Semi-parallel Logistic Regression for GWAS on Encrypted Data

Miran Kim[1], Yongsoo Song[2], Baiyu Li[3], and Daniele Micciancio[3]

[1] University of Texas, Health Science Center at Houston, Houston, USA
`Miran.Kim@uth.tmc.edu`
[2] Microsoft Research, Redmond, USA
`Yongsoo.Song@microsoft.com`
[3] University of California, San Diego, USA
`{baiyu,daniele}@cs.ucsd.edu`

Abstract. The sharing of biomedical data is crucial to enable scientific discoveries across institutions and improve health care. For example, genome-wide association studies (GWAS) based on a large number of samples can identify disease-causing genetic variants. The privacy concern, however, has become a major hurdle for data management and utilization. Homomorphic encryption is one of the most powerful cryptographic primitives which can address the privacy and security issues. It supports the computation on encrypted data, so that we can aggregate data and perform an arbitrary computation on an untrusted cloud environment without the leakage of sensitive information.
This paper presents a secure outsourcing solution to assess logistic regression models for quantitative traits to test their associations with genotypes. We adapt the semi-parallel training method by Sikorska et al., which builds a logistic regression model for covariates, followed by one-step parallelizable regressions on all individual single nucleotide polymorphisms (SNPs). In addition, we modify our underlying approximate homomorphic encryption scheme for performance improvement.
We evaluated the performance of our solution through experiments on real-world dataset. It achieves the best performance of homomorphic encryption system for GWAS analysis in terms of both complexity and accuracy. For example, given a dataset consisting of 245 samples, each of which has 10643 SNPs and 3 covariates, our algorithm takes about 43 seconds to perform logistic regression based genome wide association analysis over encryption. We demonstrate the feasibility and scalability of our solution.

Keywords: Homomorphic encryption · Genome-wide association studies · Logistic regression.

1 Introduction

Since National Institutes of Health (NIH) released the Gemonic Data Sharing policy allowing the use of cloud computing services for storage and analysis of controlled-access data [1], we are getting more challenge to ensure security and privacy of data in cloud computing systems. In the United States, the Health Insurance Portability and Accountability Act regulates medical care data sharing [28]. A community effort has been made to protect the privacy of genomic data, for example, iDASH (integrating Data for Analysis, Anonymization, Sharing) has hosted secure genome analysis competition for the past 5 years. This contest has encouraged cryptography experts to develop practical yet rigorous solutions for privacy preserving genomic data analysis. As a result, we could demonstrate the feasibility of secure genome data analysis using various cryptographic primitives such as homomorphic encryption (HE), differential privacy, multi-party computation, and software guard extension. In particular, HE has emerged as one of the promising solutions for secure outsourced computation over genomic data in practical biomedical applications [12, 20, 21, 4].

In this work, we provide a solution for the second track of iDASH 2018 competition, which aims to develop a method for outsourcing computation of Genome Wide Association Studies (GWAS) on homomorphically encrypted data. We propose a practical protocol to assess logistic regression model to compute p-values of different single nucleotide polymorphisms (SNPs). We investigate the association of genotypes and phenotypes by adjusting the models on the basis of covariates. The results will be used for identifying genetic variants that are statistically correlated with phenotypes of interest.

In iDASH 2017 competition, participants of the third task were challenged to train a single logistic regression model on encrypted data. Although significant performance improvements over existing solutions have been demonstrated [19, 8], it is still computationally intensive to perform logistic regression based GWAS. A straightforward implementation would require building one model for each SNP, incurring a high performance overhead of secure computation. This motivates the use of the semi-parallel algorithm, which was previously discussed in [26, 27]. Following the approach, our algorithm proceeds in two steps over encrypted data: (1) construct a logistic regression model by applying the gradient descent method of [19] while taking only the covariates into account, (2) compute the regression parameters of logistic regression corresponding to SNPs with one additional update of Newtons method. The model in the first step can be computed very efficiently and can be used for all SNPs in the subsequent step. In the second step, we apply various techniques to enable computing the logistic regression updates for all SNPs in many parallel sub-steps. This approach enables us to obtain logistic regression based models for thousands of SNPs all in one.

Our solution is based on a homomorphic scheme by Cheon et al. [9] with support for approximate fixed-point arithmetic over the real numbers. Recently, a significant performance improvement was made in [8] based on the Residue Number System (RNS). The authors modified homomorphic operations so that they do not require any expensive RNS conversions. In this paper, we propose another RNS variant of approximate HE scheme which has some advantages for this task. Specifically, we adapt a different key-switching method which is a core operation in homomorphic multiplication or permutation. The earlier studies [9, 8] were based on the key-switching technique of [16] which introduces a special modulus. A special modulus had approximately the same bit-size as a ciphertext modulus to reduce the noise of key-switching procedure, but we observed that it is not the best option when the depth of an HE scheme is small. Instead, we combine the special modulus technique with RNS-friendly decomposition method [3]. As a result, we could minimize the parameter and thereby improve the performance while guaranteeing the same security level. We further leverage efficient packing techniques and parallelization approaches to reduce the storage requirement and running time.

Related Works. There are a number of recent research articles on HE-based machine learning applications. Kim et al. presented the first secure outsourcing method to train a logistic regression model on encrypted data [22] and the follow-up showed remarkably good performance with real data [19, 8]. For example, the training of a logistic regression model took about 3.6 minutes on encrypted data consisting of 1579 samples and 18 features. A slightly different approach is taken in [7], where the authors use Gentry's bootstrapping technique in fully homomorphic encryption, so that their solution can run for an arbitrary number of iterations of gradient descent algorithm.

2 Background

The binary logarithm will be simply denoted by $\log(\cdot)$. We denote vectors in bold, e.g. $\mathbf{a}$, and matrices in upper-case bold, e.g. $\mathbf{A}$. For an $n \times m$ matrix $\mathbf{A}$, we use $\mathbf{A}_i$ to denote the i-th row of $\mathbf{A}$, and $\mathbf{a}_j$ the j-th column of $\mathbf{A}$. For a $d_1 \times d$ matrix $\mathbf{A}_1$ and a $d_2 \times d$ matrix $\mathbf{A}_2$, $(\mathbf{A}_1; \mathbf{A}_2)$ denotes the $(d_1 + d_2) \times d$ matrix obtained by concatenating two matrices in a vertical direction. If two matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ have the same number of rows, $(\mathbf{A}_1 | \mathbf{A}_2)$ denotes a matrix formed by horizontal concatenation. We let λ denote the security parameter throughout the paper: all known valid attacks against the cryptographic scheme under scope should take $\Omega(2^\lambda)$ bit operations.

2.1 Logistic Regression

Logistic regression is a widely used statistical model when the response variable is categorical with two possible outcomes [13]. In particular, it is very popular in biomedical informatics research and serve as the foundation of many risk calculators [29, 15, 18].

Let the observed phenotype be given as a vector $\mathbf{y} \in \{\pm 1\}^n$ of length n, the states of p many SNPs as the $n \times p$ matrix $\mathbf{S}$, and the states of k many covariates as the $n \times k$ matrix $\mathbf{X}$. Suppose that an intercept is included in the matrix of covariates, that is, $\mathbf{X}$ contains a column of ones. For convenience, let $\mathbf{u}_i = (\mathbf{X}_i, s_{ij}) \in \mathbb{R}^{k+1}$ for $i = 1, \ldots, n$. For each $j \in [p]$, logistic regression aims to find an optimal vector $\boldsymbol{\beta} \in \mathbb{R}^{k+1}$ which maximizes the likelihood estimator

$$\prod_{i=1}^{n} \Pr[y_i | \mathbf{u}_i] = \prod_{i=1}^{n} \sigma(-y_i \cdot \mathbf{u}_i^T \boldsymbol{\beta}),$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid function, or equivalently minimizes the loss function, defined as the negative log-likelihood:

$$L(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i \cdot \mathbf{u}_i^T \boldsymbol{\beta})).$$

Note that $\boldsymbol{\beta} = (\boldsymbol{\beta}_{\mathbf{X}} | \beta_j)$ depends on the index j, and we are particularly interested in the last component β_j that corresponds to the j-th SNP.

There is no closed form formula for the regression coefficients that minimizes the loss function. Instead, we employ an iterative process: we begin with some initial guess for the parameters and then repeatedly update them to make the loss smaller until the process converges. Specifically, the gradient descent (GD) takes a step in the direction of the steepest decrease of L. The method of GD can face a problem of zig-zagging along a local optima and this behavior of the method becomes typical if it increases the number of variables of an objective function. We can employ Nesterov's accelerated gradient [23] to address this phenomenon, which uses moving average on the update vector and evaluates the gradient at this looked-ahead position.

