Scalable and Secure Logistic Regression via Homomorphic Encryption

Yoshinori Aono Takuya Hayashi Le Trieu Phong[⋆] Lihua Wang

National Institute of Information and Communications Technology (NICT), Japan
{aono, takuya.hayashi, phong, wlh}@nict.go.jp

Abstract. Logistic regression is a powerful machine learning tool to classify data. When dealing with sensitive data such as private or medical information, cares are necessary. In this paper, we propose a secure system for protecting both the training and predicting data in logistic regression via homomorphic encryption. Perhaps surprisingly, despite the non-polynomial tasks of training and predicting in logistic regression, we show that only additively homomorphic encryption is needed to build our system. Indeed, we instantiate our system with Paillier, LWE-based, and ring-LWE-based encryption schemes, highlighting the merits and demerits of each instance. Our system is very scalable in both the dataset size and dimension, tolerating big size for example of hundreds of millions (10^8s) records. Besides examining the costs of computation and communication, we carefully test our system over real datasets to demonstrate its accuracies and other related measures such as F-score and AUC.

Keywords: Logistic regression, homomorphic encryption, Paillier, LWE, ring-LWE, outsourced computation, accuracy, F-score, area under curve (AUC).

1 Introduction

1.1 Background

Logistic regression is a standard method in supervised machine learning to classify data. It is widely applied in various fields of science and engineering. Specifically, it has been successfully used in medical research to help deciding, for example, a patient has a disease or not.

In several tasks using logistic regression, data contributors (and the data analyst as well) are geographically distributed, raising the need of a central server to *receive*, *store*, *share*, and *process* the data as shown in Figure 1. However, the large data collection at the central server may easily become a significant target for attacks. Even if the access to the server is properly controlled, the server itself cannot be completely trusted due to the sensitive nature of the data.

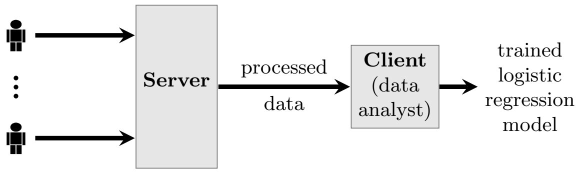

Fig. 1. Data flows.

This work is a continuation of the efforts, detailed in Section 1.3 of related works, of protecting the data collection at the server via homomorphic encryption.

[⋆] Corresponding author

1.2 Our contributions

We build a system for secure logistic regression. Both training data and predicting data in our system are protected under encryption, so data secrecy is ensured. In addition, the output of our system can achieve differential privacy. Specifically, our secure system is constructed via following technical steps and contributions:

(1) We turn the original logistic regression into what we call homomorphism-aware logistic regression via function approximations.
(2) We show how to use additively homomorphic encryption with the homomorphism-aware logistic regression. We also point out how to add differential privacy into the system.
(3) We show how to instantiate our system with Paillier, LWE-based, and ring-LWE-based encryption, and highlight their merits and demerits.
(4) We conduct experiments with real datasets from the UCI repository to compare accuracies, F-scores, and AUCs of both original logistic regression (over unencrypted data) and our system (encrypted data, optionally with differential privacy).

The effect of our design is that the system is both *secure* and *scalable*, being able to handle very large dataset size up to 10^8s. For example, with a synthetic dataset of 10^8 items each of 40 features, our commodity server finishes in 20 minutes and its communication to the client transmits from 20 Kbytes to 1 Mbyte depending on the used encryption scheme.

1.3 Related works

The works [4, 17] consider private prediction in logistic regression, given that the regression coefficients (i.e. θ^* in Section 2.1) are publicly known. This work complements [4] by protecting the training data in producing the regression coefficients.

The work [10] considers polynomial learning algorithms on encrypted data, assuming that multiplications over ciphertexts are supported by the underlying homomorphic encryption scheme. In addition, [10] cites [17] for logistic regression and examines other classifiers such as Linear Means and FLD. This work continues the efforts in [10] by showing that, via approximations and data arrangements, the training phase of logistic regression can be done without any multiplications over ciphertexts.

Approximate versions of logistic regression have appeared in other contexts such as large-scale text categorization [24] and differential privacy [25]. This work complements [25] by showing the secrecy of training data can be gained via encryption. In addition, this work combines well with the functional mechanism for differential privacy in [25] to obtain both data secrecy and output privacy. See Section 4.2.

Bost et al. [5] examines several classifiers in the setting of two-party computation, in which the server has a secret θ (model) and the client has a secret x (data). Both interacts in a protocol with many rounds so that at the end the client learns $\mathsf{Circuit}(\theta, x)$ where $\mathsf{Circuit}$ is one of the classifiers.

The papers [6, 7, 18] are also in the setting of secure two-party computation in which each party has a secret training set. Both wish to combine those sets in the training phase to obtain the trained model, but do not want to reveal their private dataset to each other.

Zhu et al. [26] also consider outsourced logistic regression via an interactive protocol in which their customer must be always active in communication in several rounds with the cloud server.

The communication cost in their protocol depends on both dataset size and dimension, and likewise the customer's computation cost. Differential privacy is not considered in [26].

Logistic regression with data privacy protection receives significant attentions from researchers in biomedical informatics [11, 22, 23], considering different settings of integrating and sharing data. Our work adds the setting of secure outsourced logistic regression to this line of research, showing the task can be done securely at scale.

Khedr et al. [12] examine classifiers using Bayesian filter and decision trees on encrypted data using homomorphic encryption supporting a few multiplications over ciphertexts.

2 Logistic regression and its approximation

2.1 Original logistic regression

Consider a (N_{data}, d)-dataset of N_{data} records and d dimension

$$\left\{x^{(i)}, y^{(i)}\right\}_{1 \le i \le N_{\text{data}}}$$

in which $x^{(i)} = \left(1, x_1^{(i)}, \ldots, x_d^{(i)}\right) \in \mathbb{R}^{d+1}$, $y^{(i)} \in \{0, 1\}$. Define the following cost function $J : \mathbb{R}^{d+1} \to \mathbb{R}$, with variable $\theta = (\theta_0, \theta_1, \ldots, \theta_d) \in \mathbb{R}^{d+1}$,

$$\begin{aligned} J(\theta) &= \\ &\frac{\lambda}{2N_{\text{data}}} \sum_{j=1}^{d} \theta_j^2 + \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \Big[-y^{(i)} \log(h_\theta(x^{(i)})) - \\ &\qquad -(1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \Big] \end{aligned} \tag{1}$$

in which $0 < h_\theta(x) < 1$ is the value of the sigmoid function $h_\theta : \mathbb{R}^{d+1} \to \mathbb{R}$ given by

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)} = \frac{1}{1 + \exp(-\sum_{j=0}^{d} \theta_j x_j)}.$$

The training phase of logistic regression aims at finding the minimizer which optimizes the cost function

$$\theta^* = \text{argmin}_\theta J(\theta).$$

The predicting phase of logistic regression, given a new data $x^{\text{new}} = (1, x_1^{\text{new}}, \ldots, x_d^{\text{new}}) \in \mathbb{R}^{d+1}$, aims at guessing the binary value $y^{\text{new}} \in \{0, 1\}$ by setting

$$y^{\text{new}} = \begin{cases} 1 \text{ if } h_{\theta^*}(x^{\text{new}}) \ge \mathsf{thres} \\ 0 \text{ if } h_{\theta^*}(x^{\text{new}}) < \mathsf{thres} \end{cases} \tag{2}$$

in which $0 < \mathsf{thres} < 1$ is a variable threshold, and typically $\mathsf{thres} = 1/2$.

2.2 Homomorphism-aware logistic regression via approximation

The cost function $J(\theta)$ in Section 2.1 includes logarithm functions, causing obstacles when all the data are in encrypted form. In this section, via approximation of the logarithm functions, we derive what we call homomorphism-aware logistic regression.

Note that,

$$\log(h_\theta(x)) = \log\left(\frac{1}{1+\exp(-\theta^T x)}\right)$$
$$\log(1-h_\theta(x)) = \log\left(\frac{1}{1+\exp(\theta^T x)}\right)$$

so if we approximate the following function of $u \in \mathbb{R}$ by a degree k (e.g., $k = 2$) polynomial

$$\log\left(\frac{1}{1+\exp(u)}\right) \approx \sum_{j=0}^{k} a_j u^j \tag{3}$$

we obtain following approximations

$$\log(1-h_\theta(x)) \approx \sum_{j=0}^{k} a_j (\theta^T x)^j \tag{4}$$
$$\log(h_\theta(x)) \approx \sum_{j=0}^{k} (-1)^j a_j (\theta^T x)^j \tag{5}$$

The approximate function of $J(\theta)$ is formed by using the polynomials at (4) and (5) to replace the corresponding logarithm functions in (1). Namely, define the following cost function

$$J_{\text{approx}}(\theta) = \frac{\lambda}{2N_{\text{data}}}\sum_{j=1}^{d} \theta_j^2 + J^*_{\text{approx}}(\theta) \tag{6}$$

in which

$$\begin{aligned}
&J^*_{\text{approx}}(\theta)\\
&= \frac{1}{N_{\text{data}}}\sum_{i=1}^{N_{\text{data}}}\Big[-y^{(i)}\sum_{j=0}^{k}(-1)^j a_j(\theta^T x^{(i)})^j\\
&\qquad\qquad -(1-y^{(i)})\sum_{j=0}^{k} a_j(\theta^T x^{(i)})^j\Big]\\
&= \frac{1}{N_{\text{data}}}\sum_{i=1}^{N_{\text{data}}}\sum_{j=0}^{k}\Big(y^{(i)} - y^{(i)}(-1)^j - 1\Big)\, a_j(\theta^T x^{(i)})^j\\
&= \frac{1}{N_{\text{data}}}\sum_{i=1}^{N_{\text{data}}}\sum_{j=1}^{k}\Big(y^{(i)} - y^{(i)}(-1)^j - 1\Big)\, a_j(\theta^T x^{(i)})^j - a_0.
\end{aligned}$$

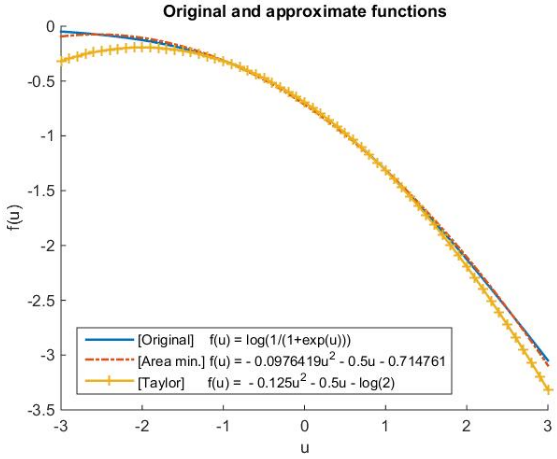

Fig. 2. Approximate functions.

