$ for the server S and $\mathcal{S}_{P_i}$ for the user P_i. We assume that the simulator has access to the ideal functionality $\mathcal{F}_{\text{LogSLG}}$. We assume that the server receives the model $\boldsymbol{\theta}$ and the user P_i receives $\mathcal{D}_i$ with size d_i from the environment.

Server S is semi-honest. When $\mathcal{A}$ controls the server, the simulator $\mathcal{S}_S$ works as follows:

- $\mathcal{S}_S$ receives $\boldsymbol{\theta}$ from the environment and sends it to $\mathcal{F}_{\text{LinSLG}}$ and receives $\mathbf{s}' = \boldsymbol{\omega}_i + \mathbf{r}' = (s'_0, s'_1, \cdots, s'_n)$.
- $\mathcal{S}_S$ runs S on input $\boldsymbol{\theta}$ and receives $\mathrm{E}(\boldsymbol{\theta}) = (\mathrm{E}(\theta_0), \cdots, \mathrm{E}(\theta_n))$.
- $\mathcal{S}_S$ constructs $\left\{z'_j\right\}_{j=1}^{d_i}$ where $z'_j = \boldsymbol{\theta} \cdot \mathbf{x}^j + c'_j$ and sends $\left\{\mathrm{E}(z'_j)\right\}$ to S. Note that the simulator does not know actual values of $\left\{c'_j\right\}$ chosen by P_i.
- $\mathcal{S}_S$ receives $\left\{z_j\right\}$ from S and computes $\left\{z_j^2\right\}$ and $\left\{\sigma_3(z_j)\right\}$ as well as $\left\{\mathrm{E}(z_j^2), \mathrm{E}(\sigma_3(z_j))\right\}$ and sends it to S
- $\mathcal{S}_S$ encrypts $\mathbf{s}$ using S's public key pk and sends to S and outputs whatever S outputs.

S's view in the real-world execution of the protocol is $\left\{z_j\right\}_{j=1}^{d_i}$ and $\mathbf{s} = (s_0, s_1, \ldots, s_n)$ after decrypting $\mathrm{E}(\mathbf{s})$ where

$$z_j = \boldsymbol{\theta} \cdot \mathbf{x}^{(j)} + c_j, \text{ and } \mathbf{s} = \boldsymbol{\omega}_i + \mathbf{r} = \Big(\sum_{j=1}^{d_i} e_j + r_0, \cdots, \sum_{j=1}^{d_i} e_j x_n^{(j)} + r_n\Big)$$

with $e_i = h(\boldsymbol{\theta}, \mathbf{x}^{(j)})$ and h is a logistic regression function. As $\mathbf{r}$ is chosen uniformly at random, any coordinate of $\mathbf{s}$, i.e., $s_l = \sum_{j=1}^{d_i} e_j x_l^{(j)} + r_l, 0 \leq l \leq n$ is indistinguishable from s'_l. Therefore, the server's view in the real-world execution is indistinguishable from that of the ideal-world execution of the protocol.

User P_i is semi-honest. When $\mathcal{A}$ controls the user P_i, the simulator $\mathcal{S}_{P_i}$ works as follows.

- $\mathcal{S}_{P_i}$ receives $\mathcal{D}_i$ from the environment and sends it to $\mathcal{F}_{\text{LogSLG}}$ and receives $\mathbf{r}$.
- $\mathcal{S}_{P_i}$ constructs $\mathrm{E}(\boldsymbol{\theta}) = (\mathrm{E}(0), \cdots, \mathrm{E}(0))$ and sends it to P_i.
- $\mathcal{S}_{P_i}$ runs P_i on the inputs $\mathcal{D}_i$ and $\mathrm{E}(\boldsymbol{\theta})$ and obtains $\left\{\mathrm{E}(z_j)\right\}$ where $\mathcal{S}_{P_i}$ generates random coins $\{c'_j\}$ for P_i.
- $\mathcal{S}_{P_i}$ constructs $\left\{(\mathrm{E}(0), \mathrm{E}(0))\right\}_{j=1}^{d_i}$ and sends to P_i and outputs whatever P_i outputs.

