UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

AGILE: ARBITRARY GRID LOGISTIC REGRESSION USING INTEL SOFTWARE GUARD EXTENSIONS

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE IN TELECOMMUNICATIONS ENGINEERING

By

CHAO JIANG
Norman, Oklahoma
2016

AGILE: ARBITRARY GRID LOGISTIC REGRESSION USING
INTEL SOFTWARE GUARD EXTENSIONS

A THESIS APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY

_____
Dr. Kam Wai Clifford Chan, Chair

_____
Dr. Pramode Verma

_____
Dr. Samuel Cheng

Acknowledgements

Foremost, I would like to express my sincere gratitude to my thesis advisor Dr. Shuang Wang for his support of this research, for his patience, motivation, and immense knowledge. Without his guidance and support, I would not have been able to complete this dissertation.

Besides my advisor, I would also like to thank my thesis committee members: Dr. Kam Wai Clifford Chan, Dr. Pramode Verma, and Dr. Samuel Cheng for their encouragement, insightful comments and suggestions.

My sincere thanks also goes to Dr. Xiaoqian Jiang for offering me the summer internship opportunities in his groups. Also I thank my lab mates at iDASH: Feng Chen, Chenwei Xie, Sijie Ding, Yao Lu, Dong Han for their advice and support of my thesis.

Last, but not least, I would like to thank my parents, Xiaochun Jiang and Xiaoyu Liu, for giving birth to me and supporting me throughout my life.

Table of Contents

List of Tables

List of Figures

Abstract

Biomedical data are often collected and stored at different sites. How to take the most advantage of the data to provide better health care for patients and to contribute to academic research becomes more and more important and challenging considering the privacy regulations association with the data. There are several barriers to sharing and exchanging information, such as complex of data formats, information leakage during the data transmission, and big data issues. In this thesis, I focus on how to conduct integrated data analysis while ensuring data privacy and security during both data transmission and integration. Through a small experiment of GLORE[1] implemented on both garbled circuits[2] and Intel® Software Guard Extensions (Intel® SGX), I found that Intel® SGX performed better than garbled circuits in time consuming. So I believe that Intel® SGX has the potential to make great progress in security multiparty computation. By applying Intel® SGX, I not only built a framework but also devised a more flexible model that lets participants more freely cooperate with each other. My model AGILE leverages Intel® SGX to deliver trustworthy computations, a feature that is unlike the existing models like GLORE and VERTIGO[3] that address the integration problem when data is either horizontally or vertically partitioned. AGILE deals with data that is arbitrarily partitioned. Furthermore, to demonstrate AGILE's performance, I evaluated the model using two real datasets. The experimental results show that AGILE provides secure and accurate computation much faster than GLORE and VERTIGO.

Chapter 1: Introduction

Biomedical data sharing and integration are vital for improving the quality of healthcare, accelerating new discoveries in research area. For example, by integrating data from different research centers like the Mobile-sensor Data-to-Knowledge (MD2K) Center, Informatics for Integrating Biology and Bedside (i2b2), and Global Alliance for Genomics and Health (GA4GH), physicians can make more informed decisions on disease prediction and treatment for each patient. A big challenge in integrating data from different institutions is the securing patient privacy. Information leakage of the data may cause problems for participants in the data exchange considering that an attacker can identify those participants by utilizing this kind of sensitive information. Many frameworks or protocols were proposed such as Yao's Garbled Circuits (GC), the protocol of Goldriech-Micali-Wigderson developed by Choi[4] (GMW) to provide secure communication between two or more participants. The approach documented in this thesis was to take Intel® SGX to implement an alternative framework for security multi-partition computation. By taking logistic regression as an example, experiment proved that it costs less time to run an algorithm on Intel® SGX compared to running it with GC. Also the proposed model is more flexible and efficient than previous model GLORE and VERTIGO models.

1.1 Motivation

In the era of big data, out the security and privacy of personal data is a major concern, especially data related to one's health. In order to encourage people to share their data for the purpose of research, an efficient, reliable, and adoptable framework for data exchange needs to be built. Existing software-based protection strategies like

GMW and GC will be costly and time consuming. Hardware-based data protection techniques have made great progress. One popular method for improving the efficiency of data sharing is to use powerful hardware-based security solutions such as trusted execution environments (TEEs)[5]. Thus, my model took advantage of Intel® SGX.

1.2 Objectives

The goal of this thesis is to provide secure data sharing and analysis across different institutions by using Intel® SGX. In order to demonstrate my assumption, I implemented the same algorithms on Intel® SGX as was implemented on GC. Also, I introduced a more flexible model, AGILE, for data sharing as compared with previous models such as GLORE and VERTIGO. Any participant can share and send its data to the server without having to adhere to a strict format.

1.3 Organization of the Thesis

This thesis is organized as follows. Chapter 2 starts with a brief introduction of Intel® SGX emphasizing the relevant features that apply in my AGILE model. Chapter 3 begins with an overview of standard logistic regression. Then I present an explanation of two previous popular models, GLORE and VERTIGO, based on logistic regression. Chapter 4 describes the implementation of three models GLORE, VERTIGO, and AGILE on the Intel® SGX and compares the model GLORE that operated between GC and Intel® SGX. Chapter 5 shows some conclusions based on the experimental results.

Chapter 2: Intel® SGX

This chapter describes Intel® SGX briefly, then goes on to describe the key features of Intel® SGX related to the design of the experimental models, i.e., local attestation and remote attestation. More details are described in Intel® SGX white paper[6].

2.1 Intel® SGX Overview

Intel® SGX is a set of new instructions and mechanisms[7] that enables applications to allocate private regions of memory, called *enclaves*. Enclaves provide a secure environment where code or data within enclaves cannot be accessed by other applications or privileged system software, such as the operating system (OS), hypervisor, and BIOS. Fig. 1 shows the framework of Intel® SGX. The application has two parts, one is *untrusted* part which can create an enclave. The communication between these two parts is through ECALL and OCALL part of the Enclave Definition Language (EDL).

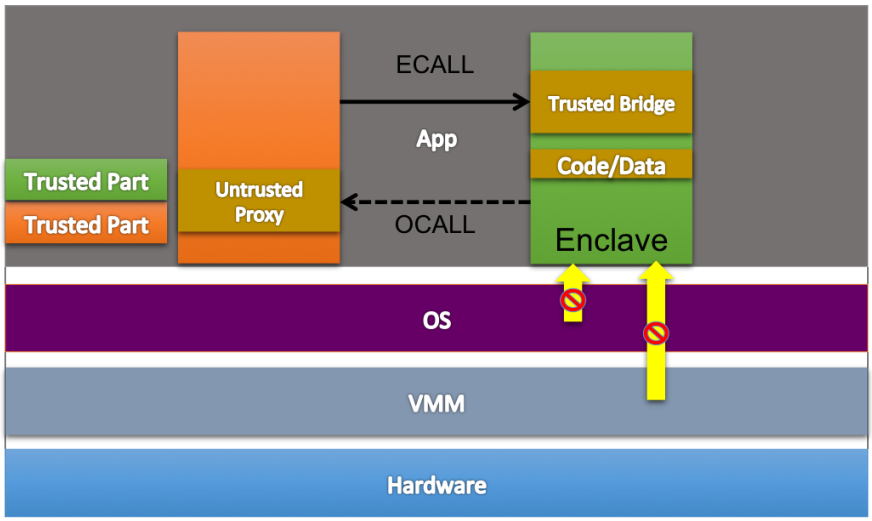

Fig. 1 SGX overview

In order to know how Intel® SGX can supply a service to more friendly

implement secure multi-partitions computing, one must know more about the enclave and attestation service.