For our secure system, we further want to expand $(\theta^T x^{(i)})^j$ for all i. For that purpose, we have with $x = (x_0, x_1, \ldots, x_d) \in \mathbb{R}^{d+1}$,

$$
\begin{aligned}
(\theta^T x)^j = \left(\sum_{r=0}^{d} \theta_r x_r \right)^j &= \sum_{0 \le r_1, \ldots, r_j \le d} (\theta_{r_1} x_{r_1}) \cdots (\theta_{r_j} x_{r_j}) \\
&= \sum_{r_1, \ldots, r_j = 0}^{d} (\theta_{r_1} \cdots \theta_{r_j})(x_{r_1} \cdots x_{r_j})
\end{aligned}
$$

and denote

$$
A_{j, r_1, \ldots, r_j} = \sum_{i=1}^{N_{\text{data}}} \left(y^{(i)} - y^{(i)}(-1)^j - 1 \right) \left(x_{r_1}^{(i)} \cdots x_{r_j}^{(i)} \right) \tag{7}
$$

we can finally express

$$
\begin{aligned}
&J^*_{\text{approx}}(\theta) = \\
&\frac{1}{N_{\text{data}}} \sum_{j=1}^{k} \sum_{r_1, \ldots, r_j = 0}^{d} a_j (\theta_{r_1} \cdots \theta_{r_j}) A_{j, r_1, \ldots, r_j} - a_0
\end{aligned} \tag{8}
$$

Since degree $k = 2$ is mainly used in later sections, we give the explicit formulas of (7) as

$$
A_{1, r_1} = \sum_{i=1}^{N_{\text{data}}} \underbrace{\left(2y^{(i)} - 1 \right) \left(x_{r_1}^{(i)} \right)}_{\text{owned by the } i^{\text{th}} \text{ data source}} \tag{9}
$$

$$
A_{2, r_1, r_2} = \sum_{i=1}^{N_{\text{data}}} \underbrace{(-1) \left(x_{r_1}^{(i)} x_{r_2}^{(i)} \right)}_{\text{owned by the } i^{\text{th}} \text{ data source}} \tag{10}
$$

for later references. Furthermore, the coefficients a_j at (3) can be chosen via Taylor expansion, namely

$$\text{(Taylor) } a_0 = -\log 2, a_1 = -0.5, a_2 = -0.125.$$

Another method is via minimizing the area between the original and the quadratic approximation, yielding[1]

$$\text{(Area min.) } a_0 = -0.714761, a_1 = -0.5, a_2 = -0.0976419.$$

Figure 2 depicts the original and the (two) approximate functions, showing that the method of area minimization is apparently better. When d is relatively small such as $d \leq 8$, both approximations work equally well in our experiments. When d is bigger such as $d = 44$, the approximation using the area tends to performs a little better (e.g., increasing the prediction accuracy by 2%).

3 Models for secure outsourced computation

3.1 Model 1: encryption only

We use the model of [20]. In the following we recap the details.

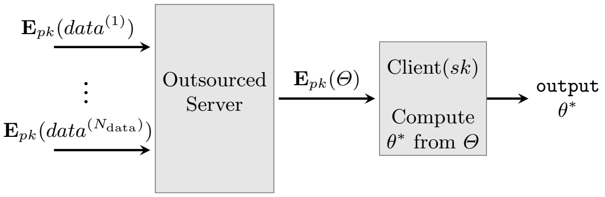
$\mathbf{E}_{pk}(data^{(1)})$
⋮
$\mathbf{E}_{pk}(data^{(N_{\text{data}})})$
Outsourced Server
$\mathbf{E}_{pk}(\Theta)$
Client(sk)
Compute θ^* from Θ
output θ^*

Fig. 3. Model 1 with encryption only (for data secrecy). The protection goal is against the honest-but-curious server.

Model outline. The general picture of the protocol is in Figure 3. We use

$$\mathbf{E}_{pk}(data^{(1)}), \ldots, \mathbf{E}_{pk}(data^{(N_{\text{data}})})$$

to represent the encryption of the data under a public key pk from various geographically distributed data contributors. After receiving the encrypted data, using homomorphic property of $\mathbf{E}_{pk}$, the computing server does necessary computations and sends the output $\mathbf{E}_{pk}(\Theta)$ to the data analyst, from which Θ is recovered by decryption, and the final result θ^* is obtained (from Θ). Note that, naively $\Theta = \theta^*$ but it is not a must[2]; indeed, what requires is that θ^* can be efficiently derived from Θ.

Key generation. The client generates the public and secret key pair (pk, sk) and publicly distributes pk.

Data encryption. Data from the client or many contributors is encrypted and sent to the outsourced server. We assume that these encryption and uploading processes are always correctly executed.

[1] found by using the function `fit` of `gnuplot`.

[2] For example, in [14, 17], $\Theta = (\theta_1^*, \theta_2^*) \in \mathbb{Z}^2$ and $\theta^* = \theta_1^*/\theta_2^* \in \mathbb{R}$, as current homomorphic encryption schemes do not support division directly.

Threat (semi-honest server) and protection goal. The outsourced server is assumed *honest-but-curious*: it is curious on any information from the data, and yet is honest in instructed computations. This curious nature of the server is considered a threat. The protection goal of our protocol in Figure 3 is to hide any information of the data from the server.

This honest-but-curious assumption is reasonable to model an economically motivated cloud service provider: it wants to provide excellent service for a successful business, but would be interested in any extra available information. On the order hand, a malicious cloud service provider can mishandle calculations, delete data, refuse to return results, collude with other parties etc. Nevertheless, it is likely to be caught in most of these malicious behaviors, and hence harms its reputation in business. Therefore, we will stick to the assumption of honest-but-curious server.

Honest client with decryption key. The client can decrypt the encrypted data, and this is always inherent in using homomorphic encryption for outsourced computation [9], as the client holds the secret key. This makes sense, e.g., in medical research, as the client can be a research institute having its own responsibility on handling the data by laws.

3.2 Model 2: encryption + differential privacy

Figure 4 extends Figure 3 by considering differential privacy, namely output privacy in the sense that the output θ^* reveals nothing meaningful from any specific input $data^{(i)}$. For that purpose, we view the server and the client and their interactions as an interactive mechanism $\mathcal{I} = (\text{Server}, \text{Client})$ with inputs $data^{(1)}, \ldots, data^{(N_{\text{data}})}$ and output θ^*.

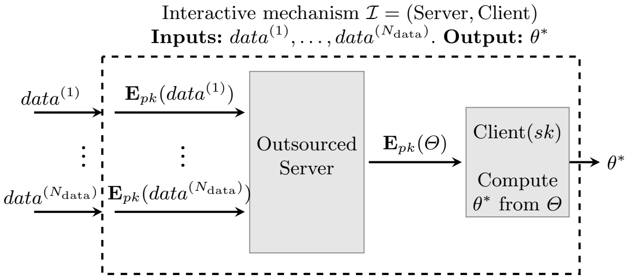
Interactive mechanism $\mathcal{I} = (\text{Server}, \text{Client})$
Inputs: $data^{(1)}, \ldots, data^{(N_{\text{data}})}$. **Output:** θ^*
$data^{(1)}$
$\mathbf{E}_{pk}(data^{(1)})$
$\vdots$
$\vdots$
$data^{(N_{\text{data}})}$
$\mathbf{E}_{pk}(data^{(N_{\text{data}})})$
Outsourced Server
$\mathbf{E}_{pk}(\Theta)$
Client(sk)
Compute θ^* from Θ
θ^*

Fig. 4. Model 2 of encryption plus differential privacy (for data secrecy and output privacy).

Let D and D_* be two neighbor databases, namely

$$D = \{data^{(1)}, \ldots, data^{(N_{\text{data}}-1)}, data^{(N_{\text{data}})}\}$$
$$D_* = \{data^{(1)}, \ldots, data^{(N_{\text{data}}-1)}, data_*^{(N_{\text{data}})}\}$$

differing only at the final item, i.e., $data^{(N_{\text{data}})} \neq data_*^{(N_{\text{data}})}$. Denote $\mathcal{I}(D) \Rightarrow \theta^*$ as a shorthand for saying that the interactive mechanism $\mathcal{I}$ in Figure 4 outputs θ^* with input D, we have the following definition.

