Privacy-Preserving Distributed Linear Regression on High-Dimensional Data

Adrià Gascón[1], Phillipp Schoppmann[2], Borja Balle[3], Mariana Raykova[4], Jack Doerner[5], Samee Zahur[6], and David Evans[7]

[1] University of Edinburgh
[2] Humboldt University of Berlin
[3] Lancaster University
[4] Yale University
[5] Northeastern University
[6] Google
[7] University of Virginia

Abstract. We propose privacy-preserving protocols for computing linear regression models, in the setting where the training dataset is vertically distributed among several parties. Our main contribution is a hybrid multi-party computation protocol that combines Yao's garbled circuits with tailored protocols for computing inner products. Like many machine learning tasks, building a linear regression model involves solving a system of linear equations. We conduct a comprehensive evaluation and comparison of different techniques for securely performing this task, including a new Conjugate Gradient Descent (CGD) algorithm. This algorithm is suitable for secure computation because it uses an efficient fixed-point representation of real numbers while maintaining accuracy and convergence rates comparable to what can be obtained with a classical solution using floating point numbers. Our technique improves on Nikolaenko et al.'s method for privacy-preserving ridge regression (S&P 2013), and can be used as a building block in other analyses. We implement a complete system and demonstrate that our approach is highly scalable, solving data analysis problems with one million records and one hundred features in less than one hour of total running time.

1 Introduction

Predictive modeling is an essential tool in decision making processes in domains such as policy making, medicine, law enforcement, and finance. To obtain more accurate models, it is common that organizations cooperate to build joint training datasets, specially when the analysis at hand involves complex phenomena whose study requires vast amounts of data. However, collaborative analyses involving private data are often limited by ethical and regulatory constraints. As an example, consider a research project consisting of a study to predict medical conditions given data related to socio-economic background. While databases holding medical, judicial, and tax records linkable by common unique identifiers (e.g., social security numbers) exist, they are often held by different institutions.

One solution is to rely on a trusted third-party that is provided with the databases from each party *in the clear*, which then performs the analysis over the data and returns the result to all parties. This approach requires all participants to agree on one party they trust, and poses an obvious privacy threat to the data. Finding effective ways of limiting the amount of trust to be placed on such third-party is an ongoing research challenge being tackled simultaneously from many perspectives.

Our goal is to enable such joint analyses without the need to expose each organization's sensitive data to any other party. We consider a natural setup where parties have incentives to collaborate in obtaining the parameters of a regression model, but are not willing to disclose any information about their data beyond what is revealed by the learned model. Secure Multi-Party Computation (MPC) protocols provide a general mechanism for multiple parties to jointly compute a function on their private inputs. However, existing out-of-the-box MPC techniques do not yet scale to problem sizes encountered in real-world data analysis applications. The goal of this work is to circumvent such limitations by developing application-specific efficient protocols with better performance than generic MPC techniques.

The motivation to join databases in the medical example above is to build a more accurate model by using a joint training dataset of *higher dimensionality* than any of the original ones. This is known as data analysis on *vertically-partitioned* datasets, as the columns (i.e. features) in the training dataset are distributed among several parties. This is in contrast with the *horizontally-partitioned* setting, where parties produce a joint dataset by contributing rows (i.e. data points) involving a fixed set of features.

Linear regression is a fundamental machine learning task that fits a linear curve over a set of high-dimensional data points. An important property of this problem is that it can be cast as an optimization problem whose solution admits a closed-form expression. Formally, linear regression can be reduced to solving a system of linear equations of the form $A\theta = b$. Interestingly, solving systems of linear equations is a basic building block of many other machine learning algorithms [55]. Thus, secure MPC solutions for solving linear systems can also be used to develop other privacy-preserving data analysis algorithms.

Contributions. In this paper, we propose a solution for securely computing a *linear regression* model from a vertically-partitioned dataset distributed among an arbitrary number of parties. Our protocols are the result of securely composing a protocol for privately computing the coefficients of the system of equations $A\theta = b$ from distributed datasets (*aggregation phase*), and a protocol for securely solving linear of systems (*solving phase*). Our main contributions are as follows:

– Scalable MPC protocols for linear regression on vertically partitioned datasets (Section 3). We design, analyze, and evaluate two hybrid MPC protocols allowing for different trade-offs between running time and the availability of external parties that facilitate the computation.
– A fast solver for high-dimensional linear systems suitable for MPC (Section 5). Our solver relies on a new Conjugate Gradient Descent (CGD) algo-

rithm that scales better, both in terms of running time and accuracy, than the best previous MPC-based alternative.
- We conduct an extensive evaluation of our system on real data from the UCI repository [46] (Section 6.4). The results show that our system can produce solutions with accuracy comparable to those obtained by standard tools without privacy constraints.

Our work is also the first to undertake a formal analysis of the effect different number representations have on the accuracy of privacy-preserving regression protocols. We further propose a thorough data pre-processing pipeline to mitigate accuracy loss (Section 4).

While the basic versions of our protocols are secure against semi-honest adversaries, in Section 7 we discuss extensions of our protocol beyond semi-honest security. We implemented a protocol extension providing stronger security guarantees for the solving phase and observed that it introduces minimal overhead over the semi-honest solution.

Code for our complete system is publicly available under open-source licenses [67].

Related work. Questions of privacy-preserving data mining and private computation of machine learning algorithms have been considered in several works [49,20,41,72], providing theoretical protocols without implementations or practical evaluations of their efficiency. Early implementations of privacy-preserving linear regression protocols [22,66] either don't use formal threat models, or leak additional information beyond the result of the computation.

Hall et al. [37] were the first to propose a protocol for linear regression with formally defined security guarantees. However, due to the dependence on expensive homomorphic encryption, the resulting system does not scale well to large datasets.

Nikolaenko et al.'s work [60] on secure computation for linear regression is the one most similar to ours. In particular, their approach also considers a protocol split into aggregation and solving phases implemented using multiple MPC primitives. However, Nikolaenko et al. only consider the horizontally-partitioned setting while we focus on the more challenging case of vertically-partitioned datasets. Our implementation of the solving phase using CGD improves the results obtained using Nikolaenko et al.'s secure Cholesky decomposition in terms of both speed and accuracy for datasets with more than 100 features (experiments in [60] were limited to datasets with at most 20 features). Furthermore, our CGD algorithm can also be used directly to improve other MPC protocols requiring a secure linear system solver [60,37,35].

Vertically-partitioned databases have been also studied from the perspective of privacy-preserving querying of statistical databases [23]. This work falls under the paradigm of differential privacy [24], where the goal is to mitigate the privacy risks incurred by revealing to an attacker the model computed by a machine learning algorithm. Our work addresses an orthogonal concern, namely that of preserving privacy during the computation the model. Combining MPC and differential privacy is an interesting problem we plan to address in future work.

2 Background

This section provides background on linear regression and MPC techniques that will be used in the sequel.

2.1 Linear Regression

Linear regression is a fundamental building block of many machine learning algorithms. It produces a model given a training set of sample data points by fitting a linear curve through them. Formally, a linear model is a real-valued function on d-dimensional vectors of the form $x \mapsto \langle \theta, x \rangle$, where $x \in \mathbb{R}^d$ is the input vector, $\theta \in \mathbb{R}^d$ is the vector of parameters specifying the model, and $\langle \theta, x \rangle = \sum_{j=1}^{d} \theta_j x_j$ is the inner product between θ and x. The term *linear* comes from the fact that the output predicted by the function $\langle \theta, x \rangle$ is a linear combination of the features represented in x. The learning algorithm has access to a *training set* of samples representing the input-output relation the model should try to replicate. These are denoted as $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$ with $x^{(i)} = (x_1^{(i)}, \ldots, x_d^{(i)}) \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$ for $i \in [n]$.

A standard approach to learn the parameters of a linear model is to solve the *ridge regression* optimization:

$$\operatorname*{argmin}_{\theta} \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \langle \theta, x^{(i)} \rangle)^2 + \lambda \|\theta\|^2 \tag{1}$$

Here, $\lambda \|\theta\|^2$ is known as a ridge (or Tikhonov) regularization term weighted by the regularization parameter $\lambda > 0$. When $\lambda = 0$, this optimization finds the parameter vector θ minimizing the *mean squared error* between the predictions made by the model on the training examples $x^{(i)}$ and the desired outputs $y^{(i)}$. The ridge penalty is used to prevent the model from overfitting when the amount of training data is too small.

The optimization problem (1) is convex and its solution admits a closed-form expression. A way to derive this solution is to reformulate the optimization problem by writing $X \in \mathbb{R}^{n \times d}$ for the matrix with n rows corresponding to the different d-dimensional row vectors ${x^{(i)}}^\top$ so that $X(i,j) = x_j^{(i)}$, and $Y \in \mathbb{R}^n$ for a column vector with n entries corresponding to the training labels $y^{(i)}$. With this notation, (1) is equivalent to

$$\operatorname*{argmin}_{\theta} \frac{1}{n} \|Y - X\theta\|^2 + \lambda \|\theta\|^2 \ .$$

Now, by taking the gradient of this objective function and equating it to zero one can see that the optimization (1) reduces to solving this system of linear equations:

$$\left(\frac{1}{n} X^\top X + \lambda I \right) \theta = \frac{1}{n} X^\top Y \tag{2}$$

where I represents the $d \times d$ identity matrix.

Therefore, ridge regression reduces to solving a system of linear equations of the form $A\theta = b$, where the coefficients $A = \frac{1}{n}X^\top X + \lambda I$ and $b = \frac{1}{n}X^\top Y$ only depend on the training data (X, Y). Since the coefficient matrix A is positive definite by construction ridge regression can be solved using one of the many known algorithms for solving positive definite linear systems.

2.2 Multi-Party Computation

Multi-Party Computation (MPC) is an area of cryptography concerned with enabling multiple parties to jointly evaluate a public function on their private inputs, without leaking any information about their respective inputs (other than their sizes and whatever can be inferred from the result of the computation). Several generic techniques have been developed for MPC including Yao's garbled circuits protocol [71], the GMW protocol [32], and other secret-sharing based techniques [9,52,42]. These differ in several aspects including the number of parties they support, the representation they use for the evaluated function (e.g., binary vs. arithmetic circuits), the function families that can be evaluated securely with these solutions (e.g., bounded-degree polynomials vs. arbitrary polynomials), as well trade-offs in terms of communication, computation, and security guarantees that they provide. In Section 3, we keep these differences in mind in order to choose the right MPC techniques for each part of the computation. Popular applications used for the empirical evaluation of practical MPC systems include graph algorithms [34,8,43,53,58], data structure algorithms [43,53], string matching and distance algorithms [38,39,68] and AES [38,39,6,68].

Other than in Section 7, this paper assumes *semi-honest adversaries*. Such adversaries are limited to observing the full trace of the protocol execution without influencing it. This contrasts with *malicious adversaries*, that take complete control of protocol participants, and may deviate from the protocol in arbitrary ways. For detailed definitions of security in both models see Goldreich [31]. To prove security, we use a simulation-based approach [31,48] where one argues that a protocol is secure if it can be simulated in an ideal environment in which there is no data leakage by definition.

For our solving phase, we use Yao's two-party garbled circuits protocol, which was formally-described and proven secure by Lindell and Pinkas [50]. In its basic form, this protocol provides semi-honest security for computing arbitrary functions represented as a binary circuit. A two-party computation protocol based on garbled circuits has three phases: circuit garbling, input garbling and garbled circuit evaluation. In the first step one of the parties (the garbler) creates a garbled representation of the evaluated Boolean circuit where wire values are represented with cryptographic keys (garbled labels) and the circuit gates are implemented as garbled tables of ciphertexts that are encrypted using the wire labels. The garbled circuit consisting of all garbled gates is provided to the other party (the evaluator) who can evaluate it on the input for which it is given the corresponding input garbled labels. These input garbled values are provided in the input garbling phase. While the garbler can directly send the garbled values for its input bits, the parties need to run an oblivious transfer (OT) protocol [63,56] to enable the evaluator to obtain the input garbled labels corresponding its input bits without revealing those inputs to the generator.

