Journal of Systems Architecture 151 (2024) 103134

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Efficient privacy-preserving Gaussian process via secure multi-party computation

Shiyu Liu [a], Jinglong Luo [b,c], Yehong Zhang [c], Hui Wang [c], Yue Yu [c], Zenglin Xu [b,c,*]

[a] University of Electronic Science and Technology of China, Chengdu, China
[b] Harbin Institute of Technology, Shenzhen, China
[c] Peng Cheng Laboratory, Shenzhen, China

ARTICLE INFO

Keywords:
Privacy-preserving machine learning
Secret sharing
Gaussian process
Secure multi-party computation

ABSTRACT

Gaussian processes (GPs), known for their flexibility as non-parametric models, have been widely used in practice involving sensitive data (e.g., healthcare, finance) from multiple sources. With the challenge of data isolation and the need for high-performance models, how to jointly develop privacy-preserving GP for multiple parties has emerged as a crucial topic. In this paper, we propose a new *privacy-preserving GP* algorithm, namely PP-GP, which employs *secret sharing* (SS) techniques. Specifically, we introduce a new SS-based exponentiation operation (PP-Exp) through confusion correction and an SS-based matrix inversion operation (PP-MI) based on Cholesky decomposition. However, the advantages of the GP come with a great computational burden and space cost. To further enhance the efficiency, we propose an efficient split learning framework for privacy-preserving GP, named Split-GP, which demonstrably improves performance on large-scale data. We leave the private data-related and SMPC-hostile computations (i.e., random features) on data holders, and delegate the rest of SMPC-friendly computations (i.e., low-rank approximation, model construction, and prediction) to semi-honest servers. The resulting algorithm significantly reduces computational and communication costs compared to PP-GP, making it well-suited for application to large-scale datasets. We provide a theoretical analysis in terms of the correctness and security of the proposed SS-based operations. Extensive experiments show that our methods can achieve competitive performance and efficiency under the premise of preserving privacy.

1. Introduction

Gaussian processes (GPs) [1–4] are widely used non-parametric models that can provide principled uncertainty representations through posterior variances. These uncertainty representations play a crucial role in various application domains, including medicine [5–7], robotics [8] or finance [9], where incorrect predictions could result in catastrophic consequences. GPs are not only well-suited for problems with limited observations; they also show considerable potential in leveraging the available information in large datasets [10].

Despite recent progress, the data used in such applications often comes from multiple sources but cannot be shared directly due to the growing privacy concerns in the machine learning (ML) community. This is also known as the data isolation problem. For instance, different hospitals with limited patient data aim to collaborate and develop a high-quality GP model for enhanced disease progression prediction [11,12]. However, due to legal restrictions, such data often contains sensitive patient information and therefore cannot be shared. Consequently, a single hospital with limited data may struggle to achieve the high precision required for effective modeling. Additionally, there are concerns regarding potential privacy breaches related to sensitive personal data, such as personal features (i.e., test input), and diagnostic outcomes (i.e., model output), particularly when specific hospitals or patients consider using the developed Gaussian processes model for diagnosis. Similarly, financial data such as consumption behavior encounters comparable privacy challenges. This raises a fundamental question: *how can GPs be utilized in a way that aligns with regulatory compliance requirements?*

Several GP approaches specifically catered towards privacy settings have been proposed to address these issues. These approaches aim to protect the privacy of both inputs and outputs throughout the process of model construction and inference. They prevent privacy leakage in three scenarios: data sharing on horizontally and vertically partitioned data, as well as privacy-preserving inference (see Fig. 1 for an illustrative example) The most straightforward solution to privacy-preserving GP is to apply enhancement techniques, including *homomorphic encryption* (HE) [13], *federated learning* (FL) [14], and *differential privacy* (DP) [15]. Although it is possible to obtain some degree of privacy,

* Corresponding author.
E-mail address: xuzenglin@hit.edu.cn (Z. Xu).

https://doi.org/10.1016/j.sysarc.2024.103134
Received 30 September 2023; Received in revised form 28 March 2024; Accepted 29 March 2024
Available online 9 April 2024

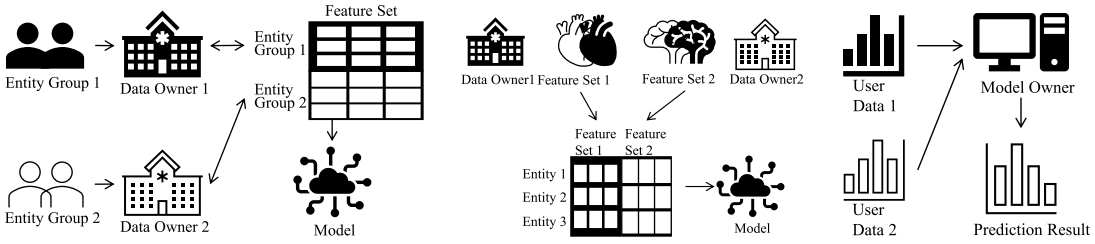


Fig. 1. Illustration of scenarios of privacy-preserving data sharing on Horizontally partitioned data, Vertically partitioned data, and Privacy-preserving inference. (a) Horizontally partitioned data (HPD): multiple parties contribute datasets of different entities but with matching features, which are then combined for model construction; (b) Vertically partitioned data (VPD): parties share distinct features for a common set of entities to collaboratively construct the model; (c) Privacy-preserving inference (PPI): before using the constructed model, participants must share their data with the model holder.

such approaches are usually ineffective owing to their limitations: the HE may suffer from high computation and communication costs, while DP is prone to compromising performance, and the FL typically provides limited security. Importantly, none of these techniques can be universally applied to all data-sharing scenarios. Specifically, the homomorphic encryption GP (HE-GP) [16] methods focus on privacy-preserving inference, while the federated learning GP (FL-GP) [17–19] methods are tailored for model construction with horizontally partitioned data; the differential privacy (DP-GP) [20,21] methods, which operate under the assumption of a single data owner, ensure privacy preservation for either the input features or the outputs.

This manuscript constitutes a significantly extended version of [22] that aims to overcome the above-mentioned issues. In this paper, we propose to study the privacy-preserving GP problem under the secure multi-party computation (SMPC) [23] perspective. The most prominent among the various types of SMPC is the secret sharing (SS) [24] algorithm, which due to its good communication efficiency, has been widely used in many ML approaches. However, the majority of existing privacy-preserving ML approaches were not specifically designed for GPs. Although linear operations (e.g., addition, or multiplication) can be achieved using existing SS-based toolkits, the non-linear operations essential for GP tasks, such as exponentiation and matrix inversion, have not been well integrated into SMPC. To overcome this challenge, we propose novel SMPC protocols that utilize SS basis operations for positive definite matrix inversion and exponential operations. Specifically, inspired by the idea of confusion correction, we propose a privacy-preserving exponentiation operation (PP-Exp) that provides a more precise approximation than conventional iterative polynomial methods. Then, we extend the Cholesky decomposition algorithm to enable the conversion of all operations into their corresponding SS-based versions. The resulting algorithm is referred to as privacy-preserving matrix inversion (PP-MI). Based on these protocols, we propose a novel *privacy-preserving GP* framework, named PP-GP, which promises robust security guarantees and protection against unwanted information exposure in various data-sharing scenarios.

However, the advantages of the GP come with great computational and space demands due to its kernel matrix calculations, leading to $\mathcal{O}(n^2)$ storage complexity and $\mathcal{O}(n^3)$ computational complexity [25]. This often results in communication bottlenecks with SMPC protocols, particularly for non-linear operations, rendering the privacy-preserving GP less practical for larger datasets ($n > 10^4$). To further enhance the efficiency, we introduce Split-GP, a highly efficient split learning framework tailored for privacy-preserving GP on large-scale data. Split learning enables us to divide the model into multiple parts, where each part is computed by a different participant, making it particularly suitable for resource-constrained environments. Hence, we split the total computation process into two parts to address memory and efficiency concerns: the SMPC-hostile computations related to data are carried out by data holders, and SMPC-friendly tasks are conducted by semi-honest servers. Specifically, data holders use SS techniques to compute

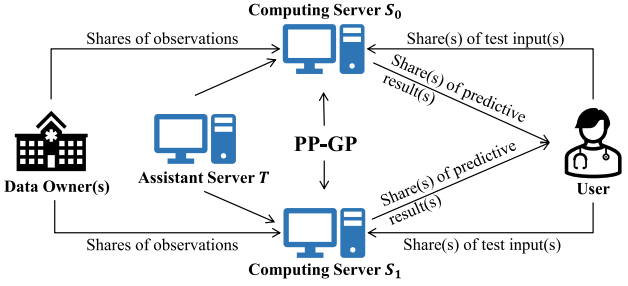


Fig. 2. Overview of our privacy-preserving GP framework with three-party SMPC architecture.

the random features [26] expansions mapping based on their private data, individually. Subsequently, the semi-honest servers combine these expansions, perform a low-rank factorization for kernel approximation, and construct the predictive model. The use of random features not only eliminates the need to share raw data and complex exponential operations but also enhances communication efficiency. Furthermore, this allows us to avoid the difficult matrix computations in the size of the number of data points, which is also the most time-consuming among the SMPC protocols, and reduce storage space while ensuring accuracy and efficiency. Consequently, the proposed Split-GP not only minimizes computation and communication costs but also provides strong security guarantees (see Fig. 2).

Contributions. In light of the above discussion, the main contributions of this paper can be summarized as follows:

- We introduce a new perspective on privacy-preserving GP through secret sharing, which broadens the design space for privacy-preserving ML algorithms. We propose a novel privacy-preserving GP algorithm named PP-GP (see Algorithm 1), which provably addresses the privacy exposure concerns in various data-sharing scenarios (Section 4).
- We design privacy-preserving protocols for non-linear operations, including exponentiation (PP-Exp, see Algorithm 2) and matrix inversion (PP-MI, see Algorithm 3). Our efficient PP-Exp, inspired by the idea of confusion correction, is up to 70 times faster than typical approximation algorithms, offering both correctness and security. Furthermore, we introduce PP-MI, the first SS-based matrix inversion algorithm using Cholesky decomposition, which matches the plaintext algorithm's accuracy with reasonable communication costs (Section 4).
- To further boost the inference efficiency, we propose Split-GP (see Algorithm 4), a highly efficient split learning algorithm tailored for privacy-preserving GP on large-scale data ($n > 10^4$). The Split-GP can have significantly lower communication and computation complexity (Section 5).

- Our theoretical analysis demonstrates the correctness and security of PP-MI and PP-Exp, which thereby provide strong security guarantees for our approaches (Section 6). We evaluate the empirical performance of PP-GP and Split-GP on several datasets from the UCI repository. The results demonstrate that although Split-GP exhibits a slight decrease in performance, it is particularly effective when applied to large datasets. It achieves up to 30× faster computation and reduces communication by $50 \sim 3000\times$(Section 7).

Organization. The rest of this paper is organized as follows: We first review and discuss the related work in Section 2 and then briefly introduce the preliminaries relevant to our discussion in Section 3. Section 4 outlines our proposed framework as well as its associated operations. In addition, we extend the proposed method to split learning in Section 5. In Section 6, we state our theoretical results for the proposed protocols. We provide the empirical performance evaluation of our methods in Section 7. All proofs are deferred to Appendix.

2. Related work

In recent times, several approaches have appeared to tackle the challenges of privacy-preserving ML and GP models. In this section, we review the relevant literature and its connections to our work.

Gaussian processes. Gaussian Processes (GPs) have demonstrated significant success across various ML applications, including Bayesian optimization [27–29], deep neural networks [30–32], reinforcement learning [8,33,34], and time-series analysis [35,36]. Their strengths lie in providing reliable uncertainty quantification, priors that require little expert intervention, and a flexible adaptability to datasets of varying sizes [1,37]. While GPs are effective for tasks with limited data, they also hold the potential to exploit the available information from expansive datasets, especially when paired with expressive kernels [38,39] or hierarchical structures [40–45].

Several follow-up works have focused on enhancing GPs in various ways for application to large-scale data (e.g.,). However, a drawback of GPs is their poor scalability with the dataset size n; training and prediction scale with complexities of $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, respectively. This constraint practically limits GPs to datasets of size around $\mathcal{O}(10^4)$ [46]. For large datasets (e.g., $n > 10^4$), sparse approximations have been successfully applied in a broad range of practical applications [47–50]. These methods typically reduce the computational burden by implicitly or explicitly using a subset of the data. A recent line of work has studied the distributed GPs [25,46,51], allowing independent updates of variational parameters across different computing nodes. To the best of our knowledge, applying GPs to training set sizes of $n \geq \mathcal{O}(10^7)$ remains impractical [52].

Privacy-preserving machine learning. Recent concerns about data privacy, coupled with the need for enhanced ML performance, have accelerated interest in privacy-preserving ML (PPML) [23,53]. PPML allows multiple entities to collaboratively develop ML models without compromising data privacy. However, the majority of existing PPML studies predominantly concentrate on linear regression [54,55], logistic regression [23,56,57], decision trees [58], and neural networks [23,59–61]. These models can be broadly categorized based on data partitioning: horizontally partitioned PPML, where each party possesses a subset of samples with identical features, and vertically partitioned PPML, where parties share the same samples but different features [53,62].

Privacy-preserving Gaussian processes. To construct privacy-preserving GPs, various enhancement techniques can be employed, including *Homomorphic Encryption* (HE) [16,63], *Differential Privacy* (DP) [15,64], and *Federated Learning* (FL) [65]. While they provide a certain degree of privacy guarantees, they come with limitations: HE suffers high computation and communication costs, DP often compromises performance, and FL provides limited security. Moreover, none of them are practical enough to protect the privacy of both the inputs and outputs, across all three data-sharing situations. Specifically, Fenner and Pyzer-Knapp [16] employ the HE algorithm to secure test data input features but consider only the PDS scenario. Due to the high computational demands of working on homomorphically encrypted data, the prediction in PP-GP involves interactive calculations between the user and the model constructor. However, this interactive approach cannot be directly extended to the HE-based PP-GP construction, as it omits the covariance matrix inversion operation.

DP is another technique widely used to achieve PP-ML models. Smith et al. [20] leveraged DP in the construction of GP and introduced the first DP-GP learning algorithm. However, this algorithm comes with certain limitations. Specifically, it can only guarantee the privacy of the model outputs, but does not extend the same privacy guarantee to intermediate values and parameters. [21] protected the input features with random projection, but required that all the observations used belong to a single party, which restricts its applicability in both HPD and VPD scenarios. In addition, when the privacy budget ϵ is limited, the DP-based method can introduce substantial noise to the original model, potentially compromising its performance [66].

Other works [17,19,67] utilize FL to protect GP observation privacy or combine FL with DP to enhance the privacy protection of the model parameters during the model construction process [18]. To adapt GP model construction for distributed or federated framework, these methods often rely on sparse approximations of the standard GP, which could compromise model performance. Furthermore, FL-based GP methods are limited to the HPD scenario. On the other hand, The privacy-preserving capabilities of FL-based GP [17,19,67] lack comprehensive theoretical analysis. This happens because the algorithm requires the server and clients to exchange intermediate results like local model parameters or gradients. Numerous research [68,69] have shown that such exchanges can inadvertently expose sensitive data. Moreover, the privacy of HE-based GP methods [16] may be at risk, particularly due to decryption steps taken to mitigate the computational burden of exponential operations, especially in PPI scenarios.