Definition 1. *(Differential privacy) The interactive mechanism $\mathcal{I}$ = (Server, Client) in Figure 4 has ϵ-differential privacy, if and only if*

$$\Pr[\mathcal{I}(D) \Rightarrow \theta^*] \leq \exp(\epsilon) \cdot \Pr[\mathcal{I}(D_*) \Rightarrow \theta^*]$$

for any output θ^*.

The above definition naturally extends the definition in the literature [8] in which $\mathcal{I}$ is a randomized algorithm. Intuitively, the definition ensures that the change in any single data item will not much affect the final output.

4 Our system for secure logistic regression

4.1 Securing data via encryption

We present our secure system for logistic regression under the model outlined in Figure 3. Overall, in the following, the workload of the parties characterized by dataset size N_{data} and dimension d ($\ll N_{\text{data}}$ presumably) is in Table 1. The server carries the heaviest computation and storage. Remarkably, by our system design, the server computations can be easily parallelized, detailed below.

Table 1. Costs in dataset size N_{data} and dimension d.

	Storage	**Computation**
Server	$O(N_{\text{data}}d^2)$	$O(N_{\text{data}}d^2)$
Client	N/A	$O(d^2)$
Data sources	N/A	$O(d^2)$

	Communication
Each data source → server	$O(d^2)$
Server → client	$O(d^2)$

Encryption at data sources. Each data source i computes real numbers $a_{1,r_1}^{(i)} = \left(2y^{(i)} - 1\right) x_{r_1}^{(i)} \in \mathbb{R}$ and $a_{2,r_1,r_2}^{(i)} = (-1)\left(x_{r_1}^{(i)} x_{r_2}^{(i)}\right) \in \mathbb{R}$ for all $0 \leq r_1, r_2 \leq d$. Counting distinctly, these are $d+1$ and $(d+1)(d+2)/2$ numbers respectively, so that the data source needs to prepare totally

$$n_d = \frac{(d+1)(d+4)}{2} \tag{11}$$

real numbers using its d-dimensional data. In later sections, for the convenience of indexing, we denote these n_d numbers from data source i as $dat^{(i)} = (dat_1^{(i)}, \ldots, dat_{n_d}^{(i)}) \in \mathbb{R}^{n_d}$.

Next, the data source encrypts the above $O(d^2)$ real numbers, using some additively homomorphic encryption scheme, to produce the ciphertext

$$CT^{(i)} = \mathbf{E}_{pk}\left(\{a_{1,r_1}^{(i)}\}_{0 \leq r_1 \leq d}, \{a_{2,r_1,r_2}^{(i)}\}_{0 \leq r_1, r_2 \leq d}\right)$$

or equivalently $CT^{(i)} = \mathbf{E}_{pk}(dat^{(i)})$, in which concrete instantiations of the encryption $\mathbf{E}_{pk}$ together with real number encodings will be described in later section.

Cloud server computation. The server receives and stores $CT^{(i)}$ for $1 \leq i \leq N_{\text{data}}$ from all data sources. It then computes the following ciphertext additions

$$CT = \sum_{i=1}^{N_{\text{data}}} CT^{(i)} \tag{12}$$

and sends CT to the client.

Client decryption. The client decrypts CT using its secret key. Due to the additive homomorphism of $\mathbf{E}_{pk}$, what the client obtains is $\sum_{i=1}^{N_{\text{data}}} dat^{(i)} \in \mathbb{R}^{n_d}$ or equivalently,

$$\sum_{i=1}^{N_{\text{data}}} a_{1,r_1}^{(i)} \in \mathbb{R}, \sum_{i=1}^{N_{\text{data}}} a_{2,r_1,r_2}^{(i)} \in \mathbb{R}$$

for all indexes $0 \le r_1, r_2 \le d$. These are exactly the coefficients given in (9) and (10) of the cost function at (8). Using the coefficients, the client then finds the minimizer $\theta^* = \text{argmin}_\theta J_{\text{approx}}(\theta)$.

Theorem 1. *The above system is secure against the semi-honest server in Figure 3 (and Section 3.1).*

Proof. The proof is straightforward since the server only handles encrypted data, so the security is reduced to the semantic security of the underlying encryption scheme (which is either Paillier's or LWE-based in the following section).

4.2 Adding differential privacy

Recall that, a noise $x \in \mathbb{R}$ has $\text{Lap}(\sigma)$ distribution if its probability density function is $\frac{1}{2\sigma}\exp(-|x|/\sigma)$. To obtain ϵ-differential privacy, the only change we need is the computation of the server given at (12). Namely, the server generates Laplace noises $e = (e_1, \dots, e_{n_d}) \in \mathbb{R}^{n_d}$ in which all components are from the $\text{Lap}(\varLambda_d/\epsilon)$ distribution where $\varLambda_d = 2n_d = (d+1)(d+4)$, and does the following computation in the replacement of (12)

$$CT = \mathbf{E}_{pk}(e) + \sum_{i=1}^{N_{\text{data}}} CT^{(i)}. \tag{13}$$

Theorem 2. *The system in Section 4.1 with the change at (13) satisfies ϵ-differential privacy (Definition 1).*

Proof. The effect of (13) is that the client obtains after decryption

$$e + \sum_{i=1}^{N_{\text{data}}} dat^{(i)} \in \mathbb{R}^{n_d}$$

or equivalently,

$$\text{Lap}(\varLambda_d/\epsilon) + \sum_{i=1}^{N_{\text{data}}} a_{1,r_1}^{(i)} \in \mathbb{R}$$
$$\text{Lap}(\varLambda_d/\epsilon) + \sum_{i=1}^{N_{\text{data}}} a_{2,r_1,r_2}^{(i)} \in \mathbb{R}$$

for all indexes $0 \le r_1, r_2 \le d$. These are exactly the Laplace-perturbed coefficients of the cost function at (8), so that the claimed differential privacy follows via the functional mechanism [25]. More details are given below.

By Definition 1, it is necessary to show that, for any output θ^*,

$$\Pr[\mathcal{I}(D) \Rightarrow \theta^*] \leq \exp(\epsilon) \cdot \Pr[\mathcal{I}(D_*) \Rightarrow \theta^*].$$

As the final computation $\theta^* = \mathrm{argmin}_\theta J_{\text{approx}}(\theta)$ at the client is deterministic, the above is equivalent to proving

$$\frac{\Pr[\mathcal{I}(D) \Rightarrow J_{\text{approx}}(\theta)]}{\Pr[\mathcal{I}(D_*) \Rightarrow J_{\text{approx}}(\theta)]} \leq \exp(\epsilon) \tag{14}$$

in which function $J_{\text{approx}}(\theta)$ here is represented by its coefficients $(A_{1,r_1}, A_{2,r_1,r_2})$ in (9) and (10). When a training set D is used, we make D explicit in those coefficients as follows

$$\begin{aligned} A_{1,r_1,D} &= \sum_{i=1}^{N_{\text{data}}} \left(2y_D^{(i)} - 1\right)\left(x_{r_1,D}^{(i)}\right) \\ A_{2,r_1,r_2,D} &= \sum_{i=1}^{N_{\text{data}}} (-1)\left(x_{r_1,D}^{(i)} x_{r_2,D}^{(i)}\right) \end{aligned}$$

and likewise for those related to the training set D_*. As the interactive mechanism $\mathcal{I}(D)$ perturbs the coefficients by noises $\mathrm{Lap}(\Lambda_d/\epsilon)$, let

$$\begin{aligned} A_{1,r_1,D}^{dp} &= \mathrm{Lap}(\Lambda_d/\epsilon) + A_{1,r_1,D} \\ A_{2,r_1,r_2,D}^{dp} &= \mathrm{Lap}(\Lambda_d/\epsilon) + A_{2,r_1,r_2,D}, \end{aligned}$$

then what requires at (14) becomes

$$\frac{\Pr[(A_{1,r_1,D}^{dp}, A_{2,r_1,r_2,D}^{dp}) = (A_{1,r_1}, A_{2,r_1,r_2}) \forall r_1, r_2]}{\Pr[(A_{1,r_1,D_*}^{dp}, A_{2,r_1,r_2,D_*}^{dp}) = (A_{1,r_1}, A_{2,r_1,r_2}) \forall r_1, r_2]} \leq \exp(\epsilon)$$

whose left hand side is $\delta_1 \cdot \delta_2$ where

$$\begin{aligned} \delta_1 &= \frac{\prod_{r_1} \exp(-\frac{\epsilon}{\Lambda_d}|A_{1,r_1,D} - A_{1,r_1}|)}{\prod_{r_1} \exp(-\frac{\epsilon}{\Lambda_d}|A_{1,r_1,D_*} - A_{1,r_1}|)} \\ \delta_2 &= \frac{\prod_{r_1,r_2} \exp(-\frac{\epsilon}{\Lambda_d}|A_{2,r_1,r_2,D} - A_{2,r_1,r_2}|)}{\prod_{r_1,r_2} \exp(-\frac{\epsilon}{\Lambda_d}|A_{2,r_1,r_2,D_*} - A_{2,r_1,r_2}|)} \end{aligned}$$

so that

$$\begin{aligned} \delta_1 &\leq \prod_{0 \leq r_1 \leq d} \exp\left(\frac{\epsilon}{\Lambda_d}|A_{1,r_1,D_*} - A_{1,r_1,D}|\right), \\ \delta_2 &\leq \prod_{0 \leq r_1, r_2 \leq d} \exp\left(\frac{\epsilon}{\Lambda_d}|A_{2,r_1,r_2,D_*} - A_{2,r_1,r_2,D}|\right). \end{aligned}$$