2.2 Newton's Method

We can alternatively use Newton algorithm to estimate parameters [24]. It can be achieved by calculating the first and the second derivatives of the loss function, followed by the update:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - (\nabla_{\boldsymbol{\beta}}^2 L(\boldsymbol{\beta}))^{-1} \cdot \nabla_{\boldsymbol{\beta}} L(\boldsymbol{\beta}).$$

Let $p_i = \sigma(\mathbf{u}_i^T \boldsymbol{\beta})$ for $i \in [n]$; then p_i represents the probability of success for each sample. We see that

$$\begin{aligned} \nabla_{\boldsymbol{\beta}} L(\boldsymbol{\beta}) &= \mathbf{U}^T(\mathbf{y} - \mathbf{p}), \\ \nabla_{\boldsymbol{\beta}}^2 L(\boldsymbol{\beta}) &= -\mathbf{U}^T \mathbf{W} \mathbf{U}, \end{aligned} \tag{1}$$

where $\mathbf{U}$ is an $n \times (k+1)$ regressor matrix whose i-th row contains the variables $\mathbf{u}_i$, $\mathbf{p} = (p_i)_{i=1}^n$ is a column vector of the estimated probabilities p_i, and $\mathbf{W}$ is a diagonal weighting matrix with elements $w_i = p_i(1 - \ p_i)$. Then the above update formula can be rewritten as

$$\begin{aligned} \boldsymbol{\beta} &\leftarrow (\mathbf{U}^T \mathbf{W} \mathbf{U})^{-1} \cdot \mathbf{U}^T \mathbf{W} \cdot (\mathbf{U}\boldsymbol{\beta} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})) \\ &= (\mathbf{U}^T \mathbf{W} \mathbf{U})^{-1} \cdot \mathbf{U}^T \mathbf{W} \mathbf{z} \end{aligned} \tag{2}$$

where $\mathbf{z} = \mathbf{U}\boldsymbol{\beta} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$. Here, the vector $\mathbf{z}$ is known as the *working response*. This method is also called *Iteratively Reweighted Least Squares*. More details can be found in [24]. On the other hand, the Fisher information $\mathbf{U}^T \mathbf{W} \mathbf{U}$ can be partitioned into a block form:

$$\begin{array}{cc} \begin{array}{r} k\left\{\vphantom{\begin{matrix}a\\a\end{matrix}}\right. \\ 1\left\{\vphantom{a}\right. \end{array} & \left[\begin{array}{c:c} \mathbf{A} & \mathbf{b} \\ \hdashline \mathbf{b}^T & c \end{array}\right] \\ & \begin{array}{cc} \underbrace{\hphantom{\mathbf{AAAA}}}_{k} & \underbrace{\hphantom{\mathbf{b}}}_{1} \end{array} \end{array}$$

where $\mathbf{A} = \mathbf{X}^T\mathbf{W}\mathbf{X}$, $\mathbf{s}_j = (s_{ij})_{i=1}^n$ is a column vector of all samples of the j-th SNP, $\mathbf{b} = \mathbf{X}^T\mathbf{W}\mathbf{s}_j$, and $c = \mathbf{s}_j^T\mathbf{W}\mathbf{s}_j$. Then the inverse of $\mathbf{U}^T\mathbf{W}\mathbf{U}$ is

$$\left[\begin{array}{c:c} * & * \\ \hdashline -\frac{1}{t}\mathbf{b}^T\mathbf{A}^{-1} & \frac{1}{t} \end{array}\right]$$

where $t = c - \mathbf{b}^T\mathbf{A}^{-1}\mathbf{b}$. Therefore, the estimated SNP effect β_j and the variance for the estimation are computed by

$$\begin{aligned} \beta_j &= -\frac{1}{t}\cdot(\mathbf{b}^T\mathbf{A}^{-1})\cdot(\mathbf{X}^T\mathbf{W}\mathbf{z}) + \frac{1}{t}\cdot(\mathbf{s}_j^T\mathbf{W}\mathbf{z}) \\ &= \frac{|\mathbf{A}|\cdot\mathbf{s}_j^T\mathbf{W}\mathbf{z} - \mathbf{b}^T\cdot\mathtt{adj}(\mathbf{A})\cdot(\mathbf{X}^T\mathbf{W}\mathbf{z})}{|\mathbf{A}|\cdot c - \mathbf{b}^T\cdot\mathtt{adj}(\mathbf{A})\cdot\mathbf{b}}, \quad (3) \\ \mathtt{var}_j &= \frac{1}{c - \mathbf{b}^T\cdot\mathbf{A}^{-1}\cdot\mathbf{b}}, \quad (4) \end{aligned}$$

where $\mathtt{adj}(\mathbf{A})$ denotes the adjugate matrix and $|\mathbf{A}|$ the determinant of $\mathbf{A}$.

3 Full RNS Variant of HEAAN, Revisited

In this paper, we apply the full RNS variant of the CKKS scheme [9], called RNS-CKKS [8], for efficient arithmetic over the real numbers. In addition, we modify some algorithms to meet our goals.

The previous RNS-CKKS scheme uses some approximate modulus switching algorithms for the key-switching procedure. The evaluation key should have a much larger modulus compared to encrypted data due to multiplicative noise. In this work, we developed and implemented a new key-switching algorithm which provides a trade-off between complexity and parameter. Our new key-switching process requires more Number Theoretic Transformation (NTT) conversions, but the HE parameters such as the ring dimension N can be reduced while keeping the same security level. In particular, our method is more efficient than the previous one when the depth of a circuit to be evaluated is small.

The following is a simple description of RNS-CKKS based on the ring learning with errors (RLWE) problem. Let $R = \mathbb{Z}[X]/(X^N + 1)$ be a cyclotomic ring for a power-of-two integer N. An ordinary ciphertext of RNS-CKKS can be represented as a linear polynomial $c(Y) = c_0 + c_1 \cdot Y$ over the ring R_Q where Q denotes the ciphertext modulus and $R_Q = R \pmod{Q}$ is the residue ring modulo Q.

• $\underline{\mathtt{Setup}(q, L, \eta; 1^\lambda)}$. Given a base integer module q, a maximum level L of computation, a bit precision η, and a security parameter λ, the Setup algorithm generates the following parameters:

- Choose a basis $\mathcal{D} = \{p_0, q_0, q_1, \ldots, q_L\}$ such that $q_i/q \in (1 - 2^{-\eta}, 1 + 2^{-\eta})$ for $1 \le i \le L$. We write $Q_\ell = \prod_{i=0}^{\ell} q_i$ for $0 \le \ell \le L$.
- Choose a power-of-two integer N.
- Choose a secret key distribution χ_{key}, an encryption key distribution χ_{enc}, and an error distribution χ_{err} over R.

We always use the RNS form with respect to the basis $\{p_0, q_0, \ldots, q_\ell\}$ (or its sub-basis) to represent polynomials in our scheme. For example, an element $a(X)$ of R_{Q_ℓ} is identified with the tuple $(a_0, a_1, \ldots, a_\ell) \in \prod_{i=0}^{\ell} R_{q_i}$ where $a_i = a \pmod{q_i}$. We point out that all algorithms in our scheme are RNS-friendly, so that we do not have to perform any RNS conversions.

The main difference of our scheme from previous work [8] is that the key-switching procedure is based on both the decomposition and modulus raising techniques. The use of decomposition allows us to use a smaller parameter, but its complexity may be increased when the level of HE scheme is large. However, we realize that the GWAS analysis does not require a huge depth, so this new key-switching technique is beneficial to obtain a better performance in this specific application. The generation of switching key and key-switching algorithms are described as follows.

• $\underline{\texttt{KSGen}(s_1, s_2)}$. Given two secret polynomials $s_1, s_2 \in R$, sample $\tilde{a}_i(X) \leftarrow U(R_{p_0 \cdot Q_L})$ and errors $\tilde{e}_i \leftarrow \chi_{err}$ for $0 \le i \le L$. Output the switching key

$$\mathsf{swk} = \{\mathsf{swk}_i = (\tilde{b}_i, \tilde{a}_i)\}_{0 \le i \le L} \in \left(R^2_{p_0 Q_L}\right)^{L+1}$$

where $\tilde{b}_i = -\tilde{a}_i \cdot s_2 + \tilde{e}_i + p_0 B_i \cdot s_1 \pmod{p_0 \cdot Q_L}$ for the integer $B_i \in \mathbb{Z}_{Q_L}$ such that $B_i = 1 \pmod{q_i}$ and $B_i = 0 \pmod{q_j}$ for all $j \neq i$.

• $\underline{\texttt{KeySwitch}_{\mathsf{swk}}(\mathsf{ct})}$. For $\mathsf{ct} = (c_0, c_1) \in R^2_{Q_\ell}$, let $c_{1,i} = c_1 \pmod{q_i}$ for $0 \le i \le \ell$. We first compute $\tilde{\mathsf{ct}} = \sum_{i=0}^{\ell} c_{1,i} \cdot \mathsf{swk}_i \pmod{p_0 Q_\ell}$, and then return the ciphertext $\mathsf{ct}' = (c_0, 0) + \lfloor p_0^{-1} \cdot \tilde{\mathsf{ct}} \rceil \pmod{Q_\ell}$.

The idea of key-switching procedure is used to relinearize a ciphertext in homomorphic multiplication algorithm below. All other algorithms including key generation, encryption and decryption are exactly same as the previous RNS-based scheme.

• $\underline{\texttt{KeyGen}(1^\lambda)}$.

- Sample $s \leftarrow \chi_{key}$ and set the secret key as $\mathsf{sk} = (1, s)$.
- Sample $a \leftarrow U(R_{Q_L})$ and $e \leftarrow \chi_{err}$. Set the public key pk as $\mathsf{pk} = (b, a) \in R^2_{Q_L}$ where $b = -a \cdot s + e \pmod{Q_L}$.
- Set the evaluation key as $\mathsf{evk} \leftarrow \texttt{KSGen}(s^2, s)$.

• $\underline{\texttt{Enc}_{\mathsf{pk}}(m)}$. Given $m \in R$, sample $v \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$. Output the ciphertext $\mathsf{ct} = v \cdot \mathsf{pk} + (m + e_0, e_1) \pmod{Q_L}$.