Relation to [22]. This paper is a substantially extended version of our previous conference paper [22], with the new contributions summarized as follows:

- We have developed an efficient framework for privacy-preserving Gaussian Processes (GPs) using split learning, which shows significant capability in processing large-scale data. Inspired by split learning, we have divided the entire computational process and approximated the kernel using random features. This approach reduces the dimension of matrix operations, thereby ensuring communication efficiency and enhancing the efficiency of subsequent computations (Section 5).
- We have conducted more experiments, incorporating additional large-scale datasets, and provided comprehensive comparisons of running time and communication within the empirical benchmark study (Section 7).
- In addition, we have presented more in-depth theoretical analyses, including more rigorous and formal proofs for PP-Exp, and detailed proofs of security and correctness for PP-MI (Section 6 and Appendix A).
- Finally, we have provided a more comprehensive review of related work about Gaussian processes and privacy-preserving machine learning. This will help us to better understand the relevant background, and make it easier to follow our research (Section 2).

This work presents a solution that effectively tackles the challenges posed by the SS-based non-linear operation, thereby ensuring comprehensive privacy protection throughout the entire inference process of privacy-preserving GPs. To the best of our knowledge, there is limited research in the field of privacy-preserving GPs that are specifically designed with SMPC techniques to fully satisfy privacy requirements in various data-sharing contexts. Our proposed approaches not only fill this gap but also have the potential to significantly enhance real-world applications of privacy-preserving GPs, especially in large-scale contexts.

3. Preliminaries

In this section, we provide an overview of the relevant background. First, we define basic notation and recall elementary concepts of the Gaussian process. Then, we briefly introduce secure multi-party computation, including secret sharing and fixed-point representation. Finally, we introduce the random features, which provide a way to tackle large-scale machine learning problems with kernel methods.

3.1. Gaussian process

Considered the observed data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ for $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. Gaussian processes (GPs) are non-parametric machine learning models that define a distribution over functions $y = f(\mathbf{x}) + \varepsilon$, where

$$f(\mathbf{x}) \sim \mathcal{GP}\big(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\big), \quad \varepsilon \sim \mathcal{N}(0, \sigma^2), \tag{3.1}$$

and $\mu(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$ is prior mean function $\mathbb{R}^d \to \mathbb{R}$. Typically, it is set to zero without loss of generality. The function $k(\mathbf{x}, \mathbf{x}')$ represents the covariance kernel $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, defined as the covariance between $f(\mathbf{x})$ and $f(\mathbf{x}')$. Common kernels include the Radial Basis Function (RBF) kernel, $k(\mathbf{x}, \mathbf{x}') := \sigma_s^2 \exp\big(-\|\mathbf{x} - \mathbf{x}'\|_2^2 / 2\ell^2\big)$, incorporate the length-scale ℓ and the signal variance σ_s^2.

Let $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\top$ be an $n \times d$ input matrix and $\mathbf{y} = (y_1, \ldots, y_n)^\top$ represent column vector of the n noisy outputs. For a test point $\mathbf{x}^*$, the GP predictive posterior distribution $p\big(f(\mathbf{x}^*)|D\big)$, assuming a Gaussian likelihood, is Gaussian and characterized by the following moments:

$$\mu_{f|D}(\mathbf{x}^*) = \mathbf{k}_*^\top (K_{XX} + \sigma_n^2 I)^{-1} \mathbf{y}, \tag{3.2a}$$

$$\sigma_{f|D}^2(\mathbf{x}^*) = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (K_{XX} + \sigma_n^2 I)^{-1} \mathbf{k}_*, \tag{3.2b}$$

where $\mathbf{k}_* = k(\mathbf{x}_*, X) = \big(k(\mathbf{x}_*, \mathbf{x}_i)\big)_{i=1}^n$ consists of kernel values between training examples and a test point $\mathbf{x}_*$. The matrix K_{XX}, with dimensions $n \times n$, represents all pairs of kernel entries, where $\big[K_{\mathbf{XX}}\big]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. To simplify notation, a hat symbol denotes a kernel matrix with added Gaussian observational noise $\hat{K}_{\mathbf{XX}} = K_{\mathbf{XX}} + \sigma_n^2 I$, and I represents the identity matrix of size $n \times n$.

3.2. Secure multi-party computation

Secure multi-party computation (SMPC) [70] allows multiple parties to collaboratively compute an operation f, while preserving any privacy of their individual data throughout the computation process. In this paper, we employ the *semi-honest security* model, sometimes referred to as the *honest-but-curious* model, which has been the backbone of numerous privacy-preserving ML algorithms [60,71–75]. Under the semi-honest setting, involved parties are expected to adhere to the SMPC protocol but might attempt to infer unexposed information using acquired data-sharing and intermediate computations. Now, we discuss the *secret sharing* technique and introduce the algebraic structure that forms the foundation of our SMPC protocol.

3.2.1. Secret sharing

Secret sharing (SS) [76,77] is a crucial component of SMPC. It allows a secret to be divided among multiple parties in such a way that the secret can only be reconstructed when a sufficient number of them collaborate. Specifically, we focus on the $(2, 2)$-additive secret sharing scheme in this work. This scheme is defined by two algorithms: $Shr(\cdot)$ for sharing and $Rec(\cdot, \cdot)$ for reconstruction. We represent the additive share of an integer u in the ring of integers modulo L, denoted as $\mathcal{Z}_L$, as $[\![u]\!] = ([u]_0, [u]_1)$. The sharing algorithm $Shr(u) \to ([u]_0, [u]_1)$ works by randomly selecting a number r from $\mathcal{Z}_L$. It sets $[u]_0 = r$ and computes $[u]_1 = (u - r) \mod L$. Due to the random choice of r, neither $[u]_0$ nor $[u]_1$ individually provides any information about the original value u. Reconstruction is straightforward with the $Rec([u]_0, [u]_1) \to u$ algorithm, which computes $([u]_0 + [u]_1) \mod L$ to obtain the original value u.

Additive secret sharing is a foundational technique in the design of SMPC protocols, particularly for machine learning operations such as addition and multiplication. In this approach, both the inputs and the outputs of the SMPC protocol are represented as additive shares. Specifically, for an operation f, denoted by the SMPC protocol π_f, the protocol transforms the additive shares of the inputs, $[inputs]_0$ and $[inputs]_1$, into the corresponding additive shares of the output, $[f]_0$ and $[f]_1$.

SS-based multiplication $u \times v$. To understand the secret sharing-based multiplication protocol, let us consider its execution by two parties: P_0 and P_1. Each party receives one additive share, denoted as $([u]_i, [v]_i)$, of the operation's inputs. Here, $i \in \{0, 1\}$ is associated with the respective party. To compute the additive shares of $u \times v$, Beaver triples (a, b, c) are used. Within these triples, a and b are random values in $\mathcal{Z}_L$, and c is derived as $a \times b \mod L$. Each party, P_i, starts by determining $[d]_i = [u]_i - [a]_i$ and $[e]_i = [v]_i - [b]_i$. Subsequently, the parties exchange their calculated values $[d]_i$ and $[e]_i$. Using these shared values, they then jointly reconstruct d through $\text{Rec}([d]_0, [d]_1)$ and e via $\text{Rec}([e]_0, [e]_1)$. The final computation for the additive share of $u \times v$ is derived as $[u \times v]_i = -j \times d \times e + [u]_i \times e + d \times [v]_i + [c]_i$. It is worth noting that for the multiplication SMPC protocol to be completed, both parties must partake in one communication round, involving a two-way exchange of two ring elements.

SS-based matrix multiplication $\mathbf{UV}$. The SS-based multiplication protocol can be straightforwardly extended to SS-based matrix multiplication [23]. Let $\mathcal{F}_{\text{MatMul}}(\mathbf{U}, \mathbf{V})$ denotes the SS-based matrix multiplication operation, where $\mathbf{U}$ is a $m \times n$ matrix and $\mathbf{V}$ is a $n \times k$ matrix. During this process, one round of bidirectional communication (between parties P_0 and P_1) is required, during which a total of $(m + k) \times n$ ring elements are transmitted.

While SS-based protocols offer efficiency for various operations, they are not universal. Purely additive secret sharing on $\mathcal{Z}_L$ encounters limitations, especially when dealing with specific operations like exponentiation and matrix inversion. To address these challenges, approximation techniques are often introduced. For instance, methods such as the Newton–Raphson iteration and Taylor expansion provide workarounds for these operations, as discussed in [78].

3.2.2. Fixed-point representation

SS-based SMPC protocols typically use a ring of integers for security. While many machine learning algorithms, like GP, favor floating-point numbers, they prove inefficient in SMPC [79]. Thus, fixed-point representation, offering better performance, is the common choice in this context.

In fixed-point encoding, all data is represented using a fixed number of bits, typically denoted as l. We can define a set of fixed-point numbers, denoted as $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$, where l_f represents the number of fractional bits (precision). The set is mapped from the ring of integers $\mathcal{Z}_L$, where $L = 2^l$. For floating-point numbers within a specific range, such as $[-2^{l-l_f-1}, 2^{l-l_f-1})$, this work adopts a rounding procedure to convert them to the nearest fixed-point numbers in $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$. Then, these fixed-point numbers are further mapped to integers in $\mathcal{Z}_L$ by multiplying them with 2^{l_f}.

Consider the example where $l = 7$ and $l_f = 4$. Given a floating-point number 1.937532 within the range $[-4, 4]$, the first step involves rounding it to a fixed-point number, resulting in 1.9375. This number belongs to $\mathcal{Q}_{<\mathcal{Z}_{2^7}, 4>}$. Subsequently, it is transformed into an element of $\mathcal{Z}_{2^7}$ using the formula $(1.9375 \times 2^4) \mod 2^7$, which equals 31. On the other hand, to convert an integer from $\mathcal{Z}_{2^7}$ to a fixed-point number in $\mathcal{Q}_{<\mathcal{Z}_{2^7}, 4>}$, you'd proceed as follows: take the integer 29 from $\mathcal{Z}_{2^7}$, and the conversion yields $29/2^4 = 1.8125$.

In this paper, all algorithms operate on $\mathcal{Z}_L$ and $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$. By selecting suitable l and l_f, fixed-point-based SMPC protocols can balance efficiency and accuracy. For simplicity, lowercase letters denote either

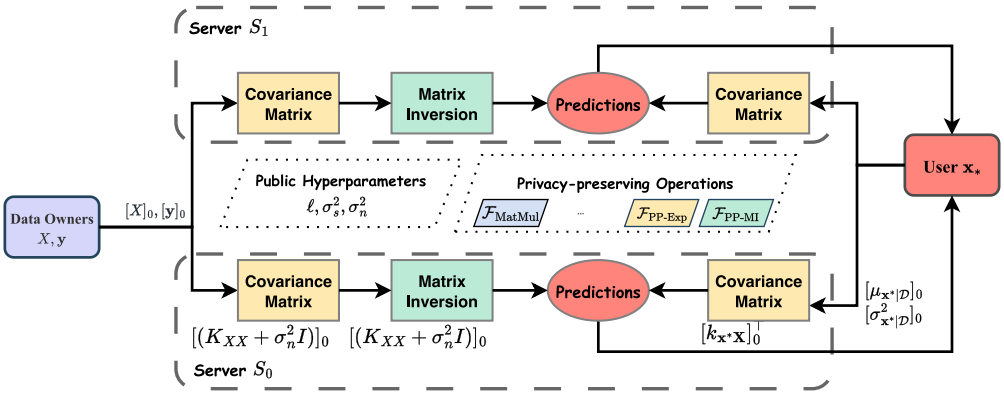


Fig. 3. Overview of our PP-GP framework. Data owners and users upload their private observations to the server in a shared representation format $([\mathbf{x}^*]_i, [X]_i, [\mathbf{y}]_i)$. The server then utilizes various SS-Based operators (e.g., $\mathcal{F}_{\text{PP-Exp}}$, $\mathcal{F}_{\text{PP-MI}}$) to collaboratively compute the kernel matrix, thereby completing the construction and inference of the privacy-preserving model $([\mu_{\mathbf{x}^*|\mathcal{D}}]_i, [\sigma^2_{\mathbf{x}^*|\mathcal{D}}]_i)$.

floating-point or integer numbers, while $\check{x}$ indicates a fixed-point number in $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$ derived from x. For simplicity, we use lowercase letters for both floating-point or integer numbers, and $\check{x}$ denotes the fixed-point representation of x in $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$. Moreover, the algorithm $Shr(x)$ will convert the input x to its corresponding representation in $\mathcal{Z}_L$, when x is a floating-point number.

3.3. Random feature

Random features (RF) [26] is a technique that is widely used to scale up kernel methods. It offers a simple and effective way to approximate the kernel function using explicit feature mapping. RF specifically applies to shift-invariant (stationary) kernels [80,81]:

$$k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}') = \int \varphi(\mathbf{x}, \mathbf{w})\varphi(\mathbf{x}', \mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w} \tag{3.3}$$

where $\varphi :\in \mathbb{R}^d \times \Omega \to \mathbb{R}$ is a continuous and bounded function with respect to $\mathbf{w}$ and $\mathbf{x}$. $p(\cdot)$ is the density function if the spectral measure. The core idea underlying RF is to approximate the kernel function $k(\mathbf{x}, \mathbf{x}')$, via Monte-Carlo estimation. Instead of directly computing the kernel function, we calculate its approximation using a set of random features:

$$\begin{aligned} k(\mathbf{x} - \mathbf{x}') &\approx \frac{1}{M}\sum_{j=1}^{M} \varphi(\mathbf{x}, \mathbf{w}_j)\varphi(\mathbf{x}', \mathbf{w}_j) \\ &= \left\langle \phi_M(\mathbf{x}), \phi_M(\mathbf{x}) \right\rangle. \end{aligned} \tag{3.4}$$

where $\phi_M(\mathbf{x}) = \frac{1}{\sqrt{M}}\left(\varphi(\mathbf{x}, \mathbf{w}_1), \ldots, \varphi(\mathbf{x}, \mathbf{w}_M)\right)^\top$ is a vector of random features for a given input, and the number of random features $M \ll n$. The vectors $\mathbf{w}_1, \ldots, \mathbf{w}_M$ are drawn independently in accordance with the density function $p(\mathbf{w})$. For many commonly used kernels, it is feasible to compute $p(\mathbf{w})$ in closed form, such as in the case of RBF kernel, which possesses zero-mean Gaussian spectral densities. The selection of the number of random features is at the user's discretion, with typical values ranging between 100–1000; more features in more precise approximations of the kernel function.

The Eq. (3.4) enables us to approximate the shift-invariant kernels with explicit feature maps through the Monte Carlo approximation of the Fourier representation. Therefore, RFs provide us with a computationally efficient approach to approximating kernel machines in machine learning tasks, which makes them one of the most popular techniques for accelerating kernel methods in large-scale problems [81].

4. Practical privacy-preserving Gaussian process

In this section, we present our privacy-preserving GP framework (PP-GP), which safeguards individual data privacy using SMPC techniques. First, we derive the architecture of PP-GP model. Then, we detail the essential SS-based non-linear operations in our approach: the exponentiation (PP-Exp) and matrix inversion (PP-MI). Finally, we analyze the communication complexity of these operations. The overview of our proposed framework is illustrated in Fig. 3.