3 Protocol description

We consider the problem of solving a ridge regression problem when the training data ($X \in \mathbb{R}^{n \times d}, Y \in \mathbb{R}^n$) is distributed vertically (by columns) among several parties. As in previous work on privacy-preserving ridge regression [37,60], we assume that the regularization parameter λ is public to all the parties, and hence the implementation of a secure data analysis pipeline including hyper-parameter tuning (e.g., cross-validation) is beyond the scope of this paper. Furthermore, we also make the assumption in the vertically partitioned case that the correspondence between the rows in the datasets owned by different parties is known a priori. This assumption is easy to enforce in databases with unique identifiers for each record, given that efficient private set intersection protocols exist. The more complex task of privacy-preserving entity resolution is beyond the scope of this paper.

As discussed in Section 2.1, solving a ridge regression problem reduces to the solution of a positive definite linear system. A superficial inspection of Equation (2) shows that this can be seen as two sequential tasks: (i) compute the matrix $A = \frac{1}{n}X^\top X + \lambda I$ and the vector $b = \frac{1}{n}X^\top Y$, and (ii) solve the system $A\theta = b$. In the distributed privacy-preserving setting, task (i) above corresponds to an *aggregation phase* where data held by different parties has to be combined. In contrast, we refer to (ii) as the *solving phase*, where the system of linear equations resulting from the aggregation is solved and all the parties obtain the solution in some form. It is important to note that in practice, ridge regression is used on problems with $n \gg d$, and that, while the input of the aggregation phase is of the order of $n \times d$, the solving phase has input size independent of n, i.e. of the order of d^2. This demonstrates the importance of having an efficient aggregation phase.

As mentioned in the introduction, the scenario where (X, Y) is partitioned horizontally among the parties was studied by Nikolaenko et al. [60]. With this partitioning, the computation of A and b can be cast as an addition of matrices computed *locally* by the parties. More concretely, in that case one can write $A = \sum_{i=1}^n A_i + \lambda I$ with $A_i = x^{(i)}{x^{(i)}}^\top/n$, and similarly $b = \sum_{i=1}^n = b_i$ with $b_i = x^{(i)}y^{(i)}/n$. The absence of the need for secure multiplication is crucial from a secure computation perspective, and guided the design of the protocol of [60], as non-linear operations are expensive in secure computation. In contrast, the setting where the data is vertically partitioned requires secure computation of inner products on vectors owned by different parties. In the following section, we analyze the exact requirements for this setting, and propose secure computation protocols to overcome this challenge.

One could consider further variants of this problem involving alternative partition schemes motivated by specific applications. For example, the variant where the dataset is secret-shared among several parties for secure storage, or the particular case of the vertically partitioned scenario where one party holds X and the other Y. Under such constraints, dedicated efficient hybrid protocols can be tailored for each case.

3.1 Aggregation Phase

For clarity of presentation, we first describe the two-party case, where X is vertically partitioned among two parties P_1 and P_2 as X_1 and X_2, and Y is held by one of the parties, say P_2. As described above, the goal of this phase is to have the parties hold additive shares of (A, b). With this notation, the functionality f of the aggregation phase can be described as follows:

$$f(X_1, X_2, Y) = \big((A - A_2, b - b_2), (A_2, b_2)\big) \ ,$$

where $(A - A_2, b - b_2)$ and (A_2, b_2) are the outputs received by the first and the second party respectively.

Our protocol implements f in a secure way. Let us now have a closer look at what needs to be computed. For simplicity, assume that each party holds a block of contiguous features. Then, the following equations show how the content of the output matrix depends on the inputs of the two parties.

$$X^\top = \begin{bmatrix} X_1^\top \\ X_2^\top \end{bmatrix} \quad X = [X_1 \ X_2]$$

$$X^\top X = \begin{bmatrix} X_1^\top X_1 & X_1^\top X_2 \\ X_2^\top X_1 & X_2^\top X_2 \end{bmatrix}$$

Observe that the upper left part of the matrix $M = X^\top X$ depends only on the input of party P_1 and the lower right part depends only on the input of party P_2. Hence, the corresponding entries of M can be computed locally by P_1, while P_2 simply has to set her shares of those entries to 0. On the other hand, for entries M_{ij} of M such that features i and j are held by distinct parties, the parties need to compute an inner product between a column vector from X_1 and a column vector from X_2. To do so, we rely on a secure inner product protocol which we present next (Section 3.1). We note also that because the two off-diagonal blocks of M are transpositions of each other, computing only one of these blocks is enough to get an additive share of M.

In the multi-party case, similarly to the two-party case, the vertical partitioning of the database implies that each party holds a subset of the columns of X and a corresponding subset of the rows of $X^\top$. Since each value in A is an inner product between a row vector of $X^\top$ and a column vector of X, this means that each value in A depends on the inputs of at most two parties. Thus, also in the multi-party case, the aggregation phase for vertically-partitioned datasets can be reduced to secure two-party computation of inner product. At the end of the aggregation phase, the parties will obtain A in a form where each entry is either known to one party or is shared between some pair of input parties.

Secure Inner Product In this section, we present protocols for two parties P_1 and P_2, holding vectors $\vec{x}$ and $\vec{y}$, respectively, to securely compute the inner product of their vectors and obtain additive shares of the result. As mentioned before, we use a fixed-point encoding for real numbers (see Section 4 for details). Thus, our numbers can be represented as elements of a finite field $\mathbb{Z}_q$, and hence the functionality we need can be described as

$$f(\vec{x},\vec{y}) = (r, \langle\vec{x},\vec{y}\rangle - r) \tag{3}$$

where $\vec{x},\vec{y} \in \mathbb{Z}_q^n$, r is a random element of $\mathbb{Z}_q$, and each party gets one of the components of the output. There are several techniques that can be used for this task as, essentially, it corresponds to secure multiplication.

Generally, MPC techniques represent the implemented functionality either as arithmetic circuits (e.g. SPDZ [17], TinyOT [59]) or as boolean circuits (e.g. Yao's protocol [71], GMW [32]). While the former makes it possible to represent arithmetic operations very efficiently, the latter is better for performing bit-level operations. A second important property of MPC protocols is round complexity. Most notably, some techniques such as Yao's garbled circuits have a *constant* number of rounds, while for others (e.g. SPDZ, TinyOT, GMW), the number of rounds increases with the multiplicative depth of the circuit being evaluated (for boolean circuits, this is the AND-depth). Note that the functionality in Equation (3) only consists of additions and multiplications. Furthermore, all multiplications can be done in parallel. Thus, a representation as an arithmetic circuit with constant multiplicative depth is the natural choice.

We propose two protocols for the functionality in Equation (3). The first is an extension of OT-based secure multiplication, which was originally proposed by Gilboa [30], and is used in different MPC contexts [42,19]. It uses OT to compute additive shares of the product of two numbers in a binary field, which can be trivially extended to securely compute the inner product of two vectors.

Our second secure inner product protocol uses only symmetric key operations and is much more efficient than the previous one. It builds upon techniques based on precomputation of multiplicative triples [5], as the secret sharing techniques mentioned above, and relies on an initializer that aids in the computation. Our protocol can also be seen as a modification of the construction presented by Du and Attalah [21] where the trusted initializer does not keep a share of the result, and is presented in Figure 1.

In both our protocols, we exploit the facts that, in our case, (i) the input vectors are each completely owned by one of the parties, and not shared among them, (ii) only two-party computations are needed, (ii) we are concerned about semi-honest security at this point. These observations lead to simpler protocols than general MPC protocols based on secret sharing.

In Section 6.3 we present an evaluation of our two protocols for secure inner product, in the context of our implementation of the aggregation phase for secure linear regression.

Parties: P_1, P_2, and Trusted Initializer TI.
Inputs: $\mathsf{P}_1 : \vec{x} \in \mathbb{Z}_q^n$; $\mathsf{P}_2 : \vec{y} \in \mathbb{Z}_q^n$.
Outputs: $\mathsf{P}_1 : s_1 \in \mathbb{Z}_q$; $B : s_2 \in \mathbb{Z}_q$ such that $s_1 + s_2 = \langle \vec{x}, \vec{y} \rangle$.

Protocol:

1. TI generates random vectors $\vec{a}, \vec{b} \in \mathbb{Z}_q^n$ and a random number $r \in \mathbb{Z}_q$, and sets $z = \langle \vec{a}, \vec{b} \rangle - r$. It sends $(\vec{a}, r)$ to P_1, and $(\vec{b}, z)$ to P_2.
2. P_1 sends $\vec{x} + \vec{a}$ to P_2.
3. P_2 sends $\vec{y} - \vec{b}$ to P_1.
4. P_1 computes its output share as $s_1 = \langle \vec{x}, \vec{y} - \vec{b} \rangle - r$.
5. P_2 computes its output share as $s_2 = \langle \vec{x} + \vec{a}, \vec{b} \rangle - z$.

Fig. 1: Secure two-party inner product/multiplication supported by a trusted initializer. Its security relies on the fact that the CSP does not collude with the data providers, as described in Lemma 1.

3.2 Solving Phase

After the aggregation phase, the parties hold additive shares of a square, symmetric, positive definite matrix A and a vector b, and the task is to securely solve $A\theta = b$.

In line with previous work [60,37], and in order to achieve constant round-complexity, we choose Yao's garbled circuits for our solving phase, and introduce two additional non-colluding parties. These aid in the computation without learning neither the result, nor anything about the parties' input. In fact, our architecture relying on two external parties can be seen as a way of reducing a multi-party problem to a two-party problem, and had been used before in [60] and [57]. Analogously to the presentation in [60], we call our additional roles Crypto Service Provider (CSP) and Evaluator. For clarity, from now on we will call our parties *data providers*. The CSP is in charge of generating a garbled circuit for solving $A\theta = b$, while the Evaluator will do the evaluation. In contrast to [60], the Evaluator does not necessarily learn the solution of the system in our protocol. The solving phase of our protocol is presented in Figure 2.

Note that the CSP and the Evaluator are only required to be non-colluding and hence, for the protocol in Figure 2, their roles could be taken by some of the data providers. In particular, for the case with 2 data providers, no additional parties are needed. We discuss security considerations of the solving phase protocol in the next section, where we present our complete protocol for secure linear regression.

Finally, let us note that we have not specified the details of the circuit $\mathcal{C}$ yet. It is important to remark that, after having designed a fast aggregation phase, the bottleneck of our protocol in terms of running time will now be in the solving phase. Hence, the concrete algorithm for solving $A\theta = b$ is a

Parties: Data providers $P_1 \dots, P_n$, CSP, and $\mathsf{Evaluator}$.
Inputs: P_i : share (A_i, b_i) of an equation $A\theta = b$.
Output: The data providers learn a solution of $A\theta = b$.

Protocol:

1. The CSP generates a garbled circuit $\mathcal{C}$ for a functionality $f((A_1, b_1), \dots, (A_n, b_n))$ that reconstructs a solves $A\theta = b$ and sends it to the $\mathsf{Evaluator}$.
2. Each data provider P_i runs oblivious transfers with the CSP to obtain garbled values $(\hat{A}_i, \hat{b}_i)$ for (A_i, b_i) in $\mathcal{C}$ and forwards them to the $\mathsf{Evaluator}$.
3. The $\mathsf{Evaluator}$ evaluates $\mathcal{C}$ and shares $\hat{\theta}$ with the data providers.
4. The CSP sends the decryption mappings for the data providers to obtain θ from $\hat{\theta}$.

Fig. 2: The solving phase of our protocol. Its security in the semi-honest threat model follows from the security proof for Yao's protocol in [50].

crucial element for the scalability of our protocol for secure linear regression. In Sections 5 and 4 we discuss desirable properties of a good algorithm for solving systems of linear equations tailored for MPC, and propose a Conjugate Gradient Descent algorithm. Finally, let us mention that, as we will see in Section 5, the circuit $\mathcal{C}$ has high degree. As mentioned in the previous section this motivates the choice of GCs as underlying MPC technique. An extensive evaluation of MPC techniques for the task of linear system solving is left for further work.

3.3 Secure Linear Regression

Our main protocol is depicted in Figure 3. In this case, the composition between the aggregation and solving phases is implemented in steps (d) and (e) by means of oblivious transfers between the CSP and the data providers. Here, the roles of Trusted Initializer and CSP in the aggregation and solving phases are taken by a single additional non-colluding party, while the role of the $\mathsf{Evaluator}$ could be taken by one of the data providers.