• $\underline{\texttt{Dec}_{\mathsf{sk}}(\mathsf{ct})}$. Given ciphertext $\mathsf{ct} = (\mathsf{ct}_j)_{0 \le j \le \ell} \in R^2_{Q_\ell}$, output $\langle \mathsf{ct}_0, \mathsf{sk} \rangle \pmod{q_0}$.

• $\underline{\texttt{Add}(\mathsf{ct}, \mathsf{ct}')}$. Given two ciphertexts $\mathsf{ct}, \mathsf{ct}' \in R^2_{Q_\ell}$, output the ciphertext $\mathsf{ct}_{\mathsf{add}} = \mathsf{ct} + \mathsf{ct}' \pmod{Q_\ell}$.

• $\underline{\texttt{Mult}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{ct}')}$. For two ciphertexts $\mathsf{ct} = (c_0, c_1)$ and $\mathsf{ct}' = (c'_0, c'_1)$, compute $d_0 = c_0 c'_0, d_1 = c_0 c'_1 + c'_0 c_1, d_2 = c_1 c'_1 \pmod{Q_\ell}$. Let $c_{2,i} = d_2 \pmod{q_i}$ for $0 \le i \le \ell$, and compute $\tilde{\mathsf{ct}} = \sum_{i=0}^{\ell} c_{2,i} \cdot \mathsf{evk}_i \pmod{p_0 Q_\ell}$. Output the ciphertext $\mathsf{ct}' = (c_0, c_1) + \lfloor p_0^{-1} \cdot \tilde{\mathsf{ct}} \rceil \pmod{Q_\ell}$.

Finally, RNS-CKKS provides the rescaling operation to round messages over encryption, thereby enabling to control the magnitude of messages during computation.

• $\underline{\texttt{ReScale}(\mathsf{ct})}$. For given $\mathsf{ct} \in \mathcal{R}^2_{Q_\ell}$, return the ciphertext $\mathsf{ct}' = \lfloor q_\ell^{-1} \cdot \mathsf{ct} \rceil \pmod{Q_{\ell-1}}$.

It is a common practice to rescale the encrypted message after each multiplication as we round-off the significant digits after multiplication in plain fixed/floating point computation. In the next section, we assume that the rescaling procedure is included in homomorphic multiplications for simpler description, but a rigorous analysis about level consumption will be provided later in the parameter setting section.

As in the original CKKS scheme, the native plaintext space can be understood as an $N/2$-dimensional complex vector space (each vector component is called a *plaintext slot*). Addition and multiplication in R correspond to component-wise addition and multiplication on plaintext slots. Furthermore, it provides an operation that shifts the plaintext vector over encryption. For a ciphertext ct encrypting a plaintext vector $(m_1, \ldots, m_\ell) \in \mathbb{R}^\ell$, we could obtain an encryption of a shifted vector $(m_{r+1}, \ldots, m_\ell, m_1, \ldots, m_r)$. Let us denote such operation by $\mathsf{Rot}(\mathsf{ct}; r)$. For more detail, we refer the reader to [8]. In the rest of this paper, we let $N_2 = N/2$ and denote by $\mathtt{E}(\cdot)$ the encryption function for convenience.

4 Our Method

4.1 Database Encoding

As noted before, the learning data are recorded into an $n \times k$ matrix $\mathbf{X}$ of covariates, an $n \times p$ binary matrix $\mathbf{S} = (s_{ij})$ of all the SNP data, and an n-dimensional binary column vector $\mathbf{y}$ of the dependent variable. In large-scale GWAS, the number of parameters of SNPs, p can be in the thousands, so we split the SNP data into several N_2-dimensional vectors, encrypt them, and send the resulting ciphertexts to the server. For simplicity, we assume in the following discussion that each row of $\mathbf{S}$ is encrypted into a single ciphertext. More specifically, for $1 \leq i \leq n$ and for $1 \leq \ell \leq k$, we encrypt

$$\mathtt{E}(x_{i\ell}\mathbf{S}_i) = \mathtt{E}(x_{i\ell}s_{i1}, \dots, x_{i\ell}s_{ip}).$$

As mentioned before, we add a column of ones to $\mathbf{X}$ to allow for an intercept in the regression; that is, we assume $x_{i1} = 1$ for all $1 \leq i \leq n$. So, when $\ell = 1$, the ciphertext $\mathtt{E}(x_{i1}\mathbf{S}_i)$ encrypts exactly the i-th SNP sample.

Next, consider the matrix $\mathbf{y}^T\mathbf{X} \in \mathbb{R}^{n\times k}$ defined as

$$\begin{aligned}\mathbf{y}^T\mathbf{X} &= \left[y_1\mathbf{X}_1; \cdots ; y_n\mathbf{X}_n\right] \\ &= \begin{bmatrix} y_1x_{11} & y_1x_{12} & \cdots & y_1x_{1k} \\ y_2x_{21} & y_2x_{22} & \cdots & y_2x_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ y_nx_{n1} & y_nx_{n2} & \cdots & y_nx_{nk} \end{bmatrix}.\end{aligned}$$

For simplicity, we assume that n and k are power-of-two integers satisfying $\log n + \log k \leq \log(N_2)$. Kim et al. [19] suggested an efficient encoding map to encode the whole matrix $\mathbf{y}^T\mathbf{X}$ in a single ciphertext in a row-by-row manner. Specifically, we will identify this matrix with a vector in $\mathbb{R}^{n\cdot k}$, that is,

$$\begin{aligned}\mathbf{y}^T\mathbf{X} &\mapsto (y_1\mathbf{X}_1|\cdots|y_n\mathbf{X}_n) \\ &= (y_1x_{11}, \dots, y_1x_{1k}, \dots, y_nx_{n1}, \dots, y_nx_{nk}).\end{aligned}$$

Similarly, we identify the matrix $\mathbf{X}$ with a vector in $\mathbb{R}^{n\cdot k}$ as follows:

$$\begin{aligned}\mathbf{X} &\mapsto (\mathbf{X}_1|\cdots|\mathbf{X}_n) \\ &= (x_{11}, \dots, x_{1k}, \dots, x_{n1}, \dots, x_{nk}).\end{aligned}$$

For an efficient implementation, we can make $N_2/(k\cdot n)$ copies of each component of $\mathbf{y}^T\mathbf{X}$ and $\mathbf{X}$ to encode them into fully packed plaintext slots. For example, we can generate the encryption of $\mathbf{y}^T\mathbf{X}$ as

$$\mathtt{E}(\mathbf{y}^T\mathbf{X}) = \mathtt{E}\left(y_1\mathbf{X}_1^{(N_2/(k\cdot n))}|\cdots|y_n\mathbf{X}_n^{(N_2/(k\cdot n))}\right),$$

where $y_i\mathbf{X}_i^{(N_2/(k\cdot n))}$ denotes an array containing $N_2/(k\cdot n)$ copies of $y_i\mathbf{X}_i$. In the case of the target vector $\mathbf{y}$, we make N_2/n copies of each entry, so that the encoding aligns y_i with each copies of $y_i\mathbf{X}_i$ and $\mathbf{X}_i$ in the ciphertexts. Let us denote the generated ciphertext by $\mathtt{E}(\mathbf{y})$.

Finally, we now consider how to encrypt the covariance matrix $\mathbf{X}^T\mathbf{X}$ which can be used for computing the adjugate matrix and determinant of $\mathbf{A} = \mathbf{X}^T\mathbf{W}\mathbf{X}$. The adjugate $\mathtt{adj}(\mathbf{A})$ is a $k\times k$ matrix whose entries are defined as $\mathtt{adj}(\mathbf{A})_{j\ell} := (-1)^{j+\ell}\cdot|\hat{\mathbf{A}}_{\ell j}|$ for $1 \leq j, \ell \leq k$, where $|\hat{\mathbf{A}}_{\ell j}|$ is the determinant of $\hat{\mathbf{A}}_{\ell j}$. Here, $\hat{\mathbf{A}}_{\ell j}$ is a $(k-\ 1)\times(k-1)$ sub-matrix obtained by removing the j-th column and ℓ-th row from $\mathbf{A}$. For example, when $k = 4$, the determinant $|\hat{\mathbf{A}}_{11}|$ is computed by $a_{22}(a_{33}a_{44} - a_{34}a_{43}) + a_{23}(a_{34}a_{42} - a_{32}a_{44}) + a_{24}(a_{32}a_{43} - a_{33}a_{42})$, which can be rewritten as a component-wise product of three vectors

$$\begin{aligned}\mathbf{A}_{1,1,1} &= (a_{22}, -a_{22}, a_{23}, -a_{23}, a_{24}, -a_{24}), \\ \mathbf{A}_{1,1,2} &= (a_{33}, -a_{34}, a_{34}, -a_{32}, a_{32}, -a_{33}), \\ \mathbf{A}_{1,1,3} &= (a_{44}, -a_{43}, a_{42}, -a_{44}, a_{43}, -a_{42}).\end{aligned}$$

In general, we can consider $(k-1)!$-dimensional vectors $\mathbf{A}_{j,\ell,1}, \mathbf{A}_{j,\ell,2}, \dots, \mathbf{A}_{j,\ell,(k-1)}$ that can be used to compute $|\hat{\mathbf{A}}_{\ell j}|$. To do so, for each $i \in [n]$, we first pre-compute the i-th covariance matrix $\mathbf{X}_i^T\mathbf{X}_i \in \mathbb{R}^{k\times k}$

and generate the corresponding vector $(\mathbf{X}_i^T\mathbf{X}_i)_{j,\ell,t}$ for $1 \le j \le \ell \le k$ and $1 \le t \le k-1$. Suppose that $N_2 \ge n \cdot (k-1)!$. Let $\phi = N_2/(n \cdot (k-1)!)$, and we encrypt the following concatenated vector

$$\Sigma_{j,\ell,t} = \left((\mathbf{X}_1^T\mathbf{X}_1)_{j,\ell,t}^{(\phi)} \mid \ \ldots \ |(\mathbf{X}_n^T\mathbf{X}_n)_{j,\ell,t}^{(\phi)} \right).$$

We denote the resulting ciphertext by $\mathtt{E}(\Sigma_{j,\ell,t})$.