4.1. Proposed method

Our privacy-preserving GP follows the framework of the standard GP algorithm. It aims to achieve comparable performance with the plaintext GP while ensuring data privacy. In practice, PP-GP adopts a three-party SMPC architecture: two computing servers, $\mathcal{S}_0$ and $\mathcal{S}_1$, and an assistant server, $\mathcal{T}$. In this setting, $\mathcal{S}_0$ and $\mathcal{S}_1$ receive an additive share of the data and process PP-GP algorithm, resulting in an additive share of the GP predictive outcomes. The assistant server, $\mathcal{T}$, primarily generates the random numbers essential for the SS-based protocols within PP-GP. The specific steps of PP-GP are detailed in Algorithm 1.

Initialization. For successful execution and to ensure coherence, all involved parties, including servers ($\mathcal{S}_0$, $\mathcal{S}_1$, and $\mathcal{T}$), data owners, and users, must establish a consensus on the algebraic structure. This consensus entails selecting suitable values for l and l_f that will define $\mathcal{Z}_{2^l}$ and $\mathcal{Q}_{<\mathcal{Z}_{2^l}, l_f>}$, respectively. After this step, data owners and users convert their private observations, denoted as $D = (X, \mathbf{y})$, and test inputs, represented by $\mathbf{x}^*$, into shared representations using the $Shr(\cdot)$ function. These shared representations $\left([\![\mathbf{x}^*]\!], [\![X]\!], [\![\mathbf{y}]\!]\right)$, are then transformed into individual shares $\left([\mathbf{x}^*]_i, [X]_i, [\mathbf{y}]_i\right)$, which are subsequently dispatched to the appropriate computing server $\mathcal{S}_i$ $(i = 0, 1)$.

Since each variable in X, $\mathbf{y}$, and $\mathbf{x}^*$ undergoes independent application of $Shr(\cdot)$, data shares can be easily computed, regardless of variable partitioning among data owners. This adaptability enables the SS-based GP algorithm to handle various data scenarios, suitable for diverse real-world applicability. GP hyperparameters $(\ell, \sigma_s^2, \sigma_n^2)$ are publicly shared between the computing servers. Once servers have data and hyperparameter shares, they initiate the SS-based protocols for PP-GP, producing shares of predictive results. With the shared data in place, the servers proceed with the construction and inference of the privacy-preserving GP model.

Model construction. During the model construction phase, the servers first calculate secret shares of the distance matrix $d(X, X') := \left[d(\mathbf{x}_i, \mathbf{x}'_j)\right]_{i,j=1,\ldots,n}$. To achieve this, we employ the SS-based protocol, denoted as $\mathcal{F}_{\text{dist}}(X, X')$. This protocol relies on standard SS-based addition and (matrix) multiplication operations. Once the distance matrix shares are computed, the servers proceed to calculate the shares $[\![K]\!]$. They utilize the privacy-preserving exponentiation operation, $\mathcal{F}_{\text{PP-Exp}}$. Following this, the servers compute the shares $[\![\mathbf{\Lambda}]\!]$ with the aid of the privacy-preserving matrix-inverse operation, $\mathcal{F}_{\text{PP-MI}}$, with the inputs $[\![K]\!] + \sigma_n^2 I$.

Algorithm 1: Privacy-Preserving Gaussian Process (PP-GP)

Setup: The servers $(\mathcal{S}_0, \mathcal{S}_1, T)$ choose appropriate l and l_f. shares $\left(\llbracket X \rrbracket, \llbracket \mathbf{y} \rrbracket, \llbracket \mathbf{x}^* \rrbracket\right)$.

Input: $\mathcal{S}_i$ has shares $\left([\mathbf{x}^*]_i, [X]_i, [\mathbf{y}]_i\right)$ and $(\ell, \sigma_s^2, \sigma_n^2)$, $i \in \{0, 1\}$.

Output: $\mathcal{S}_i$ gets $\left[\mu_{\mathbf{x}^*|D}\right]_i$, $\left[\sigma^2_{\mathbf{x}^*|D}\right]_i$.

/* Construction stage */

$\llbracket d(X, X) \rrbracket \leftarrow \mathcal{F}_{\text{dist}}(\llbracket X \rrbracket, \llbracket X \rrbracket)$

$\llbracket K \rrbracket \leftarrow \sigma_s^2 \cdot \mathcal{F}_{\text{PP-Exp}}(\llbracket -d(X,X)/2\ell^2 \rrbracket)$

$\llbracket \mathbf{\Lambda} \rrbracket \leftarrow \mathcal{F}_{\text{PP-MI}}(\llbracket (K + \sigma_n^2 I) \rrbracket)$

/* Prediction stage */

$\llbracket d(\mathbf{x}^*, \mathbf{x}^*) \rrbracket \leftarrow \mathcal{F}_{\text{dist}}(\llbracket \mathbf{x}^* \rrbracket, \llbracket \mathbf{x}^* \rrbracket)$, $\llbracket d(\mathbf{x}^*, X) \rrbracket \leftarrow \mathcal{F}_{\text{dist}}(\llbracket \mathbf{x}^* \rrbracket, \llbracket X \rrbracket)$

// Compute the kernel matrix

$\llbracket k(\mathbf{x}^*, \mathbf{x}^*) \rrbracket \leftarrow \sigma_s^2 \cdot \mathcal{F}_{\text{PP-Exp}}(\llbracket -d(\mathbf{x}^*, \mathbf{x}^*)/2\ell^2 \rrbracket)$

$\llbracket \mathbf{k}_* \rrbracket \leftarrow \sigma_s^2 \cdot \mathcal{F}_{\text{PP-Exp}}(\llbracket -d(\mathbf{x}^*, X)/2\ell^2 \rrbracket)$

$\llbracket \mathbf{k}_*^\top \mathbf{\Lambda} \rrbracket \leftarrow \mathcal{F}_{\text{MatMul}}(\llbracket \mathbf{k}_*^\top \rrbracket, \llbracket \mathbf{\Lambda} \rrbracket)$

// Compute the predictive moments

$\llbracket \mu^2_{\mathbf{x}^*|D} \rrbracket \leftarrow \mathcal{F}_{\text{MatMul}}\left(\llbracket \mathbf{k}_*^\top \mathbf{\Lambda} \rrbracket, \llbracket \mathbf{y} \rrbracket\right)$

$\llbracket \sigma^2_{\mathbf{x}^*|D} \rrbracket \leftarrow \llbracket k(\mathbf{x}^*, \mathbf{x}^*) \rrbracket - \mathcal{F}_{\text{MatMul}}\left(\llbracket \mathbf{k}_*^\top \mathbf{\Lambda} \rrbracket, \llbracket \mathbf{k}_* \rrbracket\right)$

Prediction. At the prediction stage, the model user invokes $Shr(\cdot)$ to generate $[\mathbf{x}^*]_0$ and $[\mathbf{x}^*]_1$ for each test input and sends each share to the corresponding computing server. Then, the servers calculate $\llbracket \mathbf{k}_* \rrbracket$ and $\llbracket k(\mathbf{x}^*, \mathbf{x}^*) \rrbracket$ using $\mathcal{F}_{\text{dist}}$ and $\mathcal{F}_{\text{PP-Exp}}$. They then obtain the shares of the predictive mean $\llbracket \mu_{\mathbf{x}^*|D} \rrbracket = \llbracket \mathbf{k}_*^\top \rrbracket \llbracket \mathbf{\Lambda} \rrbracket \llbracket \mathbf{y} \rrbracket$ and variance $\llbracket \sigma^2_{\mathbf{x}^*|D} \rrbracket = \llbracket k(\mathbf{x}^*, \mathbf{x}^*) \rrbracket - \llbracket \mathbf{k}_*^\top \rrbracket \llbracket \mathbf{\Lambda} \rrbracket \llbracket \mathbf{k}_* \rrbracket$ using the matrix multiplication operation $\mathcal{F}_{\text{MatMul}}$. Finally, the computing servers send the shares of the prediction results to the model user. This allows the user to obtain the prediction results locally through $Rec(\cdot)$.

Algorithm 2: Privacy-preserving exponentiation ($\mathcal{F}_{\text{PP-Exp}}$)

Setup: The servers $(\mathcal{S}_0, \mathcal{S}_1, T)$ choose appropriate l, l_f, u_{min} and $\check{r}_{max}$.

Input: $\mathcal{S}_i$ has share $[u]_i$, $i \in \{0, 1\}$.

Output: $\mathcal{S}_i$ gets share $[\exp(u)]_i$.

/* Offline phase executed on assistant server T */

T draws $\check{r}$ in the range $[-\check{r}_{max}, \check{r}_{max})$ randomly, computes $r \leftarrow \check{r} \cdot 2^{l_f}$, and generates $\left([r]_0, [r]_1\right) \in \mathcal{Z}_L$

T computes $\exp(-\check{r})$ in $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$ and generates $\left([\exp(-\check{r})]_0, [\exp(-\check{r})]_1\right) \in \mathcal{Z}_L$

T sends $[r]_i$ and $[\exp(-\check{r})]_i$ to $\mathcal{S}_i (i = 0, 1)$

/* Online phase */

for $i = 0, 1$ *in parallel* **do**

  $\mathcal{S}_i$ computes $[d]_i \leftarrow [u]_i + [r]_i$ and sends $[d]_i$ to each other

  $\mathcal{S}_i$ computes $d \leftarrow Rec\left([d]_0 + [d]_1\right)$ and $\check{d} \leftarrow d/2^{l_f}$

  $\mathcal{S}_i$ computes $\exp(\check{d})$ in $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$ and

  $[\exp(u)]_i \leftarrow \exp(\check{d}) \cdot 2^{l_f} \cdot [\exp(-\check{r})]_i$

end

4.2. Privacy-preserving operation construction

To facilitate secure interactions between servers $\mathcal{S}_0$ and $\mathcal{S}_1$, we design efficient sub-protocols based on the additive SS techniques. We focus on the essential privacy-preserving operations, the exponentiation $\mathcal{F}_{\text{PP-Exp}}$ and matrix inversion $\mathcal{F}_{\text{PP-MI}}$, for PP-GP algorithm.

4.2.1. Privacy-preserving exponentiation

While additive SS can handle operations like addition and multiplication directly, it struggles with exponentiation. A commonly employed solution is to approximate the exponentiation using its Taylor expansion: $\exp(u) = \sum_{k=0}^{\infty} u^k/k!$. This transforms the exponentiation into a series of additions and multiplications. However, the exponential function grows much faster than any polynomial, which means that this Taylor series approximation can introduce substantial errors. Increasing the degree of the polynomial can enhance accuracy, but this comes at the cost of greater communication, especially given the need for information exchange in SS-based multiplication. To tackle this, Knott et al. [78] proposed a new method. They utilized the limit approximation: $\exp(u) = \lim_{k\to\infty}(1 + u/2^k)^{2^k}$. This approach employs the repeated squaring algorithm, efficiently generating higher-order polynomials iteratively. However, even with this refined technique, obtaining precise approximation results demands significant communication and computational overhead.

We introduce a novel approach for constructing a privacy-preserving exponentiation operation, termed PP-Exp. This approach draws on the concept of confusion-correction. In the PP-Exp operation, each computing server, $\mathcal{S}_0$ and $\mathcal{S}_1$, receives an additive share $[u]_j$ of a private number u within the range $[u_{min}, 0]$. The objective is to enable these servers to deduce the additive shares of $\exp(u)$ in a confidential manner, aided by random numbers supplied by a trusted entity T. The PP-Exp algorithm unfolds in several steps:

1. The servers apply a masking operation on the share of u, adding a random value r which results in the share $u - r$.
2. They then collaboratively reveal this obfuscated value $u - r$.
3. With the obfuscated value in hand, each server computes the obfuscated outcome $\exp(u - r)$.
4. Finally, each server corrects its share of $\exp(u - r)$ by stripping away the mask, ultimately obtaining the share of $\exp(u)$.

The pseudo-code for the PP-Exp operation is provided in Algorithm 2. It is important to note that PP-Exp is specifically designed to handle negative input values u. This choice aligns with the fact that commonly used kernel functions in GP, such as RBF kernel, involve the exponentiation of negative values. The correctness and security of the proposed PP-Exp operation can be achieved by carefully selecting appropriate values for $[-\check{r}_{max}, \check{r}_{max})$, which will be discussed in detail later.

4.2.2. Privacy-preserving matrix inversion

Previous research [78] attempted to approximate matrix inversion using the Newton–Raphson iteration, a local optimization technique. However, this approach heavily depends on the initial value of the algorithm. This becomes challenging in secure computation, where no information about the original input matrix is accessible to determine an initial inverted matrix that satisfies the convergence condition.

PP-MI converts the matrix inversion process into SMPC-suited operations, primarily multiplication and division, using the Cholesky decomposition. Specifically, with $\mathbf{K} + \sigma_n^2\mathbf{I}$ as a positive definite matrix, its inversion is computed as $\mathbf{K} + \sigma_n^2\mathbf{I} = \mathbf{L}\mathbf{D}\mathbf{L}^\top$, where $\mathbf{L}$ and $\mathbf{D}$ are a lower triangular and diagonal matrix, respectively. This method, which only requires addition, multiplication, and division between matrix elements, is then securely adapted to the SS-based version. In PP-MI, each server $\mathcal{S}_i$ $(i = 0, 1)$ processes an additive share $[\mathbf{U}]_i$ of matrix $\mathbf{U} \in \mathcal{Z}_L^{n\times n}$ as input to privately compute a share of $\mathbf{U}^{-1}$.

Let $\mathbf{U}$ denote an $n\times n$ positive definite matrix, with its entries defined as $u_{h,k}$, for $h, k = 1, 2, \ldots, n$. It can be factorized into the form $\mathbf{U} = \mathbf{L}\mathbf{D}\mathbf{L}^\top$, where $\mathbf{L}$ represents a unit lower triangular matrix, and $\mathbf{D}$ is a diagonal matrix. Consider d_k to represent the k the diagonal element of $\mathbf{D}$. The matrices $\mathbf{L}$ and $\mathbf{D}$ can be computed as follows:

$$\begin{cases} d_k = u_{k,k} - \sum_{m=1}^{k-1} l_{k,m}^2 \times d_m \,, \\ l_{h,k} = \left(u_{h,k} - \sum_{m=1}^{k-1} l_{h,m} \times l_{k,m} \times d_m\right)/d_k \,. \end{cases} \tag{4.1}$$

Assuming that $\mathbf{U}\mathbf{\Lambda} = \mathbf{I}$, where $\mathbf{I}$ is an $n \times n$ identity matrix, we now outline the procedure for computing $\mathbf{\Lambda}$ based on $\mathbf{I}$, $\mathbf{L}$, and $\mathbf{D}$.

Consequently, we can obtain $\mathbf{U}^{-1} = \mathbf{\Lambda}$. Define $\mathbf{V} := \mathbf{D}\mathbf{L}^{\top}\mathbf{\Lambda}$. Then, we have $\mathbf{U}\mathbf{\Lambda} = (\mathbf{L}\mathbf{D}\mathbf{L}^{\top})\mathbf{\Lambda} = \mathbf{L}\mathbf{V} = \mathbf{I}$, where $\mathbf{V}$ is a unit lower triangular matrix. Moreover, for $k = 1, \ldots, n$ and $h = k+1, \ldots, n$, we have each elements of $\mathbf{V}$,

$$v_{h,k} = -\sum_{m=1}^{h-1} v_{m,k} \times l_{h,m}. \tag{4.2}$$

Finally, we compute the matrix $\mathbf{\Lambda}$,

$$\mathbf{\Lambda} = \left(\mathbf{L}\mathbf{D}\mathbf{L}^{\top}\right)^{-1} = (\mathbf{L}^{-1})^{\top}\mathbf{D}^{-1}\mathbf{L}^{-1} = \mathbf{V}^{\top}\mathbf{D}^{-1}\mathbf{V}. \tag{4.3}$$

Combining Eqs. (4.1)–(4.3), we can achieve matrix inversion using privacy-preserving multiplication ($\mathcal{F}_{\text{MatMul}}$) and privacy-preserving division ($\mathcal{F}_{\text{Div}}$) appropriately [78]. Building upon this intuition, we propose a novel privacy-preserving matrix multiplication algorithm, named PP-MI, which provably addresses the issues of initial value. The pseudo-code for the PP-MI operation is provided in Algorithm 3.