2.2 Enclave Identity

An enclave created by an application is identified by the value MRENCLAVE, which is a SHA-256[8] digest of an internal logbook that archives all the activity performed while the enclave is made[9]. After the initialization of the enclave through the EINIT instruction, no more updates can be made to MRENCLAVE. This SHA-256 digest identifies cryptographically the code, data, and stack placed inside the enclave, the order and position in which the enclave's pages were placed, and the security properties of each page. Once changes for these variables are complete, the value of MRENCLAVE is also modified.

Another identity of the enclave is "Sealing Identity" used for data protection. It contains a "Sealing Authority," a product ID and a version number. The function of "Sealing Authority" is signing the enclave prior to distribution. Then the hardware can check the signature. After passed the check, a hash of the public key of the Sealing Authority stores in the MRSIGNER register.

2.3 Attestation

To exchange information, the identification must be attested first. In Intel® SGX, attesting means checking that an application has been properly instantiated on the platform. Another party can believe that only exact software is running within an enclave on a support host. For attestation, Intel® SGX architecture produces an attestation assertion that contains the identification of the software environment, the

detail of any non-measureable state, and the data associated with the environment. The last one is a cryptographic binding with the host.

Two mechanisms are used for authentication: One is for two enclaves running on the same platform (local attestation); another one is for attestation on different platforms (remote attestation).

2.3.1 Local Enclave Attestation

We start to portray the process of two enclaves, A and B running on the same Intel® SGX enable host, verify the identity of the other as Fig. 2 shows.

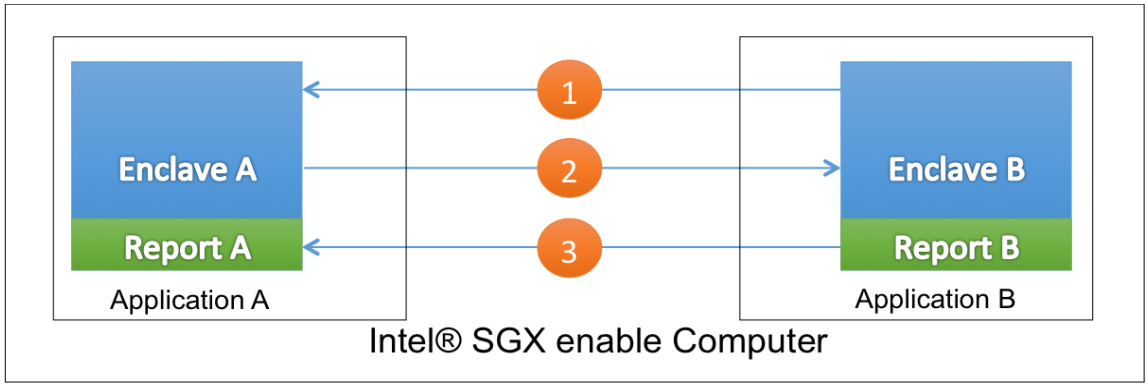

Fig. 2 Local attestation

1. First, an unsecured communication path was established between enclave A and B. Enclave A obtains the identity of B.
2. Then, enclave A uses the EREPORT instruction to produce a REPORT data structure that consists of the hash value of both two enclaves (enclave identities), the signed identity of enclave A, the public key of the signer, some user data, and a message authentication code (MAC) over the data structure. The MAC is constructed with a report key, known only to the target enclave and the EREPORT instruction on the same device. After that,

enclave A sends its REPORT to enclave B via an untrusted communication path.

3. After receiving the REPROT from enclave A, enclave B can retrieve its report key to re-compute the MAC over the REPORT structure and authenticates that the REPORT is valid to ensure that they are running on the same host. It then validates the identity of A (i.e., its content is not tainted). Finally, enclave B reciprocates this process for mutual verification.

2.3.2 Remote Enclave Attestation

Different from local attestation that uses a symmetric key system where only enclave B and the EREPORT instruction of enclave A have access to the authentication key, remote attestation needs an asymmetric cryptography to identify a specific platform outside the host. Here Intel introduced an extension to the Direct Anonymous Attestation scheme used by the TPM[10], [11] called Intel® Enhanced Privacy ID[12] (EPID) to sign enclave attestation. This attestation will be certified by an EPID backend infrastructure. In order to make sure that the signed enclave actually resides in the platform, Intel® SGX designed a particularly provisioned enclave, named *quoting enclave*, which will verify the enclave as local attestation. After that, quoting enclave signs it with the EPID key. Considering that EPID key is bound to the version of the processor's firmware, the quoting enclave can present the processor itself. A shared key will be created after the procedure of remote attestation by taking the method Diffie-Hellman key exchange.

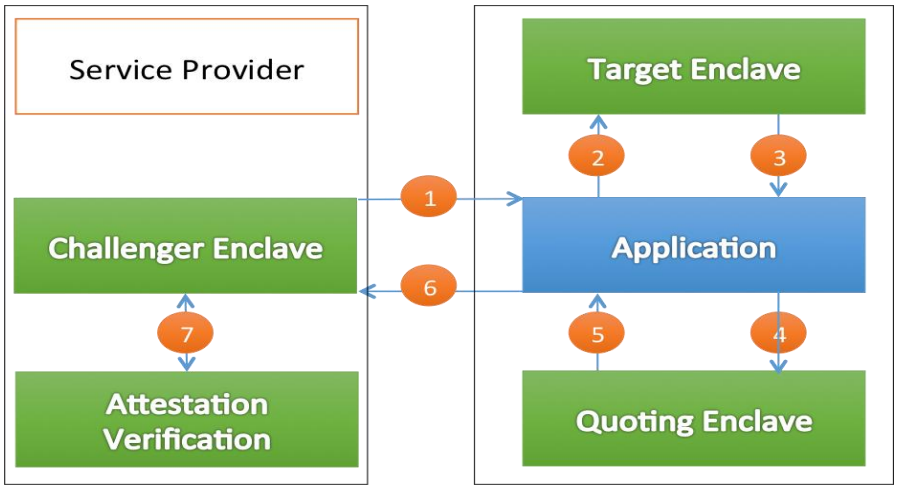

Fig. 3 Remote attestation

Fig. 3 shows an illustration of how a service provider requests remote attestation from an outsider platform.

1. A remote attestation request arrives from a challenger enclave.
2. Application passes the provider's challenge to its enclave.
3. Enclave creates a REPORT that includes the response to the challenge. This response usually is used for building the secret key.
4. The application forwards the REPORT to Quoting Enclave for signing.
5. Quoting Enclave veriies that it's the enclave created by this platform and signs it with its EPID key.
6. The application sends the QUOTE to the service challenger.
7. The challenger requests attestation verification service to validate the signature over the Quote. The challenger enclave then checks the response information to get the secret key.

2.4 Diffie-Hellman key exchange

Diffie-Hellman protocol[13] is a method of securely exchanging cryptographic keys over a public channel for two computer users. I used an example to explain how

the algorithm works. Assume that Alice and Bob want to share private key. First, they agree on two prime number g and p, where p is usually very large and g is a primitive root modulo p. These two numbers are public to other users. Now, Alice chooses a large random number a as her private key, and Bob takes b as his private key. Then Alice computes $A = g^a (mod\ p)$; Bob computes $B = g^b (mod\ p)$. After sending the computed A and B to each other, they can compute their shared key $K = g^{ab} (mod\ p)$. If any potential eavesdropper (Eve) wants to know that share key, she needs to get a from $A = g^a (mod\ p)$ and b from $B = g^b (mod\ p)$. But she cannot get a and b from $A\ and\ B$ as that is the discrete logarithm problem.