P_i's view in the real-world execution of the protocol is given by $\mathrm{E}(\boldsymbol{\theta}), \left\{(\mathrm{E}(z_j^2), \mathrm{E}(\sigma_3(z_j))\right\}_{j=1}^{d_i}$, which is computationally indistinguishable from that of the real-world execution of the protocol, due to the semantic security of E. □

B.2 Security of Training Protocols

In this section, we prove the security of the training protocol against semi-honest adversaries in the simulation paradigm by providing a construction of a simulator. We use an aggregation protocol π_{DeA} that securely implements the ideal functionality in Figure 12. In [7, 44], the security of the aggregation protocol is proved against semi-honest and active adversaries. We only need the security of π_{DeA} against semi-honest adversaries. The security of the training protocol in the server-only threat model is a special case of Theorem 3. So we omit it.

Inputs

Public param: a set of users $\mathcal{U}$ with $|\mathcal{U}| = m$, a threshold value $t < m$

Server (S): Nothing ($\perp$)

User (P_i): Private input $\mathbf{s}_i$, $i \in \mathcal{U}$

Outputs

User (P_i): Nothing ($\perp$)

Server (S): $\mathbf{s} = \sum_{u \in \mathcal{V}} \mathbf{s}_u$, $\mathcal{V} \subseteq \mathcal{U}$ **if** $|\mathcal{V}| \geq t$ and $\perp$ otherwise.

Figure 12: Ideal functionality $\mathcal{F}_{\text{AGG}}$.

Lemma 1. *Assume that the aggregate-sum is ϵ-private. For the regression models, the gradient $\boldsymbol{\omega}$ on $\{\mathcal{D}_i\}_{i \in \mathcal{B}}$ and $\mathcal{B} \subseteq [m]$ with $|\mathcal{B}| \geq 2t$ does not leak any information about an honest user's $\mathcal{D}_i$ with $|\mathcal{D}_i| \geq 1$ for a coalition of up to $(t-1)$ users and the server, where $t = \lceil \frac{m}{3} \rceil$.*

Proof. For the linear and logistic regression, the local gradient $\boldsymbol{\omega}_i$ on $\mathcal{D}_i$ with size d_i can be written as

$$\boldsymbol{\omega}_i = \Big(\sum_{j=1}^{d_i} e^{(j)}, \sum_{j=1}^{d_i} e^{(j)} x_1^{(j)}, \cdots, \sum_{j=1}^{d_i} e^{(j)} x_n^{(j)}\Big)$$

where $e^{(j)} = (h(\boldsymbol{\theta}, \mathbf{x}^{(j)}) - y^{(j)})$ and h is a linear or logistic regression function. The gradient $\boldsymbol{\omega} = (\omega_0, \cdots, \omega_n)$ on the dataset $\{\mathcal{D}_i\}_{i \in \mathcal{B}}$ is given by

$$\boldsymbol{\omega} = \sum_{i \in \mathcal{B}} \boldsymbol{\omega}_i = \Big(\sum_{i \in \mathcal{B}} \sum_{j=1}^{d_i} e^{(j)}, \sum_{i \in \mathcal{B}} \sum_{j=1}^{d_i} e^{(j)} x_1^{(j)}, \cdots, \sum_{i \in \mathcal{B}} \sum_{j=1}^{d_i} e^{(j)} x_n^{(j)}\Big).$$

The first component of $\boldsymbol{\omega}$ can be written as $\omega_0 = \sum_{i \in \mathcal{B} \backslash C} \sum_{j=1}^{d_i} e^{(j)} + \sum_{i \in C} \sum_{j=1}^{d_i} e^{(j)}$. For a set of corrupted users of size up to $t-1$, the number of terms in $\sum_{i \in \mathcal{B} \backslash C} \sum_{j=1}^{d_i} e^{(j)}$ is $\sum_{i \in \mathcal{B} \backslash C} d_i$, and similarly for other components. In worst case, when $|\mathcal{D}_i| = 1$ and the number of the honest users in $\mathcal{B} \backslash C$ is at least $t \geq \epsilon$, the gradient leaks no information about $\boldsymbol{\omega}$ due to ϵ-privacy of the aggregate. □

Proofs of Theorems 3 and 4. Below we provide the proofs of the security theorems of the training protocols.

Proof of Theorem 3. We will prove the security of the protocols in $(\mathcal{F}_{\text{SLG}}, \mathcal{F}_{\text{AGG}})$-hybrid model using the standard hybrid argument. For π_{LinTrain} and π_{LogTrain}, the ideal functionality $\mathcal{F}_{\text{SLG}}$ is replaced by $\mathcal{F}_{\text{LinSLG}}$ and $\mathcal{F}_{\text{LogSLG}}$, respectively.

