Practical Privacy-Preserving Gaussian Process Regression via Secret Sharing

Jinglong Luo[1,2] **Yehong Zhang**[*2] **Jiaqi Zhang**[2] **Shuang Qin**[2] **Hui Wang**[2] **Yue Yu**[2] **Zenglin Xu**[*1,2]

[1]Harbin Institute of Technology, Shenzhen, China
[2]Peng Cheng Laboratory, Shenzhen, China

Abstract

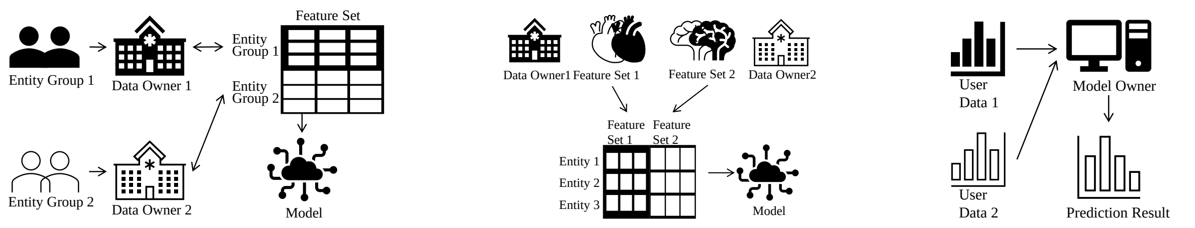
Gaussian process regression (GPR) is a non-parametric model that has been used in many real-world applications that involve sensitive personal data (e.g., healthcare, finance, etc.) from multiple data owners. To fully and securely exploit the value of different data sources, this paper proposes a privacy-preserving GPR method based on *secret sharing* (SS), a *secure multi-party computation* (SMPC) technique. In contrast to existing studies that protect the data privacy of GPR via homomorphic encryption, differential privacy, or federated learning, our proposed method is more practical and can be used to preserve the data privacy of both the model inputs and outputs for various data-sharing scenarios (e.g., horizontally/vertically-partitioned data). However, it is non-trivial to directly apply SS on the conventional GPR algorithm, as it includes some operations whose accuracy and/or efficiency have not been well-enhanced in the current SMPC protocol. To address this issue, we derive a new SS-based exponentiation operation through the idea of "confusion-correction" and construct an SS-based matrix inversion algorithm based on Cholesky decomposition. More importantly, we theoretically analyze the communication cost and the security of the proposed SS-based operations. Empirical results show that our proposed method can achieve reasonable accuracy and efficiency under the premise of preserving data privacy.

1 INTRODUCTION

Gaussian process regression (GPR) [Rasmussen and Williams, 2006, Yan et al., 2011, Xu et al., 2015, Zhe et al., 2016, Zhang et al., 2016, Zhe et al., 2015] is a Bayesian non-parametric model that has been widely used in various real-world applications such as disease progression prediction [Ortmann et al., 2019, Shashikant et al., 2021], traffic prediction [Chen et al., 2015], and finance [Yang et al., 2015], etc. In practice, the data of the above applications may belong to different parties and cannot be shared directly due to the increasing privacy concerns in the *machine learning* (ML) community. For example, two hospitals that have a small amount of patient data would like to jointly construct a high-quality GPR model for better disease progression prediction. However, such data usually contain patients' personal information and cannot be shared between hospitals due to legal regulations. In addition, when other hospitals or patients consider to use the constructed GPR model for diagnosis, privacy leakage of the personal feature (i.e., test input) and the diagnostic result (i.e., model output) is also a concern. Similarly, in finance, a bank that owns users' financial behaviors (e.g., income, credit, etc.) may hope to build a GPR model for risk control prediction by exploiting the users' consuming behaviors from the e-commerce companies. Obviously, neither the financial nor the consuming behaviors of the users should be shared directly due to their high information privacy.

The need of information sharing in the above examples has motivated the development of a practical GPR model that can preserve the data privacy of both the model inputs and outputs in three data-sharing scenarios (Fig. 1): (a) *Horizontal data-sharing* (HDS): Each party has a set of data for different entities with the same features and share them for model construction; (b) *Vertical data-sharing* (VDS): Each party has different features of the same set of entities and shares them for model construction; and (c) *Prediction data-sharing* (PDS): A party who aims to use the constructed model needs to share his data with the model holder for prediction. At present, a few privacy enhancement techniques such as *fully homomorphic encryption* (FHE) [Gentry, 2009], *federated learning* (FL) [Konečný et al., 2016], and *differential privacy* (DP) [Dwork, 2006, Abadi et al.,

[*]Corresponding author



(a) Horizontal data-sharing (b) Vertical data-sharing (c) Prediction data-sharing

Figure 1: Diagrams of different data-sharing scenarios.

2016] have been exploited for avoiding privacy leakage in GPR. However, none of them is general enough to achieve privacy-preserving GPR for all three data-sharing scenarios. Specifically, the FHE-GPR [Fenner and Pyzer-Knapp, 2020] and FL-GPR [Dai et al., 2020, 2021, Kontoudis and Stilwell, 2022, Yue and Kontar, 2021] approaches only focus on PDS and HDS scenarios, respectively. The DP-GPR methods [Kharkovskii et al., 2020, Smith et al., 2018] assume all the data belong to a single party and can only protect the privacy of either the input features [Kharkovskii et al., 2020] or the outputs [Smith et al., 2018]. See Section 6 for detailed discussions.

To fully and securely exploit the value of different data sources in the aforementioned data-sharing scenarios for a GPR model, this paper proposes to exploit the *secure multi-party computation (SMPC)* [Yao, 1986] which can deal with different data-sharing scenarios with a theoretical security guarantee. Among the various types of SMPC approaches [Evans et al., 2018, Goldwasser, 1987, Yao, 1982], *secret sharing* (SS) [Shamir, 1979] is exploited in this work due to its good communication efficiency and widely applications in other ML models [Mohassel and Zhang, 2017, Wagh et al., 2019]. As the name implies, an SS-based ML approach converts all the original operations (e.g., addition, multiplication, comparison, etc.) in the ML models (e.g., neural network) into its privacy-preserving alternatives which take secretly shared data as input and produce secretly shared results with secure information communication among parties (see Section 2.2.1 for details).

Although many SS-based operations have been developed in existing privacy-preserving ML works, they are not sufficient for constructing a privacy-preserving GPR model since the matrix inversion and exponentiation operations are essential for GPR (Section 3) but have not been well adapted to SMPC. To be specific, some works [Knott et al., 2021] designed SMPC protocols of these two operations based on approximation methods such as Newton-Raphson iteration method and Taylor expansion, which significantly reduces their accuracy and/or efficiency. To address this issue, this work proposes new SMPC protocols for positive-definite matrix inversion and exponentiation based on SS and integrate them into the existing SS-based operations for achieving an efficient and theoretically secure GPR model.

The specific contributions of this work include:

- To the best of our knowledge, this is the first work that considers to protect the privacy of a GPR model via secret sharing which can be used for various data-sharing scenarios (Section 3).
- Based on the SS technique, we propose an efficient *privacy-preserving exponentiation* algorithm through the idea of "confusion-correction", which is shown to be $10 \sim 70$ times faster than commonly-used approximation algorithms and can achieve theoretical correctness and security guarantees (Section 4).
- We propose the first SS-based matrix inversion algorithm via Choseky decomposition and show that its accuracy is comparable to the plaintext algorithm with acceptable communication cost (Section 4).
- Empirical results on two real-world datasets show that the proposed SS-based GPR algorithm can achieve accurate prediction results within a reasonable time (Section 5).