2.5 Summary of Intel® SGX

By getting an outline of Intel® SGX through the above explain of Intel® SGX, I can implement the authentication of remote users. What is more, data transmitted are all encrypted. Only the correct enclave can decrypt the data. Thus, Intel® SGX enables service providers to transmit privacy data over the air and to know with confidence that their data are properly protected.

Chapter 3: Grid Logistic Regression

This chapter begins with logistic regression (LR). Then I explain two logistic regression models (GLORE and VERTIGO). In the remainder of this thesis, I use regular symbols as scalar variables and bold symbols as vectors or matrices.

3.1 Logistic Regression (LR)

LR is a very common model applied in biomedical data analysis. The assumption is that there is a training dataset $\mathcal{D} = \{(\boldsymbol{X}, \boldsymbol{y})\}$ of m records for patients, where $\boldsymbol{X}$ is a matrix with dimension of $m \times n$, $where\ n$ represents the number of features and $\boldsymbol{y}$ is the observed binary outcome for each records. In the LR model, the likelihood function of $y_i = 1$ given $\boldsymbol{x}_i$ can be stated as follows:

$$P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\beta}) = \frac{1}{1 + e^{-\boldsymbol{\beta}^T x_i}}, \tag{1}$$

where $\boldsymbol{\beta}$ is a weight vector that measures the relationship between the response variable y_i and covariates $\boldsymbol{x}_i$. Note that $P(y_i = 0|\boldsymbol{x}_i, \boldsymbol{\beta}) = 1 - P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\beta})$ for binary response variable y_i. Given the training dataset $\mathcal{D}$, $\boldsymbol{\beta}$ can be estimated by maximizing the following log likelihood function.

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmax}} \left(l(\boldsymbol{\beta}) = \sum_{i=1}^{n} y_i P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\beta}) + (1 - y_i)(1 - P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\beta})) \right) \tag{2}$$

Here, I used $\mathrm{l}(\boldsymbol{\beta})$ to represent the log-likelihood function. Because there is no closed-form solution for $\boldsymbol{\beta}$, iterative numerical solutions are required to obtain the optimal parameters. In a centralized model, the Newton-Raphson method is broadly used to find $\widehat{\boldsymbol{\beta}}$. The iterative maximization is accomplished by calculating the first and second derivatives of the log-likelihood function $\mathrm{l}(\boldsymbol{\beta})$. In the t-th iteration, the current estimation $\boldsymbol{\beta}^{(\mathrm{t})}$ is updated by

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \left[l''\left(\boldsymbol{\beta}^{(t)}\right)\right]^{-1} l'\left(\boldsymbol{\beta}^{(t)}\right) \quad (3)$$

3.2 Grid Binary Logistic Regression (GLORE)

The model GLORE, proposed by Yuan in 2012, iterates the parameters in a distributive fashion, which is unlike the classical LR model that has limitations operating on federated data sets or on distributed data because is necessary to receive all the data from participants before training the model. By integrating intermediary aggregated information from locally hosted databases, servers can get the same result as normal without losing any information during this process.

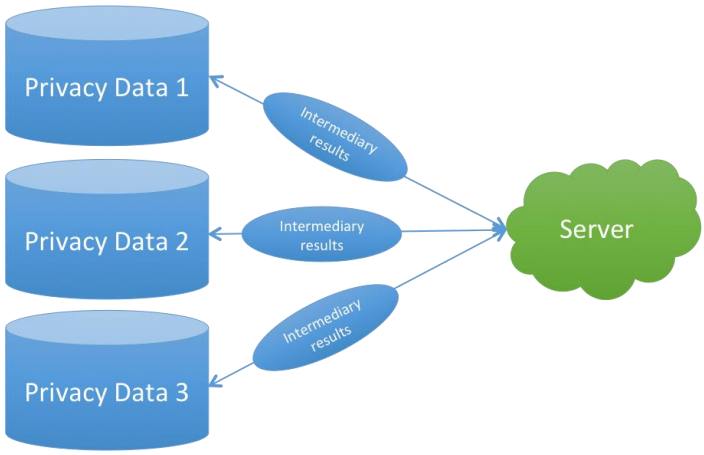

Fig. 4 GLORE

Fig. 4 shows the diagram of GLORE's workflow. The represented process supposes that I have three different institutions, each with its own database. The database record has the same attributions of a patient. Each inistitution can compute the intermediary results based on its own data and send the results to a central server. After receiving all the data from the institutions, the central server updates the training parameters and sends that data to all the institutions.

Table 1 GLORE algorithm

Input: Each party p_j provides encrypted data $\boldsymbol{X}^j$.
Output: Learned model coefficients $\boldsymbol{\beta}$

1. Initialize $\boldsymbol{\beta}^{(s)}, s = 0$ as an all-zero vector on both server and local
2. **Repeat steps 3-6 until the parameters converge**
3. Local party i: Compute a local Hessian matrix H_i and first derivative D_i. Then send them to the server.
4. Server: After receiving data from all local parties, calculate the inverse of global Hessian matrixH^{-1} and global first derivative $\boldsymbol{D}$. Then update $\boldsymbol{\beta}$ based on Equation (3)
5. Send the new $\boldsymbol{\beta}^{s+1}$ to all local parties.
6. $s = s + 1$

Table 1 shows the procedure for GLORE. The following equations explain how GLORE works. Supposing that I have l institutions.

$$\begin{aligned}\boldsymbol{\beta}^{(t+1)} &= \boldsymbol{\beta}^{(t)} - \left[l''\left(\boldsymbol{\beta}^{(t)}\right)\right]^{-1} l'\left(\boldsymbol{\beta}^{(t)}\right) \\ &= \boldsymbol{\beta}^{(t)} + \left[\overline{\boldsymbol{X}}^T \boldsymbol{W}(\overline{\boldsymbol{X}}, \boldsymbol{\beta}^{(t)}\overline{\boldsymbol{X}})\right]^{-1} \overline{\boldsymbol{X}}^T [\overline{\boldsymbol{Y}} - P(\overline{\boldsymbol{X}}, \boldsymbol{\beta}^{(t)})] \\ &= \boldsymbol{\beta}^{(t)} + \left[\sum_{t=1}^{l} \overline{\boldsymbol{X}}_{\boldsymbol{t}}^{\boldsymbol{T}} \boldsymbol{W}_{\boldsymbol{t}}(\overline{\boldsymbol{X}}_{\boldsymbol{t}}, \boldsymbol{\beta}^{(t)}\overline{\boldsymbol{X}}_t)\right]^{-1} \cdot \{\sum_{t=1}^{l} \overline{\boldsymbol{X}}_{\boldsymbol{t}}^{\boldsymbol{T}} [\overline{\boldsymbol{Y}}_t - P(\overline{X}_t, \boldsymbol{\beta}^{(t)})]\}\end{aligned} \tag{4}$$

where $\boldsymbol{\beta}$ is the weight vector, $\boldsymbol{W}$ is a diagonal matrix made up of elements calculated from probabilities.

Equation (4) shows that the computation can be decomposed at each iterative step. In other words, the first and second derivatives of the log likelihood function can be calculated separately for a subset of observations and then combined with the same result as if they were calculated on the complete set. So combining intermediary results from all local sites can finish the update step of GLORE. Due to the intermediary results from individual sites without losing any information, GLORE can still get the accurate estimation of parameters through summation. GLORE provides the advantage that data are transmitted as aggregated, intermediary results, rather than the raw data; hence, GLORE preserves data privacy.