We provide a construction of a simulator through a sequence of hybrids which are constructed by subsequent modifications and argue that every two subsequent hybrids are computationally indistinguishable. The simulator $\mathcal{S}$ runs the adversary $\mathcal{A}$ internally and provides the corrupted users inputs, and can emulates the honest parties' inputs as the actual inputs of the honest parties are unknown. $\mathcal{S}$ has access to $\mathcal{F}_{\text{SLG}}$ and $\mathcal{F}_{\text{AGG}}$. We denote by C the set of corrupted parties and $|C| \leq t$.

Hyb 0: This hybrid is a random variable corresponding to the joint view of $\mathcal{A}$ in the real-world execution of the protocol.

Hyb 1: This hybrid is identically same as the previous one, except the key agreement phase. For the honest users in $\mathcal{U}_0 \backslash C$, instead of using the DH key agreement algorithm (KeyAgree), $\mathcal{S}$ uses a pair of uniformly random keys for encryption/decryption and one-time key generation. The Decisional Diffie-Hellman assumption ensures that this hybrid is indistinguishable from the previous one.

Hyb 2: Note that $\mathbf{X}_{sum} = \mathbf{X}_{sum}^{C} + \mathbf{X}_{sum}^{\mathcal{U}_1 \backslash C}$ where $\mathbf{X}_{sum}^{C}$ is the sum of the corrupted user inputs and $\mathbf{X}_{sum}^{\mathcal{U}_1 \backslash C}$ is the sum of the honest users inputs. When $\mathbf{X}_{sum} = \perp$, $\mathcal{S}$ aborts. In this hybrid, $\mathcal{S}$ samples $\{Z_u\}_{u \in \mathcal{U}_1 \backslash C}$ such that $\sum_{u \in \mathcal{U}_1 \backslash C} Z_u = \mathbf{X}_{sum}$. Instead of sending $\{X_u\}_{u \in \mathcal{U}_1 \backslash C}$ as inputs for the honest users, $\mathcal{S}$ sends $\{Z_u\}_{u \in \mathcal{U}_1 \backslash C}$ as inputs to $\mathcal{F}_{\text{AGG}}$. As $|\mathcal{U}_1 \backslash C| \geq t$, the ϵ-privacy of the aggregation-sum ensures that the distributions $\{Z_u\}_{u \in \mathcal{U}_1 \backslash C}$ and $\{X_u\}_{u \in \mathcal{U}_1 \backslash C}$ are identical, where the number of inputs in the sum of the honest users' inputs is $\ell t \geq \epsilon$. Thus, this hybrid is indistinguishable from **Hyb 1**.

Hyb 3: In this hybrid, we change the encryption of $\boldsymbol{\theta}$ by $\mathrm{E}(\mathbf{0})$. $\mathcal{A}$'s view in the real-world execution of the protocol contains $\mathrm{E}(\boldsymbol{\theta})$. The semantic security of the encryption scheme ensures that this hybrid is indistinguishable from the previous one.

Hyb 4: In this hybrid, for each honest user, the input $\mathcal{D}_u$ is sampled as $\mathcal{D}'_u$ s.t. $Z_u = \left(\sum_{j=1}^{|\mathcal{D}'_u|} x_1^{(j)'}, \cdots, \sum_{j=1}^{|\mathcal{D}'_u|} x_n^{(j)'^2}\right)$, $u \in \mathcal{U}_1 \backslash C$. Note that $\mathcal{S}$ uses the knowledge of $\{Z_u\}$ in **Hyb 2**. $\mathcal{S}$ constructs $\mathcal{V}$ by randomly choosing M users with $M \geq 2t$. $\mathcal{S}$ aborts if $|\mathcal{V}| < M$. For each honest user in $\mathcal{V} \backslash C$, $\mathcal{S}$ sends $\mathcal{D}'_u$ and $\boldsymbol{\theta}$ to $\mathcal{F}_{\text{SLG}}$ and receives a random share $\mathbf{s}'_u$. $\mathcal{S}$ runs polynomial-many $\mathcal{F}_{\text{SLG}}$ and obtains $\{\mathbf{s}'_u\}_{u \in \mathcal{V} \backslash C}$. $\mathcal{A}$'s view in the real-world execution contains $\{\mathbf{s}_u\}_{u \in \mathcal{V} \backslash C}$ with $\mathbf{s}_u = \boldsymbol{\omega}_u + \mathbf{r}_u$ where $\{\mathbf{r}_u\}$ is randomly generated. Thus, the distribution $\{\mathbf{s}'_u\}_{u \in \mathcal{V} \backslash C}$ is identically distributed to $\{\mathbf{s}_u\}_{u \in \mathcal{V} \backslash C}$. Thus, this hybrid is indistinguishable from **Hyb 3**.