2 BACKGROUND AND NOTATIONS

2.1 GAUSSIAN PROCESS REGRESSION (GPR)

Let $\mathcal{X}$ denote a d-dimensional input domain. For each $\mathbf{x} \in \mathcal{X}$, we assume its corresponding output $y(\mathbf{x}) \sim \mathcal{N}(f(\mathbf{x}), \sigma_n^2)$ is a noisy observation of a function $f(\mathbf{x})$ with noise variance σ_n^2. Then, the function $f(\mathbf{x})$ can be modeled using a *Gaussian process* (GP), that is, every finite subset of $\{f(\mathbf{x})\}_{\mathbf{x}\in\mathcal{X}}$ follows a multivariate Gaussian distribution. Such a GP is fully specified by its *prior* mean $\mu(\mathbf{x}) \triangleq \mathbb{E}[f(\mathbf{x})]$ and covariance $k(\mathbf{x}, \mathbf{x}') \triangleq \text{cov}[f(\mathbf{x}), f(\mathbf{x}')]$ for all $\mathbf{x}, \mathbf{x} \in \mathcal{X}$. Usually, we assume that $\mu(\mathbf{x}) = 0$ and the covariance is defined by a kernel function. One example of the widely-used kernel function is the *squared exponential* (SE) kernel:

$$k(\mathbf{x}, \mathbf{x}') \triangleq \sigma_s^2 \exp(-d(\mathbf{x}, \mathbf{x}')/2\ell^2) \tag{1}$$

where $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_2^2$, ℓ is the length-scale and σ_s^2 is the signal variance.

Supposing we have a set $\mathcal{D}$ of n observations: $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{i=1}^n\}$ where $y_i \triangleq y(\mathbf{x}_i)$, a GPR model can perform probabilistic regression by providing a predictive distribution $p(f(\mathbf{x}_*)|\mathcal{D}) \triangleq \mathcal{N}(\mu_{\mathbf{x}_*|\mathcal{D}}, \sigma^2_{\mathbf{x}_*|\mathcal{D}})$ for any test input $\mathbf{x}_* \in \mathcal{X}$. Let $\mathbf{X} \triangleq (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top$ be an $n \times d$ input matrix and $\mathbf{y} = (y_1, \ldots, y_n)^\top$ be a column vector of the n noisy outputs. Then, the *posterior* mean and variance of the predictive distribution $p(f(\mathbf{x}_*)|\mathcal{D})$ can be computed analytically:

$$\begin{aligned} \mu_{\mathbf{x}_*|\mathcal{D}} &\triangleq \mathbf{k}_*^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y}\ , \\ \sigma^2_{\mathbf{x}_*|\mathcal{D}} &\triangleq k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{k}_*, \end{aligned} \tag{2}$$

where $\mathbf{k}_* \triangleq k(\mathbf{x}_*, \mathbf{X}) = (k(\mathbf{x}_*, \mathbf{x}_i))_{i=1}^n$ is a column vector of n-dimension, $\mathbf{K} \triangleq k(\mathbf{X}, \mathbf{X}) = (k(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1,\ldots,n}$ is an $n \times n$ gram matrix, and $\mathbf{I}$ is an identity matrix of size n.

2.2 SECURE MULTI-PARTY COMPUTATION

Secure multi-party computation (SMPC) is a type of cryptography technique for multiple parties to jointly compute an operation f without exposing the privacy of the data to any of them during the computation. In this work, we adopt the *semi-honest security* (also known as *honest-but-curious*) model which is one of the standard security models in SMPC and has been widely used in existing privacy-preserving machine learning algorithms [Liu et al., 2017, Mohassel and Rindal, 2018, Mohassel and Zhang, 2017, Ryffel et al., 2020, Wagh et al., 2019]. In a *semi-honest* security model, the parties are assumed to follow the SMPC protocol but can try to use the obtained data-sharing and intermediate results to infer the information that is not exposed to them during the execution of the protocol. Next, we will first present a *secret sharing* technique designed based on the *semi-honest security* model to construct SMPC protocols and then, introduce the algebraic structure used for designing the SMPC protocal in this work.

2.2.1 Secret Sharing

Secret sharing (SS) is a technique independently proposed by Shamir [1979] and Blakley [1979] with its full name called (t, m)-threshold secret sharing schemes, where m is the number of parties and t is a threshold value. The security of SS requires that any less than t parties cannot obtain any secret information jointly. As a special case of secret sharing, $(2, 2)$-*additive* secret sharing contains two algorithms: $Shr(\cdot)$ and $Rec(\cdot, \cdot)$. Let $\mathcal{Z}_L$ denote the ring of integers modulo L and $[\![u]\!] = ([u]_0, [u]_1)$ be the additive share of any integer u on $\mathcal{Z}_L$. $Shr(u) \rightarrow ([u]_0, [u]_1)$ is used to generate the share by randomly selecting a number r from $\mathcal{Z}_L$, letting $[u]_0 = r$, and computing $[u]_1 = (u - r) \mod L$. Note that due to the randomness of r, neither a single $[u]_0$ nor $[u]_1$ can be used to infer the original value of u. The algorithm $Rec([u]_0, [u]_1) \rightarrow u$ is used to reconstruct the original value from the additive shares, which can be done by simply calculating $([u]_0 + [u]_1) \mod L$.

The additive secret sharing technique has been widely used to construct SMPC protocols for ML operations (e.g., addition, multiplication, etc.) such that both the inputs and outputs of the protocol can be *additive* shares of the original inputs and outputs: $\pi_f([inputs]_0, [inputs]_1) \rightarrow [f]_0, [f]_1$ where π_f denotes an SMPC protocol of the operation f. To further elaborate the SS technique, we briefly introduce the SS-based multiplication protocol below, which is essential in many privacy-preserving ML algorithms and will also be widely used in this work.

SS-based multiplication $u \cdot v$ **.** Let P_j with $j \in \{0, 1\}$ be two parties that are used to execute the SMPC protocol. Each party P_j will be given one additive share $([u]_j, [v]_j) \in \mathcal{Z}_L$ of the operation inputs for $j \in \{0, 1\}$. Then, the additive shares of $u \cdot v$ can be computed with Beaver-triples [Beaver, 1991]: (a, b, c) where $a, b \in \mathcal{Z}_L$ are randomly sampled from $\mathcal{Z}_L$ and $c = a \cdot b \mod L$. Specifically, for each $j \in \{0, 1\}$, P_j first calculates $[d]_j = [u]_j - [a]_j$ and $[e]_j = [v]_j - [b]_j$. Then, they send the $[d]_j$ and $[e]_j$ to each other and reconstruct $d = Rec([d]_0, [d]_1)$ and $e = Rec([e]_0, [e]_1)$. Finally, the additive share of $u \cdot v$ can be computed using $[u \cdot v]_j = -jd \cdot e + [u]_j \cdot e + d \cdot [v]_j + [c]_j$. To complete the SS-based multiplication, both parties need to spend 1 round of two-way communication and transmit two ring elements.

The SS-based multiplication protocol is extended to matrix multiplication in the work of Mohassel and Zhang [2017]. Let $\mathcal{F}_{matMul}$ denote the SS-based matrix multiplication functionality, $\mathbf{U}$ and $\mathbf{V}$ be two matrices of size $m \times n$ and $n \times k$, respectively. The SS-based matrix multiplication $\mathcal{F}_{matMul}(\mathbf{U}, \mathbf{V})$ still requires only 1 rounds of bidirectional communication between parties P_0 and P_1 but the transmitted ring elements are of size $(m + n) \times k$.