An alternative choice is to encrypt SNPs, covariates, and phenotype vectors in a separate way. The server can reconstruct the aforementioned encryptions by applying homomorphic operations, but it requires additional levels for the computation. So, we used the former encryption algorithm in the implementation, thereby saving on the depth and time in the evaluation. Our encoding system has another advantage, in that it can be applied to *horizontally partitioned* data where each party has a subset of the rows in dataset. In this case, each party encrypts their locally computed quantities on their data and sends them to the server. Then the server aggregates them to obtain encryptions of the shared data as the ones in our encryption method.

4.2 Homomorphic Evaluation of Logistic Regression

The main idea of the semi-parallel logistic regression analysis [26, 27] is to assume that the probabilities predicted by a model without SNP will not change much once SNP is included to the model. We will follow their approach, where the first step is to construct a logistic regression model taking only the covariates into account, and the second step is to compute the model coefficients of the logistic regression corresponding to the SNP in a semi-parallel way.

We start with a useful aggregation operation across plaintext slots from the literature [17, 10, 11]. This algorithm is referred as `AllSum`, which is parameterized by integers ψ and α. See Algorithm 1 for an implementation. Let $\ell = \psi \cdot \alpha$. Given a ciphertext ct representing a plaintext vector $\mathbf{m} = (m_1, \ldots, m_\ell) \in \mathbb{R}^\ell$, the `AllSum` algorithm outputs a ciphertext ct' encrypting

$$\begin{aligned}\mathbf{m}' = (&\sum_{j=0}^{\alpha-1} m_{\psi j+1}, \sum_{j=0}^{\alpha-1} m_{\psi j+2}, \ldots \sum_{j=0}^{\alpha-1} m_{\psi(j+1)},\\ &\sum_{j=0}^{\alpha-1} m_{\psi j+1}, \sum_{j=0}^{\alpha-1} m_{\psi j+2}, \ldots \sum_{j=0}^{\alpha-1} m_{\psi(j+1)}, \ldots),\end{aligned}$$

i.e., $m_i' = \sum_{j=0}^{\alpha-1} m_{\psi j+i}$ for $1 \le i \le \psi$, and $m_{\psi j+i}' = m_i'$ for $1 \le j \le \alpha - 1$. For example, when $\psi = 1$, it returns an encryption of the sum of the elements of $\mathbf{m}$.

As mentioned before, our algorithm consists of two steps to perform the semi-parallel logistic regression training while taking as input the following ciphertexts: $\{\mathtt{E}(x_{i\ell}\mathbf{S}_i)\}$, $\mathtt{E}(\mathbf{y}^T\mathbf{X})$, $\mathtt{E}(\mathbf{X})$, $\mathtt{E}(\mathbf{y})$, and $\{\mathtt{E}(\Sigma_{j,\ell,t})\}$, for $1 \le i \le n$, $1 \le j \le \ell \le k$, and $1 \le t \le k-1$.

Algorithm 1 `AllSum`$(\mathsf{ct}, \psi, \alpha)$

```
Input: ct, input ciphertext, the unit initial amount by which the ciphertext shifts $\psi$, the number of
    summands $\alpha$
1: for $i = 0, 1, \ldots, \log\alpha - 1$ do
2:     Compute $\mathsf{ct} \leftarrow \mathsf{Add}(\mathsf{ct}, \mathsf{Rot}(\mathsf{ct}; \psi \cdot 2^i))$
3: end for
4: return ct
```

4.2.1 Logistic Regression Model Training for Covariates

The best solution to train a logistic regression model from homomorphically encrypted dataset is to evaluate Nesterov's accelerated gradient descent method [19, 8]. We adapt their evaluation strategy to train a model for covariates.

Step 0: For simplicity, let $\mathbf{v}_i = y_i \mathbf{X}_i$ and $\ell = N_2/(k \cdot n)$. Since the input ciphertext $\mathtt{E}(\mathbf{y}^T\mathbf{X})$ represents ℓ copies of $\mathbf{v}_i$, Step 6 in [19] outputs the following ciphertext that encrypts the same number of copies of the vectors $\sigma(\mathbf{v}_i^T \boldsymbol{\beta}_{\mathbf{X}}) \cdot \mathbf{v}_i$:

$$\mathtt{ct}_6 = \mathtt{E}\begin{bmatrix} \sigma_3(\mathbf{v}_1^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{11} & \cdots & \sigma_3(\mathbf{v}_1^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{1k} \\ \vdots & \ddots & \vdots \\ \sigma_3(\mathbf{v}_1^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{11} & \cdots & \sigma_3(\mathbf{v}_1^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{1k} \\ \vdots & \ddots & \vdots \\ \sigma_3(\mathbf{v}_n^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{n1} & \cdots & \sigma_3(\mathbf{v}_n^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{nk} \\ \vdots & \ddots & \vdots \\ \sigma_3(\mathbf{v}_n^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{n1} & \cdots & \sigma_3(\mathbf{v}_n^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{nk} \end{bmatrix}.$$

Then Step 7 in [19] is changed from $\mathtt{AllSum}(\mathtt{ct}_6, k, n)$ into $\mathtt{ct}_7 = \mathtt{AllSum}(\mathtt{ct}_6, N_2/n, n)$, so that the output ciphertext is as follows:

$$\mathtt{ct}_7 = \mathtt{E}\begin{bmatrix} \sum_i \sigma_3(\mathbf{v}_i^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{i1} \cdots \sum_i \sigma_3(\mathbf{v}_i^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{ik} \\ \sum_i \sigma_3(\mathbf{v}_i^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{i1} \cdots \sum_i \sigma_3(\mathbf{v}_i^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{ik} \\ \vdots \qquad\qquad \ddots \qquad\qquad \vdots \\ \sum_i \sigma_3(\mathbf{v}_i^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{i1} \cdots \sum_i \sigma_3(\mathbf{v}_i^T\boldsymbol{\beta}_{\mathbf{X}})\cdot v_{ik} \end{bmatrix}.$$

In the end, the model parameters $\boldsymbol{\beta}_{\mathbf{X}}$ are encrypted as a ciphertext with fully-packed plaintext slots. More precisely, it yields encrypted model parameters $\mathtt{E}(\boldsymbol{\beta}_{\mathbf{X}})$ that represent a plaintext vector containing $N_2/k = \ell \cdot n$ copies of $\boldsymbol{\beta}_{\mathbf{X}}$ as follows:

$$\mathtt{E}(\boldsymbol{\beta}_{\mathbf{X}}) = \begin{bmatrix} \beta_{X1} & \beta_{X2} & \cdots & \beta_{Xk} \\ \beta_{X1} & \beta_{X2} & \cdots & \beta_{Xk} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{X1} & \beta_{X2} & \cdots & \beta_{Xk} \end{bmatrix}.$$

4.2.2 Parallel Logistic Regression Model Building for SNPs

Starting with $\boldsymbol{\beta} = (\boldsymbol{\beta}_{\mathbf{X}}, 0) \in \mathbb{R}^{k+1}$, we will perform one step of Newton's method for regression with SNPs. This implies that the regression coefficients multiplied by the values of the predictor are $\mathbf{U}\boldsymbol{\beta} = \mathbf{X}\boldsymbol{\beta}_{\mathbf{X}}$, so for all $i \in [n]$, if we let the predicted value be $\hat{y}_i = \mathbf{u}_i^T\boldsymbol{\beta}$, then we have $\hat{y}_i = \mathbf{x}_i^T\boldsymbol{\beta}_{\mathbf{X}}$. We note that

$$\begin{aligned}(\mathbf{W}\mathbf{z})_i &= w_i \cdot z_i \\ &= p_i(1-p_i)\cdot\left(\hat{y}_i + \frac{y_i - p_i}{p_i\cdot(1-p_i)}\right) \\ &= p_i(1-p_i)\cdot\hat{y}_i + (y_i - p_i). \end{aligned} \tag{5}$$

with $p_i = \sigma(\hat{y}_i)$. In the following, we describe how to securely evaluate these variables from the model parameters $\boldsymbol{\beta}_{\mathbf{X}}$. In the end, the server outputs encryptions of the numerator and the denominator of Equation (3), denoted by $\beta_j^\star$ and $\beta_j^\dagger$.