4.3. Analysis of communication complexity

Now, we analyze the theoretical communication rounds and communication volume for the proposed PP-Exp and PP-MI. We assume that the assistant server T has pre-generated a sufficient number of random numbers for the PP-GP calculation process during the offline phase. These random number shares have been transmitted in advance to the respective computing servers. The PP-Exp algorithm on an n-dimensional vector $\mathbf{u}$, as outlined in Algorithm 2, requires only one communication round at Line 6. The data exchanged during this interaction amounts to $\mathcal{O}(2nl)$.

In PP-MI, we employ Cholesky decomposition to break down the matrix inversion process into basic operations such as addition, multiplication, and division. A single-element SS-based multiplication operation involves only one round of bidirectional communication, transmitting $2l$ volume. For singular element division, we adopt the privacy-preserving division in Crypten [78], requiring 17 rounds and a communication volume of $\mathcal{O}(l)$. For an $n \times n$ positive definite matrix in PP-MI, there are n division rounds and $5n-6$ multiplication rounds, totaling $22n-6$ communication rounds. Computationally, PP-MI demands $\mathcal{O}(n^3)$ element multiplications and $\mathcal{O}(n^2)$ element divisions. Thus, the overall communication volume for the PP-MI algorithm is $\mathcal{O}(n^3 l)$.

5. Efficient privacy-preserving Gaussian process

In this section, we explore methods to expand the proposed privacy-preserving GP algorithm to accommodate larger-scale datasets. The overall framework of the method is illustrated in Fig. 4.

In general, the advantages of the GP come with significant computational and space demands. Exact GP inference on substantial datasets necessitates confronting the time and space constraints associated with linear system solutions. Specifically, the GP posterior evaluation, involving a matrix inversion, demands a time complexity of $\mathcal{O}(n^3)$ time complexity for n data points. Moreover, the Cholesky decomposition consumes $\mathcal{O}(n^2)$ memory, accounting for both the lower-triangular factor L and the kernel matrix. With $n = 500,000$, this decomposition necessitates an entire terabyte of memory and an extensive amount of computational resources, as highlighted by [60]. Consequently, employing SMPC protocols introduces pronounced communication bottlenecks, particularly with non-linear operations like exponentiation and matrix inversion, rendering the privacy-preserving GP largely unsuitable for extensive datasets.

In the context of GP, one widespread strategy for accommodating kernel methods on expansive datasets is to substitute the kernel matrix K with its approximate low-rank factorization. Such a substitution effectively transforms the kernel function k into a finite-dimensional inner product $k(\mathbf{x}, \mathbf{x}') \approx \langle \boldsymbol{\phi}_M(\mathbf{x}), \boldsymbol{\phi}_M(\mathbf{x}') \rangle$, derived from a feature map $\boldsymbol{\phi}(\cdot) : \mathbb{R}^d \to \mathbb{R}^M$. This approximation considerably speeds up the downstream training phase. However, pinpointing an optimal feature map remains a complex endeavor. A well-regarded method in this context is the random Fourier features approach [26], which offers an approximation to the Gaussian (RBF) Kernel.

Algorithm 3: Privacy-preserving matrix inversion (PP-MI)

Setup: The servers $(\mathcal{S}_0, \mathcal{S}_1, T)$ choose appropriate l and l_f.
Input: $\mathcal{S}_i$ has the share $[\mathbf{U}]_i$, $i \in \{0, 1\}$.
Output: $\mathcal{S}_i$ gets share $[\mathbf{\Lambda}]_i$.

`/* Offline phase */`

1 T generates $\mathbf{A}, \mathbf{B} \in Z_L^{n \times n}$ randomly, and computes $\mathbf{C} = \mathbf{A}\mathbf{B}$
2 T sends $([\mathbf{A}]_i, [\mathbf{B}]_i, [\mathbf{C}]_i)$ to $\mathcal{S}_i$

`/* Online phase */`

`// Step 1:Privacy-preserving Cholesky decomposition.` $\mathcal{S}_i$ `gets` $[\mathbf{L}]_i$, $[\mathbf{D}]_i$

3 **for** $i = 0, 1$ *in parallel* **do**
4 $\quad [d_1]_i \leftarrow [u_{1,1}]_i$
5 $\quad \mathcal{S}_i$ computes $[l_{1:n,1}]_i \leftarrow [u_{1:n,1}]_i / [u_{1,1}]_i$ using $\mathcal{F}_{\text{Div}}$
6 $\quad$ **for** $k = 2, 3, \ldots, n$ **do**
7 $\quad\quad \mathcal{S}_i$ computes $[d_k]_i \leftarrow [u_{k,k}]_i - [\sum_{m=1}^{k-1} l_{k,m}^2 d_m]_i$ using $\mathcal{F}_{\text{MatMul}}$
8 $\quad\quad \mathcal{S}_i$ computes $[\hat{l}_{k+1:n,k}]_i \leftarrow [u_{k+1:n,k}]_i - [\sum_{m=1}^{k-1} l_{k,m} d_m l_{k+1:n,m}]_i$ using $\mathcal{F}_{\text{MatMul}}$
9 $\quad\quad \mathcal{S}_i$ computes $[l_{k+1:n,k}]_i \leftarrow [\hat{l}_{k+1:n,k}]_i / [d_k]_i$ using $\mathcal{F}_{\text{Div}}$
10 $\quad$ **end**
11 **end**

`// Step 2:Privacy-preserving Forward and Backward.` $\mathcal{S}_i$ `gets` $[\mathbf{V}]_i$ `and` $[\mathbf{\Lambda}]_i$

12 **for** $i = 0, 1$ *in parallel* **do**
13 $\quad v_{1,1} \leftarrow 1$
14 $\quad$ **for** $k = 2, 3, \ldots, n$ **do**
15 $\quad\quad \mathcal{S}_i$ computes $[v_{k,1:k-1}]_i \leftarrow -[\sum_{m=1}^{k-1} v_{m,1:k-1} l_{k,m}]_i$ using $\mathcal{F}_{\text{MatMul}}$
16 $\quad$ **end**
17 $\quad \mathcal{S}_i$ computes $[\mathbf{\Lambda}]_i \leftarrow [\mathbf{V}^{\top}]_i [\mathbf{D}^{-1}]_i [\mathbf{V}]_i$ using $\mathcal{F}_{\text{Div}}$ and $\mathcal{F}_{\text{MatMul}}$
18 **end**

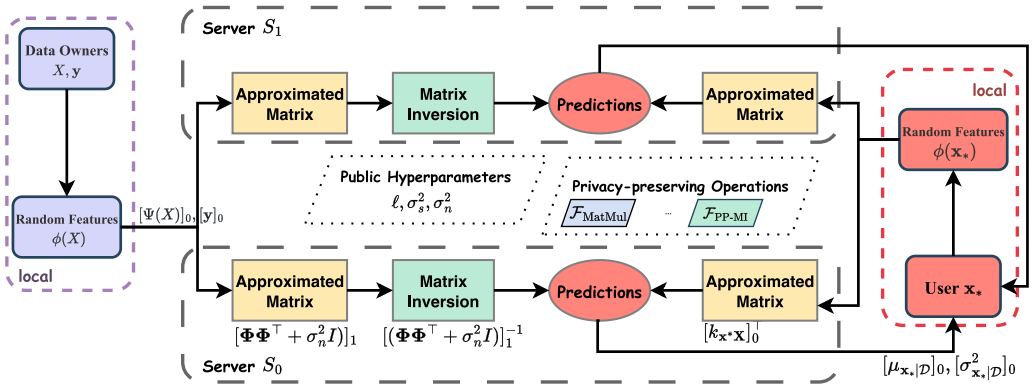


Fig. 4. Overview of our Split-GP framework. Data holders and users use SS techniques to compute the random features based on their private data ($[\mathbf{\Phi}]_i$), individually. Subsequently, servers combine these random expansions, perform a low-rank factorization for kernel approximation $[\mathbf{\Phi}\mathbf{\Phi}^\top + \sigma_n^2 I]_i$, and construct the predictive model ($[\mu_{\mathbf{x}^*|\mathcal{D}}]_i, [\sigma^2_{\mathbf{x}^*|\mathcal{D}}]_i$).

Random Fourier features. Given $k(\mathbf{x}, \mathbf{x}') = G(\mathbf{x} - \mathbf{x}')$, where $G(z) = e^{-\frac{1}{2\sigma^2}\|z\|^2}$, for $\sigma > 0$, then we have

$$G(\mathbf{x} - \mathbf{x}') = \frac{1}{2\pi Z} \iint_0^{2\pi} \sqrt{2}\cos(\mathbf{w}^\top \mathbf{x} + b) \times \sqrt{2}\cos(\mathbf{w}^\top \mathbf{x}' + b) e^{-\frac{\sigma^2}{2}\|\mathbf{w}\|^2} \mathrm{d}\mathbf{w}\mathrm{d}b \tag{5.1}$$

where Z is a normalizing factor. For the RBF kernel, the feature map is defined as:,

$$\boldsymbol{\phi}_M(\mathbf{x}) = M^{-1/2} \times \left(\sqrt{2}\cos(\mathbf{w}_1^\top \mathbf{x} + b_1), \ldots, \sqrt{2}\cos(\mathbf{w}_M^\top \mathbf{x} + b_M)\right), \tag{5.2}$$

with $\mathbf{w}_1, \ldots, \mathbf{w}_M$ and $b_1, \ldots, b_M$ sampled independently from $p(\mathbf{w}) = \frac{1}{Z} e^{-\sigma^2\|\mathbf{w}\|^2/2}$ and uniformly within $[0, 2\pi]$, respectively.

With this on hand, we achieve significant computational benefits. For the predictive distribution, we utilize a more efficient form:

$$\mu_{f|D}(\mathbf{x}^*) = \boldsymbol{\phi}_M(\mathbf{x})^\top \left(\mathbf{\Phi}\mathbf{\Phi}^\top + \sigma_n^2 I\right)^{-1} \mathbf{\Phi}\bar{\mathbf{y}}, \tag{5.3a}$$

$$\sigma^2_{f|D}(\mathbf{x}^*) = \boldsymbol{\phi}_M(\mathbf{x})^\top \left(\mathbf{\Phi}\mathbf{\Phi}^\top + \sigma_n^2 I\right)^{-1} \boldsymbol{\phi}_M(\mathbf{x}), \tag{5.3b}$$

where $\mathbf{\Phi} = \frac{1}{\sqrt{n}}\left(\boldsymbol{\phi}_M(\mathbf{x}_1), \ldots, \boldsymbol{\phi}_M(\mathbf{x}_n)\right)$, $\bar{\mathbf{y}} = \frac{1}{\sqrt{n}}(y_1, y_2, \ldots, y_n)^\top$.

This motivates the development of efficient privacy-preserving GPs to address concerns related to privacy and efficiency. Inspired by the existing work in split learning, we introduce a novel splitting privacy-preserving GP learning paradigm to address these challenges, termed split-GP. This paradigm splits the whole computation process into two distinct parts: we allocate privacy-sensitive and SMPC-hostile computations to data holders while delegating SMPC-friendly computations to a semi-honest server, for computational efficiency and memory concerns. In particular, data holders first compute the random features $\boldsymbol{\phi}_M(\mathbf{x})$ via SMPC techniques using local private data, individually. Servers combine the locally derived random features from data holders to derive the kernel's approximate low-rank factorization, collaboratively. Then, the server conducts follow-up computations using the proposed matrix inversion operation. In the end, servers derive the final prediction distribution.

The resulting algorithm, which we refer to as split-GP, is outlined in Algorithm 4. Specifically, there are several changes compared to PP-GP:

- At the initialization, data owners and users construct the random features on our private observation after the consensus step, individually. Then, data owners and users convert their random features, $\mathbf{\Phi}$ and $\boldsymbol{\phi}_M(\mathbf{x})$, into shared representations using the $Shr(\cdot)$ function. These shared representations are then transformed into individual shares, $[\mathbf{\Phi}]_i$, $[\boldsymbol{\phi}_M(\mathbf{x}^*)]_i$, $[\mathbf{y}]_i$, which are subsequently dispatched to the appropriate computing server S_i $(i = 0, 1)$.
- During the model construction phase, servers first compute the shares $[\![\mathbf{\Phi}\mathbf{\Phi}^\top]\!]$ and $[\![\mathbf{\Lambda}]\!]$ with the aid of the privacy-preserving matrix multiplication $\mathcal{F}_{\text{MatMul}}$ and matrix inverse operation $\mathcal{F}_{\text{PP-MI}}$.
- At the prediction stage, the model user invokes $Shr(\cdot)$ to generate $[\boldsymbol{\phi}_M(\mathbf{x}^*)]_0$ and $[\boldsymbol{\phi}_M(\mathbf{x}^*)]_1$, instead of $[\mathbf{x}^*]_0$ and $[\mathbf{x}^*]_1$, and sends these shares to the corresponding computing server. Then, servers compute $[\![\boldsymbol{\phi}_M^\top \mathbf{\Lambda}]\!]$ and obtain the shares of the predictive mean $[\![\mu^2_{\mathbf{x}^*|D}]\!] = \mathcal{F}_{\text{MatMul}}([\![\boldsymbol{\phi}_M^\top \mathbf{\Lambda}]\!], [\![\mathbf{\Phi}\mathbf{y}]\!])$ and variance $[\![\sigma^2_{\mathbf{x}^*|D}]\!] = \mathcal{F}_{\text{MatMul}}([\![\boldsymbol{\phi}_M^\top \mathbf{\Lambda}]\!], [\![\boldsymbol{\phi}_M(\mathbf{x}^*)]\!])$. Similarly, servers finally send the shares of the prediction results to the model user.

Algorithm 4: Efficient Privacy-Preserving Gaussian Process via Split Learning (Split-GP)

Setup: The servers (S_0, S_1, T) choose appropriate l and l_f. The shares $([\![\boldsymbol{\phi}_M(\mathbf{x}^*)]\!], [\![\mathbf{\Phi}]\!], [\![\mathbf{\Phi}\mathbf{y}]\!])$.

Input: S_i has shares $([\boldsymbol{\phi}_M(\mathbf{x}^*)]_j, [\mathbf{\Phi}]_j, [\mathbf{\Phi}\mathbf{y}]_j)$ and $(\ell, \sigma_s^2, \sigma_n^2)$, $i \in \{0, 1\}$.