3.3 Vertical gird logistic regression (VERTIGO)

Another grid logistic regression is VERTIGO that corresponds to vertically partitioned databases. It also does not need to transmit the patient data to server VERTIGO works by taking the standard LR model's format as a primal and solving the decomposable dual format. The following is a description of a brief proof [14] of dual problem of logistic regression.

Given a training dataset $\{y_i, x_i\}, i = 1, \cdots, m, x_i \in R^{n\times 1}$, the MLE parameter solution β of the logistic regression model can be derived by maximizing the following penalized likelihood function:

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{m} \log p(y_i|z_i) - \frac{\lambda}{2}\boldsymbol{\beta}^T\boldsymbol{\beta} \tag{5}$$

where $z_i = \boldsymbol{\beta}^T x_i$, $p(y_i|z_i) = \tau(y_i\boldsymbol{\beta}^T x_i) = 1/(1+\exp(-y_i\boldsymbol{\beta}^T x_i))$, and λ is a regularization parameter which is the weight of penalty for over fitting. The item in the covariance matrix is $Cov = x_i^T x_j$, and the solution $\boldsymbol{\beta}$ is unique since $l(\boldsymbol{\beta})$ is strictly concave in $\boldsymbol{\beta}$. With $\eta_i = y_i z_i$, it can be proved that $\log p(y_i|z_i)$ is a concave, continuous, differentiable function of η_i. According to the Legendre transformation [15], there exists a function $L(\alpha_i)$ such that the following two equations hold:

$$\log \tau(\eta_i) = \min_{\alpha_i}\{\alpha_i\eta_i - L(\alpha_i)\} \tag{6}$$

$$L(\alpha_i) = \min_{\eta_i}\{\alpha_i\eta_i - \log \tau(\eta_i)\} \tag{7}$$

The function $L(\alpha_i)$ is known as the *conjugate function*. The next step is to get the derived $L(\boldsymbol{\alpha})$ to compute the minimum value of $L(\boldsymbol{\alpha})$. The first step is to take the derivative with respect to η_i and equate it to zero:

$$\alpha_i - \frac{1}{\exp(\eta_i) + 1} = 0$$

Then

$$\eta_i = \log\left(\frac{1}{\alpha_i} - 1\right), 0 < \alpha_i < 1$$

Substituting η_i back into Eq. (7), I have:

$$L(\alpha_i) = \alpha_i \log\left(\frac{1}{\alpha_i} - 1\right) - \log(1 - \alpha_i) = -\alpha_i \log(\alpha_i) - (1 - \alpha_i)\log(1 - \alpha_i) \quad (8)$$

Therefore, we get the new objective function by combine Eq. (1) and Eq. (2),

$$l(\boldsymbol{\beta}, \boldsymbol{\alpha}) = \sum_{i=1}^{m} \{\alpha_i y_i \boldsymbol{\beta}^T \boldsymbol{x_i} - L(\alpha_i)\} - \frac{\lambda}{2}\boldsymbol{\beta}^T\boldsymbol{\beta} \quad (9)$$

It can be demonstrated that $\max_{\beta}\min_{\alpha} l(\boldsymbol{\beta}, \boldsymbol{\alpha}) = \min_{\alpha}\max_{\beta} l(\boldsymbol{\beta}, \boldsymbol{\alpha})$, and $J(\boldsymbol{\alpha}) = \max_{\beta} l(\boldsymbol{\beta}, \boldsymbol{\alpha})$. Recall that $z_i = \boldsymbol{\beta}^T x_i$, which indicates that for any fixed setting of $\boldsymbol{\alpha}$, Eq. (9) is a quadratic function of the parameter $\boldsymbol{\beta}$. Thus by taking the derivative of Eq. (9) with respect to $\boldsymbol{\beta}$ and equate it to zero, I can get the maximization of $\boldsymbol{\beta}$:

$$\sum_{i=1}^{m} \{\alpha_i y_i \boldsymbol{x_i} - 0\} - \lambda\boldsymbol{\beta} = 0 \quad (10)$$

Therefore, $\boldsymbol{\beta} = \lambda^{-1}\sum_i \alpha_i y_i x_i$ and substituting $\boldsymbol{\beta}$ back into $l(\boldsymbol{\beta}, \boldsymbol{\alpha})$, I derive the dual problem:

$$\min_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}) = \frac{1}{2\lambda}\sum_{i,j} \alpha_i\, \alpha_j\, y_i\, y_j\, \boldsymbol{x_j^T}\, \boldsymbol{x_i} - \sum_i L(\alpha_i) \quad (11)$$

Table 2 is the detailed procedure of the algorithm of VERTIGO. The variable E in Table 2 is computed by $E^i = Y\beta^i(\alpha^s)X,^i$ where the variable $\boldsymbol{\alpha}$ is the dual parameter. Because our dataset has more features compared with the sample size, I implemented a regularized logistic regression to avoid over fitting. If I do not need the effect of regularization, just let λ be enough smaller when running the algorithm.

Table 2 VERTIGO Algorithm

Input: Local data in each party X^i, Y^i
Output: Learned model coefficients $\boldsymbol{\beta}$

1. Initialize dual solution at server and local parties with $\alpha^s, s = 0$ and parameter λ.
2. Local party i: compute a local linear kernel matrix $K^i = X^i X^{i'}$ then send them to Server.
3. Server: calculate the global gram matrix K which is necessary to compute the first derivative of the dual likelihood function $J\ (\boldsymbol{\alpha})$.
4. **Repeat steps 5-8 until the parameters converge**
5. Local party i: compute E^i, and then send it to Server.
6. Server: after receiving data from all local parties, calculating the inverse of hessian matrix H^{-1} and global first derivative D,
7. Then update $\boldsymbol{\alpha}$ based on Equation (3), and send the new $\boldsymbol{\alpha}^{s+1}$ to all local parties.
8. $s = s + 1$
9. $\boldsymbol{\beta} = \boldsymbol{\lambda^{-1}} \sum_i \boldsymbol{\alpha_i x_i y_i}$

3.4 Summary of GLORE and VERTIGO

After knowing how GLORE and VERTIGO work, it is obvious that only data partitioned horizontally or vertically can be used. In practice, there are many other situations that require a more flexible method can handle any partitioned data. In this case, there is no guarantee that some algorithm can deliver the same advantage of only aggregated intermediary result needed be passed to the server. So we came up with a more simple idea that aligning all the data at the server after receiving all the data from clients with more information about the position in the whole database. At the same time, the security is promised by implementing it on Intel® SGX.

Chapter 4: Secure Multi-Party Logistic Regression

For secure multiparty computation, Yao's garbled circuit that was proposed in 1986 is an example. I will first introduce the basic concept of garbled circuits and how GLORE was implemented on it. Finally, I will explain how to transfer all the models to Intel® SGX will be described.