Step 1: Let $\hat{\mathbf{y}} = (\hat{y}_i)_{i=1}^n$ be a column vector of the predicted values. The goal of this step is to generate its encryption. The server first performs homomorphic multiplication between two ciphertexts $\mathtt{E}(\boldsymbol{\beta}_{\mathbf{X}})$ and $\mathtt{E}(\mathbf{X})$, and then applies $\mathtt{AllSum}$ to the resulting ciphertext:

$$\mathtt{E}(\hat{\mathbf{y}}_\star) \leftarrow \mathtt{AllSum}(\mathtt{E}(\boldsymbol{\beta}_{\mathbf{X}})\cdot\mathtt{E}(\mathbf{X}), 1, k). \tag{6}$$

The output ciphertext $\mathtt{E}(\hat{\mathbf{y}}_\star)$ encrypts the values $\hat{y}_i$ at $(t \cdot k + 1)$ positions for $(i-1)\cdot \ell \le t < i \cdot \ell$ and some garbage values in the other entries, denoted by $\star$, i.e.,

$$\mathtt{E}(\hat{\mathbf{y}}_\star) = \mathtt{E}\begin{bmatrix} \hat{y}_1 & \star & \cdots & \star \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_1 & \star & \cdots & \star \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_n & \star & \cdots & \star \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_n & \star & \cdots & \star \end{bmatrix}.$$

The server then performs a constant multiplication by c to annihilate the garbage values. The polynomial $c \leftarrow \mathtt{Encode}(\mathbf{C})$ is the encoding of the following matrix, where $\mathtt{Encode}(\cdot)$ is a standard procedure in [9] to encode a real vector as a ring element in R:

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{bmatrix}.$$

The next step is to replicate the values $\hat{y}_i$ to other columns:

$$\mathtt{E}(\hat{\mathbf{y}}) \leftarrow \mathtt{AllSum}(\mathtt{CMult}(\mathtt{E}(\hat{\mathbf{y}}_\star); c), -1, k),$$

denoted by $\mathtt{CMult}(\cdot)$ a scalar multiplication. So, the output ciphertext $\mathtt{E}(\hat{\mathbf{y}})$ has $N_2/n = \ell \cdot k$ copies of $\hat{y}_i$:

$$\mathtt{E}(\hat{\mathbf{y}}) = \mathtt{E}\begin{bmatrix} \hat{y}_1 & \hat{y}_1 & \cdots & \hat{y}_1 \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_1 & \hat{y}_1 & \cdots & \hat{y}_1 \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_n & \hat{y}_n & \cdots & \hat{y}_n \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_n & \hat{y}_n & \cdots & \hat{y}_n \end{bmatrix}.$$

Step 2: This step is simply to evaluate the approximating polynomial of the sigmoid function by applying the pure SIMD additions and multiplications:

$$\mathtt{E}(\mathbf{p}) \leftarrow \sigma_3(\mathtt{E}(\hat{\mathbf{y}})).$$

Then the server securely computes the weights w_i and carries out their multiplication with the working response vector $\mathbf{z}$ using Equation (5):

$$\begin{aligned} \mathtt{E}(\mathbf{w}) &\leftarrow \mathtt{E}(\mathbf{p}) \cdot (1 - \mathtt{E}(\mathbf{p})), \\ \mathtt{E}(\mathbf{Wz}) &\leftarrow \mathtt{E}(\mathbf{w}) \cdot \mathtt{E}(\hat{\mathbf{y}}) + (\mathtt{E}(\mathbf{y}) - \mathtt{E}(\mathbf{p})). \end{aligned} \tag{7}$$

Here the two output ciphertexts containing N_2/n copies of the values w_i and $w_i z_i$, respectively:

$$\mathtt{E}(\mathbf{w}) = \mathtt{E}\begin{bmatrix} w_1 & w_1 & \cdots & w_1 \\ \vdots & \vdots & \ddots & \vdots \\ w_1 & w_1 & \cdots & w_1 \\ \vdots & \vdots & \ddots & \vdots \\ w_n & w_n & \cdots & w_n \\ \vdots & \vdots & \ddots & \vdots \\ w_n & w_n & \cdots & w_n \end{bmatrix}, \quad \mathtt{E}(\mathbf{Wz}) = \mathtt{E}\begin{bmatrix} w_1 z_1 & w_1 z_1 & \cdots & w_1 z_1 \\ \vdots & \vdots & \ddots & \vdots \\ w_1 z_1 & w_1 z_1 & \cdots & w_1 z_1 \\ \vdots & \vdots & \ddots & \vdots \\ w_n z_n & w_n z_n & \cdots & w_n z_n \\ \vdots & \vdots & \ddots & \vdots \\ w_n z_n & w_n z_n & \cdots & w_n z_n \end{bmatrix}.$$

Step 3: The goal of this step is to generate trivial encryptions $\mathtt{E}(w_i)$ such that for $i \in [n]$, $\mathtt{E}(w_i)$ has w_i in all positions of its plaintext vector. We employ the hybrid algorithm of [17] for replication, denoted by $\mathtt{Replicate}(\cdot)$. The server outputs n ciphertexts

$$\{\mathtt{E}(w_i)\}_{1 \le i \le n} \leftarrow \mathtt{Replicate}(\mathtt{E}(\mathbf{w})).$$

Similarly, the server takes the ciphertext $\mathtt{E}(\mathbf{Wz})$ and performs another replication operation:

$$\{\mathtt{E}(w_i z_i)\}_{1 \le i \le n} \leftarrow \mathtt{Replicate}(\mathtt{E}(\mathbf{Wz})).$$

Step 4: For all $j \in [p]$, we define the vector $\mathbf{b}_j = \mathbf{X}^T\mathbf{W}\mathbf{s}_j \in \mathbb{R}^k$ and denote the ℓ-th component of $\mathbf{b}_j$ by $b_{j\ell}$. We note that $b_{j\ell} = \mathbf{x}_\ell^T\mathbf{W}\mathbf{s}_j = \sum_{i=1}^n (x_{i\ell} \cdot w_i \cdot s_{ij})$, where $\mathbf{x}_\ell = (x_{i\ell})_{i=1}^n$ is the j-th column of the design matrix $\mathbf{X}$. Then, for all $\ell \in [k]$, the server generates encryptions of the vectors $\mathbf{B}_\ell = \mathbf{x}_\ell^T\mathbf{W}\mathbf{S} = (b_{1\ell}, b_{2\ell}, \ldots, b_{p\ell})$ by computing

$$\mathtt{E}(\mathbf{B}_\ell) \leftarrow \sum_{i=1}^n \mathtt{E}(w_i) \cdot \mathtt{E}(x_{i\ell}\mathbf{S}_i). \tag{8}$$

On the other hand, since we add a column of ones to the matrix $\mathbf{X}$, we have $c_j = \mathbf{s}_j^T\mathbf{W}\mathbf{s}_j = \sum_{i=1}^n w_i \cdot s_{ij} = \sum_{i=1}^n x_{i1} \cdot w_i \cdot s_{ij} = b_{1j}$ for $j \in [p]$, which implies that $\mathtt{E}(\mathbf{B}_1)$ can be understood as an encryption of $(c_1, c_2, \ldots, c_p)$.

Step 5: This step is to securely compute the values $\mathbf{s}_j^T\mathbf{Wz} = \sum_{i=1}^n s_{ij} \cdot w_i \cdot z_i$ for $j \in [p]$. Specifically, the server performs the following computation:

$$\mathtt{E}(\mathbf{s}_1^T\mathbf{Wz}, \ldots, \mathbf{s}_p^T\mathbf{Wz}) \leftarrow \sum_{i=1}^n \mathtt{E}(w_i z_i) \cdot \mathtt{E}(x_{i1}\mathbf{S}_i). \tag{9}$$

Step 6: The goal of this step is to securely compute the vector $\mathbf{X}^T\mathbf{Wz}$ such that the ℓ-th element is obtained by $\mathbf{x}_\ell^T\mathbf{Wz} = \sum_{i=1}^n (x_{i\ell} \cdot w_i \cdot z_i)$ for $\ell \in [k]$. The server first performs the pure SIMD multiplication between two ciphertexts $\mathtt{E}(\mathbf{X})$ and $\mathtt{E}(\mathbf{Wz})$:

$$\mathtt{E}(\mathbf{X} \odot \mathbf{Wz}) \leftarrow \mathtt{E}(\mathbf{X}) \cdot \mathtt{E}(\mathbf{Wz}). \tag{10}$$

Here, the output ciphertext $\mathtt{E}(\mathbf{X} \odot \mathbf{Wz})$ encrypts the values $x_{i\ell}w_i z_i$:

$$\mathtt{E}(\mathbf{X} \odot \mathbf{Wz}) = \mathtt{E}\begin{bmatrix} x_{11}w_1z_1 & x_{12}w_1z_1 & \cdots & x_{1k}w_1z_1 \\ \vdots & \vdots & \ddots & \vdots \\ x_{11}w_1z_1 & x_{12}w_1z_1 & \cdots & x_{1k}w_1z_1 \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1}w_nz_n & x_{n2}w_nz_n & \cdots & x_{nk}w_nz_n \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1}w_nz_n & x_{n2}w_nz_n & \cdots & x_{nk}w_nz_n \end{bmatrix}.$$

Then the server aggregates the values in the same column to obtain a ciphertext encrypting $\mathbf{x}_\ell^T\mathbf{Wz}$:

$$\mathtt{E}(\mathbf{X}^T\mathbf{Wz}) \leftarrow \mathtt{AllSum}(\mathtt{E}(\mathbf{X} \odot \mathbf{Wz})), N_2/(k \cdot n), n).$$

Notice that this ciphertext contains the scalar $\mathbf{x}_\ell^T\mathbf{Wz}$ in every entry of the ℓ-th column, for $1 \le \ell \le k$:

$$\mathtt{E}(\mathbf{X}^T\mathbf{Wz}) = \mathtt{E}\begin{bmatrix} \mathbf{x}_1^T\mathbf{Wz} & \mathbf{x}_2^T\mathbf{Wz} & \cdots & \mathbf{x}_k^T\mathbf{Wz} \\ \mathbf{x}_1^T\mathbf{Wz} & \mathbf{x}_2^T\mathbf{Wz} & \cdots & \mathbf{x}_k^T\mathbf{Wz} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_1^T\mathbf{Wz} & \mathbf{x}_2^T\mathbf{Wz} & \cdots & \mathbf{x}_k^T\mathbf{Wz} \end{bmatrix}.$$