As D and D_* differ only at the final (N_{data}-th) item, we have

$$\begin{aligned}
&|A_{1,r_1,D_*} - A_{1,r_1,D}| \\
&= \left|\left(2y_{D_*}^{(N_{\text{data}})} - 1\right) x_{r_1,D_*}^{(N_{\text{data}})} - \left(2y_{D}^{(N_{\text{data}})} - 1\right) x_{r_1,D}^{(N_{\text{data}})}\right| \\
&\le \left|\left(2y_{D_*}^{(N_{\text{data}})} - 1\right) x_{r_1,D_*}^{(N_{\text{data}})}\right| + \left|\left(2y_{D}^{(N_{\text{data}})} - 1\right) x_{r_1,D}^{(N_{\text{data}})}\right| \\
&\le 1 + 1 = 2
\end{aligned}$$

assuming that $|x_{r_1,D_*}^{(N_{\text{data}})}|, |x_{r_1,D}^{(N_{\text{data}})}| \le 1$ (via data normalization) and $y_D^{(N_{\text{data}})}, y_{D_*}^{(N_{\text{data}})} \in \{0,1\}$. Also,

$$\begin{aligned}
&|A_{2,r_1,r_2,D_*} - A_{2,r_1,r_2,D}| \\
&= \left|(-1)\left(x_{r_1,D_*}^{(N_{\text{data}})} x_{r_2,D_*}^{(N_{\text{data}})}\right) - (-1)\left(x_{r_1,D}^{(N_{\text{data}})} x_{r_2,D}^{(N_{\text{data}})}\right)\right| \\
&= \left|x_{r_1,D}^{(N_{\text{data}})} x_{r_2,D}^{(N_{\text{data}})} - x_{r_1,D_*}^{(N_{\text{data}})} x_{r_2,D_*}^{(N_{\text{data}})}\right| \\
&\le 2
\end{aligned}$$

again assuming that $|x_{r_1,D_*}^{(N_{\text{data}})}|$, $|x_{r_1,D}^{(N_{\text{data}})}|$, $|x_{r_2,D}^{(N_{\text{data}})}|$, $|x_{r_2,D_*}^{(N_{\text{data}})}| \le 1$ via data normalization. Therefore,

$$\begin{aligned}
\delta_1 &\le \prod_{0 \le r_1 \le d} \exp\left(\frac{2\epsilon}{\Lambda_d}\right) = \exp\left(\frac{2(d+1)\epsilon}{\Lambda_d}\right) \\
\delta_2 &\le \prod_{0 \le r_1, r_2 \le d} \exp\left(\frac{2\epsilon}{\Lambda_d}\right) = \exp\left(\frac{(d+1)(d+2)\epsilon}{\Lambda_d}\right)
\end{aligned}$$

so that

$$\delta_1 \cdot \delta_2 \le \exp\left(\frac{(d+1)(d+4)\epsilon}{\Lambda_d}\right) = \exp(\epsilon)$$

since $\Lambda_d = (d+1)(d+4)$, finishing the proof.

4.3 Securing data in prediction

The output θ^* in Sections 4.1 (or 4.2) can be used for secure prediction in the sense that

- $\theta^* = (\theta_0^*, \ldots, \theta_d^*)$ can be made public, and
- data used in prediction $x = (x_1, \ldots, x_d)$ is encrypted,

and yet the output

$$h_{\theta^*}(x) = \frac{1}{1 + \exp(-\theta_0^* - \sum_{j=1}^d \theta_j^* x_j)}$$

can be computed.

To compute $h_{\theta^*}(x)$, it suffices to know $\sum_{j=1}^d \theta_j^* x_j$. If the data for prediction is encrypted, it is necessary to compute

$$\sum_{j=1}^d \theta_j^* \mathbf{E}_{pk}(x_j)$$

which is discussed below:

– If $\mathbf{E}_{pk}(\cdot)$ supports only ciphertext addition, then it is necessary to multiply a greatest common divisor to convert the (fixed precision) scalars $\theta_j^* \in \mathbb{R}$ into integers. Then ciphertext additions are applied to obtain the encrypted result.
– If $\mathbf{E}_{pk}(\cdot)$ supports one multiplication such as in the (below) LWE and ring-LWE cases, it suffices to compute the sum of ciphertext product $\sum_{j=1}^{d} \mathbf{E}_{pk}(\theta_j^*)\mathbf{E}_{pk}(x_j)$.

In both cases, the data holder with the secret key sk decrypts to obtain $\sum_{j=1}^{d} \theta_j^* x_j$ and finally computes $h_{\theta^*}(x)$.

5 Instantiations of our system

In this section we use additively homomorphic encryption schemes to instantiate our system in Section 4. We employ three schemes: Paillier encryption, LWE-based encryption, and ring-LWE-based encryption. The Paillier scheme is not quantum-safe, so LWE-based and ring-LWE-based ones are the choices whenever quantum-resistant security is expected. Regarding the quantum-safe schemes, the ring-LWE-based one has smaller public key size while the LWE-based yields surprisingly smaller ciphertexts in communication, showed below. These features enable flexible choices to employ our system in practice.

5.1 Using Paillier encryption

Paillier encryption. With public key $pk = n$, the encryption of an integer $m \in \{0, \dots, n-1\}$ is $\mathsf{PaiEnc}_{pk}(m) = r^n(1+n)^m \mod n^2$ in which r is chosen randomly from $\{0, \dots, n-1\}$. The encryption is additively homomorphic because the ciphertext product $\mathsf{PaiEnc}_{pk}(m_1)\mathsf{PaiEnc}_{pk}(m_2) \mod n^2$ becomes an encryption of $m_1 + m_2 \mod n$.

Packing data in encryption. As the plaintext space has $\log_2 n \geq 2048$ bits, we can pack many integers $dat_1, \dots, dat_t$ each of prec bits into each Paillier plaintext as follows.

$$\mathsf{PaiEnc}_{pk}\Big(\overbrace{\underbrace{[dat_1 0_{\mathsf{pad}}]}_{\mathsf{prec}+\mathsf{pad}\text{ bits}} \cdots \underbrace{[dat_t 0_{\mathsf{pad}}]}_{\mathsf{prec}+\mathsf{pad}\text{ bits}}}^{\lfloor \log_2 n \rfloor \text{ bits}} \Big)$$

in which 0_{pad} is the zero padding of pad bits, which helps preventing overflows in ciphertext additions. Typically, $\mathsf{pad} \approx \log_2 N_{\text{data}}$ as we need N_{data} additions of ciphertexts. Moreover, as the number of plaintext bits must be less than $\log_2 n$, it is necessary that $t(\mathsf{prec} + \mathsf{pad}) \leq \log_2 n$. Therefore,

$$t = \left\lfloor \frac{\lfloor \log_2 n \rfloor}{\mathsf{prec} + \mathsf{pad}} \right\rfloor$$

which is the upper-bound of packing prec-bit integers into one Paillier plaintext. As a real number $0 \leq r < 1$ can be represented as an integer of form $\lfloor r \cdot 2^{\mathsf{prec}} \rfloor$, the above packing method can be used to encrypt around $\lfloor \log_2 n \rfloor / (\mathsf{prec} + \mathsf{pad})$ real numbers in the range $[0, 1)$ with precision prec, tolerating around 2^{pad} ciphertext additions.

Communication cost (data source to server). In our system in Section 4, each data source needs to encrypt and send $n_d = O(d^2)$ real numbers to the server. Therefore, with the above packing method, the number of Paillier ciphertexts sent from each data source is

$$\frac{n_d}{t} = \frac{n_d(\mathsf{prec} + \mathsf{pad})}{\lfloor \log_2 n \rfloor}$$

which is around

$$\mathsf{CommCostPaillier} = 2n_d(\mathsf{prec} + \mathsf{pad}) \text{ (bits)} \tag{15}$$

where n_d is at (11), since each Paillier ciphertext is of $2\lfloor \log_2 n \rfloor$ bits.

Ciphertext additions (server). The ciphertext additions on the server can be seen as follows:

$$
\begin{array}{c}N_{\text{data}}\\ \text{rows}\end{array}\left\{\overbrace{\begin{array}{ccc}
\mathsf{PaiEnc}_{pk}\Big(\overbrace{\underbrace{[dat_1^{(1)}0_{\mathsf{pad}}]}_{\mathsf{prec}+\mathsf{pad}\text{ bits}}\cdots\underbrace{[dat_t^{(1)}0_{\mathsf{pad}}]}_{\mathsf{prec}+\mathsf{pad}\text{ bits}}}^{\lfloor \log_2 n \rfloor\text{ bits}}\Big) & , & \cdots \\
\vdots & & \vdots \\
+ & & + \\
\vdots & & \vdots \\
\mathsf{PaiEnc}_{pk}\Big(\overbrace{\underbrace{[dat_1^{(N_{\text{data}})}0_{\mathsf{pad}}]}_{\mathsf{prec}+\mathsf{pad}\text{ bits}}\cdots\underbrace{[dat_t^{(N_{\text{data}})}0_{\mathsf{pad}}]}_{\mathsf{prec}+\mathsf{pad}\text{ bits}}}^{\lfloor \log_2 n \rfloor\text{ bits}}\Big) & , & \cdots
\end{array}}^{\frac{n_d}{t}\text{ columns of }\mathsf{PaiEnc}_{pk}}\right.
$$

whose computational cost is around

$$N_{\text{data}} \cdot \frac{n_d}{t} \cdot \mathbf{T}_{\mathsf{PaiAdd}} \tag{16}$$

where $\mathbf{T}_{\mathsf{PaiAdd}}$ is the time of adding two Paillier ciphertexts. The resulted n_d/t ciphertext sums in columns are sent to the client, which also requires $O(2d^2(\mathsf{prec} + \mathsf{pad}))$ bits.