4.1 Garbled circuit

The key idea of circuit-based computation is based on the fact that operations in almost all modern digital computers are implemented by circuits combining basic logic gates such as AND, OR, NOT etc., where inputs and outputs of a gate may be TRUE or FALSE for certain propositions. Then, one can design a garbled circuit counterpart [16] to protect the data and the computation. Fig. 5 shows an example of diagnosing gestational diabetes based on blood glucose level (BGL) in a standard circuit representation, which consists of three gates (i.e., G_i with $i = 1, 2, 3$) and six wires (i.e., w_j with $j = 1, 2, \dots, 6$). Using Boolean algebra [17] and truth tables of the three basic gates shown in the figure, the circuit can calculate (gestational diabetes) = (NOT (non-pregnant women)) AND ((fasting BGL ≥ 95 mg/dl) OR (1 hour BGL≥ 180 mg/dl)). In theory, one can build circuits of any complexity using basic logical gates to evaluate functions (e.g., secure distributed logarithm, exponent, etc.) or algorithms (e.g., secure distributed logistic regression models in this section). A garbled circuit [18], [2] is a specially designed circuit, which enables two (or more) parties to securely compute a function $f(x_A, x_B)$ without exposing their private secrets (e.g., x_A and x_B are inputs from party A and party B, respectively). All parties here are assumed to be semi-honest

collaborators, which means that they follow the protocol honestly but may try to deduce additional information from the received messages during the protocol execution.

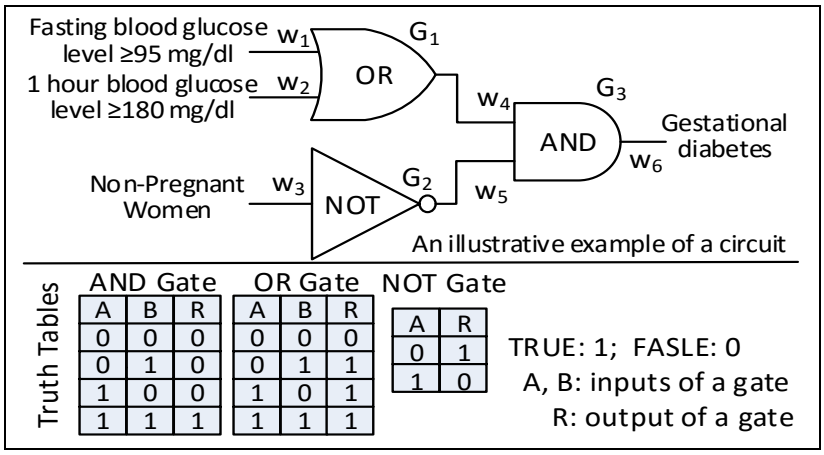

Fig. 5 An example of a garbled circuit

4.2 GLORE on garbled circuits

4.2.1 Matrix Multiplication

I transferred the matrix inversion problem into an iterative procedure of matrix multiplication and addition. Optimizing the implementation of matrix multiplication would definitely improve the efficiency of the proposed framework. In this subsection, I adopted the Strassen algorithm for matrix multiplication.

Let us denote A and B two square $n \times n$ matrices and $C = AB$ their matrix product. Here, n is suggested to be a power of 2, namely $n = 2^k$, $k \in \mathbb{N}$. A, B, and C can be partitioned into equally sized block matrices as

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{A}_{2,1} & \boldsymbol{A}_{2,2} \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_{1,1} & \boldsymbol{B}_{1,2} \\ \boldsymbol{B}_{2,1} & \boldsymbol{B}_{2,2} \end{bmatrix}, \boldsymbol{C} = \begin{bmatrix} \boldsymbol{C}_{1,1} & \boldsymbol{C}_{1,2} \\ \boldsymbol{C}_{2,1} & \boldsymbol{C}_{2,2} \end{bmatrix}, \tag{12}$$

where $A_{i,j}$, $B_{i,j,}$ and $C_{i,j}$ are all $(n/2) \times (n/2)$ matrices. According to the definition of block matrix multiplication, $C_{i,j}$ can be represented by $A_{i,j}$ and $B_{i,j}$ for $i, j = 1,2$.

$$\begin{aligned} \boldsymbol{C}_{1,1} &= \boldsymbol{A}_{1,1}\boldsymbol{B}_{1,1} + \boldsymbol{A}_{1,2}\boldsymbol{B}_{2,1}, \quad \boldsymbol{C}_{1,2} = \boldsymbol{A}_{1,1}\boldsymbol{B}_{1,2} + \boldsymbol{A}_{1,2}\boldsymbol{B}_{2,2}, \\ \boldsymbol{C}_{2,1} &= \boldsymbol{A}_{2,1}\boldsymbol{B}_{1,1} + \boldsymbol{A}_{2,2}\boldsymbol{B}_{2,1}, \quad \boldsymbol{C}_{2,2} = \boldsymbol{A}_{2,1}\boldsymbol{B}_{1,2} + \boldsymbol{A}_{2,2}\boldsymbol{B}_{2,2}. \end{aligned} \tag{13}$$

According to (13), the calculation of C requires the same number of multiplications as the standard definition of matrix multiplication $\mathrm{C} = \mathrm{AB}$. To reduce the number of multiplications, I introduced the Strassen algorithm, which defines some new matrices based on $\mathrm{A_{i,j}}$ and $\mathrm{B_{i,j}}$.

$$\begin{gathered} \boldsymbol{M}_1 = \boldsymbol{A}_{1,1}(\boldsymbol{B}_{1,2} - \boldsymbol{B}_{2,2}),\ \ \boldsymbol{M}_2 = (\boldsymbol{A}_{1,1} + \boldsymbol{A}_{1,2})\boldsymbol{B}_{2,2}, \\ \boldsymbol{M}_3 = (\boldsymbol{A}_{2,1} + \boldsymbol{A}_{2,2})\boldsymbol{B}_{1,1},\ \ \boldsymbol{M}_4 = \boldsymbol{A}_{2,2}(\boldsymbol{B}_{2,1} - \boldsymbol{B}_{1,1}), \\ \boldsymbol{M}_5 = (\boldsymbol{A}_{1,1} + \boldsymbol{A}_{2,2})(\boldsymbol{B}_{1,1} + \boldsymbol{B}_{2,2}),\ \ \boldsymbol{M}_6 = (\boldsymbol{A}_{1,2} - \boldsymbol{A}_{2,2})(\boldsymbol{B}_{2,1} + \boldsymbol{B}_{2,2}), \\ \boldsymbol{M}_7 = (\boldsymbol{A}_{1,1} - \boldsymbol{A}_{2,1})(\boldsymbol{B}_{1,1} + \boldsymbol{B}_{1,2}) \end{gathered} \tag{14}$$

Equation (14) requires only 7 matrix multiplications between $(\mathrm{n}/2) \times (\mathrm{n}/2)$ square matrices (one for each $\mathrm{M_l}, \mathrm{l} = 1, \cdots, 7$) to calculate $\mathrm{C} = \mathrm{AB}$, which reduces the number of multiplications by $\mathrm{n}^3/8$. The product C can be recovered from $\mathrm{M_l}$, $\mathrm{l} = 1, \cdots, 7$ by

$$\begin{gathered} \boldsymbol{C}_{1,1} = \boldsymbol{M}_5 + \boldsymbol{M}_4 - \boldsymbol{M}_2 + \boldsymbol{M}_6,\ \ \boldsymbol{C}_{1,2} = \boldsymbol{M}_1 + \boldsymbol{M}_2, \\ \boldsymbol{C}_{2,1} = \boldsymbol{M}_3 + \boldsymbol{M}_4,\ \ \boldsymbol{C}_{2,2} = \boldsymbol{M}_5 + \boldsymbol{M}_1 - \boldsymbol{M}_3 - \boldsymbol{M}_7. \end{gathered} \tag{15}$$

The matrices can be iteratively partitioned k times, when $\mathrm{n} = 2^{\mathrm{k}}$. Thus, the Strassen algorithm can reduce the complexity of matrix multiplication from $\mathrm{O}(\mathrm{n}^3)$ to $\mathrm{O}\ (\mathrm{n}^{2.8})$.