Finally, it outputs k ciphertexts, each encrypting $\mathbf{x}_\ell^T \mathbf{W}\mathbf{z}$ for $1 \le \ell \le k$, by applying the replication operation as follows:

$$\{\mathtt{E}(\mathbf{x}_\ell^T \mathbf{W}\mathbf{z})\}_{1 \le \ell \le k} \leftarrow \mathtt{Replicate}(\mathtt{E}(\mathbf{X}^T \mathbf{W}\mathbf{z})).$$

Step 7: The goal of this step is to compute the encryptions of the adjugate matrix and the determinant of $\mathbf{A} = \mathbf{X}^T\mathbf{W}\mathbf{X}$. We note that

$$\mathbf{A}_{r,s,t} = (\sum_{i=1}^{n} w_i \cdot \mathbf{X}_i^T \mathbf{X}_i)_{r,s,t} = \sum_{i=1}^{n} w_i \cdot (\mathbf{X}_i^T \mathbf{X}_i)_{r,s,t}$$

for $1 \le r \le s \le k$ and $1 \le t \le k-1$. The server first multiplies the ciphertexts $\mathtt{E}(\Sigma_{r,s,t})$ with the ciphertext $\mathtt{E}(\mathbf{w})$ to obtain

$$\mathtt{E}(\Sigma'_{r,s,t}) \leftarrow \mathtt{E}(\mathbf{w}) \cdot \mathtt{E}(\Sigma_{r,s,t}). \tag{11}$$

Here, the ciphertext $\mathtt{E}(\Sigma'_{r,s,t})$ encrypts n vectors $w_i \cdot (\mathbf{X}_i^T \mathbf{X}_i)_{r,s,t}$ for $1 \le i \le n$. Then we apply `AllSum` to aggregate these vectors and obtain $\mathbf{A}_{r,s,t}$:

$$\mathtt{E}(\mathbf{A}_{r,s,t}) \leftarrow \mathtt{AllSum}(\mathtt{E}(\Sigma'_{r,s,t}), \phi, n).$$

Next, the server performs multiplications between the ciphertexts $\mathtt{E}(\mathbf{A}_{r,s,t})$ as follows:

$$\mathtt{E}(\Sigma_{r,s}) \leftarrow \prod_{t=1}^{k-1} \mathtt{E}(\mathbf{A}_{r,s,t}). \tag{12}$$

The adjugate matrix can be obtained by aggregating $(k-1)!$ many values in $\mathtt{E}(\Sigma_{r,s})$:

$$\mathtt{E}(\mathtt{adj}(\mathbf{A})_{r,s}) \leftarrow \mathtt{AllSum}(\mathtt{E}(\Sigma_{r,s}), 1, (k-1)!).$$

In addition, the server computes

$$\mathtt{E}(x_{1r}\mathbf{W}) \leftarrow \mathtt{AllSum}(\mathtt{E}(x_{1r}) \cdot \mathtt{E}(\mathbf{w}), N_2/n, n)$$

for $1 \le r \le k$, and obtains a trivial encryption of the determinant of $\mathbf{A}$ as follows:

$$\mathtt{E}(|\mathbf{A}|) \leftarrow \sum_{r=1}^{k} \mathtt{E}(x_{1r}\mathbf{W}) \cdot \mathtt{E}(\mathtt{adj}(\mathbf{A})_{1r}).$$

Step 8: The final step is to securely compute the encryptions of $\boldsymbol{\beta}^*$ and $\boldsymbol{\beta}^\dagger$ by pure SIMD additions and multiplications. We note that multiplication of the vectors $\mathbf{B}_j$ from the left side and $\mathbf{X}^T\mathbf{W}\mathbf{z}$ from the right side with the matrix $\mathtt{adj}(A)$ can be written as

$$\mathbf{B}_j^T \cdot \mathtt{adj}(A) \cdot (\mathbf{X}^T\mathbf{W}\mathbf{z}) = \sum_{r,s=1}^{k} b_{jr} \cdot (\mathtt{adj}(A))_{r,s} \cdot (\mathbf{X}^T\mathbf{W}\mathbf{z})_s.$$

So, the server evaluates the numerator of Equation (3) to get the encryption of $\boldsymbol{\beta}^*$:

$$\mathtt{E}(\boldsymbol{\beta}^*) \leftarrow \mathtt{E}(|\mathbf{A}|) \cdot \mathtt{E}(\mathbf{s}_1^T\mathbf{W}\mathbf{z}, \ldots, \mathbf{s}_p^T\mathbf{W}\mathbf{z}) - \sum_{r,s=1}^{k} \mathtt{E}(\mathbf{B}_r) \cdot \mathtt{E}(\mathtt{adj}(\mathbf{A})_{rs}) \cdot \mathtt{E}(\mathbf{x}_s^T\mathbf{W}\mathbf{z}). \tag{13}$$

Then the output ciphertext $\mathtt{E}(\boldsymbol{\beta}^*)$ encrypts the values $\boldsymbol{\beta}_j^*$'s in a way that $\mathtt{E}(\boldsymbol{\beta}^*) = \mathtt{E}(\beta_1^*, \beta_2^*, \ldots, \beta_p^*)$. Similarly, we evaluate the denominator of Equation (3) to get an encryption of $\boldsymbol{\beta}^\dagger$:

$$\mathtt{E}(\boldsymbol{\beta}^\dagger) \leftarrow \mathtt{E}(|\mathbf{A}|) \cdot \mathtt{E}(c_1, c_2, \ldots, c_p) - \sum_{r,s=1}^{k} \mathtt{E}(\mathbf{B}_r) \cdot \mathtt{E}(\mathtt{adj}(\mathbf{A})_{rs}) \cdot \mathtt{E}(\mathbf{B}_s). \tag{14}$$

Hence, the output ciphertext $\mathtt{E}(\boldsymbol{\beta}^\dagger)$ represents the values $\boldsymbol{\beta}_j^\dagger$ in a way that $\mathtt{E}(\boldsymbol{\beta}^\dagger) = \mathtt{E}(\beta_1^\dagger, \beta_2^\dagger, \ldots, \beta_p^\dagger)$.

4.2.3 Output Reconstruction

The server sends the resulting ciphertexts $\mathtt{E}(\boldsymbol{\beta}^*)$, $\mathtt{E}(\boldsymbol{\beta}^\dagger)$, and $\mathtt{E}(|\mathbf{A}|)$ to the authority who has the secret key of the underlying HE scheme. Afterwards, the authority decrypts the values and computes the test statistics by using the Wald z-test, which are defined by the coefficient estimates divided by the standard errors of the parameters: $\beta_j/\sqrt{\mathtt{var}_j} = \beta_j^*/\sqrt{|\mathbf{A}| \cdot \beta_j^\dagger}$ for all $j \in [p]$. In the end, the p-values can be obtained from the definition $2 \cdot \mathtt{pnorm}(|\beta_j|/\sqrt{\mathtt{var}_j})$.

It includes some post computations after decryption, however, we believe that this is a reasonable assumption for the following reasons. Its complexity is even less than that of decryption, so this process does not require any stronger condition on the computing power of the secret key owner. Meanwhile, the output ciphertexts are encrypting $(2p+1)$ scalar values, which is two times more information compared to the ideal case. Our solution relies on the heuristic assumption that no sensitive information beyond the desired p-values can be extracted from decrypted results. One alternative is that the server can use a masking (sampling random values $r_j^*, r_j^\dagger, r_A$ such that ${r_j^*}^2 = r_j^\dagger \cdot r_A$ and multiplying them to $\beta_j^*, \beta_j^\dagger$ and $|\mathbf{A}|$, respectively) on resulting ciphertexts before sending them to the secret key owner to weaken this assumption.

4.3 Threat Model

We consider the following threat models. Firstly, we assume that the computing server is semi-honest (i.e., honest but curious). If we can ensure the semantic security of the underlying HE scheme, there is no information leakage from encrypted data even in malicious setting. Secondly, we assume that the secret key owner does not collude with the server.

5 Results

In this section, we explain how to set the parameters and report the performance of our regression algorithms.

5.1 Dataset Description

The dataset provided by the iDASH competition organizers consists of 245 samples, partitioned into two groups by the condition of high cholesterol, 137 under control group and 108 under disease group. Each sample contains a binary phenotype along with 10643 SNPs and 3 covariates (age, weight, and height). This data was extracted from Personal Genome Project [2]. The organizers changed the input size in terms of SNPs, cohort size, and threshold of significance to test the scalability of submitted solutions.

We may assume that the imputation and normalization are done in the clear prior to encryption. More precisely, we impute the missing covariate values with the sample mean of the observed covariates. We also center the covariates matrix $\mathbf{X}$ by subtracting the minimum from each column and dividing by a quantity proportional to the range.

5.2 Parameters Settings

We explain how to choose the parameter sets for building secure semi-parallel logistic regression model. We begin with a parameter L which determines the largest bitsize of a fresh ciphertext modulus. Since the plaintext space is a vector space of real numbers, we multiply a scale factor of p to plaintexts before encryption. It is a common practice to perform the rescaling operation by a factor of p on ciphertexts after each (constant) multiplication in order to preserve the precision of the plaintexts. This means that a ciphertext modulus is reduced by $\log p$ bits after each multiplication or we can say that a multiplication operation consumes one level.