Unfortunately, there exist many operations (e.g., exponentiation, matrix inversion, etc.) that cannot be constructed using purely additive secret sharing on $\mathcal{Z}_L$. Some approximation methods such as Newton-Raphson iteration method and Taylor expansion have been exploited for designing additive SS-based protocols of these operations. Details of the approximation methods and other SMPC protocols can be found in the work of Knott et al. [2021].

2.2.2 Fixed-Point Representation

As has been shown above, the SS-based SMPC protocols are constructed in a ring of integers due to security reasons. In practice, the ML algorithms such as GPR are usually implemented using floating-point numbers. However, it has been shown that the SMPC protocols designed based on floating-point numbers are inefficient [Aliasgari et al., 2012] and fixed-point representation is a better choice.

Specifically, the fixed-point encoding method represents all data as l bits. Let $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$ be a set of fixed-point numbers with a precision of l_f (i.e., l_f fractional bits) mapped from

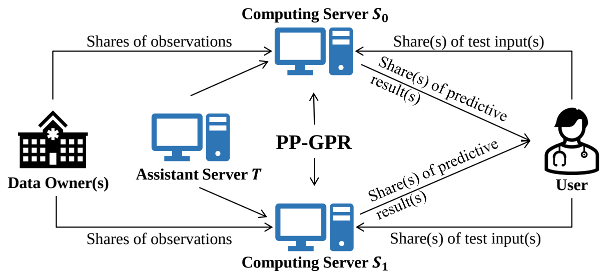


Figure 2: The overall framework of PP-GPR.

$\mathcal{Z}_L$ and $L = 2^l$. For floating-point numbers in the range[1] $[-2^{l-l_f-1}, 2^{l-l_f-1})$, this work will first round them to the nearest fixed-point numbers in $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$ and then, map them to the integers in $\mathcal{Z}_L$ by multiplying the converted fixed-point numbers with 2^{l_f}. For example, given $l = 5$ and $l_f = 3$, a floating-point number 1.125123 in $[-2, 2]$ is firstly rounded to the fixed-point number 1.125 in $\mathcal{Q}_{<\mathcal{Z}_{2^5}, 3>}$ and then converted to an element in $\mathcal{Z}_{2^5}$ by $(1.125 \times 2^3) \mod 2^5 = 9$. Conversely, an integer $11 \in \mathcal{Z}_{2^5}$ can be converted in $\mathcal{Q}_{<\mathcal{Z}_{2^5}, 3>}$ as $11/2^3 = 1.375$.

All the algorithms in this paper are performed on $\mathcal{Z}_L$ and $Q_{<\mathcal{Z}_L, l_f>}$. By choosing appropriate l and l_f, the fixed-point-based SMPC protocols can achieve a desirable compromise between efficiency and accuracy. To ease notations, we will use lowercase letters to represent either floating-point or integer numbers and $\check{x}$ to represent a fixed-point number in $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$ converted from x. The algorithm $Shr(x)$ in Section 2.2.1 will first convert x to its corresponding representation in $\mathcal{Z}_L$ if the input x is a floating-point number.

3 PRIVACY-PRESERVING GPR

In this section, we propose to exploit the SMPC technique for constructing a *privacy-preserving GPR* (PP-GPR) algorithm. The overall framework of the *Privacy-preserving GPR* (PP-GPR) model is shown in Fig. 2. As can be seen, the PP-GPR adopts a three-party SMPC architecture with two computing servers and one assistant server. Let S_0 and S_1 represent the two computing servers and T be the assistant server. Each computing server takes one additive share of the data as input, performs the calculations according to the steps of the PP-GPR algorithm, and then outputs the additive share of the GPR predictive results. The assistant server is responsible for generating random numbers required during the execution of the SS-based protocols in the PP-GPR algorithm. The proposed algorithm exploits the SS-based operations for achieving privacy-preserving GPR on all three data-sharing scenarios shown in Fig. 1. The complete steps are illustrated in Algorithm 1.

[1]We assume that all the numbers appeared in an ML algorithm are in this range. Appropriate l and l_f need to be selected for avoiding underflow and overflow issues.

Algorithm 1 Privacy-preserving GPR

Setup: The servers determine $\mathcal{Z}_L$ and $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$. The data owners convert their private observations $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ and predicted samples $\mathbf{x}_*$ into $(\llbracket \mathbf{x}_* \rrbracket, \llbracket \mathbf{X} \rrbracket, \llbracket \mathbf{y} \rrbracket)$.
Input: For $j \in \{0, 1\}$, S_j holds the shares $([\mathbf{x}_*]_j, [\mathbf{X}]_j, [\mathbf{y}]_j)$, and the hyperparameters $(\ell, \sigma_s^2, \sigma_n^2)$.

```
1:  // Model construction stage.
2:      $\llbracket d(\mathbf{X}, \mathbf{X}) \rrbracket \leftarrow \mathcal{F}_{dist}(\llbracket \mathbf{X} \rrbracket, \llbracket \mathbf{X} \rrbracket)$
3:      $\llbracket \mathbf{K} \rrbracket \leftarrow \sigma_s^2 \cdot \mathcal{F}_{PPExp}(\llbracket -d(\mathbf{X}, \mathbf{X})/2\ell^2 \rrbracket)$
4:      $\llbracket Inv \rrbracket \leftarrow \mathcal{F}_{MatInv}(\llbracket (\mathbf{K} + \sigma_n^2 \mathbf{I}) \rrbracket)$
5:  // Prediction stage.
6:      $\llbracket d(\mathbf{x}_*, \mathbf{X}) \rrbracket \leftarrow \mathcal{F}_{dist}(\llbracket \mathbf{x}_* \rrbracket, \llbracket \mathbf{X} \rrbracket)$
7:      $\llbracket d(\mathbf{x}_*, \mathbf{x}_*) \rrbracket \leftarrow \mathcal{F}_{dist}(\llbracket \mathbf{x}_* \rrbracket, \llbracket \mathbf{x}_* \rrbracket)$
8:    // Compute the kernel matrices.
9:      $\llbracket \mathbf{k}_* \rrbracket \leftarrow \sigma_s^2 \cdot \mathcal{F}_{PPExp}(\llbracket -d(\mathbf{x}_*, \mathbf{X})/2\ell^2 \rrbracket)$
10:     $\llbracket k(\mathbf{x}_*, \mathbf{x}_*) \rrbracket \leftarrow \sigma_s^2 \cdot \mathcal{F}_{PPExp}(\llbracket -d(\mathbf{x}_*, \mathbf{x}_*)/2\ell^2 \rrbracket)$
11:   // Compute the predictive mean and variance.
12:     $\llbracket \mu^2_{\mathbf{x}_*|\mathcal{D}} \rrbracket \leftarrow \mathcal{F}_{matMul}(\mathcal{F}_{matMul}(\llbracket \mathbf{k}_*^\top \rrbracket, \llbracket Inv \rrbracket), \llbracket \mathbf{y} \rrbracket)$
13:     $\llbracket \Lambda \rrbracket \leftarrow \mathcal{F}_{matMul}(\mathcal{F}_{matMul}(\llbracket \mathbf{k}_*^\top \rrbracket, \llbracket Inv \rrbracket), \llbracket \mathbf{k}_* \rrbracket)$
14:     $\llbracket \sigma^2_{\mathbf{x}_*|\mathcal{D}} \rrbracket \leftarrow \llbracket k(\mathbf{x}_*, \mathbf{x}_*) \rrbracket - \llbracket \Lambda \rrbracket$
```

Output: S_j outputs the share $[\mu_{\mathbf{x}_*|\mathcal{D}}]_j, [\sigma^2_{\mathbf{x}_*|\mathcal{D}}]_j$ for $j \in \{0, 1\}$.