Decryption (client). The client decrypts the n_d/t sums to obtain the sums $\sum_{i=1}^{N_{\text{data}}} dat_1^{(i)} \in \mathbb{Z}$, $\dots$, $\sum_{i=1}^{N_{\text{data}}} dat_t^{(i)} \in \mathbb{Z}$, $\dots$, $\sum_{i=1}^{N_{\text{data}}} dat_{n_d}^{(i)} \in \mathbb{Z}$. As $\mathsf{pad} = \lceil \log_2 N_{\text{data}} \rceil$, there will be no overflows in the integer sums as required.

5.2 Using a LWE-based encryption

In this section, $\mathbb{Z}_p$ are integers in $(-p/2, p/2]$ and likewise for $\mathbb{Z}_q$.

LWE-based encryption. We use the scheme in [3] due to its flexibility in choosing the plaintext length. Each plaintext is a vector in $\mathbb{Z}_p^l$ where p and l are almost independent with security parameters, so that it is possible to set $p \approx N_{\text{data}}$ and $l \approx O(d^2)$. Concretely, for a plaintext $m \in \mathbb{Z}_p^{1\times l}$,

$$\mathsf{lweEnc}_{pk}(m) = e_1[A|P] + p[e_2|e_3] + [0_{n_{\text{lwe}}}|m] \in \mathbb{Z}_q^{1\times(n_{\text{lwe}}+l)}$$

in which $e_1 \in \mathbb{Z}^{1\times n_{\text{lwe}}}, e_2 \in \mathbb{Z}^{1\times n_{\text{lwe}}}, e_3 \in \mathbb{Z}^{1\times l}$ are Gaussian noise vectors of deviation s; $[A|P]$ is the matrix concatenation of public matrices $A \in \mathbb{Z}_q^{n_{\text{lwe}}\times n_{\text{lwe}}}$ and $P \in \mathbb{Z}_q^{n_{\text{lwe}}\times l}$ given in the public key $pk = (A, P, (p, l), (n_{\text{lwe}}, s, q))$.

The encryption is additively homomorphic because

$$\begin{aligned}
&\mathsf{lweEnc}_{pk}(m) + \mathsf{lweEnc}_{pk}(m') \\
&= e_1[A|P] + p[e_2|e_3] + [0_{n_{\text{lwe}}}|m] + \\
&\quad e_1'[A|P] + p[e_2'|e_3'] + [0_{n_{\text{lwe}}}|m'] \\
&= (e_1 + e_1')[A|P] + p[e_2 + e_2'|e_3 + e_3'] + [0_{n_{\text{lwe}}}|m + m']
\end{aligned}$$

in $\mathbb{Z}_q^{1\times(n_{\text{lwe}}+l)}$, which is the encryption of $m + m' \in \mathbb{Z}_p^{1\times l}$.

Data encoding and encryption. Real numbers $a \in \mathbb{R}$ can be represented in $\mathsf{prec} = (L + \ell + 1)$ signed bits as

$$a = \sum_{k=-L}^{\ell} a_k 2^k$$

which is the inner product of following vectors

$$\begin{aligned}
\mathsf{Pow}(2, L, \ell) &= [2^{-L}, \ldots, 2^0, \ldots, 2^{\ell}] \in \mathbb{Z}^{\mathsf{prec}} \\
\mathsf{Bits}(a) &= [a_{-L}, \ldots, a_0, \ldots a_{\ell}] \ \in \{-1, 0, 1\}^{\mathsf{prec}}.
\end{aligned}$$

in which all signed bits are in $\{0, 1\}$ if $a \geq 0$ and in $\{-1, 0\}$ if $a < 0$.

Encryption from each data source corresponds to each row in (18). Namely, each data source i having n_d real numbers takes all signed bits of these numbers, concatenating them to form a vector in $\{-1, 0, 1\}^{n_d\cdot\mathsf{prec}}$ and encrypts that vector using lweEnc. Note that $\{-1, 0, 1\} \subset \mathbb{Z}_p$ it is sufficient to set the plaintext length $l = n_d \cdot \mathsf{prec}$.

Communication cost (data source to server). Each data source sends to the server a row in (18), whose size is

$$(n_{\text{lwe}} + n_d \cdot \mathsf{prec}) \log_2 q \text{ (bits)} \tag{17}$$

which is $O(d^2\mathsf{prec})$ as n_{lwe} and q are fixed as parameters of the scheme.

Ciphertext additions (server). The N_{data} ciphertext additions on the server can be seen the row additions as follows, where each row comes from each data source.

$$\begin{array}{c}N_{\text{data}}\\ \text{rows}\end{array}\left\{\begin{array}{c}
\overbrace{\mathsf{lweEnc}_{pk}\Big(\underbrace{\mathsf{Bits}(dat_1^{(1)})}_{\in\{-1,0,1\}^{\mathsf{prec}}} \cdots \underbrace{\mathsf{Bits}(dat_{n_d}^{(1)})}_{\in\{-1,0,1\}^{\mathsf{prec}}}\Big)}^{1 \text{ column of } \mathsf{lweEnc}_{pk} \in \mathbb{Z}_q^{1\times(n_{\text{lwe}}+n_d\cdot\mathsf{prec})}} \\
\vdots \\
+ \\
\vdots \\
\mathsf{lweEnc}_{pk}\Big(\underbrace{\mathsf{Bits}(dat_1^{(N_{\text{data}})})}_{\in\{-1,0,1\}^{\mathsf{prec}}} \cdots \underbrace{\mathsf{Bits}(dat_{n_d}^{(N_{\text{data}})})}_{\in\{-1,0,1\}^{\mathsf{prec}}}\Big)
\end{array}\right. \tag{18}$$

whose computational cost is around

$$N_{\text{data}} \cdot \mathbf{T}_{\mathsf{lweAdd}} = N_{\text{data}} \cdot (n_{\text{lwe}} + n_d \cdot \mathsf{prec}) \cdot \mathbf{t}_{\text{add}\mathbb{Z}_\text{q}} \tag{19}$$

where $\mathbf{T}_{\mathsf{lweAdd}}$ is the adding time of two vectors in the set $\mathbb{Z}_q^{n_{\text{lwe}}+n_d \cdot \mathsf{prec}}$ and $\mathbf{t}_{\text{add}\mathbb{Z}_\text{q}}$ is the time for adding two elements in $\mathbb{Z}_q$. The resulted sum is sent to the client and its length is the same as in (17).

Decryption (client). The client decrypts the sum to obtain following sums of vectors (over $\mathbb{Z}$)

$$\begin{aligned} \mathsf{Bits}(dat_1^{(1)}) + \cdots + \mathsf{Bits}(dat_1^{(N_{\text{data}})}) &\in [-N_{\text{data}}, N_{\text{data}}]^{\mathsf{prec}} \\ &\vdots \\ \mathsf{Bits}(dat_{n_d}^{(1)}) + \cdots + \mathsf{Bits}(dat_{n_d}^{(N_{\text{data}})}) &\in [-N_{\text{data}}, N_{\text{data}}]^{\mathsf{prec}} \end{aligned}$$

and then takes the inner products with $\mathsf{Pow}(2, L, \ell)$ to get following $n_d = O(d^2)$ real numbers

$$\begin{aligned} & dat_1^{(1)} + \cdots + dat_1^{(N_{\text{data}})} \in \mathbb{R} \\ & \vdots \\ & dat_{n_d}^{(1)} + \cdots + dat_{n_d}^{(N_{\text{data}})} \in \mathbb{R} \end{aligned}$$

as required. The condition for not overflowing here is that $[-N_{\text{data}}, N_{\text{data}}] \subseteq \mathbb{Z}_p$, meaning the plaintext of the resulted ciphertext after additions must be in $\mathbb{Z}_p^{n_d \cdot \mathsf{prec}}$. Therefore $N_{\text{data}} < p/2$ is necessary.

5.3 Using a ring-LWE-based encryption

Ring-LWE-based encryption. We use the ring-LWE-based scheme described in [10]. Define ring $R = \mathbb{Z}[x]/f(x)$, $f(x) = x^{n_{\text{rlwe}}} + 1$, and quotient rings $R_q = R/q$, $R_p = R/p$. The notion $R_{(0,s)}$ stands for polynomials in R with small Gaussian coefficients of mean 0 and deviation s.

The encryption of $\mathbf{m} \in R_p$ under public key $pk = (\mathbf{a}, \mathbf{p}) \in R_q^2$ is as follows

$$\mathsf{rlweEnc}_{pk}(\mathbf{m}) = (\mathbf{e}_1\mathbf{a} + \mathbf{e}_2, \mathbf{e}_1\mathbf{p} + \mathbf{e}_3 + \lfloor q/p \rfloor \mathbf{m}) \in R_q^2$$

in which $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 \in R_{(0,s)}$ are noises.

The addition of ciphertexts can be done naturally as

$$\mathsf{rlweEnc}_{pk}(\mathbf{m}) + \mathsf{rlweEnc}_{pk}(\mathbf{m}') \in R_q^2$$

in which $\mathbf{m}, \mathbf{m}' \in R_p$ can also be seen as vectors of length n_{rlwe} over $\mathbb{Z}_p = (-p/2, p/2] \cap \mathbb{Z}$.

Data encoding and encryption. Data encoding is identical to Section 5.2. Encryption of the encoding is different since the plaintext length is fixed with n_{rlwe}. Namely each data source needs to pack their data items into the ring-LWE encryption. Referring to the rows of (21), let t be the number of data items packed into one $\mathsf{rlweEnc}$, we have $t \cdot \mathsf{prec} = n_{\text{rlwe}}$ so that $t = \left\lfloor \frac{n_{\text{rlwe}}}{\mathsf{prec}} \right\rfloor$.