In this work, the Strassen algorithm is implemented for matrix inversion, which has a significant effect on computational complexity. For other ordinary matrix multiplications, I still used the standard method. However, it is also possible to extend the Strassen algorithm to employ it in ordinary matrix multiplication, e.g. multiplication between non-square matrices.

4.2.2 The First Derivative of the Maximum Likelihood Function

In the t-th iteration, the first derivative $\mathrm{l}'(\boldsymbol{\beta})$ of the maximum likelihood function has to be updated with current $\boldsymbol{\beta}^{(\mathrm{t})}$. The k-th element of $\mathrm{l}'(\boldsymbol{\beta})$ can be obtained in a distributed manner.

$$\frac{\partial l}{\partial \beta(k)}\Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(t)}} = \sum_{j=1}^{h}\sum_{i=1}^{n_i}\left(y_i^j - P\left(\boldsymbol{x}_i^j, \boldsymbol{\beta}^{(t)}\right)\right)x_i^j(k), \tag{16}$$

where $x_i^j(k)$ and $\beta(k)$ are the k-th element of x_i^j and $\beta^{(t)}$, respectively. Equation (13) shows that I can allow each party to separately compute its own part of the first derivative based on local data, and I add these results [1]. However, such an approach will leak the information of $\beta^{(t)}$ at each iteration. Therefore, I need to securely evaluate (16) without releasing any intermediary $\beta^{(t)}$. As a result, I need to securely evaluate the exponential function e^x in the garble circuits. In the proposed framework, I used the Taylor series to approximate the evaluation of $e,^x$ such that I only need to handle multiplication and addition operations.

$$e^x = \sum_{i=1}^{\infty}\frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots \tag{17}$$

To simplify the computation and avoid overflow, I set a filter to bound the exponential within the interval between -5 and 5. When the exponentials are greater than 5 or smaller than -5, the evaluation results of the logit function (i.e., $1/(1+e^x)$) would be smaller than 6.7×10^{-2} according to (1). Thus, I will not lose considerable accuracy by using this bound.

Simulations in MATLAB show that the Taylor series could achieve an approximated result with an error less than $10,^{-2}$ when the maximal order for the expansion is set to 15. To reduce the number of multiplications, I transformed the Taylor series to a recursion algorithm.

$$e^x \approx 1 + x\left(1 + \frac{x}{2}\left(\ldots\left(1 + \frac{x}{14}\left(1 + \frac{x}{15}\right)\right)\right)\right). \tag{18}$$

I also built a look-up table storing the inversion of integers from 1 to 15 to avoid divisions and to speed up the calculation. For the other divisions involved in the logistic function, I treated them as a matrix with size 1.

It is worth mentioning that all the above computations in this section are carried out in a customized Boolean circuit, where the OT protocol and the garbled circuits protect all the inputs and intermediary information exchange. The only outputs in plaintext are the learned model parameter β in the proposed SMAC-GLORE.

4.3 GLORE, VERTIGO and AGILE on Intel® SGX

I developed a distributed system based on Intel® SGX by implementing logistic regression as a demonstration. By designing three forms of data construction, one is assuming that the data are divided horizontally, consistent with the GLORE model. Another form is dividing whole data vertically, consistent with the VERTIGO. The last framework considers that data can be partitioned horizontally and vertically. That is my AGILE model. The primary data set used to validate the methods AGILE, GLORE, and VERTIGO is the Edinburgh database; more detail is presented in the next section.

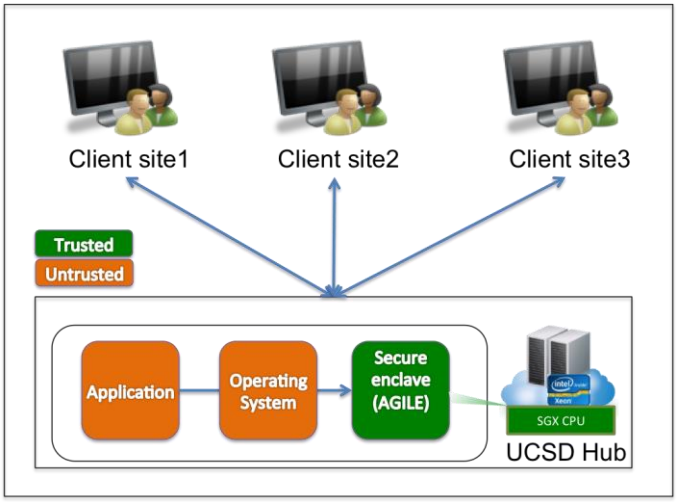

Fig. 6 Framework of AGILE using SGX

Fig. 6 shows the AGILE framework using Intel® SGX. Suppose that three clients have their own privacy data. Now they want to contribute their data for research. Data encrypted before transmission can only be decrypted within an enclave in the UCSD hub. All data and algorithm are stored and executed within the protected memory, so security is promised.

Table 3 AGILE Algorithm

Input: Local data in each party X^i, Y^i
Output: Learned model coefficients $\boldsymbol{\beta}$

1. Initialize solution at server and local parties with $\boldsymbol{\beta}^s, s = 0$.
2. Local party i: send data X^i with observation id and max, min of attribute number to Server.
3. Server: scan all received data to get the dimension of the whole data then filling the data matrix with received data from local party through side information
4. **Repeat steps 5-7 until the parameters converge**
5. Server: after receiving data from all local parties, calculating the inverse of hessian matrix H^{-1} and global first derivative D,
6. Then update $\boldsymbol{\beta}$ based on Equation (3)
7. $s = s + 1$
8. **Send $\boldsymbol{\beta}$ to all local parties**

The other two models are implemented the same as AGILE. Putting all the sensitive data and code within the enclave and all the intermediary result transmitted outside the enclave is encrypted. Table 3 contains the algorithm procedure for AGILE where the alignment algorithm will be more specifically stated below as Table 4 shows.

Table 4 Alignment Algorithm

Input: Data with side information from each participant (Client)
Output: Whole data matrix

1. Iterative all clients to get the dimension of whole data and initialize the four variables: $min\_feature\_label, max\_feature\_label, min\_patient\_id, max\_patient\_id$. These will be used to locate the position of the data in the whole data matrix.
2. Iterative each client with k
3. Iterative each element in client with (i, j)
4. Row = clients [k]->feature_label [i] - min_feature_label
5. Column = clients [k]->patient_id [j] – min_patient_id
6. Whole [row][column]=client [i][j]
7. End

Chapter 5: Results and Conclusions

In this section, we will conduct our experiments on GLORE, VERTIGO and AGILE for two different real datasets separately by implementing all of these three models based on Intel® SGX.

5.1 Environment

Our server is hosted on UCSD-DBMI with a Windows 2012 operating system, an SGX-capable 3.60 GHz Xeon CPU and 64GB memory. Local party are simulated by using Amazon EC2 instances with the type of t2.micro, where these sites are located at California, Virginia, Ireland, Singapore, and San Paulo. Below is the data partition pattern for our three models; the horizontally partitioned data corresponding to GLORE, for the model of VERTIGO, the vertically partitioned data will be used; the last three scenes are designed for our own model AGILE which stand for any partition of the data.