Output: S_i returns $[\mu_{\mathbf{x}^*|D}]_i$, $[\sigma^2_{\mathbf{x}^*|D}]_i$.

```
/* construction stage */
[[ΦΦ^⊤]] ← σ_s^2 · F_MatMul([[[[Φ]], [[Φ]]^⊤]])
[[Λ]] ← F_PP-MI([[(ΦΦ^⊤ + σ_n^2 I)]])
/* Prediction stage */
[[φ_M^⊤ Λ]] ← F_MatMul([[φ_M^⊤(x*)]], [[Λ]])
// Compute the predictive moments
[[μ²_{x*|D}]] ← F_MatMul([[φ_M^⊤ Λ]], [[Φy]])
[[σ²_{x*|D}]] ← F_MatMul([[φ_M^⊤ Λ]], [[φ_M(x*)]])
```

It requires $\mathcal{O}(M^3)$ and $\mathcal{O}(M^2 n)$ time to solve the inverse of $(\mathbf{\Phi}\mathbf{\Phi}^\top + \sigma_M^2 I)$ and the matrix multiplication $\mathbf{\Phi}\mathbf{\Phi}^\top$ in model construction phase. The predictive distribution is computed in $\mathcal{O}(M^2 n)$, with the predictive mean and variance for an extra test point calculated in $\mathcal{O}(M)$ and $\mathcal{O}(M^2)$, respectively. The storage costs are also reduced, since we store only the $n \times M$ design matrix $\mathbf{\Phi}$, rather than the full $n \times n$ covariance matrix.

6. Correctness and security proof

In this section, we present our main theoretical contributions. First, we confirm the integrity and security of the proposed operations, $\mathcal{F}_{\text{PP-Exp}}$ and $\mathcal{F}_{\text{PP-MI}}$. Subsequently, based on universally composable [82] security, we deduce the security of both PP-GP and Split-GP algorithms. All proofs are deferred to the Appendix A.

Like previous works [83–86], we provably provide the security of proposed operations against a static semi-honest probabilistic polynomial time (PPT) adversary $\mathcal{A}$ following the simulation paradigm [87–89]. Specifically, a computationally bounded adversary $\mathcal{A}$ passively corrupts either servers at the beginning of the protocol Π but follows the protocol specification honestly. The simulation paradigm defines

two distinct worlds: a real world, where servers conduct the protocol as per the specification in the presence of $\mathcal{A}$, and an ideal world, where the parties send their inputs to a trusted party that computes the functionality f faithfully. The executions in both worlds are coordinated by the environment Env, which selects the inputs for the parties and assumes the role of distinguishing between the real and ideal executions. Security requires that for any adversary, the real-world distribution is *computationally indistinguishable* from the ideal-world distribution. Namely, for any adversary $\mathcal{A}$ in the real world, there is a simulator Sim in the ideal world, such that no environment Env can distinguish between real and ideal worlds. We recap the definition of a private protocol in [85,90] as below:

Definition 1. A protocol Π between servers, which have as input the shares of features X (or random features $\phi_M(X)$) from data owners and input vector $\mathbf{x}$ from the user, is a private inference protocol (against honest-but-curious adversaries) if it satisfies the following guarantees.

- **Correctness**: On every set of features X (or $\phi_M(X)$) and every input sample $\mathbf{x}$, the output of the user at the end of the protocol is the correct inference $f(\mathbf{x})$.
- **Security**: We require that the corrupted, semi-honest servers do not learn anything about the private input of one another. Formally, for any corrupted server $S_j (j \in \{0,1\})$, we require the existence of an efficient simulator Sim_j such that

$$\left\{Sim_j(x_j, f_j(x_0, x_1)), \lambda\right\}_{x_j, x_1 \in Z_L} \overset{c}{\equiv} \left\{view_0^{\Pi}(x_0, x_1)\right\}_{x_0, x_1 \in Z_L},\ j \in \{0,1\},$$

where $view_j^{\Pi}(x_0, x_1)$ is the *view* of the server S_j in the execution of Π, $f = (f_0, f_1)$ is the functionality with f_j executed on Server S_j, λ denotes some public parameters, and $\overset{c}{\equiv}$ to denote the *computationally indistinguishable* [87].

We aim to formalize the idea that a protocol is considered secure if any computation performed by a party participating in the protocol can be solely derived from its input and output. Hence, security here can be formalized by saying that a party's view during a protocol execution could be simulated based on its input and output. This formulation suggests that the parties gain no additional knowledge from the protocol execution beyond what they can deduce from their given input and expected output.

It is worth noting that the honest-but-curious security proof for PP-GP and Split-GP according to the above definition, can be straightforwardly derived from the sequential composability of individual sub-protocols [78,82]. Consequently, our focus is directed towards furnishing a security proof specifically for our non-linear and matrix operation protocols, i.e., PP-Exp and PP-MI.

To ensure the validity of the proposed PP-Exp operation, our foremost concern is to prevent any potential overflow or underflow in the fixed-point calculations (e.g., Line 2 ($\exp(-\check{r}) \in \mathcal{Q}_{<\mathcal{Z}_L, l_f>}$) and Line 7 ($\exp(\check{d}) \in \mathcal{Q}_{<\mathcal{Z}_L, l_f>}$) in Algorithm 2). Thus, it is imperative that the chosen values for $[-\check{r}_{max}, \check{r}_{max})$ and l_f adhere to the following relationship.

Theorem 6.1 (*Correctness of PP-Exp*). *For any value of u within the range $[u_{\min}, 0]$, under the condition that $(\check{r}\max - u\min)\log_2^{\exp} \leq l_f < \frac{l-1}{2}$, the PP-Exp operation can accurately compute $\left([\exp(u)]_0, [\exp(u)]_1\right)$ from $([u]_0, [u]_1)$. Then, it holds that $[\exp(u)]_0 + [\exp(u)]_1 = \exp(u)$.*

Proof. We defer the proof to Appendix A.1 □

Remark. As an illustrative example, when working with the set of integers $\mathcal{Z}_{2^{64}}$, and assuming that u takes values within the interval $[-4, 0]$, while $\check{r}$ is constrained within $[-16, 16)$, setting $l_f = 29$ is sufficient to guarantee the correctness of the PP-Exp operation.

Moving forward, our attention shifts to the security analysis of the PP-Exp operation. Within Algorithm 2, it is noteworthy that Line 6 (i.e. $d \leftarrow Rec([d]_0 + [d]_1)$) represents the sole step in which the variable d is reconstructed in the fixed-point domain, thereby introducing a potential risk of information leakage concerning the value of u.

To elaborate further, given the maximal ranges for u and $\check{r}$ (i.e., $[u_{min}, 0]$ and $[-\check{r}_{max}, \check{r}_{max})$), the knowledge of d could be leveraged to narrow down the feasible range of u. Such an inference would constitute an instance of information leakage. For instance, we assume that u can assume values from the set $\{-2, -1, 0\}$, r can be drawn from $\{-1, 0, 1\}$, and we have $d = u + r$. Under these conditions, if it is known that $d = -2$, one can deduce that u must be either -2 or -1. To rigorously assess the extent of privacy leakage, the concept of the degree of information leakage can be stated as follows:

Definition 2. Let $\mathcal{U}$ be a finite set and $u \in \mathcal{U}$. Then, the degree of information leakage about u can be defined as $\frac{1}{|\mathcal{U}|}$.

We consider an algorithm secure when the amount of information leaked regarding the input remains consistent throughout the entire execution of the algorithm. Given a fixed precision level l_f, let m_u and m_r denote the number of fixed-point numbers that can be represented within the intervals $[u_{min}, 0]$ and $[-\check{r}_{max}, \check{r}_{max})$, respectively. The following theorem gives us the security of the PP-Exp operation:

Theorem 6.2 (*Security of PP-Exp*). *For any fixed u within the range $[u_{min}, 0]$, the PP-Exp holds secure, with the probability $\frac{m_r - m_u + 1}{m_r}$. Additionally, the expected degree of information leakage on u is $\frac{m_u + m_r - 1}{m_u \cdot m_r}$.*

Proof. The proof of Theorem 6.2 is deferred to Appendix A.3. □

Remark. The result in Theorem 6.2 demonstrates that the security of PP-Exp operation is influenced by the relative sizes of the sets of values for u and r (i.e., $m_r - m_u$). Specifically, when the number of values of r significantly surpasses the number of values of u, indicated by a larger $m_r - m_u$, the PP-Exp operation guarantees a minimal probability of privacy breaches.

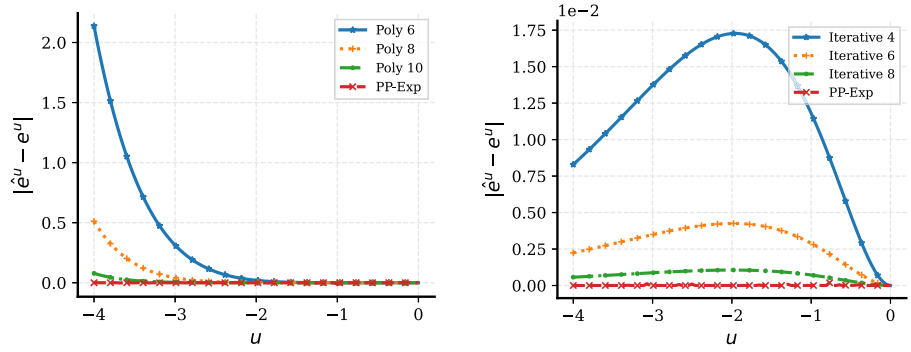
Opting for a wider range for $\check{r}$ (i.e., a larger $\check{r}_{max}$) can substantially mitigate the degree of information leakage. However, it is important to note that a larger $\check{r}_{max}$ may lead to a larger l_f and subsequently a larger l, as outlined in Theorem 6.1. Consequently, this can result in increased communication costs, as will be discussed in Section 4.3.

It is also worth noting that the PP-Exp operation only leaks the exact value of u with a probability of $\frac{2}{m_u m_r}$. In other words, this occurs when both u and r assume the maximum or minimum values within their respective ranges. As an illustrative example, when $l_f = 29$ and considering that the input u can range from $[-4, 0]$ and r can range from $[-16, 16)$, we find that $m_u = 2^{31} + 1$ and $m_r = 2^{34}$. In this scenario, the PP-Exp operation achieves security with a probability of $\frac{7}{8}$. The degree of information leakage is calculated as $\frac{9}{2^{34}+8}$, indicating an increase of $\frac{1}{2^{34}+8}$ over the secure state. Furthermore, the probability of revealing a specific value of u is less than $\frac{1}{2^{64}}$.

Theorem 6.3 (*Correctness and Security of PP-MI*). *Given a positive definite matrix $U \in Z_L^{n \times n}$, the PP-MI operation securely derives $([U^{-1}]_0, [U^{-1}]_1)$ from $([U]_0, [U]_1)$, satisfying $[U^{-1}]_0 + [U^{-1}]_1 = U^{-1}$.*

Proof. The proof is also deferred to Appendix A.2. □

Remark. The correctness and security of PP-MI hinge on the precision and security of the underlying PP-MM and PP-Div operations. The results in Theorem 6.3 indicate that the PP-MI operation satisfies correctness and security, which provide a further theoretical guarantee for our proposed approaches.



(a) Polynomial approximation methods (b) Iterative approximation methods

Fig. 5. The performance of the proposed PP-Exp in comparison to (a) polynomial approximation methods using Taylor expansions, and (b) iterative approximation methods employing the limit approximation of the exponential function.

In light of the above discussion, these protocols are numerically precise, which preserves the model accuracy of plaintext. Additionally, we provide a formal security proof for our designed protocols to demonstrate their security guarantee. While PP-GP presents a minimal probability of leakage risk, Split-GP is completely devoid of such risks.

7. Numerical experiments

We now empirically evaluate the proposed privacy-preserving GPs and their underpinning protocols. We evaluated the performance of PP-GP and Split-GP in comparison with standard GP methods on various large-scale datasets from the UCI dataset repository. Our experiments demonstrate that: (1) the proposed operators (PP-MI and PP-Exp) are both correct and efficient; (2) there is a negligible performance difference between PP-GP and standard GP, which further confirms the correctness of our proposed method; and (3) while Split-GP exhibits some performance loss, likely due to kernel approximation, it significantly reduces running time and communication volume, particularly in larger datasets ($n > 10^4$). In addition, we explore the impact of the number of random features on the performance of Split-GP, and prove that effective results can be achieved with some features significantly smaller than the sample size. Finally, we show that DP-GP incurs considerably higher computational errors due to the added DP noise, in contrast to the proposed PP-GP, which exhibits lower error rates.

7.1. Experimental setup

Baselines. We benchmark PP-GP with standard GP models that are trained using isolated data (local) and combined plaintext data (Exact). For standard GP, we use the GPyTorch[1] [25] package, which conducts all computations in plaintext. This allows us to compare the performance and accuracy of our privacy-preserving algorithms against the conventional approach. Meanwhile, our privacy-preserving algorithms leverage ciphertext operations from the open-source Crypten[2] [78] framework for privacy-preserving ML.

Implementation. Moreover, we determine the values of the hyperparameters $\sigma_s^2, \sigma_n^2, \ell$ based on cross-validation. We take a single data owner as an example, which is easy to expand to multi-data owners or multi-users. To ensure accuracy and avoid overflow errors in our experimentation, we set the parameters $l = 64$ and $l_f = 24$. Our experiments are conducted on three servers equipped with an NVIDIA Tesla V100 GPU. Additionally, we perform the experiments on a local area network with a communication latency of 0.2 ms and a bandwidth of 10GB/s. These server configurations and network conditions enable us to carry out realistic evaluations of the proposed algorithms under practical settings.

Datasets. We evaluate the proposed models on ten datasets sourced from the UCI dataset repository [91]. These datasets consist of various domains and contain up to 48,827 training examples (the maximum possible before all implementations exhausted GPU memory [25]). These datasets include Diabetes, Airfoil, Skillcraft, Parkinsons, PoleTele, Elevators, Bike, Kin40K, Protein, and KeggDirected. For PP-GP evaluations, we downsample the larger datasets ($n \geq 15,000$) to meet the memory requirements of ciphertext computation.

7.2. Performance of PP-Exp and PP-MI

Firstly, we demonstrate the accuracy and computational efficiency of the proposed operations.

Evaluation of PP-Exp. We evaluate the performance of the proposed PP-Exp by comparing it with these baselines: (1) *Poly*, an SS-based exponential operation method utilizing Taylor expansion polynomials; (2) *Crypten*, an SS-based iterative method utilizing the limit approximation for the exponential function; and (3) *Plaintext*, which directly executes plaintext exponential operations, or in other words, lossless exponential operations.

To evaluate the effectiveness of the PP-Exp algorithm, we consider both its accuracy and efficiency. For accuracy evaluations, we test Poly using various polynomial degrees and Crypten with different iteration counts. The metric $|\hat{e}_u - e_u|$ denotes the discrepancy between $\hat{e}_u$ produced by PP-based exponential operation and e_u derived from plaintext. Fig. 5 shows our PP-Exp achieves a level of accuracy similar to that of plaintext while also outperforming the other tested algorithms. It is also worth noting that, the error rate of PP-Exp remains consistent across varying input (u). This is because PP-Exp is an unbiased operation founded on the principle of confusion-correction, setting it apart from other approximation methods. These methods typically incur both approximation and precision errors. While the approximation error can be reduced by increasing computations, it cannot be eliminated. In contrast, PP-Exp primarily faces precision errors. However, as previously discussed, this comes with a potential minor privacy risk, which is relatively small.