Communication cost (data source to server). Each data source sends to the server a row in (21), whose size is

$$\frac{n_d}{t}(2n_{\text{rlwe}} \log_2 q) \approx 2n_d \cdot \mathsf{prec} \cdot \log_2 q \text{ (bits)} \tag{20}$$

since each ring-LWE ciphertext is of $2n_{\text{rlwe}}\log_2 q$ (bits). It is worth noting that the cost at (20) in ring-LWE case is bigger than (17) in LWE case if $n_{\text{lwe}} < n_d \cdot \mathsf{prec}$.

Ciphertext additions (server). The N_{data} ciphertext additions on the server can be seen the row additions as follows, where each row comes from each data source.

$$\begin{array}{c}N_{\text{data}}\\ \text{rows}\end{array}\left\{\begin{array}{c}\overbrace{\hphantom{\mathsf{rlweEnc}_{pk}\Big(\mathsf{Bits}(dat_1^{(N_{\text{data}})})\cdots\mathsf{Bits}(dat_t^{(N_{\text{data}})})\Big),\ldots}}^{\frac{n_d}{t}\text{ columns of }\mathsf{rlweEnc}_{pk}\in R_q^2}\\ \mathsf{rlweEnc}_{pk}\Big(\underbrace{\mathsf{Bits}(dat_1^{(1)})}_{\in\{-1,0,1\}^{\mathsf{prec}}}\cdots\underbrace{\mathsf{Bits}(dat_t^{(1)})}_{\in\{-1,0,1\}^{\mathsf{prec}}}\Big),\ldots\\ \vdots\\ +\\ \vdots\\ \mathsf{rlweEnc}_{pk}\Big(\underbrace{\mathsf{Bits}(dat_1^{(N_{\text{data}})})}_{\in\{-1,0,1\}^{\mathsf{prec}}}\cdots\underbrace{\mathsf{Bits}(dat_t^{(N_{\text{data}})})}_{\in\{-1,0,1\}^{\mathsf{prec}}}\Big),\ldots\end{array}\right. \tag{21}$$

whose computational cost is around

$$N_{\text{data}}\cdot\frac{n_d}{t}\cdot\mathbf{T}_{\mathsf{rlweAdd}}\approx N_{\text{data}}\cdot\frac{n_d\cdot\mathsf{prec}}{n_{\text{rlwe}}}\cdot\mathbf{T}_{\mathsf{rlweAdd}} \tag{22}$$

in which $\mathbf{T}_{\mathsf{rlweAdd}}$ is the time of adding two $\mathsf{rlweEnc}$ ciphertexts (seen as polynomials) in R_q^2.

Decryption (client). This is the same as Section 5.2 and the condition $N_{\text{data}} < p/2$ is again necessary to prevent overflows in adding the columns.

5.4 Costs of computation and communication

The computation and communication costs are depicted in Figure 5, in which the size N_{data} is intentionally set as large as 10^8s to demonstrate the *scaling feasibility* of our system. Therefore, for real datasets with $N_{\text{data}} < 10^8$ (as in Section 6 below), our system is very efficient in both communication and computation.

Below are details on the parameters to produce that figure. Timings are based on a commodity server of 2.60 GHz x 2 CPU, 128 GB RAM. It is worth noting that the computation cost of $O(N_{\text{data}}d^2)$ can be further reduced to

$$O(N_{\text{data}}d^2/n_{\mathsf{threads}})$$

if n_{threads} parallelized threads can be used. With our server, $n_{\mathsf{threads}} = 20$. Also, we exclude the time for memory access in comparing computation costs of the instantiations.

Using Paillier's scheme. We take the Paillier modulus n of 3072 bits, which is standard to gain 128-bit security, so that $\log_2 n \approx 3072$. The communication cost (each data source to server; server to client) is computed using (15) with precision $\mathsf{prec} = 64$ and padding number $\mathsf{pad} = \lceil\log_2(N_{\text{data}})\rceil$. The computation cost is computed via (16) divided by $n_{\mathsf{threads}} = 20$, with $\mathbf{T}_{\mathsf{PaiAdd}} = 10\cdot 10^{-6}$ seconds our commodity server.

Using the LWE-based scheme. The communication cost (each data source to server; server to client) is computed using (17) in which $n_{\text{lwe}} = 3530$, $\mathsf{prec} = 64$, $\log_2 q = 114$. The parameters of

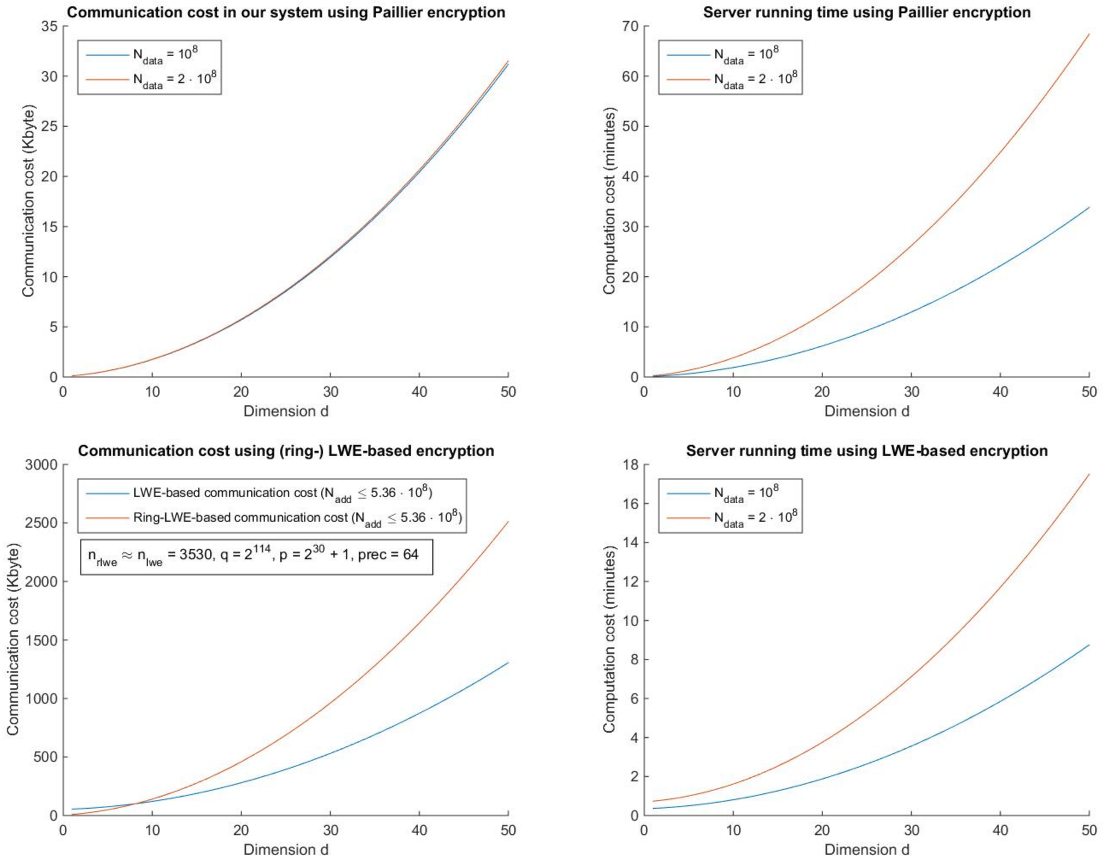

Fig. 5. Simulated costs of communication and computation for very large datasets of size 10^8s.

(n_{lwe}, q) and Gaussian deviation $s = 8.0$ are set to gain around 128-bit security with respect to current attacks [2,15,16]. The computation cost is gained via (19) divided by $n_{\mathsf{threads}} = 20$, in which the time $\mathbf{t}_{\text{add}\mathbb{Z}_\text{q}}$ of adding to element over $\mathbb{Z}_q$ is $\mathbf{t}_{\text{add}\mathbb{Z}_\text{q}} = 1.146 \cdot 10^{-9}$ seconds on our server, using our own implementation for the dedicated q.

Using the Ring-LWE-based scheme. We use the same parameters as in the LWE case for 128-bit security. In particular, $n_{\text{rlwe}} \approx 3530$. The communication cost (20) is depicted using these parameters in Figure 5. As expected, the ring-LWE-based communication cost is larger than the LWE-based one when the data dimension d increases (e.g. ≥ 10).