3.1 THE ALGORITHM SETUPS

To ensure a coherent execution of the algorithm, consensus must be reached among the servers (S_0, S_1, and T), data owners, and users regarding the algebraic structure to be employed. Specifically, an appropriate choice of l and l_f needs to be made for $\mathcal{Z}_{2^l}$ and $\mathcal{Q}< \mathcal{Z}_{2^l}, l_f >$, respectively. Once consensus is established, the data owners and users proceed with the conversion of their private observations $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ and test inputs $\mathbf{x}_*$ into shared representations denoted as $(\llbracket \mathbf{x}_* \rrbracket, \llbracket \mathbf{X} \rrbracket, \llbracket \mathbf{y} \rrbracket)$. This conversion is accomplished using the function $Shr(\cdot)$ and each resulting share $([\mathbf{x}_*]_j, [\mathbf{X}]_j, [\mathbf{y}]_j)$ is transmitted to the respective computing server S_j for $j \in \{0, 1\}$. Afther that, the servers perform GPR's privacy-preserving model construction and inference.

Let us consider an illustrative example to showcase the process. Supposing $l = 5$ and $l_f = 3$, the model user aims to privately predict the output of a test input $\mathbf{x}_* = (0.625, 0.375, 0.375)$. To achieve this, the user firstly converts $\mathbf{x}_*$ into $\mathcal{Z}_{2^3}$ by $\mathbf{x}_* \cdot 2^3 = (5, 3, 3)$, independently generates random values $[\mathbf{x}_*]_0 = (6, 9, 6)$, and then calculates $[\mathbf{x}_*]_1 = ((\mathbf{x}_* - [\mathbf{x}_*]_0) \mod 32) = (31, 26, 29)$. The computed value $[\mathbf{x}_*]_j$ is then transmitted to the computing server S_j for $j \in \{0, 1\}$. In a similar manner, the data owners employ the $Shr(\cdot)$ mechanism to send all the values pertaining to their private observations $\mathcal{D}$ to the respective computing servers.

Note that since $Shr(\cdot)$ is applied independently to each variable in $\mathbf{X}$, $\mathbf{y}$ and $\mathbf{x}_*$, the shares of the data can be computed easily no matter how the variables in $\mathbf{X}$, $\mathbf{y}$ and $\mathbf{x}_*$ are partitioned among the data owners. Therefore, the SS-based GPR algorithm can handle HDS, VDS, and PDS scenarios straightforwardly, which makes it practical enough to be used in various real-world applications.

The GPR hyperparameters $(\ell, \sigma_s^2, \sigma_n^2)$ are assumed to be known a priori and publicly shared between the computing servers. The privacy-preserving optimization of the hyperparameters will be considered in future work.

3.2 THE ALGORITHM EXECUTION STEPS

Once the computing servers receive the shares of all the data and hyperparameters, they start to execute the SS-based protocols for PP-GPR and output the shares of the predictive results. Similar to the conventional GPR, the PP-GPR algorithm contains two stages: model construction and prediction. At the model construction stage (Lines 1-4), the servers first compute secret shares of the distance matrix $d(\mathbf{X}, \mathbf{X}) \triangleq (d(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1,\dots,n}$. Let $\mathcal{F}_{dist}(\mathbf{X}, \mathbf{X}')$ be the SS-based protocol for computing the shares of the distance matrix between $\mathbf{X}$ and $\mathbf{X}'$. $[\![d(\mathbf{X}, \mathbf{X})]\!] = \mathcal{F}_{dist}(\mathbf{X}, \mathbf{X}) = ([\![d(\mathbf{x}_i, \mathbf{x}_j)]\!])_{i,j=1,\dots,n}$ where $\mathcal{F}_{dist}$ can be constructed using conventional SS-based addition and (matrix) multiplication operations. Then, the servers compute $[\![\mathbf{K}]\!]$ by calling a *privacy-preserving exponentiation* algorithm denoted as $\mathcal{F}_{PPExp}$ and compute $[\![Inv]\!] = [\![(\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1}]\!]$ by calling a *privacy-preserving matrix-inverse* algorithm $\mathcal{F}_{MatInv}$ with the inputs $[\![\mathbf{K}]\!]$ and σ_n^2. The design of $\mathcal{F}_{PPExp}$ and $\mathcal{F}_{MatInv}$ will be discussed later in Section 4.

At the prediction stage (Lines 5-14), the servers first compute $[\![\mathbf{k}_*]\!]$ and $[\![k(\mathbf{x}_*, \mathbf{x}_*)]\!]$ by calling $\mathcal{F}_{dist}$ and $\mathcal{F}_{PPExp}$ and then, obtain the shares of the predictive mean $[\![\mu_{\mathbf{x}_*|\mathcal{D}}]\!] = [\![\mathbf{k}_*^\top]\!][\![Inv]\!][\![\mathbf{y}]\!]$ and variance $[\![\sigma^2_{\mathbf{x}_*|\mathcal{D}}]\!] = [\![\mathbf{k}_*^\top]\!] - [\![k_{\mathbf{x}_*, \mathbf{x}}]\!][\![Inv]\!][\![\mathbf{k}_*]\!]$ according to (2) by calling $\mathcal{F}_{matMul}$.

4 PRIVACY-PRESERVING OPERATION CONSTRUCTION

As has been shown in Section 3, the privacy-preserving exponentiation $\mathcal{F}_{PPExp}$ and matrix inversion $\mathcal{F}_{MatInv}$ are essential for the PP-GPR algorithm. In this section, we will analyze the issues of existing methods for constructing these two operations, introduce the proposed algorithms, and analyze their computational complexity.

4.1 PRIVACY-PRESERVING EXPONENTIATION

As aforementioned in Section 2.2.1, the exponentiation cannot be constructed directly via additive SS. A commonly-used method to resolve this issue is to approximate the exponentiation using its Taylor expansion $e^u = \sum_{k=0}^{\infty} \frac{1}{k!} u^k$ such that the exponentiation can be converted into addition and multiplication operations. However, the fact that the exponential grows much faster than the polynomial may lead to large errors in the Taylor series approximation. Although increasing the degree of the polynomial can increase the approximation accuracy, the communication cost will also increase due to the information exchange needed for SS-based multiplication. The work of Knott et al. [2021] mitigated this problem via the limit approximation $e^u = \lim_{k\to\infty}(1 + \frac{u}{2^k})^{2^k}$ and exploited the repeated squaring algorithm to iteratively generate polynomials of higher order quickly. However, achieving accurate approximation results with this approach still incurs high communication and computational costs.