In the two-party case, the roles of the CSP and the $\mathsf{Evaluator}$ can be taken by the data providers, and the composition of the two phases is straightforward, while the role of Trusted Initializer in the aggregation phase is taken by an additional non-colluding party that aids in the computation.

A variant of our protocol described above implements the aggregation phase using the protocol based on OT (Section 3.1). In this way, we remove the need for an external semi-trusted party. We will consider the performance of this variant in the experimental evaluation, and revisit it when we consider extensions of our protocol to withstand malicious adversaries.

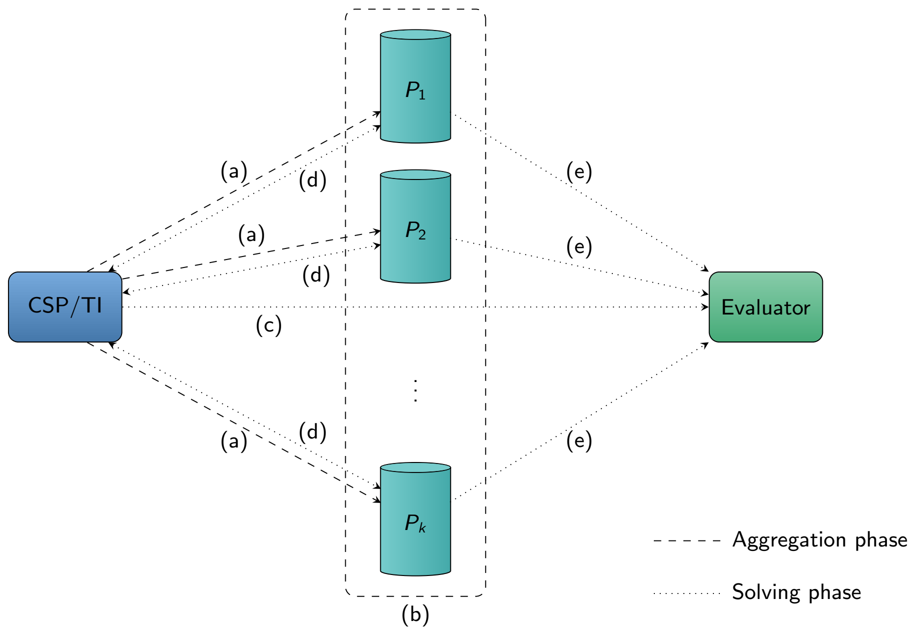


Fig. 3: The multi-party protocol: (a,b) The data providers compute additive shares of $A\theta = b$, by doing local precomputations and executing several instances of the inner product protocol of Figure 1. (c) The CSP generates a garbled circuit $\mathcal{C}$ for a functionality that reconstructs and solves $A\theta = b$ and sends it to the Evaluator. (d) Each data provider P_i obtains garbled values for their shares using OT and forwards them to the Evaluator. (e) The Evaluator evaluates $\mathcal{C}$ and shares the garbling of the result $\hat{\theta}$ with the data providers. Finally, The CSP sends the decryption mappings for the data providers to obtain θ from $\hat{\theta}$.

Up to this point we have focused on design and correctness aspects of our protocols, without discussing their security guarantees. We undertake that task in the following section.

Security against semi-honest adversaries Similarly to how we presented our protocols, we argue security of our linear regression protocol with a modular approach, by proving the security of the protocol based on the security of its subprotocols.

Let us thus first argue security of our protocols for the inner product functionality: while for the security of the OT-based protocol we refer the reader to [30,19], we state the correctness and security of the protocol based on a Trusted Initializer in the following lemma.

Lemma 1. *The protocol in Figure 1 is a secure two-party computation protocol against semi-honest adversaries for the functionality $f(\vec{x}, \vec{y}) = (r, \langle \vec{x}, \vec{y} \rangle - r)$. where $(\vec{x}, \vec{y})$ and $(r, \langle \vec{x}, \vec{y} \rangle - r)$ are the inputs and outputs of the parties.*

Proof. Correctness can be easily verified from the definition of the resulting shares. To prove security, we construct the following two simulators Sim_1 and Sim_2 which simulate the views of the two parties.
Sim_1 generates a view for P_1 given $\vec{x}$ and P_1's output in the ideal functionality, denoted s_1^*. The message from the TI is simulated as $(\vec{a'}, \langle \vec{x}, \vec{b'} \rangle - s_1^*)$, while the message from P_2 is simulated as $\vec{b'}$, where $\vec{a'}, \vec{b'} \in U(\mathbb{Z}_q^n)$. The resulting view has the same distribution as the P_1's view in the real protocol execution.
Similarly, Sim_2 generates a view for P_2 given $\vec{y}$ and s_2^* as follows: the message from the TI is simulated as $(\vec{b'}, \langle \vec{a'}, \vec{b'} \rangle - s_2^*)$, while the message from P_1 is simulated as $\vec{a'}$, where $\vec{a'}, \vec{b'} \in U(\mathbb{Z}_q^n)$. These messages have the same distribution as P_2's view in the real execution, which completes the proof. $\square$

The security of our main protocol depicted in Figure 3 is stated in the following theorem. Informally, the theorem states that, as long as the participants do not deviate from the protocol, and the external parties (CSP/TI and Evaluator) are non-colluding, none of the parties learn anything from the inputs of the data providers that cannot be deduced from the result of the computation, namely the regression parameter θ. Moreover, recall that the external parties do not obtain θ.

Theorem 1 (Security). *Let Π_1 be a secure two-party computation protocol for the inner product functionality defined in Section 3.1) with security against semi-honest adversaries, and Π_2 be a two-party computation protocol based on garbled circuits with semi-honest security. The construction in Figure 3 is a secure computation protocol that provides security against semi-honest adversaries, in the setting where the* CSP/TI *and the Evaluator do not collude with neither the data providers nor each other.*

Proof. We need to show simulation security against a semi-honest non-colluding adversary that controls the CSP, a semi-honest non-colluding adversary that controls the Evaluator, and a semi-honest adversary that controls a subset of the input parties.

The security of the inner product protocol Π_1 implies that there exist simulators $\mathsf{Sim}_{\mathsf{P}_1}$ and $\mathsf{Sim}_{\mathsf{P}_2}$ which can simulate without any inputs the view of each of the two parties in the execution of the protocol. The security of the two party computation protocol based on garbled circuits Π_2 implies the existence of simulators $\mathsf{Sim}_{\mathsf{garb}}$ and $\mathsf{Sim}_{\mathsf{eval}}$ which work as follows. $\mathsf{Sim}_{\mathsf{garb}}$ simulates the view of the garbling party, which consists of the execution of the OT protocols as a sender. $\mathsf{Sim}_{\mathsf{eval}}$ simulated the view of the Evaluator which consists of the garbled circuit itself as well at the view in the execution of the OT protocols as a receiver. We will use this simulators in our proof.
Semi-honest CSP. The view of the CSP in the secure computation protocol consists only of the messages exchanged in the garbled circuit computation where it has no input. Hence we can use $\mathsf{Sim}_{\mathsf{garb}}$ to simulate its view.
Semi-honest Evaluator. Similarly to the case of the CSP, the view of the Evaluator in the execution of the protocol consists of the messages exchanged in the execution of the garbled circuit evaluation where it has no inputs. Therefore, we can use the simulator $\mathsf{Sim}_{\mathsf{eval}}$ that can simulate its view.
Semi-honest Input Parties. The view of an adversary who controls a subset of the input parties in the protocol in Figure 2 consists of the messages exchanged in the executions of the inner product protocol in the aggregation phase with other input parties not controlled by the adversary, the messages in the OT executions with the CSP in the solving phase together with mappings for the output garbled labels as well as the garbled output the parties receive from the Evaluator. We can use these simulators $\mathsf{Sim}_{\mathsf{P}_1}$ and $\mathsf{Sim}_{\mathsf{P}_2}$ to obtain the view of the adversary in the aggregation phase. Additionally we can use $\mathsf{Sim}_{\mathsf{garb}}$ for Π_2 to simulate the view of the adversary in the OT executions. Given the value β of the output solution, which the parties will receive, the simulator $\mathsf{Sim}_{\mathsf{garb}}$ can also generate mappings for the output garbled labels to real values and a set of garbled labels for the output that open to β. This completes the proof. □

The above security argument can be easily adapted to the case where the evaluator is one of the data providers, as long as CSP/TI does not collude with any other party.

We can further extend the security guarantees of our protocol, to the setting where the roles of the CSP and the $\mathsf{Evaluator}$ are executed by two of the parties, as long as these two parties are semi-honest and non-colluding with each other. In this scenario, we use the OT-based protocol for the aggregation phase, as it does not require a Trusted Initializer (which was instantiated by the CSP). The security guarantees for this variant in which *no external parties are required*, are stated in the next theorem.

Theorem 2 (Security without external parties). *Let Π be an instantiation of the protocol described in Figure 3 where the roles of* CSP *and the* $\mathsf{Evaluator}$ *are implemented by two distinct input providers, and the aggregation phase is* *instantiated with the OT-based inner product protocol. Then, Π provides security against semi-honest adversaries in the setting where the* CSP *and the* Evaluator *do not collude.*

In Section 6 we provide an experimental evaluation of the protocols of the two theorems above where we quantify the speed-up obtained by relying on an offline phase performed by a non-colluding external party (TI).

4 Number Representation

Implementing secure MPC protocols to work with numerical data poses a number of challenges. In our case, the most significant of those challenges is choosing an encoding for numerical data providing a good trade-off between efficiency and accuracy. The IEEE 754 floating-point representation is the standard choice used by virtually every numerical computation software because of its high accuracy and numerical stability when working with numbers across a very large range of orders of magnitude. However, efficient implementations of IEEE 754-compliant secure MPC protocols are a subject of on-going research [18,62], with the current state-of-the-art yielding 32 bit floating-point multiplication circuits running in time comparable to computers from the 1960's [70]. Unfortunately, these implementations do not yet scale to the throughput required for data analysis tasks involving large datasets, and forces us to rely on fixed-point encodings like previous work in this area [37,60,12]. The rest of this section presents the details of the particular fixed-point encoding used by our system, with a particular emphasis on its effect on the accuracy of the protocol. We also discuss some important data normalization and scaling steps that are useful in the particular context of linear regression.

4.1 Fixed-point Arithmetic

Our use of fixed-point encodings follows closely previous works on secure linear regression [37,60,12]. However, unlike these, we provide a formal analysis of the errors such encodings introduce when used to simulate operations on real numbers.

In a fixed-point signed binary encoding scheme each number is represented using a fixed number of bits b. These bits are subsequently split into three parts: one sign bit, b_f bits for the fractional part, and b_i bits for the integral part, with the obvious constraint $b = b_i + b_f + 1$. For each possible value of b_f and b_i one gets an encoding capable of representing all the numbers in the interval $[-2^{b_i}+2^{-b_f-1}, 2^{b_i+1}-2^{-b_f-1}]$ with accuracy 2^{-b_f}. In general, arithmetic operations with fixed-point numbers can be implemented using signed integer arithmetic, though special care is required to deal with operations that might result in overflow.

In order to analyze the error incurred by operations working with fixed-point representations we can represent the encoding process using two steps: a first step mapping reals into integers, and a second step mapping integers to integers modulo q with $q = 2^b$. The purpose of the first step is to represent all reals up to a certain precision using integers, thus creating equivalence classes of reals that cannot be distinguished using the given precision. The second step models the fact that a computer with finite memory can only represent finitely many integers, and gives a way to simulate operations on integer-encoded reals in a finite ring provided q is large enough to avoid overflows. In addition, using the fact that one can sample uniformly at random from finite rings it is possible to leverage this representation to construct statistically secure multi-party computation protocols. The encoding and decoding maps that we introduce next are summarized in the following diagram:

$$\mathbb{R} \underset{\tilde{\phi}_\delta}{\overset{\phi_\delta}{\rightleftarrows}} \mathbb{Z} \underset{\tilde{\varphi}_q}{\overset{\varphi_q}{\rightleftarrows}} \mathbb{Z}_q \tag{4}$$

The effect of the precision when encoding real numbers using integers is controlled by the parameter $\delta = 2^{-b_f}$ which is used to define the mapping $\phi_\delta(r) = [r/\delta]$ from reals to integers, where $[\cdot]$ returns the rounding of a real number to the closest integer with ties favoring the largest integer. For example, ϕ_δ maps the interval $[-\delta/2, \delta/2)$ to 0, $[\delta/2, 3\delta/2)$ to 1, and $[-3\delta/2, -\delta/2)$ to -1. So in a sense δ is the smallest number that can be represented by our encoding ϕ_δ. A decoding of integers into real numbers acting as an approximate inverse of ϕ_δ up to precision δ is given by $\tilde{\phi}_\delta(z) = z\delta$. This encoding-decoding pair satisfy $|r - \tilde{\phi}_\delta(\phi_\delta(r))| \leq \delta$ for any real number $r \in \mathbb{R}$.