Hyb 5: In this hybrid, as $\mathbf{r}_{sum} = \mathbf{r}_{sum}^{C} + \mathbf{r}_{sum}^{\mathcal{V}'' \backslash C}$, $\mathcal{S}$ emulates the honest users' inputs $\left\{\mathbf{r}'_u\right\}_{u \in \mathcal{V}'' \backslash C}$ such that $\sum_{u \in \mathcal{V}'' \backslash C} \mathbf{r}'_u = \sum_{u \in \mathcal{V}'' \backslash C} \mathbf{r}_u = \mathbf{r}_{sum}^{\mathcal{V}'' \backslash C}$, where $\left\{\mathbf{r}_u\right\}_{u \in \mathcal{V}'' \backslash C}$ is randomly distributed, and for the honest users in $\mathcal{V}' \backslash (\mathcal{V}'' \backslash C)$, it randomly samples arbitrary values and sends all to $\mathcal{F}_{\text{AGG}}$. Note that $\boldsymbol{\omega} = \boldsymbol{\omega}^{C} + \boldsymbol{\omega}^{\mathcal{V}'' \backslash C} = \sum_{u \in C} (\mathbf{s}_u - \mathbf{r}_u) + \sum_{u \in \mathcal{V}'' \backslash C} (\mathbf{s}_u - \mathbf{r}_u)$, and the number of honest users in $\mathcal{V}'' \backslash C$ is $|\mathcal{V}'' \backslash C| = M - \delta \geq \rho$. Therefore, $\boldsymbol{\omega}^{\mathcal{V}'' \backslash C}$ is the sum of at least $\rho \cdot \ell = \epsilon$ inputs. In the real-world execution, any leakage about an individual $\mathbf{s}_u$ can happen from the execution of the aggregation protocol with a negligible probability. The security of the aggregation protocol and the ϵ-privacy of the aggregate-sum ensures that the distributions $\left\{\mathbf{r}'_u\right\}_{u \in \mathcal{V}'' \backslash C}$ and $\left\{\mathbf{r}_u\right\}_{u \in \mathcal{V}'' \backslash C}$ are identical and so do $\left\{\boldsymbol{\omega}_u\right\}_{u \in \mathcal{V}'' \backslash C}$ and $\left\{\boldsymbol{\omega}'_u\right\}_{u \in \mathcal{V}'' \backslash C}$ with $\sum_{u \in \mathcal{V}'' \backslash C} \boldsymbol{\omega}'_u = \boldsymbol{\omega}^{\mathcal{V}'' \backslash C}$ emulated by $\mathcal{S}$. Therefore, this hybrid is indistinguishable from the previous one.

Hyb 6: We repeat **Hyb 3** to **Hyb 5** sequentially $(R-1)$ times (polynomial many times), and it is easy to observe that each subsequent modification of the hybrids is indistinguishable, by applying the above arguments.

This concludes the construction of the simulator. Thus, the output of the simulator is computationally indistinguishable from the output of the real-world execution of the protocol. Hence the proof follows. □

Proof of Theorem 4. This can be proved by constructing a simulator in a similar way, as of Theorem 3. We only emphasize the main behavior changes of the simulator and omit other details. Since the adversary corrupts only the set of users in C, the joint view of the adversary, after the data scaling phase, contains $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, which involves the honest users inputs. $\mathcal{S}$ emulates the honest users' (in $\mathcal{U}_1 \backslash C$) inputs $\mathcal{D}'_u$ such that $\sum_{u \in C} W_u + \sum_{u \in \mathcal{U}_1 \backslash C} Y_u = d_1 \cdot \boldsymbol{\mu}$ where $Y_u = \left(\sum_{j=1}^{|\mathcal{D}'_u|} x_1^{(j)'}, \cdots, \sum_{j=1}^{|\mathcal{D}'_u|} x_n^{(j)'}\right)$ and $W_u = \left(\sum_{j=1}^{|\mathcal{D}_u|} x_1^{(j)}, \cdots, \sum_{j=1}^{|\mathcal{D}_u|} x_n^{(j)}\right)$ and uses the true inputs $\mathcal{D}_u$ for the corrupted users $u \in C$, and these inputs are used in the rest of the simulation. Note that in the training phase no output is received by the users in C. The simulator can use a dummy vector for the sum of the honest users' local gradient $\boldsymbol{\omega}^{\mathcal{V}'' \backslash C}$. □