We subsequently evaluate the efficiency of PP-Exp against Poly (utilizing a polynomial degree of 10) and Crypten (employing 8 iterations), with the test conducted on varying input variable sizes, specifically on $e^{\mathbf{U}}$ for different dimensions of $\mathbf{U}$. The computational time for the tested algorithms is illustrated in Fig. 6. Notably, PP-Exp requires considerably less time compared to both the polynomial and iterative methods. When handling large input sizes, PP-Exp proves to be up to 70 times swifter than Poly 10 and as much as 38 times more rapid than Crypten 8.

[1] https://gpytorch.ai
[2] https://crypten.ai/

Table 1
Root Mean Square Error (RMSE) for conventional and privacy-preserving GP models on UCI datasets, employing an RBF kernel with an independent length scale for each dimension. Results were obtained by averaging over 5 trials, each with different splits and seeds.

Dataset	n	d	Local	Exact	PP-GP	Split-GP
Diabetes	442	10	$0.596\ (\pm 4.2\times10^{-2})$	$\mathbf{0.538}\ (\pm 2.0\times10^{-3})$	$\mathbf{0.538}\ (\pm 2.0\times10^{-3})$	$0.585\ (\pm 2.1\times10^{-3})$
Airfoil	1503	5	$0.459\ (\pm 4.7\times10^{-2})$	$\mathbf{0.137}\ (\pm 7.2\times10^{-4})$	$\mathbf{0.137}\ (\pm 7.2\times10^{-4})$	$0.333\ (\pm 2.0\times10^{-3})$
Skillcraft	3338	18	$0.670\ (\pm 2.2\times10^{-2})$	$\mathbf{0.537}\ (\pm 6.1\times10^{-3})$	$\mathbf{0.537}\ (\pm 6.0\times10^{-3})$	$0.582\ (\pm 6.6\times10^{-3})$
Parkinsons	5875	19	$0.515\ (\pm 6.7\times10^{-3})$	$\mathbf{0.290}\ (\pm 2.8\times10^{-4})$	$\mathbf{0.290}\ (\pm 2.8\times10^{-4})$	$0.519\ (\pm 6.1\times10^{-4})$
PoleTele	15,000	26	$0.124\ (\pm 1.2\times10^{-3})$	$\mathbf{0.092}\ (\pm 1.3\times10^{-5})$	$0.110\ (\pm 1.6\times10^{-5})$	$0.143\ (\pm 2.5\times10^{-5})$
Elevators	16,599	16	$0.247\ (\pm 4.9\times10^{-3})$	$\mathbf{0.151}\ (\pm 1.2\times10^{-5})$	$0.163\ (\pm 9.8\times10^{-5})$	$\mathbf{0.151}\ (\pm 1.2\times10^{-5})$
Bike	17,379	17	$0.201\ (\pm 3.7\times10^{-3})$	$\mathbf{0.133}\ (\pm 8.5\times10^{-6})$	$0.202\ (\pm 1.6\times10^{-5})$	$0.172\ (\pm 6.8\times10^{-5})$
Kin40K	40,000	8	$0.101\ (\pm 3.7\times10^{-3})$	$\mathbf{0.066}\ (\pm 5.6\times10^{-5})$	$\mathbf{0.066}\ (\pm 5.6\times10^{-5})$	$0.111\ (\pm 1.4\times10^{-4})$
Protein	45,730	9	$0.711\ (\pm 7.3\times10^{-3})$	$\mathbf{0.617}\ (\pm 4.5\times10^{-4})$	$0.682\ (\pm 3.2\times10^{-3})$	$0.622\ (\pm 1.8\times10^{-5})$
KeggDirected	48,827	20	$0.034\ (\pm 6.3\times10^{-3})$	$0.023\ (\pm 2.5\times10^{-5})$	$0.023\ (\pm 2.5\times10^{-5})$	$\mathbf{0.017}\ (\pm 5.0\times10^{-5})$

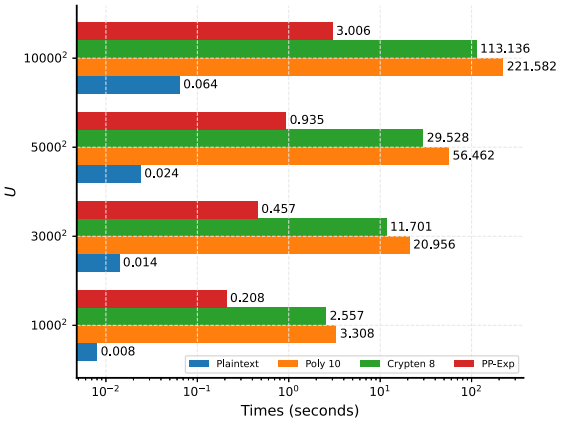


Fig. 6. The runtime of the tested methods for computing $e^{\mathbf{U}}$ based on varying sizes of $\mathbf{U}$.

Evaluation of PP-MI. We evaluate the efficacy of PP-MI using randomly generated covariance matrices, comparing its performance against two baselines: (i) Plaintext-Cholesky, which utilizes Cholesky decomposition for matrix inversion, and (ii) Plaintext-inv, found within the torch.linalg library. Firstly, we randomly sample an input matrix $X \in [-10, 10]^{n\times d}$ with $d = 2$. Using the RBF kernel with parameters $\sigma_s^2 = 1$, $\ell = 1$, and $\sigma_n^2 = 0.1$, we then compute $K + \sigma_n^2 I$. For evaluating the accuracy of a matrix inversion algorithm, we consider the output matrix as $\mathbf{\Lambda}$ and adopt $\|(K+\sigma_n^2 I)\mathbf{\Lambda} - I\|^2$ as the accuracy measure. The inversion accuracy and runtime, averaged over five independent runs and varying n, are presented in Fig. 7.

Observably, PP-MI introduces a tolerable level of accuracy deviation (approximately 0.0001 when $n = 400$) while maintaining a reasonable computational cost. This minor inaccuracy comes from the approximation entailed by SS-based division and the immutable steps of fixed-point encoding, inherent to numerous SS-based algorithms. On the other hand, PP-MI consistently demonstrates that with increasing matrix size, both error and computation time remain stable. This suggests our operator has the potential for large-scale computations.

7.3. Performance of privacy-preserving GPs

Subsequently, we benchmark the privacy-preserving GP models on real-world datasets sourced from the UCI repository. We report results averaged from 5 runs using various random seeds and initialization.

Accuracy. In Table 1, we report test prediction error (root mean square error, RMSE) for both plaintext and privacy-preserving models, including standard GP models that are trained using isolated data (local) and combined plaintext data (Exact). In general, as data volume increases, GP demonstrates improved prediction accuracy. Hence, the need arises for practical collaborative GP models that prioritize privacy across diverse private data sources. Across all small datasets ($n < 10^4$), PP-GP has comparable performance with the model that is trained on the mixed plaintext data. Essentially, this is because the PP-Exp and PP-MI protocols provide sufficient accuracies that are approximately equal to those of plaintext functions. Consequently, the operators we have proposed not only serve as a robust basis for building privacy-preserving GPs but also have potential applicability to a broader range of privacy-preserving ML algorithms.

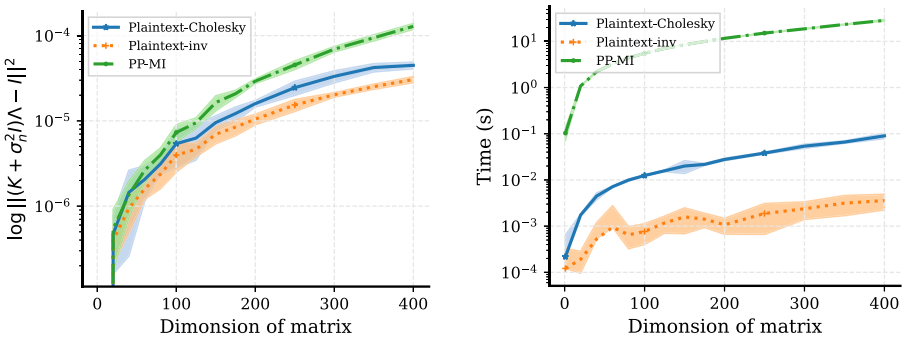
On the other hand, while the predictive accuracy of Split-GP slightly diminishes, the degradation is not pronounced. Especially for large-scale data ($n > 10^4$), the prediction accuracy of Split-GP is about the same compared to exact GP wants. The principal reason for this is the employment of a low-rank approximation for the kernel, which inherently favors computational efficiency at the cost of optimal performance. However, by tuning the number of random features, we can achieve a trade-off between performance and efficiency.

Table 2
Error rate (%) for conventional and privacy-preserving GP models on UCI datasets, employing an RBF kernel with an independent length scale for each dimension.

Dataset	PP-GP	Split-GP
Diabetes	$0.12872\ (\pm 4.1\times10^{-2})$	$\mathbf{0.00071}\ (\pm 1.7\times10^{-6})$
Airfoil	$0.06049\ (\pm 5.3\times10^{-3})$	$\mathbf{0.00356}\ (\pm 5.6\times10^{-5})$
Skillcraft	$11.29721\ (\pm 2.51\times10^{2})$	$\mathbf{0.00007}\ (\pm 6.7\times10^{-7})$
Parkinsons	$0.08547\ (\pm 4.5\times10^{-3})$	$\mathbf{0.00198}\ (\pm 2.4\times10^{-5})$
PoleTele	$0.48856\ (\pm 1.5\times10^{-1})^*$	$\mathbf{0.0093}\ (\pm 1.5\times10^{-4})$
Elevators	$0.80595\ (\pm 3.2\times10^{-1})^*$	$\mathbf{0.3658}\ (\pm 5.3\times10^{-1})$
Bike	$0.00483\ (\pm 8.8\times10^{-6})^*$	$\mathbf{0.0019}\ (\pm 2.9\times10^{-6})$
Kin40K	$0.22336\ (\pm 1.5\times10^{-2})^*$	$\mathbf{0.03355}\ (\pm 2.4\times10^{-3})$
Protein	$242.06341\ (\pm 1.6\times10^{5})^*$	$\mathbf{0.00038}\ (\pm 3.1\times10^{-6})$
KeggDirected	$11.24339\ (\pm 2.2\times10^{2})^*$	$\mathbf{0.00586}\ (\pm 3.2\times10^{-3})$

Error comparison. Let $\mu_{\mathbf{x}|D}$ and $\hat{\mu}_{\mathbf{x}|D}$ represent the predictive mean of the privacy-preserving GPs in plaintext and ciphertext, respectively. The relative difference between the predictive metric is given by: $\frac{1}{|\mathbf{x}|}(|\mu_{\mathbf{x}|D} - \hat{\mu}_{\mathbf{x}|D}|/\mu_{\mathbf{x}|D})$. To assess the performance of PP-GPs across varying data scales, we randomly sampled observations and test data from each dataset, with n indicating the sample size. We averaged results over 5 random runs. The error rate of the predictive results is shown in Table 2. It is evident that PP-GP offers a predictive mean closely matching that of the conventional GP, except for the Protein dataset. The observed discrepancies arise from approximations in some SS-based operations, such as division, as well as the fixed-point encoding step. Notably, the computational errors in Split-GP are marginally lower than in other SS-based GPs. This can be attributed to the kernel's low-rank approximation, which reduces the matrix size, thereby significantly minimizing losses stemming from encoding and SS-based operations.

Speedup. We compare Split-GP with PP-GP in terms of runtime and communication. Table 3 outlines the runtime and communication cost overheads. For each dataset, we have recorded the total time required,



(a) The accuracy of the proposed PP-MI (b) The runtime of the proposed PP-MI

Fig. 7. The performance and runtime of the proposed PP-MI in compared against plaintext-Cholesk and plaintext-inv, with error bars indicating a 95% confidence interval based on 5 runs with different initializations and random seeds.

Table 3
Total computation time and communication cost of conventional and privacy-preserving GP models on UCI datasets using an RBF kernel with independent length scale for each dimension.

Dataset	Runtime				Communication		
	Exact	PP-GP	Split-GP	Speedup	PP-GP	Split-GP	Speedup
Diabetes	0.1 s	30.6 s	6.7 s	5×	3.74 Gb	0.06 Gb	64×
Airfoil	0.2 s	2.1 min	15.3 s	8×	143.45 Gb	0.27 Gb	536×
Skillcraft	0.2 s	5.2 min	13.8 s	22×	1563.97 Gb	0.66 Gb	2379×
Parkinsons	2.4 s	10.3 min	19.1 s	32×	4363.63 Gb	1.49 Gb	2938×
PoleTele	1.5 s	13.6* min	1.2 min	12×	5252.52* Gb	41.30 Gb	128×
Elevators	1.8 s	15.1* min	1.3 min	12×	7111.97* Gb	41.69 Gb	171×
Bike	2.0 s	13.8* min	1.7 min	8×	6359.30* Gb	107.91 Gb	59×

which includes all stages such as pre-computation, model construction, and inference phases. As evident from Table 3, Split-GP outpaces its privacy-preserving counterparts (PP-GP) – an anticipated outcome. The acceleration for Split-GP ranges between 5 to 32×, averaging around 10× for larger datasets. Moreover, the execution speed remains unaffected by escalating sample sizes. This trend could be attributed to the additional time consumed during random feature formulation. Moreover, we assess the communication volume on the server side during computations. Split-GP drastically curtails communication expenses, rendering it apt for real-world communication settings, especially in edge scenarios where communication bandwidth is constricted. After that, we find that matrix inversion is the primary cause of the communication burden and time overhead. Finally, it is important to note that while Split-GP may not quite match plaintext performance, it considerably reduces both communication overhead and running time compared to PP-GP. This reduction notably aligns with the decreased dimensionality in matrix operations.

Evaluation of RF. The random feature approximation enables us to develop a learning framework for privacy-preserving GP models which significantly reduces communication costs and runtime. We have studied the impact of varying numbers of random features on Split-GP, and the results are shown in Fig. 8. It can be seen the performance of Split-GP improves as the dimension of random features increases. Consequently, We can select the appropriate number of random features, and the level of approximation can be tuned based on constraints on runtime or hardware. In most cases, we can set the dimension to $1000 \sim 1500$ for most large-scale data to reduce communication while ensuring performance.

Compare with DP-GP. Differential privacy is a widely recognized approach that aims to preserve privacy in machine learning by adding noise within a privacy budget [66]. This provides privacy but can reduce model accuracy. Smith et al. [20] introduced a privacy-preserving algorithm, yet it only secures the model outputs $\hat{\mathbf{y}}$ by adding noise within this budget. This approach does not fully protect both inputs and outputs of the model. To evaluate the effectiveness of the proposed

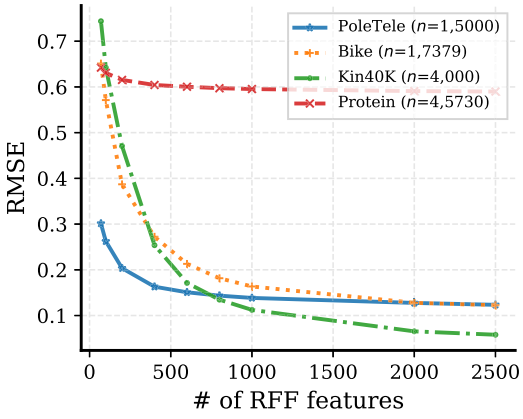


Fig. 8. RMSE as a function of the number of features.

method, we contrast it against the DP-based method. We measure this by comparing the error in the predicted mean of DP-GP across varying levels of DP guarantees, represented by different values of ϵ, using the diabetes dataset. Generally, a higher ϵ indicates a more generous privacy budget. Table 4 shows that even with a substantial privacy budget, specifically $\epsilon = 1.0$, DP-GP suffers from considerable computational errors due to added DP noise, compared to our proposed PP-GP. It is worth mentioning that the relative error rate of our PP-GP consistently stays below 0.01%.