Now we turn to the question of how well operations on encoded real numbers can approximate the corresponding operations in real arithmetic. Addition is simple because the linearity of the decoding map implies that for any reals r, r' one has $|(r + r') - \tilde{\phi}_\delta(\phi_\delta(r) + \phi_\delta(r'))| \leq 2\delta$. Multiplications are slightly more involved because in general $2b_f$ fractional bits are needed to represent exactly the fractional part of the result of multiplying two fixed-point number. Taking this into account, one gets that the error introduced by performing the multiplication of two reals r, r' in fixed-point arithmetic is bounded by $|(rr') - \tilde{\phi}_{\delta^2}(\phi_\delta(r)\phi_\delta(r'))| \leq (|r| + |r'|)\delta + \delta^2$. This bound depends on the magnitude of the numbers being multiplied and the decoding must be done using precision δ^2.

When using ϕ_δ to encode reals in a bounded interval $[-M, M]$ with $M = 2^{b_i} - 2^{-b_f-1}$ we obtain integers in the finite range $[-M/\delta] \leq \phi_\delta(r) \leq [M/\delta]$. There are exactly $K = 2M/\delta + 1 = 2^b$ integers in this range, and therefore it is possible to map them injectively into the ring $\mathbb{Z}_q$ of integers modulo q with $q \geq K$ using the map $\varphi_q(z) = z \mod q$. For integers in the range $-q/2 \leq z \leq q/2$ this map is given by $\varphi_q(z) = z$ if $z \geq 0$ and $\varphi_q(z) = q + z$ for $z < 0$. Thus, it makes sense to define a decoding map $\tilde{\varphi}_q : \mathbb{Z}_q \to \mathbb{Z}$ with $\tilde{\varphi}_q(u) = u$ if $0 \leq u \leq q/2$ and $\tilde{\varphi}_q(u) = u - q$ for $q/2 < u \leq q - 1$. Then we have $\tilde{\varphi}_q(\varphi_q(z)) = z$ for any $-q/2 \leq z \leq q/2$.

Although φ_q is a ring homomorphism mapping operations in $\mathbb{Z}$ into operations in $\mathbb{Z}_q$, decoding from $\mathbb{Z}_q$ to $\mathbb{Z}$ after operating on encoded integers might not yield the desired result due to overflows. To avoid such overflows one must check that the result falls in the interval where the coding φ_q is the inverse of $\tilde{\varphi}_q$. In particular, if z, z' are integers such that $|z|, |z'| \leq q/2$, then

1. $|z + z'| \leq q/2$ implies $z + z' = \tilde{\varphi}_q(\varphi_q(z) + \varphi_q(z'))$, and
2. $|z \cdot z'| \leq q/2$ implies $z \cdot z' = \tilde{\varphi}_q(\varphi_q(z) \cdot \varphi_q(z'))$.

These properties will be used in the next section to analyze the error of an inner product protocol working with fixed-point encoding and provide guidelines to select the number of bits in the encoding. We note that the comparison here is against real arithmetic, while in the experimental section we will be comparing against floating-point arithmetic, which can represent real arithmetic to high degrees of accuracy across a much larger range of numbers [44].

4.2 Accuracy of Inner Product

We can now provide a bound on the accuracy of the result of the inner product protocol given in Figure 1. At the end of the protocol, both parties have an additive share of of $\langle \vec{x}, \vec{y} \rangle$ for some integer vectors $\vec{x}, \vec{y} \in \mathbb{Z}_q^n$. Therefore, we want to show that when the input vectors $\vec{x}$, $\vec{y}$ provided by the parties are fixed-point encodings of some real vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$, then the result of the inner product over $\mathbb{Z}_q$ can be decoded into a real number approximating $\langle \vec{u}, \vec{v} \rangle$.

According to the previous section, there are two sources of error when working with fixed-point encoded real numbers: the systematic error induced by working with finite precision (which can be mitigated by increasing b_f), and the occasional error that occurs when the numbers in $\mathbb{Z}_q$ overflow (which can be mitigated by increasing b_i). To control the overflow error we will assume that $\vec{u}$ and $\vec{v}$ are n-dimensional real vectors with entries bounded by R, $|u_i|, |v_i| \leq R$ for $i \in [n]$, and $\vec{x} = \varphi_q(\phi_\delta(\vec{u}))$ and $\vec{y} = \varphi_q(\phi_\delta(\vec{v}))$ are encodings of those vectors with precision δ.

Theorem 3. *If $q > 2nR^2/\delta^2$, then:*

$$|\langle \vec{u}, \vec{v} \rangle - \tilde{\phi}_{\delta^2}(\tilde{\varphi}_q(\langle \vec{x}, \vec{y} \rangle))| \leq 2nR\delta + n\delta^2 \ .$$

Proof. In the first place we observe that any number occurring in the computation of the inner product $\langle \phi_\delta(\vec{u}), \phi_\delta(\vec{v}) \rangle$ of the integer encodings of u and v is bounded by nR^2/δ^2. Therefore, the condition on q ensures there are no overflows in the computation in $\mathbb{Z}_q$ and we have $\langle \phi_\delta(u), \phi_\delta(v) \rangle = \tilde{\varphi}_q(\langle \vec{x}, \vec{y} \rangle)$. Now we use the formulas for the error of sum and product of integer encodings from previous section to show that

$$\begin{aligned}
&\left| \langle \vec{u}, \vec{v} \rangle - \tilde{\phi}_{\delta^2} \left(\langle \phi_\delta(\vec{u}), \phi_\delta(\vec{v}) \rangle \right) \right| \\
&\quad = \left| \sum_{i=1}^{n} u_i v_i - \tilde{\phi}_{\delta^2} \left(\sum_{i=1}^{n} \phi_\delta(u_i) \phi_\delta(v_i) \right) \right| \\
&\quad \leq \sum_{i=1}^{n} \left| u_i v_i - \tilde{\phi}_{\delta^2}(\phi_\delta(u_i)\phi_\delta(v_i)) \right| \\
&\quad \leq n(2R\delta + \delta^2) \ ,
\end{aligned}$$

where the first inequality follows by the triangle inequality and linearity of the decoding map $\tilde{\phi}_{\delta^2}$. $\square$

We note that the assumption of a bound R on the entries of the vectors $\vec{u}$ and $\vec{v}$ is not very stringent: if both parties agree on a common bound R, or there is a publicly known R, then the vectors can be normalized locally by each party before the execution of the protocol. Thus, in practice it is convenient to normalize the real vectors u and v such that their entries are bounded by $C/\sqrt{n}$ for some constant $C > 0$. In this case, the bounds above can be used to show that if one requires the final inner product to have error at most ε, this can be achieved by taking $\delta = \varepsilon/2C\sqrt{n}$ and $q = 8C^2n/\varepsilon^2$. Using these expressions we see that an encoding with $b = O(\log(n/\varepsilon))$ bits is enough to achieve accuracy ε when performing n-dimensional inner products with fixed-point encoded reals.

4.3 Data Standardization and Scaling

The statistical theory behind ridge regression generally assumes that the covariates in the columns of X are normally distributed with zero mean and unit variance. In practical applications this is not always satisfied. However, it is a common practice to standardize each column of the training data to have zero mean and unit variance. This column-wise procedure is implemented as part of the local pre-processing done by each party in our linear regression protocol.

In order to standardize the jth column $X_j \in \mathbb{R}^n$ of X, the party owning that column will compute its sample mean μ_{X_j} and its corrected sample standard deviation σ_{X_j}, where

$$\begin{aligned} \mu_{X_j} &= \frac{1}{n}\sum_{i=1}^{n} x_j^{(i)} \\ \sigma_{X_j} &= \sqrt{\sum_{i=1}^{n}(x_j^{(i)} - \mu_{X_j})^2/(n-1)}, \end{aligned}$$

and use it to compute the standardized column $\bar{X}_j$ with entries given by $\bar{X}_j(i) = (x_j^{(i)} - \mu_{X_j})/\sigma_{X_j}$. We use $\bar{X}$ to denote the data matrix obtained after standardization. The importance of proper data scaling to prevent overflows in the aggregation phase is the key message from Theorem 3 (Section 4.2).

The standardization and scaling steps described above imply that our protocols effectively solve the linear system

$$\left(\frac{1}{nd}\bar{X}^\top\bar{X} + \lambda I\right)\theta = \frac{1}{nd}\bar{X}^\top Y \ .$$

This scaling has no effect on the linear regression problem being solved beyond changing the scale of the regularization parameter. On the other hand, the standardization step needs to be reversed if one wants to make predictions on test data following the same distribution as the original training data. This task corresponds to a simple linear computation that can be efficiently implemented in MPC, both in the case where the model θ is disclosed to the parties and the case where it is kept shared among the parties and the prediction task using θ is itself implemented in a secure way using MPC.

In particular, to predict the output $\hat{y}$ corresponding to a new feature vector x one needs to use the following formula:

$$\hat{y} = \langle \tilde{\theta}, x \rangle + \Big(\langle \tilde{\theta}, \mu_X \rangle + \mu_Y \Big) \quad , \tag{5}$$

where $\mu_X = (\mu_{X_1}, \ldots, \mu_{X_d})$ is the vector of columns means for X, $\mu_Y = \sum_{i=1}^{n} y^{(i)}/n$ is the mean of Y, and $\tilde{\theta} = (\bar{\theta}_1/\sigma_{X_1}, \ldots, \bar{\theta}_d/\sigma_{X_d})$ is the parameter vector $\bar{\theta}$ where each coordinate is re-scaled with the corresponding standard deviation from X.

The post-processing of the parameter vector $\bar{\theta}$ and the computation of the offset $\langle \tilde{\theta}, \mu_X \rangle + \mu_Y$ can be implemented in two ways. One is to have each party contribute as input to the solving phase their parts of μ_X, σ_X, and μ_Y and consider the re-scaled parameter vector and the offset as the output of the solving phase. Note that this represents very little overhead in terms of computation and network usage. A second solution is to distribute at the end of the solving phase the components of the parameter vector $\bar{\theta}$ only to the party that contributed the corresponding columns of X; that is, $\bar{\theta}_j$ is given only to the party who contributed the jth column of X. In this case the parties can compute the coordinates of $\tilde{\theta}$ and an additive share of the offset locally. This is appealing in the setting where the new feature vector x for which a response needs to be predicted is also distributed among the different parties using the same scheme as the one used for the training data.

5 Solving Linear Systems

As discussed in Section 3.2, the solving phase of our protocol involves solving positive definite linear systems of the form $A\theta = b$ in a garbled circuits protocol. A wide variety of solutions to this problem can be found in the numerical analysis literature, though not all of them are suitable for MPC. For example, any variant of Gaussian elimination involving data-dependent pivoting can be immediately ruled out because implementing non-data-oblivious algorithms inside garbled circuits produces unnecessary blow-ups in the circuit size. This limitation has already been recognized by previous works on private linear regression using garbled circuits and other MPC techniques [37,60]. In particular, these works consider two main alternatives: computing the full inverse of A, or solving the linear system directly using the Cholesky decomposition of A as an intermediate step. Although both of these methods have an asymptotic computational cost of $O(d^3)$, in practice Cholesky is faster and numerically more stable. This justifies the choice of Cholesky as a procedure for solving linear systems in garbled circuits made in [60]. However, as we show in Section 6.2, Cholesky's cubic cost in the dimension can become prohibitive when working with high-dimensional data, in which case one must resort to iterative approximate algorithms like Conjugate Gradient Descent (CGD).