Algorithm 2 Privacy-preserving exponentiation ($\mathcal{F}_{PPExp}$)

Setup. The servers determine $\mathcal{Z}_L$, $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$, the range $[u_{min}, 0]$ of input u, and the range $[-\check{r}_{max}, \check{r}_{max})$.
Input. S_0 holds the share $[u]_0$; S_1 holds the share $[u]_1$.

```
1: // Offline phase executed on assistant server $T$:
2:     Draw $\check{r}$ in the range $[-\check{r}_{max}, \check{r}_{max})$ randomly
3:     $r \leftarrow \check{r} \cdot 2^{l_f}$                     ▷ $\check{r} \in \mathcal{Q}_{<\mathcal{Z}_L, l_f>}$
4:     Generate $([r]_0, [r]_1) \in \mathcal{Z}_L$
5:     Calculate $e^{-\check{r}}$ in $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$
6:     Generate $([e^{-\check{r}}]_0, [e^{-\check{r}}]_1) \in \mathcal{Z}_L$
7:     Send $[r]_j$ and $[e^{-\check{r}}]_j$ to $S_j$ for $j \in \{0, 1\}$
8: // Online phase:
9:     $S_j$ calculates $[d]_j \leftarrow [u]_j + [r]_j$ for $j \in \{0, 1\}$
10:    $S_0$ and $S_1$ sends $[d]_0$ and $[d]_1$ to each other
11:    $d \leftarrow Rec([d]_0 + [d]_1)$       ▷ Executed by both $S_0$ and $S_1$
12:    $\check{d} \leftarrow d/2^{l_f}$                ▷ Executed by both $S_0$ and $S_1$
13:    Calculate $e^{\check{d}}$ in $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$
14:    $S_j$ calculates $[e^u]_j \leftarrow e^{\check{d}} \cdot 2^{l_f} \cdot [e^{-\check{r}}]_j$ for $j \in \{0, 1\}$
```

Output. S_j outputs the share $[e^u]_j$ for $j \in \{0, 1\}$.

In this work, we propose to construct a *privacy-preserving exponentiation* (PP-Exp) operation by adopting the idea of *confusion-correction*. In PP-Exp, given a private number $u \in [u_{min}, 0]$, each computing server S_j for $j \in \{0, 1\}$ takes the additive share $[u]_j$ of u as input and then, deduces the additive shares of e^u privately with some random numbers generated by T. The algorithm includes the following steps: (1) The computing servers mask the share of u with a random value r to obtain $[\![u - r]\!]$; (2) The computing servers jointly reveal the obfuscated value $u - r$; (3) Each computing server uses the obfuscated value to calculate the obfuscated target e^{u-r}; and (4) Each computing server corrects the share of e^{u-r} by removing the mask and obtains the share of e^u. The pseudo-code of the PP-Exp is in Algorithm 2.

Note that Algorithm 2 considers only negative input u since the commonly used kernel function (e.g., (1)) of GPR involves only exponentiation of negative values. According to the range of u, the proposed PP-Exp can achieve correctness and security by selecting appropriate $[-\check{r}_{max}, \check{r}_{max})$ and l_f as will be discussed later.

Firstly, to guarantee the correctness of the proposed PP-Exp algorithm, we need to make sure that all the fixed-point calculations (i.e., Line 5 and Lines 13-14) can not overflow or underflow. Consequently, the selected $[-\check{r}_{max}, \check{r}_{max})$ and l_f need to satisfy the following relationship.

Theorem 1 (**Correctness**). *For any number u in the range $[u_{min}, 0]$, if $(\check{r}_{max} - u_{min}) \log_2^e \le l_f < \frac{l-1}{2}$, the PP-Exp algorithm can correctly derive $([e^u]_0, [e^u]_1)$ from $([u]_0, [u]_1)$, satisfying $[e^u]_0 + [e^u]_1 = e^u$.*

For example, on the ring of integers $\mathcal{Z}_{2^{64}}$, assuming that the input u takes values in the range $[-4, 0]$ and $\check{r}$ takes values in the range $[-16, 16)$. By setting $l_f = 29$, the correctness of the PP-Exp algorithm can be ensured. See Appendix A for the proof.

Next we will analyze the security of the PP-Exp algorithm. In Algorithm 2, Line 11 is the only step that will reconstruct d in the fixed-point domain and has the risk of leaking the information of u. Specifically, given the maximal range of u and $\check{r}$ (i.e., $[u_{min}, 0]$ and $[-\check{r}_{max}, \check{r}_{max})$), the value of d may be exploited to reduce the feasible range of u, which is an information leakage. For example, supposing $u \in \{-2, -1, 0\}$, $r \in \{-1, 0, 1\}$, and $d = u + r$, one can infer that u must be -2 or -1 if $d = -2$.

To formally analyze the amount of privacy leaked, we define the *degree of information leakage* as follows:

Definition 1. *Supposing u is known to be an element of a finite set $\mathcal{U}$, the* degree of information leakage *on u is $\frac{1}{|\mathcal{U}|}$.*

We consider an algorithm to be *secure* if the *degree of information leakage* of the input remains constant during the algorithm. Given a fixed precision l_f, let m_u and m_r be the amount of fixed-point numbers that can be represented in $[u_{min}, 0]$ and $[-\check{r}_{max}, \check{r}_{max})$, respectively. The security of the PP-Exp algorithm satisfies the following theorem.

Theorem 2 (**Security**). *For any fixed number u in the range $[u_{min}, 0]$, the PP-Exp algorithm is secure with the probability $\frac{m_r - m_u + 1}{m_r}$. The expected degree of information leakage on u is $\frac{m_u + m_r - 1}{m_u \cdot m_r}$.*

See Appendix A for the proof. Theorem 2 shows that the more the number of values of r is greater than the number of values of u (i.e., $m_r - m_u$ is larger), the PP-Exp has a larger probability to be secure. Selecting a larger range of $\check{r}$ (i.e., larger $\check{r}_{max}$) can significantly reduce the degree of information leakage. However, a larger $\check{r}_{max}$ may result in larger l_f and larger l due to the results in Theorem 1 and consequently, increase the amount of communication cost as will be shown in Section 4.3.

Note that the PP-Exp algorithm leaks the exact value of u with probability only $\frac{2}{m_u m_r}$, i.e., both u and r take the maximum or minimum value of the range in which they are located. For example, when $l_f = 29$, supposing the input u takes values in the range $[-4, 0]$ and r takes values in the range $[-16, 16)$, we have $m_u = 2^{31} + 1$ and $m_r = 2^{34}$. The PP-Exp is secure with $\frac{7}{8}$ probability. The degree of information leakage is $\frac{9}{2^{34}+8}$, an increase of $\frac{1}{2^{34}+8}$ over the secure one. The probability of revealing a particular value of u is less than $\frac{1}{2^{64}}$.

4.2 PRIVACY-PRESERVING MATRIX INVERSION

In existing work [Knott et al., 2021], the matrix inversion is approximated via *Newton-Raphson* iteration which is a local optimization method such that its performance highly depends on the initial value of the algorithm. However, in the state of SS, we cannot know any information about the original input matrix such that it is difficult to find the initial inverted matrix that satisfies the convergence condition.

Note that $\mathbf{K} + \sigma_n^2 \mathbf{I}$ is a positive definite matrix whose inversion can be computed via Cholesky decomposition $\mathbf{K} + \sigma_n^2 \mathbf{I} = \mathbf{L}\mathbf{D}\mathbf{L}^\top$ where $\mathbf{L}$ is a lower triangular matrix and $\mathbf{D}$ is a diagonal matrix. To theoretically guarantee the security of a PP-MI algorithm, we choose to go into the Cholesky decomposition algorithm and convert all the operations to their corresponding SS-based version. Since the entire process of computing $\mathbf{L}$, $\mathbf{D}$, and their inverse involves only addition, multiplication, and division between matrix elements, we can exploit the existing SS-based addition, multiplication, division, and their composability [Canetti, 2001] for constructing a PP-MI algorithm. In the PP-MI, each computing server S_j for $j \in \{0, 1\}$ takes one additive share $[\mathbf{U}]_j$ of matrix $\mathbf{U} \in Z_L^{n \times n}$ as input and deduces the share of $\mathbf{U}^{-1}$ privately. The detailed steps and pseudo-code of the PP-MI are shown in Appendix B.