8. Conclusion

In this paper, we proposed PP-GP, a novel and general framework for privacy-preserving GP based on secret sharing. We finish this by introducing two additive SS-based functions (PP-Exp and PP-MI) designed to seamlessly integrate with existing SS operations, ensuring a secure

Table 4
Comparison between our algorithm and the DP-based method in terms of relative error rate(%).

n	Test	DP-GP($\epsilon = 1.0$)	DP-GP($\epsilon = 0.5$)	DP-GP($\epsilon = 0.2$)	PP-GP
80	20	18.6(±23.4)	42.9(±188.3)	96.6(±584.8)	$0.0007(\pm 3.6 \times 10^{-4})$
150	50	27.8(±19.4)	61.2(±311.6)	148.6(±1457.7)	$0.0018(\pm 1.3 \times 10^{-3})$
300	100	18.3(±10.5)	33.5(±23.7)	80.6(±133.3)	$0.0058(\pm 2.5 \times 10^{-3})$

and efficient GP model. Our theoretical analysis demonstrates the security and computational efficiency of our approach. To further boost the efficiency, we introduce Split-GP, a split-learning framework incorporating random features. The resulting framework effectively reduces the computational and communication costs, making privacy-preserving GP feasible for larger datasets. We conduct comprehensive experiments with ten real-world datasets sourced from the UCI dataset repository. Experiments on these datasets reveal the impressive efficiency of Split-GP in terms of speed and communication, while the proposed PP-GP showcases competitive performance compared to standard GP.

CRediT authorship contribution statement

Shiyu Liu: Writing – original draft, Visualization, Validation, Methodology, Investigation, Formal analysis, Conceptualization. **Jinglong Luo:** Validation, Software, Methodology, Investigation, Data curation. **Yehong Zhang:** Writing – review & editing, Supervision, Project administration, Funding acquisition. **Hui Wang:** Supervision, Project administration, Funding acquisition. **Yue Yu:** Supervision, Resources, Project administration, Funding acquisition. **Zenglin Xu:** Supervision, Resources, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was partially supported by the National Key Research and Development Program of China (No. 2022ZD0115301), the National Natural Science Foundation of China (No. 62206139), the National Key Research and Development Program of China (No. 2018AAA0100204), a key program of fundamental research from Shenzhen Science and Technology Innovation Commission (No. JCYJ20200109113403826), and an Open Research Project of Zhejiang Lab (NO. 2022RC0AB04).

Appendix

The appendices are structured as follows. We provide proofs for two of our main results, the Correctness and Security of PP-Exp in Appendices A.1 and A.2, respectively. The proof of Theorem 6.3 is sketched in Appendix A.3.

Appendix A. Proofs

A.1. Proof of Theorem 6.1: Correctness of PP-Exp

Proof of Theorem 6.1. For algorithm correctness, we must address potential overflow or underflow errors in Algorithm 2. The PP-Exp algorithm operates on two algebraic structures: the ring of integers $\mathcal{Z}_L$ mod L and the fixed-point set $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$. Share operations occur on $\mathcal{Z}_L$, and others on $\mathcal{Q}_{<\mathcal{Z}_L, l_f>}$.

Underflow. The risk in PP-Exp arises when calculating $\exp(-\check{r})$ and $\exp(\check{d})$. The lower bound is $\exp(u_{min} - \check{r}_{max})$. For no underflow, this value should be expressible as a fixed-point number with precision l_f, which results in the condition $l_f \geq (\check{r}_{max} - u_{min}) \log_2^{\exp}$.

Overflow. The potential overflow in PP-Exp occurs when computing $[\exp(u)]_j, j \in \{0, 1\}$. The maximum reconstructed value is

$$\begin{aligned} &[\exp(u_{max})]_0 + [\exp(u_{max})]_1 \\ &= \exp(u_{max} + \check{r}) \times 2^{l_f} \times \left([\exp(-\check{r})]_0 + [\exp(-\check{r})]_1\right) \\ &= \exp(u_{max}) \times 2^{2l_f}, \end{aligned} \tag{A.1}$$

which can be seen in Algorithm 2. To prevent overflow during reconstruction, the condition $\exp(u_{max}) \times 2^{2l_f} \leq 2^{l-1}$ must hold. Since $u < 0$, PP-Exp avoids overflow if $l_f < \frac{(l-1)}{2}$. □

A.2. Proof of Theorem 6.2 : Security of PP-Exp

Proof of Theorem 6.2. The proof of Theorem 6.2 is based on the following lemma, which has been widely used in existing privacy-preserving ML works [23,60].

Lemma A.1 (*Feng et al. [92]*). *For any a in Z_L, if b is uniformly distributed in Z_L and independent of a, then $a + b$ is also uniformly distributed in Z_L and independent of a.*

Note that during the execution of the privacy-preserving algorithms, T only generates random numbers that meet the public conditions in the offline phase. Therefore, there is a PPT simulator Sim_T which can randomly generate corresponding random numbers to simulate T according to public information, making the semi-honest adversary $\mathcal{A}$ cannot distinguish between the simulated view and the real view.

Theorem 6.2 holds if and only if the PP-Exp algorithm satisfies the security requirements under the semi-honest model defined in this paper. We need to prove that during the execution of the PP-Exp, the view of each computing server is simulatable except for the probability $\frac{m_r - m_u + 1}{m_r}$ and the amount of privacy leakage is $\frac{m_u + m_r - 1}{m_u \times m_r}$. Specifically, we assume that a computing server is controlled by the adversary $\mathcal{A}$ during the execution of the PP-Exp. Without loss of generality, we assume that S_1 is corrupted by $\mathcal{A}$.

The view of S_1 is $\{view_1^{PP-Exp}(u_0, u_1)\}_{u_0, u_1 \in z_L} = \{[u]_1, [r]_1, [\exp(-\check{r})]_1, d\}$ where $[r]_1$ and $[\exp(-\check{r})]_1$ are random values generated by T. It does not contain any information of the privacy data u. Therefore, there is a PPT simulator Sim_{S_1} which can simulate $\{view_1^{PP-Exp}(u_0, u_1)\}_{u_0, u_1 \in z_L}$ using the input of S_1 and some public information, making the semi-honest adversary $\mathcal{A}$ cannot distinguish between the simulated view and the real view. We continue to analyze the probability that d leaks private data u and the amount of leakage.

The variables u and r can take on m_u and m_r possible values respectively. The chance that the algorithm leaks privacy information about u is

$$\frac{(1 + 2 + \cdots + m_u) \times 2}{m_u m_r} = \frac{m_u \times (m_u - 1)}{m_u m_r} = \frac{m_u - 1}{m_r}. \tag{A.2}$$

Hence, the security probability of PP-Exp is $1 - \frac{m_u - 1}{m_r} = \frac{m_r - m_u + 1}{m_r}$. Due to the nature of the RBF kernel, $u \propto -(x - x')^2$, the potential values of u are restricted on $\mathbb{R}^+$. This further reduces the risk of leaking input u. For instance, when $l_f = 2^{29}$ and u falls within $[-4, 0]$ while r is between $[-16, 16)$, the number of possible values for u is $2^{31}+1$ and for r is $2^{34}+1$. Then, the algorithm's security probability is $\frac{2^{34} - 2^{31}}{2^{34}} = \frac{7}{8}$.

The *expected degree of information leakage* measures the potential privacy vulnerability of PP-Exp. For clarity, consider an example: Given $u \in \{-2,-1,0\}$, $r \in \{-3,-2,-1,0,1,2,3\}$, and $d = u+r$, the probabilities of d are:

$$\begin{aligned} &\mathrm{P}(d=-3) = \mathrm{P}(d=-2) = \mathrm{P}(d=-1) \\ &\qquad = \mathrm{P}(d=0) = \mathrm{P}(d=1) = 3/21, \\ &\mathrm{P}(d=-5) = \mathrm{P}(d=3) = 1/21, \\ &\mathrm{P}(d=-4) = \mathrm{P}(d=2) = 2/21. \end{aligned} \tag{A.3}$$

For $d \in \{-3,-2,-1,0,1\}$, any of the u values in $\{-2,-1,0\}$ can be chosen without revealing extra information. Hence, the leakage degree for u is $5\times\frac{3}{21}\times\frac{1}{3} = \frac{5}{21}$. For $d \in \{-5,3\}$, each u value is distinct, resulting in a leakage degree of $\frac{2}{21}$. For $d \in \{-4,2\}$, there are two potential u values, making the leakage $\frac{2}{21}$. Therefore, the degree of information leakage of u averaged over specific d is $\frac{9}{21}$.

In the PP-Exp, the chances of excluding $m_u - k$ possible u values due to d are determined by the formula $\frac{2k}{m_u m_r}$, with $k = 1,2\ldots,(m_u-1)$. Consequently, the *average degree of information leakage* for u can be calculated as:

$$\begin{aligned} &\left(1-\frac{m_u-1}{m_r}\right)\times\frac{1}{m_u} + \sum_{k=1}^{m_u-1}\left(\frac{2k}{m_u m_r}\right)\times\frac{1}{k} \\ &= \frac{m_u+m_r-1}{m_u\times m_r}. \end{aligned} \tag{A.4}$$

More importantly, the PP-Exp only reveals u's exact value when both its maximum and minimum are in the range where both u and r values overlap. The probability of this occurrence is less than:

$$\frac{2}{mn} = \frac{2}{(2^{31}+1)\times(2^{34}+1)} < \frac{1}{2^{64}}. \tag{A.5}$$

Therefore, there is a PPT simulator Sim_0 that can perfectly simulate the view of S_0 by knowing the inputs and outputs of S_0 except for the probability $\frac{m_r-m_u+1}{m_r}$. In the same way, it can be proved that the view of S_0 is simulatable during the execution of the PP-Exp except for the probability $\frac{m_r-m_u+1}{m_r}$. □

A.3. Proof of Theorem 6.3 : Correctness and security of PP-MI

Proof of Theorem 6.3. Theorem 6.3 holds if and only if the PP-MI algorithm satisfies the correctness and security requirements under the semi-honest model as defined in this paper. Recall that the correctness and security of PP-MI depend on the correctness and security of privacy-preserving multiplication ($\mathcal{F}_{\mathrm{MatMul}}$) and privacy-preserving division ($\mathcal{F}_{\mathrm{Div}}$).

In this subsection, we start by establishing the correctness and security of the $\mathcal{F}_{\mathrm{MatMul}}$ algorithm. To perform the privacy-preserving multiplicative computation of the matrices U and $V \in \mathcal{Z}_L^{n\times n}$, the computation server $S_j, j \in \{0,1\}$ takes $[U]_j$ and $[V]_j$ as inputs and outputs $[UV]_j$ by executing $\mathcal{F}_{\mathrm{MatMul}}$. To compute $[UV]_j$, the assistant server T generates the beaver triples A, B, and C, where A and B are random values in $\mathcal{Z}_L^{n\times n}$, and C is obtained by calculating $A\times B \mod L$. Each party, P_i, starts by determining $[D]_i = [U]_i - [A]_i$ and $[E]_i = [V]_i - [B]_i$. Subsequently, the parties exchange their calculated values $[D]_i$ and $[E]_i$, and using these shared values, they jointly reconstruct D through $\mathrm{Rec}([D]_0,[D]_1)$ and E through $\mathrm{Rec}([E]_0,[E]_1)$. The final computation for the additive share of $U\times V$ is obtained as $[U\times V]_i = -j\times D\times E + [U]_i\times E + D\times[V]_i + [C]_i$.

Hence, the correctness of $\mathcal{F}_{\mathrm{MatMul}}$ can be proven by:

$$\begin{aligned} &[U\times V]_0 + [U\times V]_1 \\ =& -D\times E + ([U]_0+[U]_1)\times E + D\times([V]_0+[V]_1) \\ &\quad + ([C]_0+[C]_1) \\ =& -(U-A)(V-B) + U(V-B) + (U-A)V + AB \\ =& -UV + UB - AV - AB + UV - UB \\ &\quad + UV - AV + AB \\ =& UV \end{aligned} \tag{A.6}$$

Next, we prove the security of $\mathcal{F}_{\mathrm{MatMul}}$. We assume that a computing server is controlled by adversary $\mathcal{A}$ during the execution of $\mathcal{F}_{\mathrm{MatMul}}$. Without loss of generality, let us assume that $\mathcal{A}$ controls S_1. The view of S_1 is denoted as $\{view_1^{\mathcal{F}_{\mathrm{MatMul}}}(U_0,U_1,V_0,V_1)\}_{U_0,U_1,V_0,V_1\in\mathcal{Z}_L^{n\times n}} = \{[U]_1,[V]_1,[A]_1,[B]_1,[C]_1,D,E\}$, where $\{[A]_1,[B]_1,[C]_1\}$ are random values generated by T. According to Lemma A.1, $D = U - A$ and $E = V - B$ are also random values in $\mathcal{Z}_L^{n\times n}$. Therefore, no information about the input data U and V is leaked during the execution of $\mathcal{F}_{\mathrm{MatMul}}$.

Additionally, Crypten converts division calculations into multiplication calculations using Newton's iterative method. Thus, $\mathcal{F}_{\mathrm{Div}}$ is implemented by invoking $\mathcal{F}_{\mathrm{MatMul}}$. According to Canetti [82], the correctness and security of $\mathcal{F}_{\mathrm{Div}}$ can be proven.

Summarizing the above discussion completes the proof of Theorem 6.3. □

References

[1] C.E. Rasmussen, C.K. Williams, et al., Gaussian Processes for Machine Learning, vol. 1, Springer, 2006.

[2] F. Yan, Z. Xu, et al., Sparse matrix-variate Gaussian process blockmodels for network modeling, 2012, arXiv preprint arXiv:1202.3769.

[3] Z. Xu, F. Yan, Y. Qi, Bayesian nonparametric models for multiway data analysis, IEEE Trans. Pattern Anal. Mach. Intell. 37 (2) (2013) 475–487.

[4] Y. Zhang, T.N. Hoang, K.H. Low, M. Kankanhalli, Near-optimal active learning of multi-output Gaussian processes, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30, 2016.

[5] P. Schulam, S. Saria, What-if reasoning with counterfactual Gaussian processes, 2017, arXiv preprint arXiv:1703.10651.

[6] L. Ortmann, D. Shi, E. Dassau, F.J. Doyle, B.J. Misgeld, S. Leonhardt, Automated insulin delivery for type 1 diabetes mellitus patients using Gaussian process-based model predictive control, in: 2019 American Control Conference, ACC, IEEE, 2019, pp. 4118–4123.

[7] R. Shashikant, U. Chaskar, L. Phadke, C. Patil, Gaussian process-based kernel as a diagnostic model for prediction of type 2 diabetes mellitus risk using non-linear heart rate variability features, Biomed. Eng. Lett. 11 (3) (2021) 273–286.

[8] M.P. Deisenroth, D. Fox, C.E. Rasmussen, Gaussian processes for data-efficient learning in robotics and control, IEEE Trans. Pattern Anal. Mach. Intell. 37 (02) (2015) 408–423.

[9] S.Y. Yang, Q. Qiao, P.A. Beling, W.T. Scherer, A.A. Kirilenko, Gaussian process-based algorithmic trading strategy identification, Quant. Finance 15 (10) (2015) 1683–1703.