It is also worth noting that the computation cost of using the ring-LWE-based scheme in our system is *larger* than that of the LWE-based one when $d \geq 9$, argued as follows. Looking (19) and (22), it suffices to show that

$$N_{\text{data}} \cdot \mathbf{T}_{\mathsf{lweAdd}} \leq N_{\text{data}} \cdot \frac{n_d \cdot \mathsf{prec}}{n_{\text{rlwe}}} \cdot \mathbf{T}_{\mathsf{rlweAdd}}$$

or equivalently

$$\mathbf{T}_{\mathsf{lweAdd}} \leq \frac{n_d \cdot \mathsf{prec}}{n_{\text{rlwe}}} \cdot \mathbf{T}_{\mathsf{rlweAdd}}.$$

Let $\mathbf{t}_{\mathbb{Z}_q}^{\text{add}}$ be the time of adding two elements in $\mathbb{Z}_q$, then we can estimate $\mathbf{T}_{\mathsf{lweAdd}} = (n_{\text{lwe}} + n_d\mathsf{prec})\mathbf{t}_{\mathbb{Z}_q}^{\text{add}}$ and $\mathbf{T}_{\mathsf{rlweAdd}} = 2n_{\text{rlwe}}\mathbf{t}_{\mathbb{Z}_q}^{\text{add}}$. The reason is that $\mathbf{T}_{\mathsf{lweAdd}}$ is the time for adding two elements in $\mathbb{Z}_q^{n_{\text{lwe}}+n_d\mathsf{prec}}$ and $\mathbf{T}_{\mathsf{rlweAdd}}$ is for adding two polynomials of degree n_{rlwe} in $\mathbb{Z}_q$. Therefore, it is sufficient to show

$$(n_{\text{lwe}} + n_d\mathsf{prec})\mathbf{t}_{\mathbb{Z}_q}^{\text{add}} \le \frac{n_d \cdot \mathsf{prec}}{n_{\text{rlwe}}} \cdot 2n_{\text{rlwe}}\mathbf{t}_{\mathbb{Z}_q}^{\text{add}}$$

which holds true as long as $n_{\text{lwe}} \le n_d\mathsf{prec}$. Since $n_{\text{lwe}} = 3530$, $\mathsf{prec} = 64$ in our parameters, the condition becomes $55 < n_d$ $(= (d+1)(d+4)/2)$, which is true if $d \ge 9$ as claimed. Therefore, we omit the ring-LWE-based computation cost and only depict the LWE-based one in Figure 5.

6 Experiments with real datasets

This section aims at showing the utility of our secure system regarding the accuracies, F-scores, and AUCs of logistic regression in real datasets.

We first remind the confusion matrix in Table 2 and its derivations for evaluating a classifier. We then conducts experiments with real UCI datasets [1] to compare the logistic regression over unencrypted data, encrypted data, and encrypted data with differential privacy, and reports the results in Table 3 and Figure 6.

Table 2. Comparing actual/predicted class.

		Actual class $y^{(i)}$	
		1	0
predicted	1	True Positive (TP)	False Positive (FP)
class $y^{(i)}$	0	False Negative (FN)	True Negative (TN)

Derivations from the confusion matrix. Referring to Table 2, the accuracy, the precision P and recall R, true positive rate TPR and false positive rate FPR are as follows

$$\text{Accuracy} = \frac{\#\mathsf{TP} + \#\mathsf{TN}}{\#\mathsf{TP} + \#\mathsf{FP} + \#\mathsf{FN} + \#\mathsf{TN}}$$

$$\text{(Precision) } P = \frac{\#\mathsf{TP}}{\#\mathsf{TP} + \#\mathsf{FP}} \tag{23}$$

$$\text{(Recall) } R = \mathsf{TPR} = \frac{\#\mathsf{TP}}{\#\mathsf{TP} + \#\mathsf{FN}} \tag{24}$$

$$\mathsf{FPR} = \frac{\#\mathsf{FP}}{\#\mathsf{FP} + \#\mathsf{TN}}$$

in which $\#\mathsf{TP}$ reads as the number of true positives (TP) and likewise for the others.

F-score. The score (aka, F_1-score) is define as

$$F_{\text{score}} = \frac{2PR}{P+R}$$

where P and R are at (23) and (24).

Table 3. Experiments with real datasets to compare accuracies, F-scores, and AUCs.

Dataset	Logistic Regression over	Cost Function	Accuracy	F-score	AUC
Pima [1,21]	Unencrypted (data)	Original at (1)	80.2%	0.688525	0.873653
	Encrypted (data)	Approximate at (8)	80.7%	0.694215	0.876347
	Encrypted + ϵ-DiffPriv	Approximate at (8)	73.4%	0.523364	0.805328
SPECTF [1,13]	Unencrypted (data)	Original at (1)	79.1%	0.876972	0.783333
	Encrypted (data)	Approximate at (8)	75.4%	0.851613	0.761628
	Encrypted + ϵ-DiffPriv	Approximate at (8)	72.1%	0.831169	0.673256

(As N_{data} is small here, computation and communication costs are negligible of only a few seconds and kilobytes.)

ROC curve and AUC. By varying the threshold $0 \leq \mathsf{thres} \leq 1$ at (2), we obtain multiple pair of (FPR, TPR) which can be plotted in a curve, called the ROC curve. Note if $\mathsf{thres} = 0$ then in (2), the prediction y^{new} will be always 1, so that $\#\mathsf{FN} = \#\mathsf{TN} = 0$, meaning $\mathsf{FPR} = \mathsf{TPR} = 1$. On the other hand, if $\mathsf{thres} = 1$,then y^{new} will be always 0, namely $\#\mathsf{TP} = \#\mathsf{FP} = 0$, so that $\mathsf{FPR} = \mathsf{TPR} = 0$. These extremes correspond to points $(0, 0)$ and $(1, 1)$ in the ROC curve. The "ideal" point is $(\mathsf{FPR}, \mathsf{TPR}) = (0, 1)$, specifying that the trained model correctly classifies all testing data.

The AUC is the area under the ROC curve. Together with accuracy and F-score, it is a measure of how well a classifier works. Having the ROC curve, the AUC can be computed via the built-in `trapz` function of Octave.

6.1 Using the Pima diabetes dataset

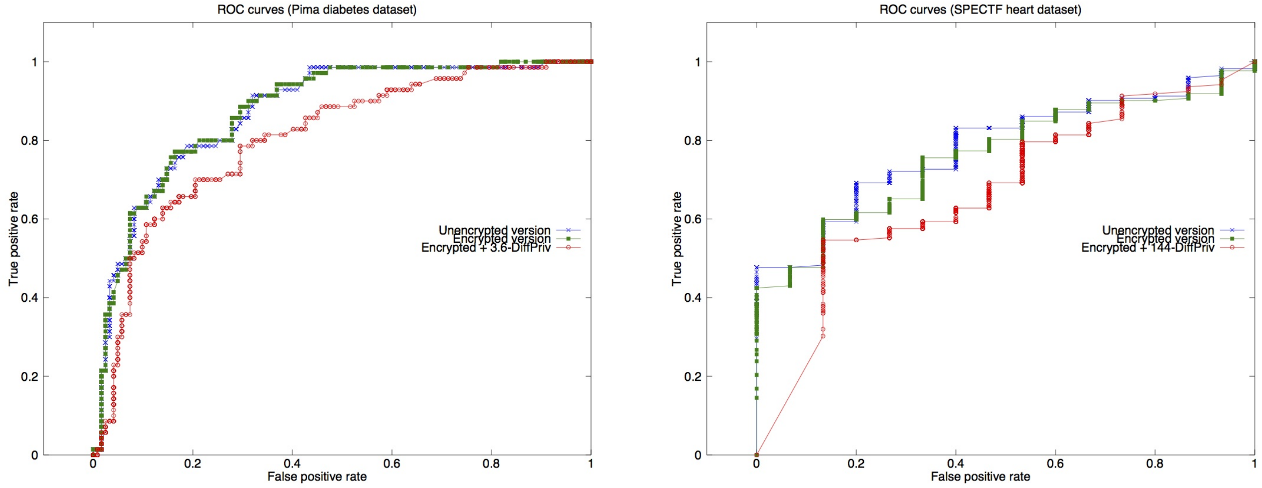

Fig. 6. ROC curves of unencrypted/encrypted/encrypted + differential-private logistic regression.

The Pima Indians diabetes dataset [1,21] consists of 768 instances and 8 features, or equivalently in our notation $(N_{\text{data}}, d) = (768, 8)$. Following [21], we split the dataset into two parts: a training set of 576 instances and a testing set of 192 instances. The model gaining by the training set is used in the testing set to compute the accuracy (when $\mathsf{thres} = 0.5$), the F-score (when $\mathsf{thres} = 0.5$), and the area under the curve (AUC) which are reported in Table 3. Note also that [21] reports a crossover of sensitivity (= TPR) and specificity (= $1 - \mathsf{FPR}$) at 0.76 when $\mathsf{thres} = 0.448$; our corresponding crossover over unencrypted data is 0.78 using the threshold for classification $\mathsf{thres} = 0.35$. The datasets in this and following sections are all normalized by their means and deviations.

For the logistic regression over unencrypted data, we implement the gradient descent algorithm (aka, steepest descent [19]) with initial model θ_{init} (Table 4 in Appendix A) and learning rate 0.1 with 200 steps of iterations. Also we choose $\lambda = 1$ in the cost function (1) to penalize big coefficients in the model. These choices are due to a few trials and errors, and are presented here since with the parameters we gain a little better accuracy (and F-score, AUC) than using the built-in `fminunc` function of Octave 3.2.4 (yielding about 79% accuracy). The gradient descent algorithm with our parameters outputs the minimizer θ^*_{unenc} (Table 4, Appendix A), reaching 80.2% accuracy and corresponding F-score, AUC as in Table 3.

For the logistic regression over encrypted data, after obtaining the coefficients of the approximate cost function, the client runs the gradient descent algorithm with the same parameters as above. Perhaps surprisingly, gradient descent with our approximate cost function at (6) performs a bit better than with the original cost function at (1), yielding 80.7% accuracy and relatively higher F-score, AUC correspondingly as in Table 3. Finally, the client obtains the minimizer θ^*_{enc} (Table 4, Appendix A).

To add differential privacy, we use distribution Lap(30) for Laplace noises at (13). This means

$$\frac{\Lambda_d}{\epsilon} = \frac{(d+1)(d+4)}{\epsilon} = 30$$

where $d = 8$ so that $\epsilon = 3.6$ in Table 3. At the end, the client obtains the minimizer of the perturbed cost function $\theta^*_{\text{enc+dp}}$ (Table 4, Appendix A). Table 3 confirms the intuition that differential privacy may decrease the performance (accuracy, F-score, AUC) of the classifier.