An exact method for solving positive definite linear systems can be obtained in terms of the Cholesky decomposition of the coefficient matrix. Recall that any positive definite matrix A admits a unique *Cholesky decomposition* of the form $A = LL^\top$, where L is a $d \times d$ lower triangular matrix with $L_{ij} = 0$ for all $i < j$ (see e.g. [33]). Given such a decomposition, the system $LL^\top \theta = b$ can be solved by first finding the solution of $L\theta' = b$ and then the solution of $L^\top \theta = \theta'$. By exploiting the lower triangular structure of L, the first system can be efficiently solved by forward substitution taking $\theta'_1 = b_1/L_{11}$ and $\theta'_i = (b_i - \sum_{j<i} L_{ij}\theta'_j)/L_{ii}$ for $i = 2, \ldots, d$. Similarly, the second system is solved by backward substitution taking $\theta_d = \theta'_d/L_{dd}$ and $\theta_i = (\theta'_i - \sum_{j>i} L_{ji}\theta_j)/L_{ii}$ for $i = d-1, \ldots, 1$. Note that these substitutions are a particular instance of Gaussian elimination where the initial coefficient matrices are already in almost echelon form, with the exception that the diagonal coefficients may be different from 1.

The Cholesky decomposition of a positive definite $d \times d$ matrix A can be computed in $O(d^3)$ operations, yielding an efficient algorithm for solving the system $A\theta = b$ since the forward and backward substitution steps also have cubic running time. Cholesky's algorithm enjoys two important properties that make it a very natural choice for solving private linear systems: it does not involve any pivoting strategy, thus yielding data-oblivious algorithms; and, it is numerically robust, making it suitable for implementations using fixed-point arithmetic. On the other hand, obtaining the Cholesky decomposition $A = LL^\top$ requires d square root computations, which are expensive to compute using MPC techniques. Nikolaenko et al. [60] use an iterative algorithm for computing square roots in garbled circuits based on a Newton-type iterative algorithm. Our implementation of Cholesky follows closely their approach and is based on the pseudo-code given in [60].

5.1 Conjugate Gradient Descent

Factorizing the coefficient matrix A in order to make the solution easier to compute, as in Cholesky's algorithm, is not the only efficient way to solve a system of linear equations. An entirely different approach relies on iterative algorithms which construct a monotonically improving sequence of approximate solutions θ_t converging to the desired solution. The main virtue of iterative algorithms is the possibility to reduce the cost of solving the system below the cubic complexity required by exact algorithms at the expense of providing only an approximate solution. In particular, for linear systems arising from ridge regression the practical importance of iterative algorithms is twofold: since intensive computations inside an MPC framework can be expensive, iterative methods provide a natural way to spend a fixed computational budget on finding an approximation to the desired solution by stopping after a fixed number of iterations; and, since the coefficients of the linear system occurring in ridge regression are inherently noisy because they are computed from a finite amount of data, finding an approximate

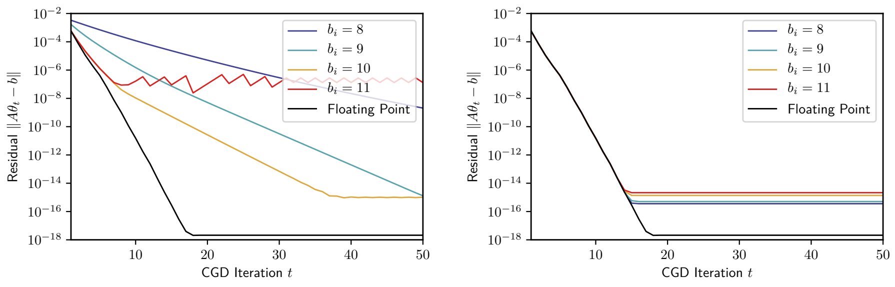


Fig. 4: Fixed-point CGD Implementations (with 64 bits on a system of dimension 50). (left) Textbook CGD [61]. (right) Our Fixed-point CGD.

solution whose accuracy is on the same order of magnitude as the noise in the coefficients is generally sufficient for optimizing the predictive performance of the model.

When the coefficient matrix of the linear system $A\theta = b$ is positive definite, a popular iterative method is the *conjugate gradient descent* (CGD) algorithm. The CGD algorithm can be interpreted as solving the system by iteratively minimizing the objective $\|A\theta - b\|^2$ with respect to the parameter vector θ using the method of conjugate gradients [61]. However, the numerical stability of CGD is in general worse than that of Cholesky's algorithm, making it very sensitive to the choice of fixed-point encoding. This can be observed in Figure 4 (left), where we plot the behavior of the residual $\|A\theta_t - b\|$ as a function of t for several settings of the number of number of integer bits b_i when solving a system with $d = 50$ using 64-bit fixed-point encodings. This plot shows that CGD is very sensitive to the choice of b_i, and even in the optimal setting ($b_i = 10$) the convergence rate is much slower than the one achieved by the corresponding floating-point implementation (dashed black curve).

After a thorough investigation of the behavior of a fixed-point implementation of CGD we concluded that the main problem occurs when the norm of the conjugate gradients decreases too fast and consequently the step sizes grow at each iteration. This motivates a variant of CGD, which we call *fixed-point CGD* (FP-CGD), and whose pseudo-code is given in Figure 5. The only difference between standard CGD and FP-CGD is the use of normalized gradients $\tilde{g}_t = g_t / \|g_t\|_\infty$ in the computation of the conjugate search directions p_t; in particular, by taking $\tilde{g}_t = g_t$ one recovers the classical textbook CGD. To the best of our knowledge, this modification of CGD has not been considered in the literature before.

It is easy to show that CGD and FP-CGD produce exactly the same sequence of approximate solutions θ_t when working in exact arithmetic. However, normalizing the search directions in FP-CGD makes the method much more stable with respect to the errors introduced by fixed-point arithmetic. This is illustrated by Figure 4 (right), where we show the evolution of the residuals for

Inputs: positive definite system (A, b), number of iterations T.
Output: approximate solution θ_T with $A\theta_T \approx b$.

Algorithm:

```
1. $\theta_0 \leftarrow 0$, $g_0 \leftarrow A\theta_0 - b$, $\tilde{g}_0 \leftarrow \frac{g_0}{\|g_0\|_\infty}$, $p_0 \leftarrow \tilde{g}_0$
2. For $t = 0 \dots T-1$ repeat:
   (a) $\theta_{t+1} \leftarrow \theta_t - \left(\frac{p_t^\top g_t}{p_t^\top A p_t}\right) p_t$.
   (b) $g_{t+1} \leftarrow g_t - \left(\frac{p_t^\top g_t}{p_t^\top A p_t}\right) A p_t$.
   (c) $\tilde{g}_{t+1} \leftarrow \frac{g_{t+1}}{\|g_{t+1}\|_\infty}$.
   (d) $p_{t+1} \leftarrow \tilde{g}_{t+1} - \left(\frac{p_t^\top \tilde{g}_{t+1}}{p_t^\top A p_t}\right) p_t$.
```

Fig. 5: Our variant of the Conjugate Gradient Descent algorithm, which is optimized for the use with fixed-point encoded numbers as described in Section 5.1.

the same system and fixed-point encodings that we used to test standard CGD. Here, we observe that we recover the same converge rate as CGD with floating-point arithmetic while only suffering a loss between 2 and 3 digits of accuracy. We will see later in our experiments with real data (Section 6.4) that this loss in accuracy is negligible in real applications, and it is comparable (or better) than the accuracy provided by a fixed-point implementation of Cholesky's method. In terms of computation, we note that each iteration of FP-CGD is slightly more expensive than an iteration of standard CGD due to the normalization step, but the asymptotic cost per iteration is in $\Theta(d^2)$ in both cases.

Further justificiation for the use of FP-CGD instead of other iterative methods is provided by its favorable theoretical properties. In exact arithmetic, FP-CGD inherits two important properties from CGD: (i) it converges to the *exact solution* of $A\theta = b$ in exactly d iterations; and (ii) it achieves the *optimal convergance rate* among all first-order methods, yielding a solution with error at most ε after $O(\sqrt{\kappa(A)}\log(1/\varepsilon))$ iterations. Property (i) is important because not all iterative algorithms for solving linear systems are guaranteed to produce an exact solution after a finite number of iterations, including standard gradient descent methods like the ones implemented in [35] using leveled homomorphic encryption. Furthermore, property (ii) says essentially that it is not possible to find a first-order method producing more accurate approximate solutions than CGD, and that only a few iterations of CGD are required to accurately solve linear systems with small condition number $\kappa(A)$. In principle, these properties might not be preserved by the fixed-point implementations of CGD and FP-CGD due to the numerical errors introduced by the finite-precision arithmetic. In the case of CGD, [54] shows that when working with a finite number of bits for the fractional part and an infinite number of bits for the integer part, several important properties of the exact version, including the convergence rate, are preserved up to small order modifications. However, the assumption of infinite bits for the integer part is not realistic in practice, and the experiments in Figure 4 show that when a fixed number of bits needs to be split between the integer and fractional part of the representation a crucial tension arises. FP-CGD is designed to alleviate such tension by making sure that a minimal number of bits in the integer part suffice to solve a properly scaled system accurately. We demonstrate this through our experimental evaluations in Section 6, where we also compare the accuracy and running time of FP-CGD against the method based in Cholesky decomposition used in [60], and study the effect of the condition number $\kappa(A)$ on both methods.

6 Experimental Results

This section presents an extensive empirical evaluation of our protocols. We start by describing the details about the concrete implementation of our system and the experimental setup. Then we present experiments for different implementations of the solving phase using FP-CGD and Cholesky, comparing them in terms of accuracy and running time across a wide range of settings. The second set of experiments evaluates two implementations for the aggregation phase: one based on the inner product protocol relying on a trusted initializer, and the one based on OT. Finally, we present an evaluation of the complete system on nine real datasets from the UCI repository with different algorithms in the solving phase and compare the resulting predictive performance of the learned model against the performance obtained in the non-private setting. Overall, our experiments highlight the importance of using FP-CGD for high-dimensional data, and provide a guide on how to choose a good fixed-point encoding depending on the characteristics of the problem.

6.1 Implementation and Experimental Setup

We implemented our MPC protocols using Obliv-C [73]. Obliv-C includes an implementation of Yao's garbled circuit protocol that incorporates recent optimizations including free XOR [45], garbled row reduction [57], fixed key block ciphers [6], and half gates [74]. Further, it features efficient implementations of OT extension, including variants such as correlated OT[2]. To support arbitrary precision arithmetic, we implemented a big integer library [1] for multi-party computation as an extension to Obliv-C. This library composes an arbitrary number of garbled machine-native integers to represent larger integer values in two's complement binary. All the arithmetic operations we require are implemented using common, efficient algorithms for extended precision arithmetic; in particular, it implements the Karatsuba-Comba [40] method for multiplication, and Knuth's algorithm D [44] for division. We extended this library to support fixed-point arithmetic via the same techniques used for native types. This arrangement incurs some overhead relative to using native types, even when the number of bits is equivalent. To compare the effects of different fixed-point encodings, we implemented two versions – with 32-bit and 64-bit numbers – of each of our protocols.

For the solving phase, we implemented the Cholesky and FP-CGD methods described in [60] and Section 5, respectively. Cholesky descomposition requires square root computation. Nikolaenko et al. [60] use an iterative algorithm for computing square roots in garbled circuits based on a Newton-type iterative algorithm. Our implementation of Cholesky's method follows closely their approach and is based on the pseudo-code given in [60].

For the experiments in Sections 6.2 and 6.3, where each phase of the protocol is evaluated separately, we use synthetically generated data. For each setting of d and n, we sample n data points $x^{(i)}$ from a standard d-dimensional Gaussian distribution and a d-dimensional vector of parameters θ^* with independent coordinates sampled uniformly in the interval $[0, 1]$. The training labels are obtained as $y^{(i)} = \langle \theta^*, x^{(i)} \rangle + \epsilon^{(i)}$, where $\epsilon^{(i)}$ is a noise term sampled from a Gaussian distribution with zero mean and variance $\sigma^2 = 0.1$. The regularization parameter in the solving phase is set to $\lambda = \sigma^2 d/n\|\theta^*\|^2$, which is the optimal choice suggested by the statistical theory of ridge regression.

For the experiments with synthetic data using 64 and 32 bit fixed-point encodings we used 60 and 28 bits for the fractional part, respectively. In the experiments with real data, the split between the fractional and integer part of the fixed-point encoding was optimized individually for each problem to obtain the best predictive performance.