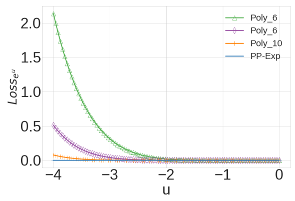
To the best of our knowledge, this is the first work that implements the SS-based PP-MI via Cholesky decomposition. A rigorous analysis of its communication cost is detailed in Section 4.3. Additionally, the performance of the proposed approach is empirically demonstrated and evaluated in Section 5.1.

4.3 COMMUNICATION COMPLEXITY

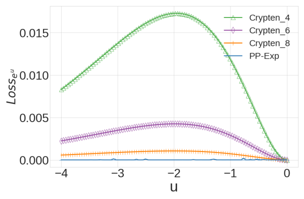
In this section, we analyze the theoretical number of communication rounds and communication volume of the proposed PP-Exp and PP-MI. We assume that the assistant server T has generated enough random numbers for the PP-GPR calculation process in the offline stage and sent their shares to the corresponding computing server a priori.

To execute PP-Exp for an n-dimensional vector $\mathbf{u}$, Algorithm 2 only requires 1 round of communication between the computing servers in Line 11 of Algorithm 2 and the amount of this communication is $2nl$.

In PP-MI, we decompose the matrix inversion into addition, multiplication, and division operations by exploiting the Cholesky decomposition. A single-element SS-based

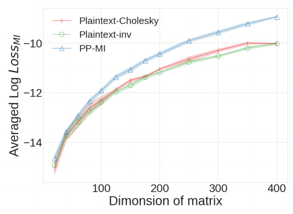


(a) PP-Exp vs. Polynomial

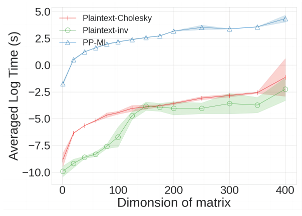


(b) PP-Exp vs. Iterative

Figure 3: Graphs of PP-Exp vs. (a) polynomial approximation methods; and (b) the iterative approximation method.



(a) PP-MI Loss



(b) PP-MI Time

Figure 4: Graphs of (a) Losses and (b) Computational time of different MI algorithms vs. dimension of matrix.

Table 1: The computational time (in the unit of second) incurred by the tested approaches in computing $e^{\mathbf{U}}$ with varying size of $\mathbf{U}$.

Size of U / Approach	1000^2	3000^2	5000^2	10000^2
Plaintext	0.008	0.014	0.024	0.064
Poly_10	3.308	20.956	56.462	221.582
Crypten_8	2.557	11.701	29.528	113.136
PP-Exp	**0.208**	**0.457**	**0.935**	**3.006**

multiplication requires 1 rounds of bidirectional communication with a traffic volume of $2l$. Implementing the division of a single element by invoking the privacy-preserving division provided in Crypten takes 17 rounds of communication and a communication volume of $\mathcal{O}(l)$. The PP-MI of an $n \times n$ positive definite matrix includes n rounds of division and $5n - 6$ rounds of multiplication. Thus, the total number of communication rounds for PP-MI is $17n + (5n - 6) = 22n - 6$. In the PP-MI, it is necessary to perform $\mathcal{O}(n^3)$ element multiplication and $\mathcal{O}(n^2)$ element division. Therefore, the total communication volume of the PP-MI algorithm is $\mathcal{O}(n^3 l)$. See Appendix C for detail.

5 EXPERIMENTS AND DISCUSSION

This section empirically evaluates the performance of the proposed privacy-preserving operations and PP-GPR. The PP-Exp, PP-MI, and PP-GPR are built upon the open-source privacy-preserving ML framework Crypten [Knott et al., 2021]. We set $l = 64$ and $l_f = 26$. The experiments are carried out on three servers with a 48-core Intel Xeon CPU running at 2.9GHz and a local area network with a communication latency of 0.2ms and bandwidth of 625MBps.

5.1 EVALUATION OF PP-EXP AND PP-MI

We first demonstrate the accuracy and computational efficiency of the proposed operations.

Evaluation of PP-Exp: We compare the performance of the proposed PP-Exp against that of (a) *Plaintext*: the conventional exponential operation; (b) *Poly*: a polynomial approximation based on Taylor expansions; and (c) *Crypten*: an iterative approach based on the limit approximation of the exponential function. We evaluate the accuracy and efficiency of the PP-Exp algorithm separately. For accuracy, Poly with varying degrees of polynomials and Crypten with a varying number of iterations are tested. $Loss_{e_u}$ is the difference between the tested algorithm and e_u computed in Plaintext. As shown in Fig. 3, the proposed PP-Exp achieves almost the same results as the plaintext and outperforms other tested algorithms. Next, we compare the efficiency of PP-Exp, Poly (with polynomial degree 10), and Crypten (with 8 iterations) on varying sizes of input variables (i.e., tested on $e^{\mathbf{U}}$ with varying sizes of $\mathbf{U}$). The computational time of the tested algorithms is shown in Table 1. As can be seen, the PP-Exp incurs significantly less time than both the polynomial and iterative approaches. In particular, PP-Exp is at most 70 times faster than Poly_10 and 38 times faster than Crypten_8 given a large size of inputs.

Evaluation of PP-MI: The performance of PP-MI is tested by generating random covariance matrices and compared against that of (a) *Plaintext-Cholesky*: Matrix inversion via Cholesky decomposition; and (b) *Plaintext-inv*: The inv function in the torch.linalg library. Specifically, we first randomly sample an input matrix $\mathbf{X} \in [-10, 10]^{n \times d}$ with $d = 2$ and then compute $\mathbf{K} + \sigma_n^2 \mathbf{I}$ using (1) with $\sigma_s^2 = 1$, $\ell = 1$, and $\sigma_n^2 = 0.1$. Let $\mathbf{\Lambda}$ be the output of a matrix inversion algorithm. $Loss_{\mathrm{MI}} \triangleq ||(\mathbf{K} + \sigma_n^2 \mathbf{I})\mathbf{\Lambda} - \mathbf{I}||_2^2$ is used as the inversion accuracy metric. The $Loss_{\mathrm{MI}}$ and wall-clock time of the tested algorithms averaged over 10 independent runs with varying n are shown in Fig. 4. The error bars are computed in the form of standard deviation. As can be seen, PP-MI incurs an acceptable level of accuracy loss (around 0.0001 for $n = 400$) with acceptable computational cost. This loss comes from the approximation of the SS-based division and the fixed-point encoding steps which cannot be avoid in most SS-based algorithms.

5.2 EVALUATION OF PP-GPR

This section empirically evaluates the performance of the proposed PP-GPR on two real-world datasets: (a) *Traffic* dataset [Chen et al., 2015] contains taxi demand information

Table 2: Evaluation results of GPR and PP-GPR using RBF kernel with varying sizes of observations and test inputs.

	Dataset Size		$Loss_\mu$	$Loss_{\sigma^2}$	Time (s)	
	n	Test	$mean(std.)$	$mean(std.)$	GPR	PP-GPR
Traffic	80	20	0.0005%(±7.5e-05)	0.0141%(±3.7e-03)	0.028	7.068
	150	50	0.0027%(±1.0e-02)	0.0061%(±4.9e-04)	0.089	13.016
	300	100	0.0057%(±3.3e-03)	0.0852%(±1.2e-02)	0.149	32.355
Diabetes	80	20	0.0007%(±3.6e-04)	0.0095%(±2.2e-03)	0.025	7.024
	150	50	0.0018%(±1.3e-03)	0.0059%(±1.2e-03)	0.104	13.901
	300	142	0.0058%(±2.5e-03)	0.0848%(±5.6e-03)	0.671	97.076

Table 3: Evaluation results of GPR and PP-GPR using Matérn kernel with varying sizes of observations and test inputs.