Kim et al. [22] proposed the least squares approach to find a global polynomial approximation of the sigmoid and presented degree 3, 5, and 7 approximation polynomial over the domain $[-8, 8]$. We observed that input values of the sigmoid in our data belong to this interval. As noted in [22], these approximations offer a trade-off between accuracy and efficiency. A low-degree polynomial requires a smaller depth for an evaluation while a high-degree polynomial has a better precision. So, we adapt the degree 3 approximation polynomials of the sigmoid function as $\sigma_3(x) = 0.5 + 0.15012x - 0.001593x^3$, which consumes roughly two levels.

Suppose that we start with $\mathbf{v}^{(0)} = \boldsymbol{\beta}_{\mathbf{X}}^{(0)} = \mathbf{0} \in \mathbb{R}^k$ and the input ciphertext $\mathtt{E}(\mathbf{y}^T\mathbf{X})$ is at level L. It follows from the parameter analysis of [19] that the ciphertext level of $\mathtt{E}(\boldsymbol{\beta}_{\mathbf{X}})$ after the evaluation of Nesterov's accelerated GD is $L-(4\cdot(\textsc{IterNum}-1)+1)$ where $\textsc{IterNum}$ denotes the number of iterations of the GD algorithm. Similarly, we expect each of Steps 1 and 2 to consume two levels for computing the ciphertexts $\mathtt{E}(\hat{\mathbf{y}})$ and $\mathtt{E}(\mathbf{p})$. This means that $\mathtt{E}(\mathbf{p})$ is at level $L-(4\cdot\textsc{IterNum}+1)$; so we get

$$\begin{aligned}\mathtt{lvl}(\mathtt{E}(\mathbf{w})) &= L-(4\cdot\textsc{IterNum}+2),\\ \mathtt{lvl}(\mathtt{E}(\mathbf{Wz})) &= L-(4\cdot\textsc{IterNum}+3).\end{aligned}$$

We now consider the replication procedure in Step 3. Although the input vector $\mathbf{w}=(w_i)_{i=1}^n$ is fully packed into a single ciphertext (i.e., the length of the corresponding plaintext vector is N_2), it suffices to produce n number of ciphertexts, each of which represents an entry w_i across the entire array. As presented in Section 4.2 of [17], the replication procedure consists of two phases of computation. The first phase is to partition the entries in the input vector into size-2^s blocks and construct $n/2^s$ number of vectors consisting of the entries in the i-th block with replicated $N_2/2^s$ times. We use a simple replication operation $n/2^s$ times, which applies multiplicative masking to extract the entry and then perform the `AllSum` operation to replicate them as in Step 1; its depth is just a single constant multiplication. The second phase is to recursively apply replication operations in a binary tree manner, such that in each stage we double the number of vectors while halving the number of distinct values in each vector; its depth is s constant multiplications. In total, we expect to consume $(s+1)$ levels during the replication procedure; so, we get

$$\begin{aligned}\mathtt{lvl}(\mathtt{E}(w_i)) &= L-(4\cdot\textsc{IterNum}+s+3),\\ \mathtt{lvl}(\mathtt{E}(w_i z_i)) &= L-(4\cdot\textsc{IterNum}+s+4).\end{aligned}$$

Later, Step 4 consumes one level from the level $\mathtt{lvl}(\mathtt{E}(w_i))$ for multiplication; so, we have

$$\mathtt{lvl}(\mathtt{E}(\mathbf{B}_\ell)) = L-(4\cdot\textsc{IterNum}+s+4). \tag{15}$$

Similarly, Step 5 consumes one more level from the computation of $\mathtt{E}(w_i z_i)$; so we get

$$\mathtt{lvl}(\mathtt{E}(\mathbf{s}_1^T\mathbf{Wz},\ldots,\mathbf{s}_p^T\mathbf{Wz})) = L-(4\cdot\textsc{IterNum}+s+5).$$

On the other hand, Step 6 requires one level of multiplication for the evaluation of the update formula (10); so we know

$$\mathtt{lvl}(\mathtt{E}(\mathbf{X}\odot\mathbf{Wz})) = \mathtt{lvl}(\mathtt{E}(\mathbf{Wz}))-1 = L-(4\cdot\textsc{IterNum}+4).$$

As discussed above, the output ciphertexts $\mathtt{E}(\mathbf{x}_\ell^T\mathbf{Wz})$ consume $(s'+1)$ levels during the replication procedure where $2^{s'}$ is the unit block size of the first step of the replication procedure; so we have

$$\mathtt{E}(\mathbf{x}_\ell^T\mathbf{Wz}) = \mathtt{lvl}(\mathtt{E}(\mathbf{X}\odot\mathbf{Wz}))-(s'+1) = L-(4\cdot\textsc{IterNum}+s'+5).$$

In Step 7, it requires one and $\log(k-1)$ levels of multiplications for the evaluation of the update formulas (11) and (12), respectively. If we let $\ell' = \mathtt{max}\{\mathtt{lvl}(\mathtt{E}(\mathbf{w})),\mathtt{lvl}(\mathtt{E}(\Sigma_{r,s,t}))\}$, then we have

$$\begin{aligned}\mathtt{lvl}(\mathtt{E}(\mathtt{adj}(\mathbf{A})_{rs}) &= \ell'-(1+\log(k-1)),\\ \mathtt{lvl}(\mathtt{E}(|\mathbf{A}|)) &= \ell'-(2+\log(k-1)).\end{aligned}$$

It follows from the update formulas (13) and (14) that it suffices to set as $\mathtt{lvl}(\mathtt{E}(\mathtt{adj}(\mathbf{A})_{rs})) = \mathtt{lvl}(\mathtt{E}(\mathbf{B}_\ell)) = 3$ for obtaining the correct results. This implies that we need to set the number of levels L to be at least $L \geq (4\cdot\textsc{IterNum}+s+4)+3$ from (15). In the implementation, we set $\textsc{IterNum}=2$, $s=4$, $s'=0$, and $L=19$. The encryption levels of data are set as follows:

– $\texttt{lvl}(\texttt{E}(\mathbf{y}^T\mathbf{X})) = L = 19$,
– $\texttt{lvl}(\texttt{E}(\mathbf{X})) = \texttt{lvl}(\texttt{E}(\boldsymbol{\beta}_{\mathbf{X}})) = 14$, from (6)
– $\texttt{lvl}(\texttt{E}(\mathbf{y})) = \texttt{lvl}(\texttt{E}(\mathbf{p})) = 10$, from (7),
– $\texttt{lvl}(\texttt{E}(x_{i\ell}\mathbf{S}_i)) = \texttt{lvl}(\texttt{E}(w_i)) = 4$, from (8),
– $\texttt{lvl}(\texttt{E}(\Sigma_{r,s,t})) = \texttt{lvl}(\texttt{E}(\mathsf{adj}(\mathbf{A}))) + 3 = 6$.

We use $\log p_0 \approx 60$, $\log q_0 \approx 51$, and $\log q_i \approx 43$ for $i = 1, \ldots, L$. Therefore, we derive a lower bound of the bit size of the largest RLWE modulus Q as

$$\log Q = \log q_0 + (L-1) \cdot \log q_i + \log p_0 \approx 885.$$

Alternatively, we may do a few less or more iterations in the GD algorithm, for example, setting IterNum = 1 or 3. We conducted tests to compare the trade-offs in using different sets of parameters.

We choose the secret key from the ternary distribution, which means to select uniformly at random from $\{-1, 0, 1\}$. The error is sampled from the discrete Gaussian distribution of standard deviation $\mathsf{stdev} = 3.2$. We follow the recommended parameters from the standardization workshop paper [6], thus providing at least 128-bits security level of our parameters. We summarize the parameters of our implementation in Table 1. For comparison, we also listed parameters when using IterNum = 1 and 3.

Table 1: HE parameter sets.

	IterNum	$\log N$	L	$\log p$	$\log q_0$	$\log p_0$	$\log Q$
Set-I	1	15	15	43	51	60	713
Set-II	2	15	19	43	51	60	885
Set-III	3	16	23	45	54	62	1106

5.3 Optimization Techniques

The standard method of homomorphic multiplication consists of two steps: raw multiplication and key-switching. The first step computes the product of two ciphertexts $\mathsf{ct}(Y) = c_0 + c_1 Y$ and $\mathsf{ct}'(Y) = c_0' + c_1' Y$ (as done in [5]), and returns a quadratic polynomial, called *extended ciphertext*, $\mathsf{ct}_{\mathsf{mult}} = c_0c_0' + (c_0c_1' + c_0'c_1)Y + c_1c_1'Y^2$. This ciphertext can be viewed as an encryption of the product of plaintexts with the extended secret $(1, s, s^2)$. Afterwards, the key-switching procedure transforms it into a normal (linear) ciphertext encrypting the same message with the secret key $(1, s)$.

We observe that the second step is much more expensive than the first one since it includes an evaluation of NTT (Fourier transformation over the modulo space), and that a simple arithmetic (e.g. linear operation) is allowed between extended ciphertexts. To reduce the complexity, we adapt the technique called *lazy key-switching*, which performs some arithmetic over extended ciphertexts instead of running the second step right after each raw multiplication. We get a normal ciphertext by performing only one key-switching operation after evaluating linear circuits over the extended ciphertexts. It can reduce the number of required key-switching algorithms as well as the total computational cost. For instance, if we add many terms after raw multiplications in the right hand side of the update (8) and apply key-switching to the output ciphertext, this takes only one key-switching rather than n.

5.4 Performance Results

We present our implementation results using the proposed techniques. All the experiments were performed on a Macbook with an Intel Core i7 running with 4 cores rated at 2.5 GHz. Our implementation exploits multiple cores when available, thereby taking the advantages of parallelization.