The timing experiments were executed using Amazon EC2 C4 instances, each having 4 CPU cores, 7.5 GiB of RAM and 1 Gbps bandwidth. All our code, including the programs used for synthetic data generation, is available under open source licenses [67].

6.2 Solving Phase

Figure 6 (top) compares the running times of the solving phase using Cholesky and FP-CGD for dimensions ranging from 10 to 500. For FP-CGD, we give the running time for $5, 10, 15$, and 20 iterations. The top plot corresponds to the 64-bit implementation and the bottom plot to the 32-bit implementation. While Cholesky is faster than CGD for lower values of d, its cubic dependence on the dimension makes running a fixed number of iterations of FP-CGD faster as d increases. Thus, for high-dimensional data the iterative FP-CGD method is preferable in terms of computation time. The time spent running oblivious transfers is also shown, and accounts for a small fraction of the running time. For example, for the 64-bit implementations with $d = 200$, FP-CGD with 15 iterations runs in less than 45 minutes, while Cholesky takes more than an hour and a half. For $d = 500$, Cholesky takes more than 24 hours while FP-CGD with 15 iterations takes less than 4.5 hours. Similar results reporting circuit size instead of running time are presented in Appendix A (Figure 7).

Next, we compare both algorithms in terms of accuracy of the results. Here, the error is measured using the Euclidean distance to the optimal solution, ob-

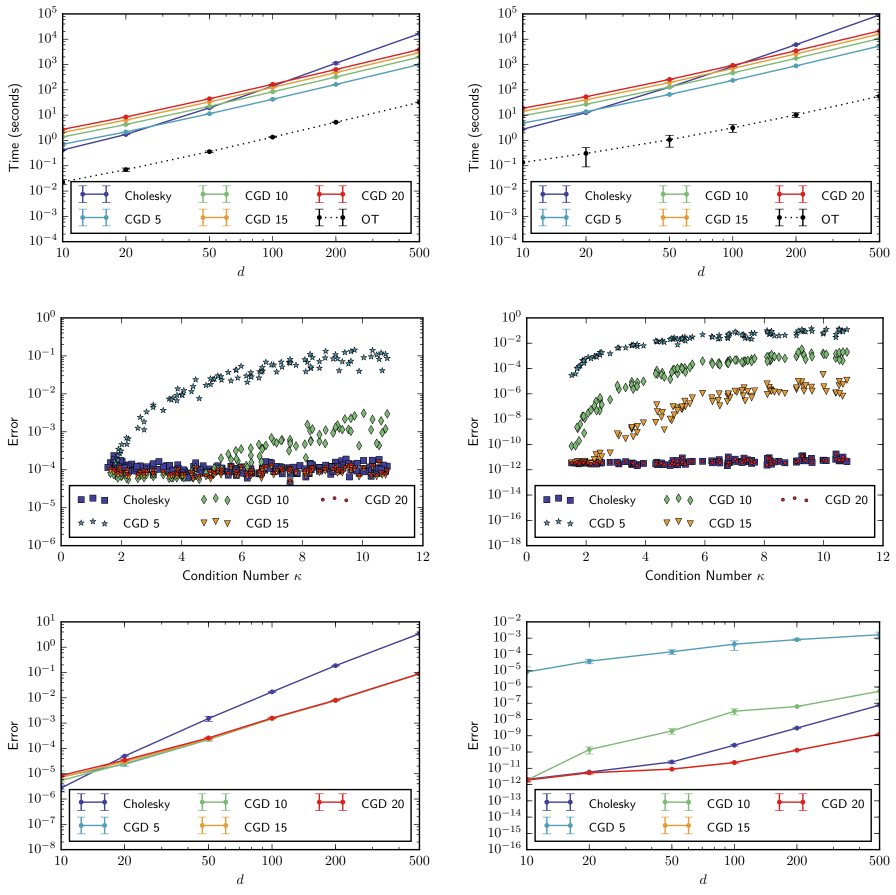


Fig. 6: (Top) Comparison between different methods for solving linear systems: Running time (seconds) of Cholesky and FP-CGD (with 5, 10, 15, and 20 iterations) as a function of input dimension. (Middle) Accuracy of Cholesky and CGD depending on the condition number of the input matrix A with $d = 20$. (Bottom) Accuracy of Cholesky and CGD, as a function of the input dimensionality d. (Left) Fixed-point numbers with $b = 32$ bits, with $b_f = 28$ in the fractional part. (Right) $b = 64, b_f = 60$.

tained via floating-point computation. Figure 6 (middle) shows how the accuracy of the result is affected by the condition number of the input matrix A, comparing Cholesky with FP-CGD for varying numbers of iterations on problems with $d = 20$. We observed that FP-CGD achieves the same accuracy as Cholesky in the 64-bit version, and has even lower error when using only 32 bits. Moreover, the influence of the input condition number decreases with the number of iterations. After convergence, the error remains the same for all tested condition numbers. This advantage over Cholesky increases with higher d, as is shown in Figure 6 (bottom). For both the 32-bit and 64-bit versions, FP-CGD is more accurate than Cholesky as soon as $d > 50$.

6.3 Aggregation Phase

Table 1 shows a comparison between the running times of the 64-bit versions of our two aggregation protocols: OT is the protocol using the OT-based inner product protocol, and *TI* is the protocol that leverages a Trusted Initializer for the inner product protocol. We vary the number of records, n, from 50,000 up to one million, and the number of features, d, from 20 to 200. As expected, the protocol that takes advantage of the Trusted Initializer performs significantly better in all cases. However, since the TI's resources are the bottleneck in this setting, the protocol does not scale well to multiple data providers, as the fraction of the coefficient matrix A that can be computed locally shrinks as the number of parties increases. The OT-based version, on the other hand, handles many parties very well, due to the fact that OTs between different pairs of parties can be performed in parallel. Timing results our protocols for the aggregation phase for a more extensive set of configurations can be found in Tables 5 and 6 of the appendix.

As a baseline, we implemented a garbled circuit protocol for performing a *single* inner product in Obliv-C. The running time for $n = 10^6$ was over 90 minutes. This implies that our protocol outperforms the naïve approach where the entire functionality is implemented in a garbled circuit by several orders of magnitude.

6.4 Experiments on Real Datasets

Although we have discussed the accuracy of our aggregation and solving protocols independently, we still have to evaluate our protocol in the task of building a ridge regresion model. We evaluate our secure multi-party ridge regression system on 9 different regression problems selected from the UCI repository [46]. Each problem comes with a set of n examples of some dimension d which are randomly split into training (70%) and test sets (30%). Details about the names, original references where the dataset appeared, dimensions and number of examples in each task are given in Table 2. The dimensions of the problems range from 7 to 384, and the number of training examples ranges from over 200 to almost 3 million.

n	d	Number of parties					
		2		3		5	
		OT	TI	OT	TI	OT	TI
$5 \cdot 10^4$	20	1m50s	1s	1m32s	2s	1m7s	2s
$5 \cdot 10^4$	100	42m12s	25s	34m39s	32s	24m58s	37s
$5 \cdot 10^5$	20	18m18s	15s	14m29s	18s	12m10s	21s
$5 \cdot 10^5$	100	7h3m56s	4m47s	5h20m52s	6m1s	4h17m8s	6m58s
$1 \cdot 10^6$	100	-	10m1s	-	12m42s	-	14m48s
$1 \cdot 10^6$	200	-	39m16s	-	49m56s	-	59m22s

Table 1: Comparison of running times between OT-based (left) and TI-based (right) aggregation protocols using 64 bit numbers. The running time of the Trusted Initializer, which is an offline preprocessing phase, is included. The complete results with additional parameter values can be found in the appendix.

Examples in the test set are used to evaluate the predictive accuracy of the learned models in terms of their root mean squared error (RMSE). The predictive accuracy of the model will depend on the particular regularization parameter λ used in the regression: larger values prevent overfitting and should be used on small datasets, while smaller values reduce the bias in the regression and should be used on large datasets. Hyper-parameter tuning algorithms can use a set of test examples to evaluate the predictive power obtained with different regularization parameters and find a near-optimal setting. In principle, any hyper-parameter tuning algorithm based on solving the regression multiple times with different regularization parameters could be run on top of our system at the cost of executing the protocol repeatedly. This would involve many repeated computations but it is also possible to design a variation of our protocol to avoid most of these repetitions, in particular the computation of $X^\top X$. We leave this optimization for future work and concentrate on evaluating our ridge regression protocol with a fixed regularization λ for each problem selected a priori using the given train and test split. Table 3 gives the values selected for each task in addition to the condition number of the resulting linear system.

In each case, the features were split evenly between two parties. The scalings and normalizations described in Section 4.3 are performed locally by each party during the aggregation phase. We use the version of our protocol with the aggregation phase implemented using the inner product protocol of Figure 1, and the reported bit width of 32 or 64 bits in each case. We compare four implementations for the solving phase: Cholesky and Fixed-Point CGD, both in their 32 and 64 bits versions. In the case of CGD, we fixed the number of iterations to 20, regardless of the dimension or any other feature of the problem. For each task, we select the number of bits b_f for the fractional part in order to make sure we have enough bits b_i in the integer part for representing the largest coefficient appearing in the linear system (A, b). The particular settings for each problem are given in Table 3.

The results are presented in Table 4. We give the total running time of the protocol and RMSE on the test set for each of the four different implementations of the solving phase. The RMSE of each algorithm is compared to the one obtained using an insecure ridge regression algorithm implemented in Matlab that uses 64-bit floating-point representations (given in the column headed *Optimal*). The percentages in parentheses next to each RMSE give the relative increase – or in very few cases, decrease – on the error incurred using the private algorithm. Remarkably, the differences in accuracy are almost negligible for both 64 bit implementations, while in the 32 bit implementations, CGD is better than Cholesky except in one task.

In terms of running time, we note that both bit settings observe the same pattern: Cholesky is faster for small dimensions $d \leq 100$, and CGD is faster for large dimensions $d > 100$. However, even in the cases where Cholesky is faster, CGD is not far behind (a bit over 3 minutes in the worst case). On the other hand, in the cases where CGD is faster Cholseky has prohibitive running times, e.g. CGD takes $\sim$ 4h on the largest problem ($d = 384$) while Cholesky takes practically half a day.

Overall, we observe that for except for the last task, Fixed-point CGD with 32 bits always obtains relative errors below 1% and runs in less than 45 minutes, thus offering a very competitive solution. For the last dataset, that has large number of training examples, 64 bits are required to achieve good accuracies, mainly due to the errors introduced in the aggregation phase when the number of examples is large and there are not enough bits to represent the intermediate values of the computation of $X^\top X$ with sufficient precision. A possible workaround to this problem would require having a different number of bits in the aggregation and solving phases; we will consider this optimization in future work.

id	Name	Reference	d	n
1	Student Performance	[13,16]	30	395
2	Auto MPG	[4]	7	398
3	Communities and Crime	[64,65]	122	1994
4	Wine Quality	[15,14]	11	4898
5	Bike Sharing Dataset	[25,26]	12	17 379
6	Blog Feedback	[11,10]	280	52 397
7	CT slices	[36]	384	53 500
8	Year Prediction MSD	[7]	90	515 345
9	Gas sensor array	[28,29]	16	4 208 261

Table 2: Specifications of UCI datasets considered in our evaluation. The number of samples n is split randomly into training (70%) and test sets (30%).

id	λ	$\kappa(A)$	$b_f(b=32)$	$b_f(b=64)$
1	$1.4 \cdot 10^{-2}$	$5.5 \cdot 10^{0}$	30	62
2	$2.2 \cdot 10^{-3}$	$9.0 \cdot 10^{1}$	28	61
3	$2.0 \cdot 10^{-4}$	$1.1 \cdot 10^{3}$	24	53
4	$2.9 \cdot 10^{-3}$	$6.8 \cdot 10^{1}$	29	60
5	$8.2 \cdot 10^{-7}$	$2.2 \cdot 10^{2}$	28	60
6	$1.1 \cdot 10^{-5}$	$1.3 \cdot 10^{4}$	25	54
7	$9.3 \cdot 10^{-6}$	$1.6 \cdot 10^{4}$	25	51
8	$1.1 \cdot 10^{-5}$	$1.7 \cdot 10^{2}$	26	58
9	$4.1 \cdot 10^{-7}$	$1.2 \cdot 10^{5}$	26	53