	Dataset Size		$Loss_\mu$	$Loss_{\sigma^2}$	Time (s)	
	n	Test	$mean(std.)$	$mean(std.)$	GPR	PP-GPR
Traffic	80	20	0.2711%(±2.3e-02)	0.0221%(±5.5e-06)	0.032	9.732
	150	50	0.2665%(±6.1e-03)	0.0241%(±1.2e-06)	0.078	13.116
	300	100	0.8288%(±1.5e-02)	0.0257%(±2.0e-06)	0.153	35.33
Diabetes	80	20	0.0548%(± 1.0e-03)	0.0193%(±4.0e-06)	0.023	8.027
	150	50	0.0424%(±6.2e-05)	0.0236%(± 7.6e-06)	0.102	15.702
	300	142	0.0545%(±4.9e-06)	0.0343%(± 3.0e-06)	0.589	99.082

of 2506 regions in a city between 9:30 p.m. and 10 p.m. on August 2, 2010; and (b) *Diabetes* dataset (under BSD License) [Efron et al., 2004] contains diabetes progression of 442 diabetes patients with 10 input features. We test the proposed PP-GPR with both the SE kernel (1) and the Matérn$_{3/2}$ kernel:

$$k(\mathbf{x},\mathbf{x}') \triangleq \sigma_s^2(1+\sqrt{3d(\mathbf{x},\mathbf{x}')}/l)\exp(-\sqrt{3d(\mathbf{x},\mathbf{x}')}/l)$$

In the diabetes experiments, we use $\sigma_s^2 = 0.8$, $\sigma_n^2 = 0.1$, and $\ell = 0.23$ for the SE kernel and $\sigma_s^2 = 0.1$, $\sigma_n^2 = 0.1$, and $\ell = 1.0$ for the Matérn$_{3/2}$ kernel. As the traffic dataset suggested, we set $\sigma_s^2 = 0.1$, $\sigma_n^2 = 0.1$, and $\ell = 1.0$ for all the traffic experiments.

Let $\mathcal{X}_*$ be a set of test inputs, $\mu_{\mathbf{x}_*|\mathcal{D}}$ ($\sigma^2_{\mathbf{x}_*|\mathcal{D}}$) and $\tilde{\mu}_{\mathbf{x}_*|\mathcal{D}}$ ($\tilde{\sigma}^2_{\mathbf{x}_*|\mathcal{D}}$) be, respectively, the predictive mean (variance) of the GPR and PP-GPR. The relative difference between the predictive results of GPR and PP-GPR is used as the performance metric: $Loss_\mu \triangleq |\mathcal{X}_*|^{-1}\sum_{\mathbf{x}_*\in\mathcal{X}_*}(|\mu_{\mathbf{x}_*|\mathcal{D}} - \tilde{\mu}_{\mathbf{x}_*|\mathcal{D}}|/\mu_{\mathbf{x}_*|\mathcal{D}})$. $Loss_{\sigma^2}$ is computed in a similar way. To test the performance of PP-GPR in different data scales, we randomly sample observations and test data from each dataset with varying n and $|\mathcal{X}_*|$. The loss of the predictive results and the wall-clock execution time (including both computation and communication time) are shown in Table 2 and Table 3. All the results are averaged over 5 random runs.

It can be observed that the PP-GPR achieves a similar predictive mean and variance compared to conventional GPR. The losses are due to the approximation of some SS-based operations (e.g., division) and the fixed-point encoding step. The computational errors of the Matérn kernel are slightly higher than that of the SE kernel but still remain at a low level. Further analysis revealed that the higher computational error in the Matérn kernel is due to the inclusion of the expression $\sqrt{d(\mathbf{x},\mathbf{x}')}$. The square root operation is a non-linear operation that must be approximated using the Newton iterative approach in Crypten Knott et al. [2021], which results in the higher computational error. In our future work, we plan to conduct further research to investigate this issue.

Furthermore, although PP-GPR incurs a longer time than GPR, especially if n is large, it can finish the model construction and prediction in a reasonable time ($<$ 2 mins) for a dataset with several hundred observations.

In Appendix D, we perform an additional empirical comparison between our algorithm and DP-based GPR under the scenario that only the model outputs are sensitive. We believe that this comparison is fair, given that both methods can theoretically preserve privacy. The other privacy-preserving GPR approaches (e.g., FHE-based and FL-based GPR) are not compared since even when operating within the same scenario (i.e., HDS, VDS, or PDS), they may have fundamentally different security assumptions to that of PP-GPR, which ultimately makes them incomparable. See Section 6 for detailed discussions.

6 RELATED WORK

To the best of our knowledge, there is no existing PP-GPR work that is designed based on SMPC techniques. As has been mentioned in Section 1, although some other privacy enhancement techniques have been applied to GPR, none of them is practical enough to protect the privacy of both the inputs and outputs of GPR for all the three data-sharing scenarios (i.e., HDS, VDS, and PDS). To be specific, Fenner and Pyzer-Knapp [2020] considers only the PDS scenario and protects the input features of the test data by *fully homomorphic encryption* (FHE) algorithm. Since performing computation on the homomorphically encrypted data incurs high computational costs, they do the PP-GPR prediction through interactive calculations between the user and the model constructor. Such an interactive method, however, cannot be generalized to FHE-based PP-GPR model construction step since the covariance matrix inversion operation is not considered.

Another technique that is widely used to achieve PP-ML models is *differential-privacy* (DP). Smith et al. [2018] proposed the first DP-GPR algorithm which can only protect the privacy of the model outputs $\mathbf{y}$. Kharkovskii et al. [2020] proposed a DP method to protect the input features of the GPR model via random projection. However, this method requires all the observations used for GPR model construction to belong to a single curator and thus, cannot be applied to either HDS or VDS scenarios. In addition, the DP-based method may incur large DP noise to the original model when the privacy budget ϵ is small, which may significantly reduce the model performance [Dwork et al., 2014].

Some other works [Dai et al., 2020, Kontoudis and Stilwell, 2022, Yue and Kontar, 2021] consider protecting the privacy of the GPR observations via *federated learning* (FL) or combine FL with DP to further protect the privacy of the model parameters [Dai et al., 2021]. To convert the GPR model construction into a distributed/federated manner, these works have to exploit some sparse approximations (e.g., random features) of the conventional GPR, which may reduce the model performance. Moreover, FL-based GPR works can only be applied to the HDS scenario.

Recently, Kelkar et al. [2022] developed a privacy-preserving exponentiation algorithm based on secret sharing techniques in a two-server setting. The communication overhead of this algorithm in the online phase is comparable to that of PP-Exp (i.e., one round of communication and transmission of two elements). However, the algorithm requires an expensive cryptographic primitive (i.e., homomorphic encryption) in the preprocessing phase to generate the random numbers needed in the online phase, resulting in excessive overhead. In addition, the algorithm suffers a certain probability of error from the use of a secure ring change procedure. In the setting of this paper (i.e., $l = 64, l_f = 26$), the probability that the error occurs is $\frac{1}{4}$, which is unacceptable.