6.2 Using the SPECTF heart dataset

The SPECTF heart dataset [1,13] consists of 267 instances and 44 features, or equivalently in our notation $(N_{\text{data}}, d) = (267, 44)$. The dataset has already existed in two parts: a training set of 80 instances and a testing set of 187 instances. The paper [13] reported a range of accuracy from 72% to 84% using a dedicated algorithm for Single Proton Emission Computed Tomography (SPECT) images called CLIP3.

Using gradient descent with the cost functions at (1) and (8), we obtain the accuracies of 79.1% (unencrypted), 73.7% (encrypted), 72.1% (encrypted + differential private) with corresponding F-scores, AUCs reported in Table 3. The ROC curves are in Figure 6. As the dimension d is relatively big, the parameter of differential privacy ϵ becomes quite big as well. Specifically, we use distribution Lap(15) for Laplace noises at (13). This means

$$\frac{\Lambda_d}{\epsilon} = \frac{(d+1)(d+4)}{\epsilon} = 15$$

where $d = 44$ so that $\epsilon = 144$ as depicted. The initial θ and final outputs are in Table 5 (Appendix A), all of which are produced via gradient descent with learning rate 0.012, regulation $\lambda = 1$, and 450 steps of iterations.

7 Conclusion

We show that secure logistic regression is efficiently possible at scale. This is a step towards helping to protect sensitive data while accelerating research which requires large datasets, collected without borders. As future works, we plan to test our system with more datasets, and deploy them into cloud servers instead of our current commodity one.

References

1. UCI Machine Learning Repository, `http://archive.ics.uci.edu/ml`.
2. Y. Aono, X. Boyen, L. T. Phong, and L. Wang. Key-private proxy re-encryption under LWE. In G. Paul and S. Vaudenay, editors, *INDOCRYPT*, volume 8250 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.
3. Y. Aono, T. Hayashi, L. T. Phong, and L. Wang. Fast and secure linear regression and biometric authentication with security update. Cryptology ePrint Archive, Report 2015/692, 2015. `http://eprint.iacr.org/`.
4. J. W. Bos, K. E. Lauter, and M. Naehrig. Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics*, 50:234–243, 2014.
5. R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015*. The Internet Society, 2015.
6. W. Du, S. Chen, and Y. S. Han. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *In Proceedings of the 4th SIAM International Conference on Data Mining*, pages 222–233, 2004.
7. D. A. duVerle, S. Kawasaki, Y. Yamada, J. Sakuma, and K. Tsuda. Privacy-preserving statistical analysis by exact logistic regression. In *2015 IEEE Symposium on Security and Privacy Workshops, SPW 2015, San Jose, CA, USA, May 21-22, 2015*, pages 7–16. IEEE Computer Society, 2015.
8. C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.
9. C. Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.
10. T. Graepel, K. E. Lauter, and M. Naehrig. ML confidential: Machine learning on encrypted data. In T. Kwon, M. Lee, and D. Kwon, editors, *Information Security and Cryptology - ICISC 2012*, volume 7839 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2012.
11. W. Jiang, P. Li, S. Wang, Y. Wu, M. Xue, L. Ohno-Machado, and X. Jiang. WebGLORE: a web service for grid logistic regression. *Bioinformatics*, 29(24):3238–3240, 2013.
12. A. Khedr, P. G. Gulak, and V. Vaikuntanathan. SHIELD: scalable homomorphic implementation of encrypted data-classifiers. *IACR Cryptology ePrint Archive*, 2014:838, 2014.
13. L. Kurgan, K. Cios, R. Tadeusiewicz, M. Ogiela, and L. Goodenday. Knowledge discovery approach to automated cardiac spect diagnosis. *Artificial Intelligence in Medicine*, 23:2:149–169, 2001.
14. K. E. Lauter, A. López-Alt, and M. Naehrig. Private computation on encrypted genomic data. In D. F. Aranha and A. Menezes, editors, *Progress in Cryptology - LATINCRYPT 2014*, volume 8895 of *Lecture Notes in Computer Science*, pages 3–27. Springer, 2014.
15. R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In A. Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.
16. M. Liu and P. Q. Nguyen. Solving BDD by enumeration: An update. In E. Dawson, editor, *CT-RSA*, volume 7779 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2013.
17. M. Naehrig, K. E. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In C. Cachin and T. Ristenpart, editors, *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, pages 113–124. ACM, 2011.
18. Y. Nardi, S. E. Fienberg, and R. J. Hall. Achieving both valid and secure logistic regression analysis on aggregated data from different private sources. *Journal of Privacy and Confidentiality*, 4:1:189–220, 2012.
19. J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, New York, 2nd edition, 2006.
20. R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.
21. J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, pages 261–265. American Medical Informatics Association, 1988.

22. S. Wang, X. Jiang, Y. Wu, L. Cui, S. Cheng, and L. Ohno-Machado. Expectation propagation logistic regression (EXPLORER): distributed privacy-preserving online model learning. *Journal of Biomedical Informatics*, 46(3):480–496, 2013.
23. Y. Wu, X. Jiang, S. Wang, W. Jiang, P. Li, and L. Ohno-Machado. Grid multi-category response logistic models. *BMC Med. Inf. & Decision Making*, 15:10, 2015.
24. J. Zhang, R. Jin, Y. Yang, and A. G. Hauptmann. Modified logistic regression: An approximation to SVM and its applications in large-scale text categorization. In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 888–895, 2003.
25. J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett. Functional mechanism: Regression analysis under differential privacy. *PVLDB*, 5(11):1364–1375, 2012.
26. X. Zhu, H. Li, and F. Li. Privacy-preserving logistic regression outsourcing in cloud computing. *IJGUC*, 4(2/3):144–150, 2013.

A Initial and trained models in our system using reals datasets

The following tables contain the initial and trained models for Section 6.

Table 4. The initial θ and outputs of our system using the Pima diabetes dataset.

θ_{init}	= [0.334781, −0.633628, 0.225721, −0.648192, 0.406207, 0.044424, −0.426648, 0.877499, −0.426819]
θ^*_{unenc}	= [−0.802939, 0.354881, 0.932210, −0.192500, 0.051789, −0.103428, 0.613109, 0.337208, 0.141407]
θ^*_{enc}	= [−0.618931, 0.272079, 0.687556, −0.164313, 0.023873, −0.078103, 0.426285, 0.215544, 0.085846]
$\theta^*_{\text{enc+dp}}$	= [−0.938223, 0.467118, 0.570093, 0.014058, −0.361076, −0.215517, 0.603078, 0.295487, −0.207777]

Table 5. The initial θ and outputs of our system using the SPECTF heart dataset.

$\theta_{\text{init}} =$ [0.921455, −0.377080, −0.313317, 0.796285, 0.992807, −0.650099, 0.865773, 0.484040, 0.021763, 0.809766, 0.222401, 0.309993, 0.375320, 0.674654, −0.961690, −0.950472, −0.753475, −0.353844, 0.717381, −0.319103, −0.664294, −0.573008, −0.401116, 0.216010, −0.810675, 0.961971, −0.412459, −0.507446, 0.585540, −0.273261, 0.899775, −0.611130, −0.223748, 0.008219, −0.758307, 0.907636, −0.547704, −0.464145, 0.677729, 0.426712, −0.862759, 0.090766, −0.421597, −0.429986, 0.410418]
$\theta^*_{\text{unenc}} =$ [0.809215, −0.140885, −0.606209, 0.203335, 0.203389, −0.531782, 0.575154, 0.064924, −0.366572, 0.835623, −0.159378, 0.043608, 0.011024, 0.613679, −0.893973, −0.742481, −0.690140, −0.333246, 0.604501, −0.054810, −0.624138, −0.443354, −0.540109, 0.172282, −0.722847, 0.703295, −0.626644, −0.508781, 0.092141, −0.585776, 0.137703, −0.685467, −0.392665, −0.072641, −0.585242, 1.029491, −0.491748, −0.274508, 0.484444, 0.171330, −1.250592, −0.016082, −0.445400, −0.551420, 0.339719]
$\theta^*_{\text{enc}} =$ [0.449506, −0.179168, −0.561430, 0.184955, 0.187654, −0.609835, 0.585331, −0.016184, −0.331420, 0.963836, 0.026561, 0.026819, 0.055403, 0.749877, −0.726896, −0.593111, −0.482201, −0.265006, 0.715793, −0.028347, −0.514324, −0.488422, −0.433774, 0.243350, −0.626253, 0.750072, −0.525558, −0.512443, 0.119176, −0.595018, 0.130557, −0.540884, −0.226714, −0.004119, −0.451977, 1.024436, −0.445246, −0.194982, 0.608593, 0.329321, −1.123225, −0.036603, −0.394657, −0.485166, 0.421146]
$\theta^*_{\text{enc+dp}} =$ [0.541275, −1.109162, −2.202257, 0.068316, 1.813504, −0.894523, 0.724268, 0.473433, −0.308237, 0.822734, 0.367747, 0.325026, −1.615040, 0.713774, −1.156806, −2.089627, 0.715980, 1.178304, 1.528983, −0.064491, −1.101812, −2.231109, −0.979761, 0.420572, −0.987508, 2.029521, −1.792476, −0.400331, 0.082781, −1.527067, 2.203595, −1.116681, 0.311569, 0.731576, −0.404809, 0.526873, −0.166109, 0.550493, 0.675732, −0.416966, −1.672484, 0.904692, −0.071968, −1.876075, −0.319403]