Table 3: Real data experimental setup. For each dataset, the regularization parameter λ and the number of fractional bits b_f were chosen as described in Section 6.1. The condition number κ was computed after data standardization and scaling (Section 4.3).

id	Optimal	FP-CGD (32 bits)		Cholesky (32 bits)		FP-CGD (64 bits)		Cholesky (64 bits)	
	RMSE	time	RMSE	time	RMSE	time	RMSE	time	RMSE
1	4.65	19s	4.65 (-0.0%)	5s	4.65 (-0.0%)	1m53s	4.65 (-0.0%)	35s	4.65 (-0.0%)
2	3.45	2s	3.45 (-0.0%)	0s	3.45 (-0.0%)	13s	3.45 (0.0%)	1s	3.45 (0.0%)
3	0.14	4m27s	0.14 (0.3%)	4m35s	0.14 (-0.0%)	24m24s	0.14 (0.2%)	26m31s	0.14 (-0.0%)
4	0.76	3s	0.76 (-0.0%)	0s	0.80 (4.2%)	23s	0.76 (-0.0%)	4s	0.76 (-0.0%)
5	145.06	4s	145.07 (0.0%)	1s	145.07 (0.0%)	26s	145.06 (0.0%)	4s	145.06 (0.0%)
6	31.89	24m5s	31.90 (0.0%)	53m24s	32.19 (0.9%)	2h3m39s	31.90 (0.0%)	4h40m23s	31.89 (-0.0%)
7	8.31	44m46s	8.34 (0.4%)	2h13m31s	8.87 (6.7%)	3h51m51s	8.32 (0.1%)	11h49m40s	8.35 (-0.0%)
8	9.56	4m16s	9.56 (0.0%)	3m50s	9.56 (0.0%)	16m43s	9.56 (0.0%)	13m28s	9.56 (0.0%)
9	90.33	48s	95.05 (5.2%)	42s	95.06 (5.2%)	1m41s	90.35 (0.0%)	1m9s	90.35 (0.0%)

Table 4: Results of the evaluation of our system on UCI datasets. For each choice of algorithm and bit width, running time is reported, and the root mean squared error (RMSE) of the solution obtained by our system and an insecure implementation of ridge regression using floating point are compared.

7 Malicious Security

The protocols that we discussed in Section 3 guarantee security in an adversarial setting where all parties, including the data providers, the CSP, and the Evaluator, are semi-honest, and the CSP and the Evaluator do not collude with each other. Next, we discuss modifications to our main protocol that strengthen its security guarantees in the setting where either the CSP or the Evaluator might behave maliciously.

A standard definition of security against malicious adversaries is given by Goldreich [31]. In this setting, a single adversary controls one or more parties, and all other parties are considered honest. If the CSP or the Evaluator are controlled by the adversary, they may deviate from the protocol in arbitrary ways. However, we maintain the restriction that they may not collude (that is, the adversary controls at most one of CSP and Evaluator). Moreover, as in the the setting used by Nikolaenko et al. [60], we assume a relaxed notion of malicious adversary that is willing to deviate from the protocol in any way, but only without getting caught. Moreover, if the protocol terminates successfully, the honest parties can be sure that their inputs were kept private. This is also known as the *covert* threat model, as formally defined by Aumann and Lindell [3].

First, note that for the aggregation phase, we need to use the inner product protocol based on OT (Section 3.1), since we can no longer assume that the CSP can act as a trusted initializer. Since neither the CSP nor Evaluator take part in this aggregation phase, it needs no further analysis.

Recall that the CSP and the Evaluator take the roles of garbler and evaluator in the garbled circuits protocol. In general, garbled circuit constructions provide security against a malicious evaluator, as garbled input and output values cannot be forged. The more challenging task is obtaining security guarantees against a malicious garbler, since it can prepare a garbled circuit (in step 1 of Figure 2) that implements a different functionality than the one intended for evaluation. Hence, in the rest of this section, we focus on the case where the Evaluator is honest, but the CSP may arbitrarily deviate from the protocol.

The general approach for constructing two-party computation based on garbled circuits that provides security against malicious adversaries employs the so-called *cut-and-choose* technique [51,47]. The basic idea is that the garbler generates multiple garbled circuits, and the evaluator chooses a subset of these circuits and checks that they were generated correctly by the garbler, while the remaining circuits are used for the secure evaluation. As the garbler does not know which circuits will be checked in advance, the evaluator gets some confidence that the garbler is not cheating in the circuit generation.

Nikolaenko et al. [60] made a perceptive observation that in the setting of linear regression, the correctness of the result can be verified simply by checking that the solution θ of the system $A\theta = b$ indeed minimizes the least squares expression of equation (1). This is done by verifying that θ evaluates to 0 in the derivative of the least squares function in equation (2), which is much cheaper than computing θ with malicious security. Concretely, this means checking that $\|2(A\theta - b)\| = 0$. As we are working on a finite ring, the equality check must be approximated as $\|2(A\theta - b)\| \in [-u, u]$, for some u chosen by the parties. We assume that the election of u is done correctly, in the sense that, if the CSP does not deviate from the protocol, then $\|2(A\theta - b)\| \in [-u, u]$ holds. If we consider the infinity norm, then the verification check of a solution θ, corresponds to checking

$$\forall i \in [d] : \vec{v}_i \in [-u, u] \tag{6}$$

where $\vec{v} = A\theta - b$, and A, b are additively shared among the data providers as $(A_1, b_1), \dots, (A_k, b_k)$ as a result of the aggregation phase. We call this check the *verification phase*, which is run after the solving phase to withstand a malicious CSP. Hence, we use a semi-honest protocol for the solving phase, and then run a verification protocol with malicious security.

An important observation is that, by proceeding in this way, we give up on checking the exact functionality implemented in the semi-honest circuit generated by a malicious CSP, which has subtle implications for security that were not explicitly addressed in the work of [60]. We also discuss those and investigate possible solutions at the end of this section.

In our protocol for the semi-honest case of Figure 3, θ is revealed to the data providers as a result of the solving phase. Consequently, an additive share of $\vec{v}$ in (6) can be precomputed locally by the parties, and hence securely evaluating (6) only requires additions and comparisons. Using this, we implemented the verification phase as follows.

1. The data providers generate a uniformly pseudorandom vector $\vec{v}_{i,1} \in \mathbb{Z}_q^d$ and locally compute $\vec{v}_{i,2} = A_i\theta - b_i - \vec{v}_{i,1}$. Then, they send $\vec{v}_{i,1}$ to the CSP and $\vec{v}_{i,2}$ to the Evaluator.
2. CSP and Evaluator add up their received shares and obtain $\vec{v}_1 = \sum_{i=1}^{k} \vec{v}_{i,1}$ and $\vec{v}_2 = \sum_{i=1}^{k} \vec{v}_{i,2}$, and then run a two-party garbled circuit protocol with malicious security. In the circuit, $\vec{v} = \vec{v}_1 + \vec{v}_2$ is recovered, and a single bit is returned, indicating whether (6) holds.
3. CSP and Evaluator send the result of the verification circuit to all data providers.

Note that since not both CSP and Evaluator cannot be malicious at the same time, the parties only need to check if both bits they received are 1. We implemented and evaluated the garbled circuit with malicious security for this verification phase, as an extension for our solving phase, using the EMP framework presented in [69]. As expected, it runs extremely fast: in our experiments, garbling and execution took less than 3 seconds for $d \leq 500$. This is in contrast with the time needed for the solving phase for $d = 500$: 30 minutes, for 10 iterations of CGD and a bit width of 32 bits (see Figure 6 (left)).

The above approach guarantees that a malicious CSP that generates a circuit for the solving phase that computes a solution that does not pass the check of the verification phase will be detected. This provides security against covert CSP when the computation of the first phase is evaluating an exact solution that is unique and this is the only value that will pass the verification. However, if the functionality is interpreted as an approximation of the linear regression functionality, there is still a difficulty that must be overcome to formally prove security: since u is known to the CSP (or at least a lower bound can be guessed reasonably well), he can change the circuit in the solving phase to encode some information about some data provider's input in the least significant bits of the elements of θ. If the changes to v are small enough, this would not be detected in the verification phase.

This is a common problem when securely computing functionalities that implement approximations, and is discussed in detail in the work of Feigenbaum et al. [27]. The authors make the observation that the outputs of the evaluation of an approximation may reveal more than the outputs of the evaluation of the exact functionality, and that simply rounding does not provide security in general. They define the notion of a functionally private approximation, which guarantees that the outputs of such an approximation are indistinguishable from those of a randomly sampled approximation, and propose a way to make an approximation functionally private by adding uniform noise to the output.

We can use the techniques proposed by Feigenbaum et al. [27] in an alternative verification phase in which a functionally private approximation of θ is provided to the parties only if the verification test passes. We leave this extension and its evaluation in terms of running time of the evaluation phase, and utility of the constructed model for further work.

8 Discussion

The problem of securely running machine learning algorithms when the training data is distributed among several parties is an important milestone for the development of privacy-preserving data analysis tools. In this paper, we focus on a linear regression task widely used in practical applications. Beyond the settings described in this paper, our implementation of secure conjugate gradient descent with fixed-point arithmetic and early stopping can also be used to deal with non-linear regression problems based on kernel ridge regression, given MPC protocols for evaluating kernel functions typically used in machine learning. Moreover, as mentioned in Section 3, an extensive evaluation of MPC techniques for the task of linear system solving, including our conjugate gradient descent algorithm, is an interesting continuation of the work proposed here. From a more theoretical perspective, the problem of providing security guarantees against malicious adversaries for approximate MPC functionalities poses interesting open challenges both in general and from the perspective of concrete machine learning tasks.

References

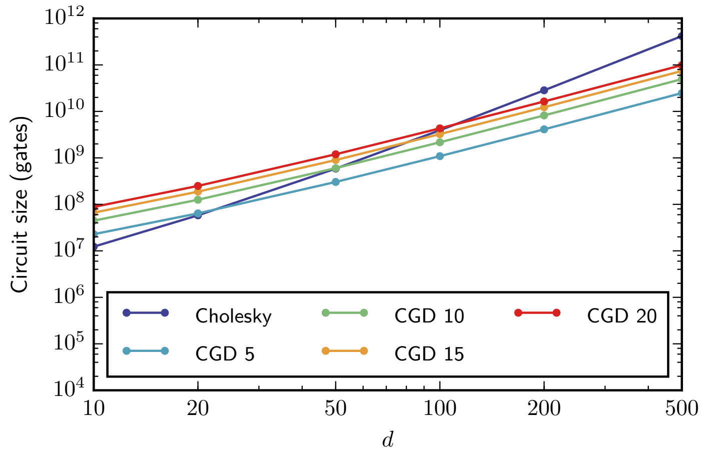
1. Absentminded crypto kit. `https://bitbucket.org/jackdoerner/absentminded-crypto-kit`.
2. G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Conference on Computer and Communications Security*, pages 535–548. ACM, 2013.
3. Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.
4. Auto MPG data set. `https://archive.ics.uci.edu/ml/datasets/Auto+MPG`, 1993.
5. D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.
6. M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society, 2013.
7. T. Bertin-Mahieux. YearPredictionMSD data set. `https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD`, 2011.
8. M. Blanton, A. Steele, and M. Aliasgari. Data-oblivious graph algorithms for secure computation and outsourcing. In *ASIACCS*, pages 207–218. ACM, 2013.
9. D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, pages 192–206. Springer, 2008.
10. K. Buza. Feedback prediction for blogs. In *GfKl*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 145–152. Springer, 2012.
11. K. Buza. BlogFeedback data set. `https://archive.ics.uci.edu/ml/datasets/BlogFeedback`, 2014.
12. M. D. Cock, R. Dowsley, A. C. A. Nascimento, and S. C. Newman. Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In *AISec@CCS*, pages 3–14. ACM, 2015.
13. P. Cortez. Student performance data set. `https://archive.ics.uci.edu/ml/datasets/Student+Performance`, 2014.
14. P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.
15. P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Wine quality data set. `https://archive.ics.uci.edu/ml/datasets/Wine+Quality`, 2009.
16. P. Cortez and A. M. G. Silva. Using data mining to predict secondary school student performance. In *Future Business Technology Conference*, pages 5–12. EUROSIS, 2008.
17. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662. Springer, 2012.
18. D. Demmler, G. Dessouky, F. Koushanfar, A. Sadeghi, T. Schneider, and S. Zeitouni. Automated synthesis of optimized circuits for secure computation. In *ACM Conference on Computer and Communications Security*, pages 1504–1517. ACM, 2015.
19. D. Demmler, T. Schneider, and M. Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS*. The Internet Society, 2015.
20. W. Du and M. J. Atallah. Privacy-preserving cooperative scientific computations. In *CSFW*, pages 273–294. IEEE Computer Society, 2001.