Note that even when operating within the same scenario (i.e., HDS, VDS, or PDS), different privacy-preserving approaches may have fundamentally different security assumptions [Yin et al., 2021, Zhang et al., 2022]. Specifically, the privacy of the FHE-based GPR algorithm Fenner and Pyzer-Knapp [2020] may be at risk even in the PDS scenario due to the decryption steps designed for reducing the high computational cost of the exponential operation in FHE. However, this work provides a solution by effectively addressing the challenges posed by the SS-based exponentiation operation. Consequently, this work guarantees complete privacy protection across the entire PDS process. The FL-based GPR [Dai et al., 2020, Kontoudis and Stilwell, 2022, Yue and Kontar, 2021] has no theoretical analysis of its privacy-preserving capabilities. This is because the intermediate results (e.g., local model parameters or gradients) generated during the algorithm need to be exchanged between the server and clients. Numerous studies [Zhu et al., 2019, Zhao et al., 2020] have demonstrated that these intermediate results pose a potential risk of revealing private data.

7 CONCLUSION

This paper describes the first SS-based privacy-preserving GPR model which can be applied to both horizontal and vertical data-sharing scenarios. We provide a detailed workflow for implementing both the model construction and prediction steps of PP-GPR. Two additive SS-based operations (i.e., PP-Exp and PP-MI) are proposed such that they can be combined with existing SS-based operations for constructing a secure and efficient GPR model. We analyze the security and computational complexity of the proposed operations in theory. Although PP-GPR incurs more computational time due to additional communications between the two computing servers and some additional computing steps, it can perform GPR in an acceptable time with a security guarantee, which is a superior alternative to existing FL and DP-based privacy-preserving GPR approaches when the scale of observations is not large.

8 ACKNOWLEDGEMENTS

This research is partially supported by the National Key Research and Development Program of China (No. 2022ZD0115301), the National Natural Science Foundation of China (No. 62206139), and a key program of fundamental research from Shenzhen Science and Technology Innovation Commission (No. JCYJ20200109113403826).

References

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proc. of the ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and

Aaron Steele. Secure computation on floating point numbers. *Cryptology ePrint Archive*, 2012.

Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

George Robert Blakley. Safeguarding cryptographic keys. In *International Workshop on Managing Requirements Knowledge*, 1979.

Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

Jie Chen, Kian Hsiang Low, Yujian Yao, and Patrick Jaillet. Gaussian process decentralized data fusion and active sensing for spatiotemporal traffic modeling and prediction in mobility-on-demand systems. *IEEE Transactions on Automation Science and Engineering*, 12(3):901–921, 2015.

Zhongxiang Dai, Bryan Kian Hsiang Low, and Patrick Jaillet. Federated Bayesian optimization via Thompson sampling. In *Proc. NeurIPS*, pages 9687–9699, 2020.

Zhongxiang Dai, Bryan Kian Hsiang Low, and Patrick Jaillet. Differentially private federated Bayesian optimization with distributed exploration. In *Proc. NeurIPS*, 2021.

Cynthia Dwork. Differential privacy. In *Proc. ICALP*, pages 1–12, 2006.

Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

David Evans, Vladimir Kolesnikov, Mike Rosulek, et al. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3): 70–246, 2018.

Peter Fenner and Edward Pyzer-Knapp. Privacy-preserving Gaussian process regression – A modular approach to the application of homomorphic encryption. In *Proc. AAAI*, pages 3866–3873, 2020.

Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proc. annual ACM symposium on Theory of computing*, pages 169–178, 2009.

Shafi Goldwasser. How to play any mental game, or a completeness theorem for protocols with an honest majority. In *Proc. ACM STOC*, pages 218–229, 1987.

Mahimna Kelkar, Phi Hung Le, Mariana Raykova, and Karn Seth. Secure Poisson regression. In *Proc. USENIX Security*, pages 791–808, 2022.

Dmitrii Kharkovskii, Zhongxiang Dai, and Bryan Kian Hsiang Low. Private outsourced Bayesian optimization. In *Proc. ICML*, pages 5231–5242, 2020.

Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. CrypTen: Secure multi-party computation meets machine learning. In *Proc. NeurIPS*, 2021.

Jakub Konečný, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

George P Kontoudis and Daniel J Stilwell. Fully decentralized, scalable Gaussian processes for multi-agent federated learning. *arXiv preprint arXiv:2203.02865*, 2022.

Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via MiniONN transformations. In *Proc. CCS*, pages 619–631, 2017.

Payman Mohassel and Peter Rindal. ABY3: A mixed protocol framework for machine learning. In *Proc. CCS*, pages 35–52, 2018.

Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy*, pages 19–38, 2017.

Lukas Ortmann, Dawei Shi, Eyal Dassau, Francis J Doyle, Berno JE Misgeld, and Steffen Leonhardt. Automated insulin delivery for type 1 diabetes mellitus patients using Gaussian process-based model predictive control. In *Proc. ACC*, pages 4118–4123, 2019.

C. E. Rasmussen and C. K. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.

Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis Bach. AriaNN: Low-interaction privacy-preserving deep learning via function secret sharing. *Proc. on Privacy Enhancing Technologies*, 2022(1):291–316, 2020.

Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

R Shashikant, Uttam Chaskar, Leena Phadke, and Chetankumar Patil. Gaussian process-based kernel as a diagnostic model for prediction of type 2 diabetes mellitus risk using non-linear heart rate variability features. *Biomedical Engineering Letters*, 11(3):273–286, 2021.

Michael T Smith, Mauricio A Álvarez, Max Zwiessele, and Neil D Lawrence. Differentially private regression with Gaussian processes. In *Proc. AISTATS*, pages 1195–1203, 2018.

Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-Party secure computation for neural network training. *Proc. on Privacy Enhancing Technologies*, 2019 (3):26–49, 2019.

Zenglin Xu, Feng Yan, and Yuan Qi. Bayesian nonparametric models for multiway data analysis. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):475–487, 2015.

Feng Yan, Zenglin Xu, and Yuan Qi. Sparse matrix-variate Gaussian process blockmodels for network modeling. In *Proc. UAI*, pages 745–752, 2011.

Steve Y Yang, Qifeng Qiao, Peter A Beling, William T Scherer, and Andrei A Kirilenko. Gaussian process-based algorithmic trading strategy identification. *Quantitative Finance*, 15(10):1683–1703, 2015.

Andrew Chi-Chih Yao. Protocols for secure computations. In *Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.

Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Annual Symposium on Foundations of Computer Science*, pages 162–167, 1986.

Xuefei Yin, Yanming Zhu, and Jiankun Hu. A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions. *ACM Computing Surveys (CSUR)*, 54(6):1–36, 2021.

Xubo Yue and Raed Al Kontar. Federated Gaussian process: Convergence, automatic personalization and multi-fidelity modeling. *arXiv preprint arXiv:2111.14008*, 2021.

Xiaojin Zhang, Hanlin Gu, Lixin Fan, Kai Chen, and Qiang Yang. No free lunch theorem for security and utility in federated learning. *ACM Transactions on Intelligent Systems and Technology*, 14(1):1–35, 2022.

Yehong Zhang, Trong Nghia Hoang, Kian Hsiang Low, and Mohan Kankanhalli. Near-optimal active learning of multi-output Gaussian processes. In *Proc. AAAI*, pages 2351–2357, 2016.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. iDLG: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.

Shandian Zhe, Zenglin Xu, Xinqi Chu, Yuan (Alan) Qi, and Youngja Park. Scalable nonparametric multiway data analysis. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, 2015.

Shandian Zhe, Yuan Qi, Youngja Park, Zenglin Xu, Ian Molloy, and Suresh Chari. DinTucker: Scaling up Gaussian process models on large multidimensional arrays. In *Proc. AAAI*, 2016.

Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Proc. NeurIPS*, 2019.