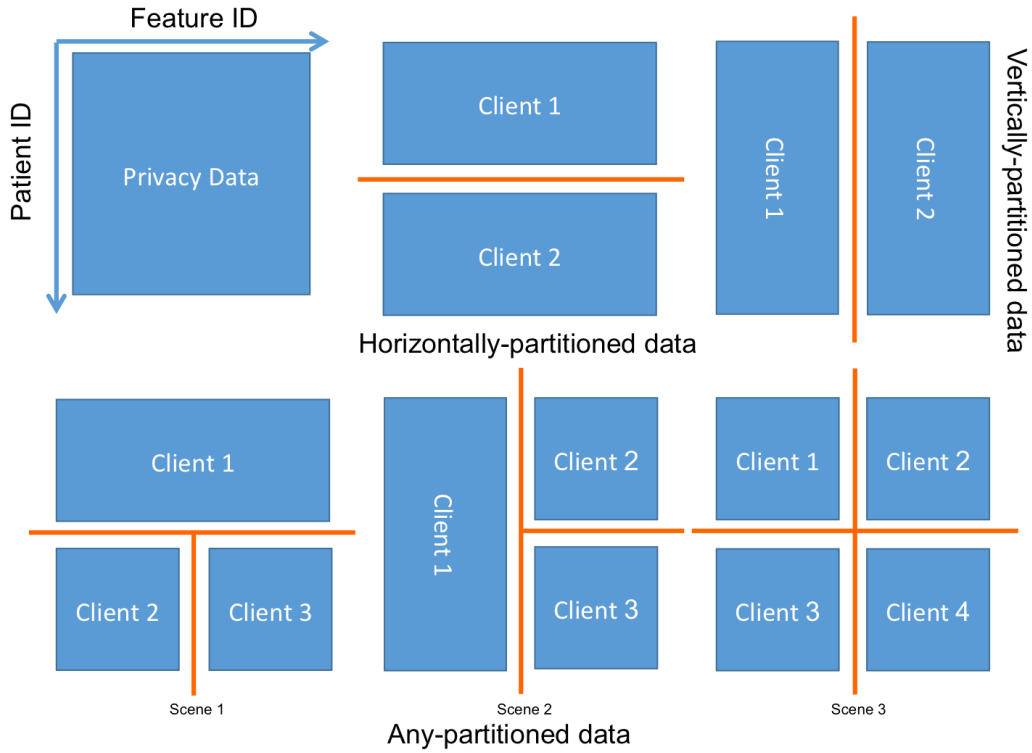

Fig. 7 Data partition for different models

5.2 Experiment

5.2.1 GLORE on Garbled circuits and Intel® SGX

This purpose of this experiment is to demonstrate that at the same settings, running time of GLORE is considerably different on two platforms as Table 5 shows. The dataset of this experiment was Edinburgh dataset whose outcome is binary. This dataset totally has 48 features and 1253 records, 3 features (T wave inversion, Sweating and Pain in right arm) plus one bias feature which are the same as our previous work (SMAC-GLORE) were selected in current experiment.

Table 5 Average running time for GLORE on Intel® SGX and GC

Method	**Number of Parties**		
	2 parties	3 parties	4 parties
GC	3545.36(sec.)	5954.16(sec.)	12029.76(sec.)
Intel® SGX	17.607(sec.)	19.091(sec.)	23.13(sec.)

As Table 5 shows, the time increased faster in GC than in Intel® SGX. Also Intel® SGX has an overwhelming advantage over GC in time and cost. Moreover, memory allocation and estimation error were considered in our next experiment. Table 6 shows that my Intel® SGX-based GLORE has less memory cost and error compared with GC. The baseline of the model I ran on MATLAB.

Table 6 Performance comparison among different implementation of GLORE

	GLORE	**SMAC-GLORE**	**SGX-GLORE**
Run Time (seconds)	0.516	12029.760	0.500
Memory	0.012MB	43GB	0.018MB (enclave only)
Estimation error (SAD)	0	5.100x10E-04	1.342x10E-05

5.2.2 GLORE, VERTIGO and AGILE on Edinburgh

I evaluated the algorithms on Edinburgh dataset with different numbers of participants. Nine non-redundant features in this data set are selected by using methods described in Kennedy et al. [19]. By setting the same converge threshold as 10^{-8} and initial parameter with 10^{-6}, I tested the algorithms without regularization about 30 times to get the average running time. The result is stated as Fig. 8. Each box corresponds to a model. showing the running time of AGILE in three different common scenes. In the middle of FIG. 8 is the running result of the GLORE model. The VERTIGO box shows the algorithm in different participants such as 1, 3, and 5. VERTIGO performance is evaluated at the same setting as GLORE.

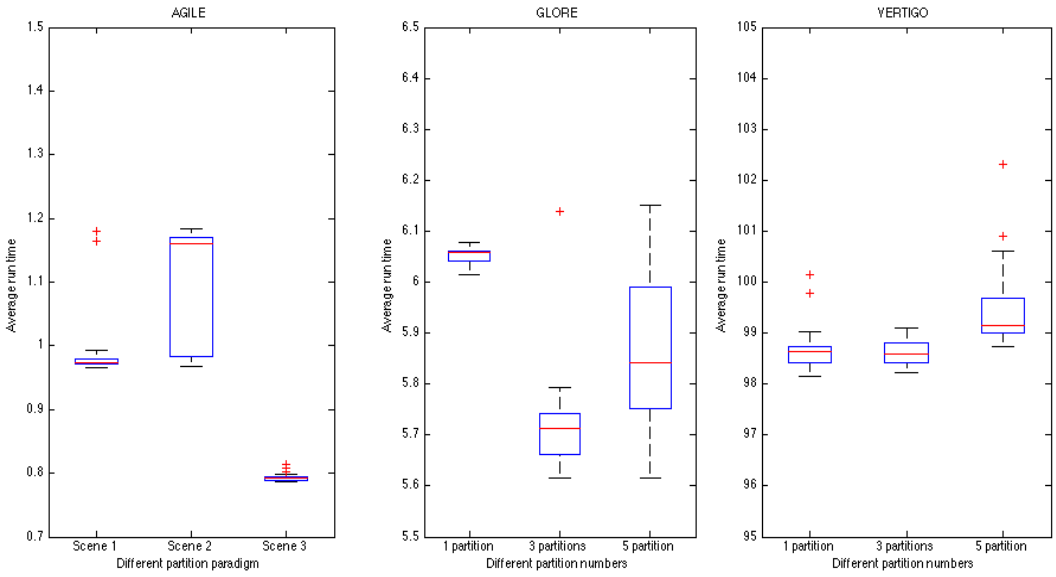

Fig. 8 Converge time for different Models in Edinbugh dataset

To delve into the result of these three models in one situation, I took 3 participants of AGILE, GLORE and VERTIGO. For the evaluation, the estimated model parameters β were compared with their centralized counterpart β using the

summation of the absolute difference (SAD) $\sum_{m=1}^{M} |\hat{\beta}(m) - \beta\ (m)|$, where m=1, M corresponds to the index of different covariates. Except for SAD, I got the iterations of convergence and average running time for each model as Table 7 shows. The centralized model is the MATLAB built-in LR function. My proposed model outperformed efficiency in both time cost, and accuracy. On average, my model is 6 times faster than GLORE and 100 times faster than VERTIGO.