In Table 2, we evaluated our model's performance based on the average running time and the memory usages in the key generation, encryption, evaluation, and decryption procedures.

We achieved very high level of accuracy in the final output (after decryption) for all three sets of parameters. The type-I (false positive) and type-II (false negative) errors of the output of our solution

Table 2: Experimental results for iDASH dataset with 245 samples, each has 10643 SNPs and 3 covariates (4 cores). ms=10^{-3} sec.

Stage	Set-I		Set-II		Set-III	
Key Generation	4.460 sec	2.321 GB	6.665 sec	3.584 GB	9.699 sec	10.721 GB
Encryption	7.059 sec	5.406 GB	7.066 sec	6.669 GB	23.023 sec	12.137 GB
Training with covariates	2.622 sec	7.176 GB	9.367 sec	7.186 GB	62.922 sec	12.137 GB
Training with all SNPs	40.442 sec	10.339 GB	42.567 sec	11.176 GB	108.24 sec	12.137 GB
Total evaluation	**43.064 sec**	–	**51.934 sec**	–	**171.162 sec**	–
Decryption	0.025 sec	10.339 GB	0.025 sec	11.176 GB	0.055 sec	12.137 GB
Reconstruction	0.794 ms	10.339 GB	0.794 ms	11.176 GB	2.821 ms	12.137 GB

Fig. 1: Comparison with the semi-parallel model (p-value cut-off: 10^{-5}).

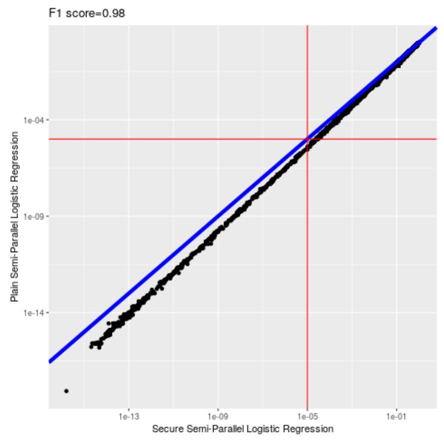
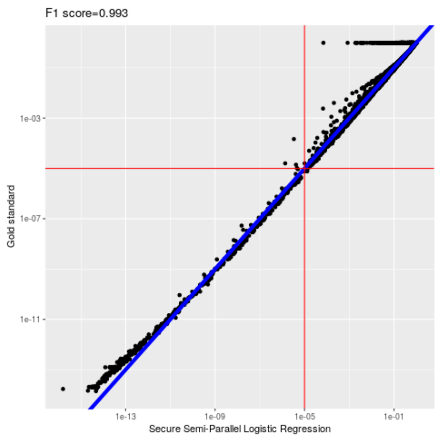


Fig. 2: Comparison with the gold standard model (p-value cut-off: 10^{-5}).



are very small when comparing to both the semi-parallel model and the gold standard model (full logistic regression) with respect to various p-value cut-off thresholds. See Figures 1 and 2 for comparisons against these two plain models with a cut-off of 10^{-5} when $\textsc{IterNum} = 2$. To better compare the estimated p-values (above or below certain cut-offs) on the encrypted model against the plaintext one (semi-parallel GWAS), we measured F_1-scores on the p-values obtained from our solution against the two plain models. The resulting F_1-scores are very close to 1 across all cases with different cut-offs (10^{-2} to 10^{-5}), which are shown in Table 3.

We also conducted the DeLong's test [14, 25] to validate our solution against the semi-parallel model. Specifically, we drawn at uniformly random about 10% of the total SNP test data and transformed the corresponding p-values to 0-1 labels according to the cut-off threshold; then we constructed the ROC (Receiver Operating Characteristic) curves for these labels and performed the DeLong's test to compare the AUCs (Area Under the Curve) of these curves. Such test was repeated 10 times to obtain the mean and the standard deviation of the p-values of the test. The results for $\textsc{IterNum} = 2$ are shown in Table 4.

6 Discussion and Conclusion

One constraint in our approach is that the matrix inverse can be computed in an efficient way when the input dimension is small. In modern GWAS, it is common to include covariates to account for such factors as gender, age, other clinical variables and population structure. A significant challenge in performing efficient secure GWAS on this generalized model is to handle large-scale matrix inversion.

Table 3: F_1-Scores on different models.

Cut-off	v.s. Plain semi-parallel model			v.s. Plain gold standard model		
	Set-I	Set-II	Set-III	Set-I	Set-II	Set-III
10^{-2}	0.9807	0.9830	0.9964	0.9818	0.9808	0.9710
10^{-3}	0.9749	0.9810	0.9975	0.9878	0.9887	0.9740
10^{-4}	0.9745	0.9798	0.9969	0.9878	0.9888	0.9729
10^{-5}	0.9828	0.9852	0.9971	0.9946	0.9970	0.9805

Table 4: DeLong's Test for AUCs of our solution with Set-II against the plain semi-parallel model.

Cut-off	Mean and stdev of the test results
10^{-2}	0.4038±0.3001
10^{-3}	0.5357±0.2704
10^{-4}	0.6404±0.2638
10^{-5}	0.8959±0.2195

In this paper, we showed the state-of-the-art performance of secure logistic regression model training for GWAS. We have demonstrated the feasibility and scalability of our model in speed and memory consumption. We expect that the performance can be improved if the underlying HE scheme is rewritten with optimized code.

References

1. admin. NIH genomic data sharing - offie of science policy. `https://osp.od.nih.gov/scientific-sharing/genomic-data-sharing/`.
2. Personal genome project. `https://www.personalgenomes.org/us`.
3. J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *International Conference on Selected Areas in Cryptography*, pages 423–442. Springer, 2016.
4. C. Bonte, E. Makri, A. Ardeshirdavani, J. Simm, Y. Moreau, and F. Vercauteren. Towards practical privacy-preserving genome-wide association study. *BMC bioinformatics*, 19(1):537, 2018.
5. Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In *Advances in Cryptology–CRYPTO 2011*, pages 505–524. 2011.
6. M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, J. Hoffstein, K. Lauter, S. Lokam, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan. Security of homomorphic encryption. Technical report, HomomorphicEncryption.org, Redmond WA, USA, July 2017.
7. H. Chen, R. Gilad-Bachrach, K. Han, Z. Huang, A. Jalali, K. Laine, and K. Lauter. Logistic regression over encrypted data from fully homomorphic encryption. *BMC medical genomics*, 11(4):81, 2018.
8. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. A full RNS variant of approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*. Springer, 2018.
9. J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017*, pages 409–437. Springer, 2017.
10. J. H. Cheon, M. Kim, and M. Kim. Search-and-compute on encrypted data. In *International Conference on Financial Cryptography and Data Security*, pages 142–159. Springer, 2015.
11. J. H. Cheon, M. Kim, and M. Kim. Optimized search-and-compute circuits and their application to query evaluation on encrypted data. *IEEE Transactions on Information Forensics and Security*, 11(1):188–199, 2016.
12. J. H. Cheon, M. Kim, and K. Lauter. Homomorphic computation of edit distance. In *International Conference on Financial Cryptography and Data Security*, pages 194–212. Springer, 2015.
13. D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242, 1958.

14. E. R. DeLong, D. M. DeLong, and D. L. Clarke-Pearson. Comparing the areas under two or more correlated receiver operating characteristic curves: A nonparametric approach. *Biometrics*, 44(3):837–845, 1988.
15. D. A. Freedman. Statistical models: theory and practice, 2009.
16. C. Gentry, S. Halevi, and N. P. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology–CRYPTO 2012*, pages 850–867. 2012.
17. S. Halevi and V. Shoup. Algorithms in HElib. In *International Cryptology Conference*, pages 554–571. Springer, 2014.
18. C. W. Hug and P. Szolovits. ICU acuity: real-time models versus daily models. In *AMIA annual symposium proceedings*, volume 2009, page 260. American Medical Informatics Association, 2009.
19. A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 11(4):83, 2018.
20. M. Kim and K. Lauter. Private genome analysis through homomorphic encryption. *BMC medical informatics and decision making*, 15(Suppl 5):S3, 2015.
21. M. Kim, Y. Song, and J. H. Cheon. Secure searching of biomarkers through hybrid homomorphic encryption scheme. *BMC medical genomics*, 10(2):42, 2017.
22. M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang. Secure logistic regression based on homomorphic encryption: design and evaluation. *JMIR medical informatics*, 6(2), 2018.
23. Y. Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.
24. C. Robert. Machine learning, a probabilistic perspective, 2014.
25. X. Robin, N. Turck, A. Hainard, N. Tiberti, F. Lisacek, J.-C. Sanchez, and M. Müller. pROC: an open-source package for R and S+ to analyze and compare ROC curves. *BMC Bioinformatics*, 12(1):77, Mar 2011.
26. A. A. Shabalin. Matrix eQTL: ultra fast eQTL analysis via large matrix operations. *Bioinformatics*, 28(10):1353–1358, 2012.
27. K. Sikorska, E. Lesaffre, P. F. Groenen, and P. H. Eilers. GWAS on your notebook: fast semi-parallel linear and logistic regression for genome-wide association studies. *BMC bioinformatics*, 14(1):166, 2013.
28. J. J. Trinckes and Jr. The definitive guide to complying with the HIPAA/HITECH privacy and security rules, 3 Dec. 2012.
29. J. Truett, J. Cornfield, and W. Kannel. A multivariate analysis of the risk of coronary heart disease in framingham. *Journal of chronic diseases*, 20(7):511–524, 1967.