21. W. Du and M. J. Atallah. Protocols for secure remote database access with approximate matching. In *E-Commerce Security and Privacy*, volume 2 of *Advances in Information Security*, pages 87–111. Springer, 2001.
22. W. Du, Y. S. Han, and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *SDM*, pages 222–233. SIAM, 2004.
23. C. Dwork and K. Nissim. Privacy-preserving datamining on vertically partitioned databases. In *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 528–544. Springer, 2004.
24. C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.
25. H. Fanaee-T and J. Gama. Bike sharing dataset data set. `https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset`, 2013.
26. H. Fanaee-T and J. Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in AI*, 2(2-3):113–127, 2014.
27. J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright. Secure multiparty computation of approximations. *ACM Trans. Algorithms*, 2(3):435–472, 2006.
28. J. Fonollosa and R. Huerta. Gas sensor array under dynamic gas mixtures data set. `https://archive.ics.uci.edu/ml/datasets/Gas+sensor+array+under+dynamic+gas+mixtures`, 2015.
29. J. Fonollosa, S. Sheik, R. Huerta, and S. Marco. Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. *Sensors and Actuators B: Chemical*, 215:618–629, 2015.
30. N. Gilboa. Two party RSA key generation. In *CRYPTO*, pages 116–129. Springer, 1999.
31. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
32. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.
33. G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.
34. S. D. Gordon, J. Katz, V. Kolesnikov, F. Krell, T. Malkin, M. Raykova, and Y. Vahlis. Secure two-party computation in sublinear (amortized) time. In *ACM Conference on Computer and Communications Security*, pages 513–524. ACM, 2012.
35. T. Graepel, K. E. Lauter, and M. Naehrig. ML confidential: Machine learning on encrypted data. In *ICISC*, pages 1–21. Springer, 2012.
36. F. Graf, H.-P. Kriegel, M. Schubert, S. Poelsterl, and A. Cavallaro. Relative location of ct slices on axial axis data set. `https://archive.ics.uci.edu/ml/datasets/Relative+location+of+CT+slices+on+axial+axis`, 2011.
37. R. Hall, S. E. Fienberg, and Y. Nardi. Secure multiple linear regression based on homomorphic encryption. *Journal of Official Statistics*, 27(4):669, 2011.
38. W. Henecka, S. Kögl, A. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: tool for automating secure two-party computations. In *ACM Conference on Computer and Communications Security*, pages 451–462. ACM, 2010.
39. Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*. USENIX Association, 2011.

40. A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. In *Proceedings of the USSR Academy of Sciences 145*, pages 293–294, 1962.
41. A. F. Karr, X. Lin, A. P. Sanil, and J. P. Reiter. Regression on distributed databases via secure multi-party computation. In *DG.O*, ACM International Conference Proceeding Series. Digital Government Research Center, 2004.
42. M. Keller, E. Orsini, and P. Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *ACM Conference on Computer and Communications Security*, pages 830–842. ACM, 2016.
43. M. Keller and P. Scholl. Efficient, oblivious data structures for MPC. In *ASIACRYPT (2)*, pages 506–525. Springer, 2014.
44. D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms.* Addison-Wesley, 1997.
45. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP (2)*, pages 486–498. Springer, 2008.
46. M. Lichman. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2013.
47. Y. Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *J. Cryptology*, 29(2):456–490, 2016.
48. Y. Lindell. How to simulate it - A tutorial on the simulation proof technique. *IACR Cryptology ePrint Archive*, 2016:46, 2016.
49. Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.
50. Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.
51. Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *J. Cryptology*, 25(4):680–722, 2012.
52. Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *CRYPTO (2)*, pages 319–338. Springer, 2015.
53. C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi. ObliVM: A programming framework for secure computation. In *IEEE Symposium on Security and Privacy*, pages 359–376. IEEE Computer Society, 2015.
54. G. Meurant. *The Lanczos and conjugate gradient algorithms: from theory to finite precision computations*, volume 19. SIAM, 2006.
55. K. P. Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012.
56. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457. ACM/SIAM, 2001.
57. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *EC*, pages 129–139, 1999.
58. K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi. Graphsc: Parallel secure computation made easy. In *IEEE Symposium on Security and Privacy*, pages 377–394. IEEE Computer Society, 2015.
59. J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2012.
60. V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE Symposium on Security and Privacy*, pages 334–348. IEEE Computer Society, 2013.
61. J. Nocedal and S. Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

62. P. Pullonen and S. Siim. Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations. In *Financial Cryptography Workshops*, pages 172–183. Springer, 2015.
63. M. Rabin. How to Exchange Secrets by Oblivious Transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.
64. M. Redmond. Communities and crime data set. `https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime`, 2009.
65. M. Redmond and A. Baveja. A data-driven software tool for enabling cooperative information sharing among police departments. *European Journal of Operational Research*, 141(3):660–678, 2002.
66. A. P. Sanil, A. F. Karr, X. Lin, and J. P. Reiter. Privacy preserving regression modelling via distributed computation. In *KDD*, pages 677–682. ACM, 2004.
67. Secure distributed linear regression. `https://github.com/schoppmp/linreg-mpc/`.
68. E. M. Songhori, S. U. Hussain, A. Sadeghi, T. Schneider, and F. Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *IEEE Symposium on Security and Privacy*, pages 411–428. IEEE Computer Society, 2015.
69. X. S. Wang, A. J. Malozemoff, and J. Katz. Faster two-party computation secure against malicious adversaries in the single-execution setting. *IACR Cryptology ePrint Archive*, 2016:762, 2016.
70. M. H. Weik. A third survey of domestic electronic digital computing systems. Technical report, DTIC Document, 1961.
71. A. C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.
72. H. Yu, J. Vaidya, and X. Jiang. Privacy-preserving SVM classification on vertically partitioned data. In *PAKDD*, pages 647–656. Springer, 2006.
73. S. Zahur and D. Evans. Obliv-C: A language for extensible data-oblivious computation. *IACR Cryptology ePrint Archive*, 2015:1153, 2015.
74. S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT (2)*, pages 220–250. Springer, 2015.

A Further Experimental Results



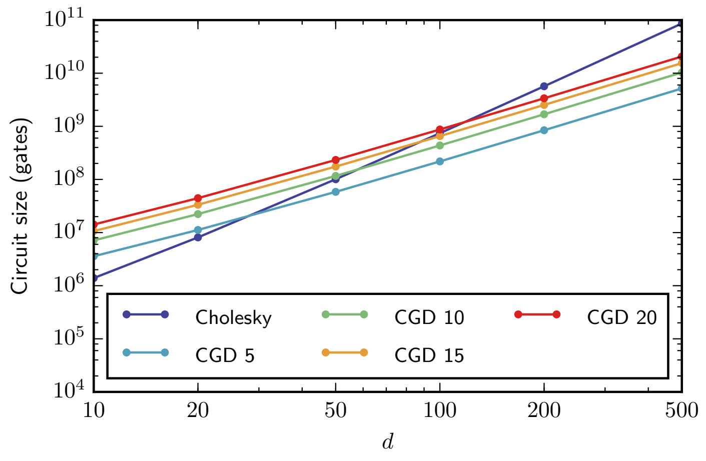


Fig. 7: Circuit sizes for 64 bits (top), 32 bits (bottom) as a function of the input dimensionality d. One can see a clear correlation between the circuit size and the execution time shown in Figure 6 (left).

n	d	Number of parties (data providers)					
		2		3		5	
		$b = 64$	$b = 32$	$b = 64$	$b = 32$	$b = 64$	$b = 32$
10000	20	24s	13s	20s	11s	14s	8s
	50	2m20s	1m16s	1m55s	1m2s	1m22s	43s
	100	9m6s	4m58s	7m18s	3m55s	5m13s	2m47s
20000	20	45s	24s	38s	20s	29s	15s
	50	4m26s	2m18s	3m39s	1m56s	2m39s	1m21s
	100	17m25s	9m58s	14m9s	7m24s	10m10s	5m14s
50000	20	1m50s	56s	1m32s	47s	1m7s	35s
	50	10m47s	5m27s	8m58s	4m37s	6m31s	3m12s
	100	42m12s	21m25s	34m39s	17m41s	24m58s	12m32s
100000	20	3m40s	1m50s	3m1s	1m33s	2m18s	1m7s
	50	21m40s	10m47s	17m25s	9m6s	13m5s	6m40s
	100	1h25m14s	42m18s	1h7m30s	35m3s	50m16s	24m12s
200000	20	7m25s	3m40s	5m57s	3m4s	4m46s	2m18s
	50	43m47s	21m43s	34m14s	18m26s	26m47s	12m40s
	100	2h52m8s	1h25m6s	2h11m35s	1h10m39s	1h43m12s	49m23s
500000	20	18m18s	9m10s	14m29s	7m27s	12m10s	5m52s
	50	1h47m59s	54m13s	1h23m49s	45m0s	1h8m8s	32m14s
	100	7h3m56s	3h32m37s	5h20m52s	2h53m54s	4h17m8s	2h4m53s

Table 5: Computation time of the aggregation phase using using the OT-based inner product protocol, for 2, 3, and 5 data providers and different values of n (number of records) and d (number of features). Note that unlike the TI-based protocol in Table 6, this algorithm scales well with the number of parties.

		Number of parties (data providers)								
n	*d*	2			3			5		
		CSP	DP	Total	CSP	DP	Total	CSP	DP	Total
100000	20	1s	2s	3s	2s	2s	3s	2s	1s	4s
	50	7s	9s	14s	9s	8s	17s	11s	6s	20s
	100	27s	31s	50s	36s	28s	1m4s	43s	21s	1m15s
	200	1m46s	1m57s	3m12s	2m22s	1m44s	4m10s	2m50s	1m17s	4m54s
	500	10m52s	11m53s	19m14s	14m40s	10m29s	25m16s	17m29s	7m37s	30m2s
200000	20	2s	4s	6s	3s	4s	7s	4s	3s	8s
	50	14s	19s	30s	18s	17s	37s	22s	13s	42s
	100	55s	1m6s	1m47s	1m11s	59s	2m16s	1m26s	44s	2m39s
	200	3m37s	4m13s	7m0s	4m44s	3m43s	9m0s	5m38s	2m43s	10m32s
	500	21m46s	25m33s	41m47s	29m11s	22m23s	54m21s	34m56s	16m6s	1h4m0s
500000	20	6s	10s	15s	8s	9s	18s	10s	8s	21s
	50	36s	48s	1m17s	47s	42s	1m35s	57s	32s	1m51s
	100	2m17s	2m51s	4m47s	3m4s	2m29s	6m1s	3m42s	1m51s	6m58s
	200	9m0s	10m55s	18m52s	12m15s	9m33s	24m8s	14m47s	6m52s	27m43s
	500	56m22s	1h6m41s	1h53m52s	1h14m37s	57m31s	2h25m55s	1h30m41s	40m56s	2h50m53s
1000000	20	12s	20s	31s	16s	18s	37s	20s	15s	44s
	50	1m12s	1m34s	2m40s	1m37s	1m24s	3m21s	1m58s	1m4s	3m53s
	100	4m45s	5m40s	10m1s	6m16s	4m58s	12m42s	7m31s	3m41s	14m48s
	200	18m49s	21m44s	39m16s	25m7s	18m48s	49m56s	30m28s	13m48s	59m22s

Table 6: Computation time of the aggregation phase using the TI-based inner product protocol with 64 bits, for 2, 3, and 5 data providers and different values of n (number of records) and d (number of features). For each number of data providers, computation time of the CSP (left) and the data providers (middle, averaged) is reported, as well as total running time (right).