Table 7 Estimation error, iterations of convergence and time costs for AGILE, GLORE and VERTIGO

Dataset	Methods	Iterations of Convergence	Estimation error (SAD)	Time Cost (second)
Edinburgh	Centralized	8	0	0.361789
	AGILE	8	2.610×10^{-4}	1.0991
	GLORE	8	2.6510×10^{-4}	5.7156
	VERTIGO	14	8.3510×10^{-4}	98.6286

5.2.3 GLORE, VERTIGO and AGILE on Genome

Another dataset I used to analyze the model is Genome data GSE3494. By removing 15 patients whose survival status was unknown, I took 236 patients; each had 222283 gene expression features on platform GPL96 and 22654 gene expression features on platform GPL97. Picking top 15 features based on the ranking of their P values (t test), as suggested by Osl[20] from each platform, I had 41 features by adding 10-dimensional phenotype features. Considering that the 41 features with only 236 patients, I ran the algorithm with regularization by taking $\lambda = 2.0$. Fig. 9 shows the results of my applying the same running environment as Edinburgh does. Also, on average, AGILE is the best among these models when considering converge time. The reason why the time cost between model GLORE and VERTIGO is not as big as previous experiments is because of the different dimension of the dataset. The time cost

for both models heavily depends on the time of operating matrix inversion. Also, the matrix size depends on the number of features on GLORE, rather than on VERTIGO whose matrix size is decided by the number of observations. So the size of matrix in the Edinburgh dataset is 1253×1253, where the size in the genome dataset is 236×236.

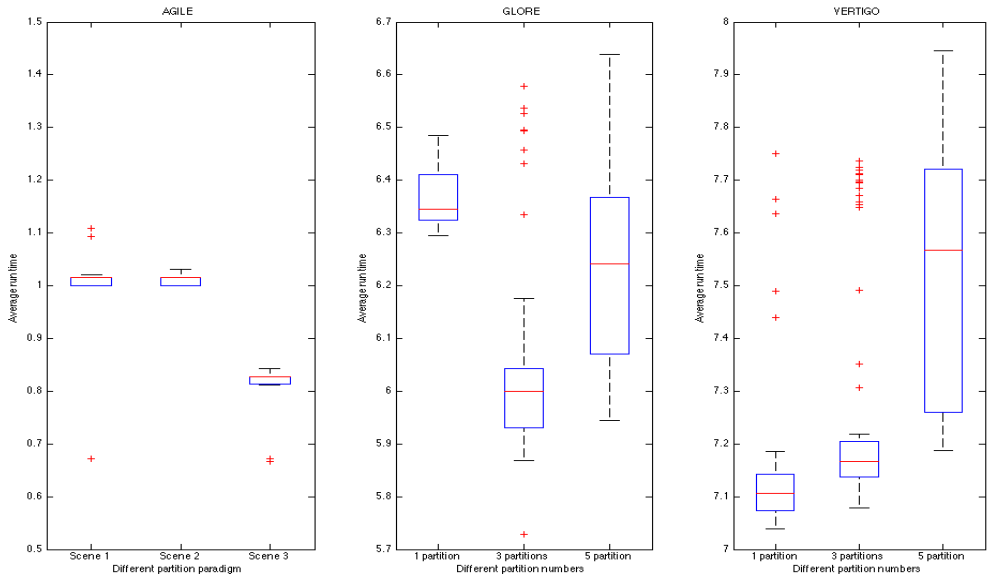

Fig. 9 Converge time for different Models in Genome dataset

As established in the previous experiment that AGILE will get the same result as the centralized model. This time I estimated the error based on AGILE. As Table 8 shows, the error and time cost are similar for GLORE and VERTIGO even though there is a big difference in iteration of convergence.

Table 8 Estimation error, iterations of convergence and time costs for GLORE and VERTIGO

Dataset	**Methods**	**Iterations of convergence**	**Estimation error vs. AGILE (SAD)**	**Time Cost (second)**
Genome	GLORE	8	2.610-6	6.0252
	VERTIGO	16	2.610-6	7.2342

5.3 Conclusions

Methods implemented both on GC and Intel® SGX have their own advantage. Although GC cost more time on running the same models GLORE, VERTIGO and AGILE, it does not need any specific hardware support such as the Intel® SGX enabled platform. Also GLORE and VERTIGO did not share the patient-level data when training the model compared with AGILE; however, my model is more flexible in data organization. Also by taking advantage of Intel® SGX, security is promised. But the enclave memory usually has limited size, typically $64MB$ or $128MB$ for now. In the future, a more extensible environment for trustworthy multi participants' computation can be built. By implementing more common practical models such as decision trees, EM algorithm, Cox model, etc. on Intel® SGX, users can more quickly and easily set up a practical model across different participants.

References

[1] Y. Wu, X. Jiang, J. Kim, and L. Ohno-Machado, “Grid binary LOgistic REgression (GLORE): Building shared models without sharing data.,” *J. Am. Med. Inform. Assoc.*, vol. 2012, no. 5, pp. 758–64, 2012.

[2] M. Bellare, V. T. Hoang, and P. Rogaway, “Foundations of garbled circuits,” in *Proc. 2012 ACM Conference on Computer and Communications Security - CCS ’12*, 2012, pp. 784–96.

[3] Y. Li, X. Jiang, S. Wang, H. Xiong, and L. Ohno-Machado, “VERTIcal grid lOgistic regression (VERTIGO),” *J. Am. Med. Informatics Assocation*, 2015, pp. 1–11.

[4] S. G. Choi, K. W. Hwang, J. Katz, T. Malkin, and D. Rubenstein, “Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7178 LNCS, pp. 416–432, 2012.

[5] S. J. Murdoch, “Introduction to Trusted Execution Environments (TEE) – IY5606.”

[6] Intel Corp., “Software Guard Extensions Programming Reference,” *Intel Corp.*, no. Ref. #329298–002, 2014.

[7] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, “Innovative instructions and software model for isolated execution.” *Proc. 2nd Int. Work. Hardw. Archit. Support Secur. Priv. - HASP ’13*, pp. 1–1, 2013.

[8] NIST, “Secure Hash Standard (SHS),” *Fed. Inf. Process. Stand.* 180-4.

[9] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, “Innovative Technology for CPU Based Attestation and Sealing,” *HASP - Proc. 2nd Int. Work. Hardw. Archit. Support Secur. Priv.*, 2013, pp. 1–7.

[10] T. Platform and M. Library, “Trusted Platform Module Library Part 1: Architecture,” 2014.

[11] T. Platform and M. Library, “Trusted Platform Module Library Part 2: Structures,” 2014.

[12] E. Brickell and J. Li, “Enhanced Privacy ID using Bilinear Pairing,” *Identity*, no. June, pp. 1–8, 2008.

[13] “Diffie–Hellman key exchange,” *Wikipedia*, 2016.

[14] T. P. Minka, “A comparison of numerical optimizers for logistic regression,”

October, vol. 2003, no. 4, pp. 1–18, 2007.

[15] R. T. Rockafellar, *Convex Analysis*. 1970.

[16] M. Gymrek, A. L. McGuire, D. Golan, E. Halperin, and Y. Erlich, "Identifying personal genomes by surname inference," *Science (80-.).*, vol. 339, no. 6117, pp. 321–324, Jan. 2013.

[17] G. Boolos, J. P. Burgess, and R. C. Jeffrey, *Computability and logic*. Cambridge University Press, 2002.

[18] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 1982, pp. 160–164.

[19] R. L. Kennedy, A. M. Burton, H. S. Fraser, L. N. Mcstay, and R. F. Harrisonf, "Early diagnosis of acute myocardial infarction using clinical and electrocardiographic data at presentation : derivation and evaluation of logistic regression models," pp. 1181–1191, 1996.

[20] M. Osl, S. Dreiseitl, J. Kim, K. Patel, C. Baumgartner, and L. Ohno-Machado, "Effect of data combination on predictive modeling: a study using gene expression data," *AMIA Annu. Symp. Proc.*, vol. 2010, pp. 567–571, 2010.