[10] H. Liu, Y.-S. Ong, X. Shen, J. Cai, When Gaussian process meets big data: A review of scalable GPs, IEEE Trans. Neural Netw. Learn. Syst. 31 (11) (2020) 4405–4423.

[11] Y. Zhao, T. Chitnis, B.C. Healy, J.G. Dy, C.E. Brodley, Domain induced Dirichlet mixture of gaussian processes: An application to predicting disease progression in multiple sclerosis patients, in: 2015 IEEE International Conference on Data Mining, IEEE, 2015, pp. 1129–1134.

[12] K. Peterson, O. Rudovic, R. Guerrero, R.W. Picard, Personalized gaussian processes for future prediction of alzheimer's disease progression, 2017, arXiv preprint arXiv:1712.00181.

[13] M. Ogburn, C. Turner, P. Dahal, Homomorphic encryption, Procedia Comput. Sci. 20 (2013) 502–509.

[14] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, Y. Gao, A survey on federated learning, Knowl.-Based Syst. 216 (2021) 106775.

[15] C. Dwork, Differential privacy, in: International Colloquium on Automata, Languages, and Programming, Springer, 2006, pp. 1–12.

[16] P. Fenner, E. Pyzer-Knapp, Privacy-preserving gaussian process regression–a modular approach to the application of homomorphic encryption, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2020, pp. 3866–3873.

[17] Z. Dai, B.K.H. Low, P. Jaillet, Federated Bayesian optimization via Thompson sampling, in: Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 9687–9699.

[18] Z. Dai, B.K.H. Low, P. Jaillet, Differentially private federated Bayesian optimization with distributed exploration, in: Advances in Neural Information Processing Systems, vol. 34, 2021, pp. 9125–9139.

[19] G.P. Kontoudis, D.J. Stilwell, Fully decentralized, scalable Gaussian processes for multi-agent federated learning, 2022, arXiv preprint arXiv:2203.02865.

[20] M.T. Smith, M.A. Álvarez, M. Zwiessele, N.D. Lawrence, Differentially private regression with Gaussian processes, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2018, pp. 1195–1203.

[21] D. Kharkovskii, Z. Dai, B.K.H. Low, Private outsourced Bayesian optimization, in: International Conference on Machine Learning, PMLR, 2020, pp. 5231–5242.

[22] J. Luo, Y. Zhang, J. Zhang, S. Qin, H. Wang, Y. Yu, Z. Xu, Practical privacy-preserving Gaussian process regression via secret sharing, in: Uncertainty in Artificial Intelligence, PMLR, 2023.

[23] P. Mohassel, Y. Zhang, Secureml: A system for scalable privacy-preserving machine learning, in: 2017 IEEE Symposium on Security and Privacy, SP, IEEE, 2017, pp. 19–38.

[24] A. Beimel, Secret-sharing schemes: A survey, in: International Conference on Coding and Cryptology, Springer, 2011, pp. 11–46.

[25] J. Gardner, G. Pleiss, K.Q. Weinberger, D. Bindel, A.G. Wilson, Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration, in: Advances in Neural Information Processing Systems, vol. 31, 2018.

[26] A. Rahimi, B. Recht, Random features for large-scale kernel machines, in: Advances in Neural Information Processing Systems, vol. 20, 2007.

[27] D. Jang, J. Yoo, C.Y. Son, D. Kim, H.J. Kim, Multi-robot active sensing and environmental model learning with distributed Gaussian process, IEEE Robot. Autom. Lett. 5 (4) (2020) 5905–5912.

[28] J. Song, Y. Chen, Y. Yue, A general framework for multi-fidelity bayesian optimization with gaussian processes, in: The 22nd International Conference on Artificial Intelligence and Statistics, PMLR, 2019, pp. 3158–3167.

[29] J. Snoek, H. Larochelle, R.P. Adams, Practical bayesian optimization of machine learning algorithms, in: Advances in Neural Information Processing Systems, vol. 25, 2012.

[30] S. Malladi, A. Wettig, D. Yu, D. Chen, S. Arora, A kernel-based view of language model fine-tuning, in: International Conference on Machine Learning, PMLR, 2023, pp. 23610–23641.

[31] G. Yang, Wide feedforward or recurrent neural networks of any architecture are gaussian processes, Adv. Neural Inf. Process. Syst. 32 (2019).

[32] M.E.E. Khan, A. Immer, E. Abedi, M. Korzepa, Approximate inference turns deep networks into gaussian processes, Adv. Neural Inf. Process. Syst. 32 (2019).

[33] J.T. Wilson, V. Borovitskiy, A. Terenin, P. Mostowsky, M.P. Deisenroth, Pathwise conditioning of Gaussian processes, J. Mach. Learn. Res. 22 (1) (2021) 4741–4787.

[34] M. Xu, W. Ding, J. Zhu, Z. Liu, B. Chen, D. Zhao, Task-agnostic online reinforcement learning with an infinite mixture of gaussian processes, Adv. Neural Inf. Process. Syst. 33 (2020) 6429–6440.

[35] C. Toth, H. Oberhauser, Bayesian learning from sequential data using gaussian processes with signature covariances, in: International Conference on Machine Learning, PMLR, 2020, pp. 9548–9560.

[36] S. Roberts, M. Osborne, M. Ebden, S. Reece, N. Gibson, S. Aigrain, Gaussian processes for time-series modelling, Phil. Trans. R. Soc. A 371 (1984) (2013) 20110550.

[37] C. Rasmussen, Z. Ghahramani, Occam's razor, Adv. Neural Inf. Process. Syst. 13 (2000).

[38] A. Wilson, H. Nickisch, Kernel interpolation for scalable structured Gaussian processes (KISS-GP), in: International Conference on Machine Learning, PMLR, 2015, pp. 1775–1784.

[39] A. Wilson, R. Adams, Gaussian process kernels for pattern discovery and extrapolation, in: International Conference on Machine Learning, PMLR, 2013, pp. 1067–1075.

[40] I. Ustyuzhaninov, I. Kazlauskaite, M. Kaiser, E. Bodin, N. Campbell, C.H. Ek, Compositional uncertainty in deep Gaussian processes, in: Conference on Uncertainty in Artificial Intelligence, PMLR, 2020, pp. 480–489.

[41] M.M. Dunlop, M.A. Girolami, A.M. Stuart, A.L. Teckentrup, How deep are deep Gaussian processes? J. Mach. Learn. Res. 19 (54) (2018) 1–46.

[42] H. Salimbeni, M. Deisenroth, Doubly stochastic variational inference for deep Gaussian processes, in: Advances in Neural Information Processing Systems, vol. 30, 2017.

[43] A.G. Wilson, Z. Hu, R.R. Salakhutdinov, E.P. Xing, Stochastic variational deep kernel learning, in: Advances in Neural Information Processing Systems, vol. 29, 2016.

[44] A. Damianou, N.D. Lawrence, Deep gaussian processes, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2013, pp. 207–215.

[45] A.G. Wilson, D.A. Knowles, Z. Ghahramani, Gaussian process regression networks, 2011, arXiv preprint arXiv:1110.4411.

[46] M. Deisenroth, J.W. Ng, Distributed gaussian processes, in: International Conference on Machine Learning, PMLR, 2015, pp. 1481–1490.

[47] M. Lázaro-Gredilla, J. Quinonero-Candela, C.E. Rasmussen, A.R. Figueiras-Vidal, Sparse spectrum Gaussian process regression, J. Mach. Learn. Res. 11 (2010) 1865–1881.

[48] Y. Gal, R. Turner, Improving the Gaussian process sparse spectrum approximation by representing uncertainty in frequency inputs, in: International Conference on Machine Learning, PMLR, 2015, pp. 655–664.

[49] K. Cutajar, E.V. Bonilla, P. Michiardi, M. Filippone, Random feature expansions for deep Gaussian processes, in: International Conference on Machine Learning, PMLR, 2017, pp. 884–893.

[50] A. Potapczynski, L. Wu, D. Biderman, G. Pleiss, J.P. Cunningham, Bias-free scalable Gaussian processes via randomized truncations, in: International Conference on Machine Learning, PMLR, 2021, pp. 8609–8619.

[51] D.-T. Nguyen, M. Filippone, P. Michiardi, Exact gaussian process regression with distributed computations, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 1286–1295.

[52] K. Wang, G. Pleiss, J. Gardner, S. Tyree, K.Q. Weinberger, A.G. Wilson, Exact Gaussian processes on a million data points, Adv. Neural Inf. Process. Syst. 32 (2019).

[53] D. Kaur, S. Uslu, K.J. Rittichier, A. Durresi, Trustworthy artificial intelligence: a review, ACM Comput. Surv. 55 (2) (2022) 1–38.

[54] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, D. Evans, Privacy-preserving distributed linear regression on high-dimensional data, 2016, Cryptology ePrint Archive.

[55] R. Hall, S.E. Fienberg, Y. Nardi, Secure multiple linear regression based on homomorphic encryption, J. Official Statist. 27 (4) (2011) 669–691.

[56] M. Kim, Y. Song, S. Wang, Y. Xia, X. Jiang, et al., Secure logistic regression based on homomorphic encryption: Design and evaluation, JMIR Med. Inform. 6 (2) (2018) e8805.

[57] C. Chen, J. Zhou, L. Wang, X. Wu, W. Fang, J. Tan, L. Wang, A.X. Liu, H. Wang, C. Hong, When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2021, pp. 2652–2662.

[58] W. Fang, D. Zhao, J. Tan, C. Chen, C. Yu, L. Wang, L. Wang, J. Zhou, B. Zhang, Large-scale secure XGB for vertical federated learning, in: Proceedings of the 30th ACM International Conference on Information and Knowledge Management, 2021, pp. 443–452.

[59] S. Wagh, D. Gupta, N. Chandran, Securenn: Efficient and private neural network training, 2018, Cryptology ePrint Archive.

[60] S. Wagh, D. Gupta, N. Chandran, SecureNN: 3-party secure computation for neural network training, Proc. Priv. Enhancing Technol. 2019 (3) (2019) 26–49.

[61] L. Zheng, C. Chen, Y. Liu, B. Wu, X. Wu, L. Wang, L. Wang, J. Zhou, S. Yang, Industrial scale privacy preserving deep neural network, 2020, arXiv preprint arXiv:2003.05198.

[62] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: Concept and applications, ACM Trans. Intell. Syst. Technol. 10 (2) (2019) 1–19.

[63] C. Gentry, Fully homomorphic encryption using ideal lattices, in: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, 2009, pp. 169–178.

[64] M. Abadi, A. Chu, I. Goodfellow, H.B. McMahan, I. Mironov, K. Talwar, L. Zhang, Deep learning with differential privacy, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 308–318.

[65] J. Konečnỳ, H.B. McMahan, F.X. Yu, P. Richtárik, A.T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, 2016, arXiv preprint arXiv:1610.05492.

[66] C. Dwork, A. Roth, et al., The algorithmic foundations of differential privacy, Found. Trends Theor. Comput. Sci. 9 (3–4) (2014) 211–407.

[67] X. Yue, R.A. Kontar, Federated Gaussian process: Convergence, automatic personalization and multi-fidelity modeling, 2021, arXiv preprint arXiv:2111.14008.

[68] L. Zhu, Z. Liu, S. Han, Deep leakage from gradients, in: Advances in Neural Information Processing Systems, vol. 32, 2019.

[69] B. Zhao, K.R. Mopuri, H. Bilen, IDLG: Improved deep leakage from gradients, 2020, arXiv preprint arXiv:2001.02610.

[70] O. Goldreich, Secure multi-party computation, Manuscr. Prelim. Version 78 (110) (1998).

[71] R. Xu, N. Baracaldo, J. Joshi, Privacy-preserving machine learning: Methods, challenges and directions, 2021, arXiv preprint arXiv:2108.04417.

[72] T. Ryffel, P. Tholoniat, D. Pointcheval, F. Bach, Ariann: Low-interaction privacy-preserving deep learning via function secret sharing, 2020, arXiv preprint arXiv:2006.04593.

[73] H. Chaudhari, R. Rachuri, A. Suresh, Trident: Efficient 4pc framework for privacy preserving machine learning, 2019, arXiv preprint arXiv:1912.02631.

[74] P. Mohassel, P. Rindal, ABY3: A mixed protocol framework for machine learning, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 35–52.

[75] J. Liu, M. Juuti, Y. Lu, N. Asokan, Oblivious neural network predictions via minionn transformations, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 619–631.

[76] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613.

[77] G.R. Blakley, Safeguarding cryptographic keys, in: International Workshop on Managing Requirements Knowledge, 1979.

[78] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, L. van der Maaten, Crypten: Secure multi-party computation meets machine learning, in: Advances in Neural Information Processing Systems, vol. 34, 2021, pp. 4961–4973.

[79] M. Aliasgari, M. Blanton, Y. Zhang, A. Steele, Secure computation on floating point numbers, 2012, Cryptology ePrint Archive.

[80] S. Bochner, et al., Lectures on Fourier integrals, vol. 42, Princeton University Press, 1959.

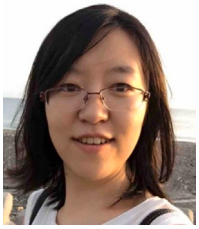
[81] F. Liu, X. Huang, Y. Chen, J.A. Suykens, Random features for kernel approximation: A survey on algorithms, theory, and beyond, IEEE Trans. Pattern Anal. Mach. Intell. 44 (10) (2021) 7128–7148.

[82] R. Canetti, Universally composable security: A new paradigm for cryptographic protocols, in: Proceedings 42nd IEEE Symposium on Foundations of Computer Science, IEEE, 2001, pp. 136–145.

[83] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, R. Sharma, Cryptflow2: Practical 2-party secure inference, in: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 325–342.
[84] D. Rathee, M. Rathee, R.K.K. Goli, D. Gupta, R. Sharma, N. Chandran, A. Rastogi, Sirnn: A math library for secure rnn inference, in: 2021 IEEE Symposium on Security and Privacy, SP, 2021, pp. 1003–1020.
[85] Z. Huang, W.-j. Lu, C. Hong, J. Ding, Cheetah: Lean and fast secure {two-party} deep neural network inference, in: 31st USENIX Security Symposium, USENIX Security 22, 2022, pp. 809–826.
[86] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, T. Zhang, Iron: Private inference on transformers, Adv. Neural Inf. Process. Syst. 35 (2022) 15718–15731.
[87] Y. Lindell, How to simulate it – A tutorial on the simulation proof technique, in: Tutorials on the Foundations of Cryptography, 2017, pp. 277–346.
[88] R. Canetti, Security and composition of multiparty cryptographic protocols, J. Cryptol. 13 (2000) 143–202.
[89] O. Goldreich, Foundations of Cryptography, Volume 2, Cambridge university press Cambridge, 2004.
[90] W.Z. Srinivasan, P. Akshayaram, P.R. Ada, DELPHI: A cryptographic inference service for neural networks, in: Proc. 29th USENIX Secur. Symp, 2019, pp. 2505–2522.
[91] A. Asuncion, D. Newman, UCI Machine Learning Repository, Irvine, CA, USA, 2007, https://archive.ics.uci.edu/.
[92] Q. Feng, D. He, Z. Liu, H. Wang, K.-K.R. Choo, SecureNLP: A system for multi-party privacy-preserving natural language processing, IEEE Trans. Inf. Forensics Secur. 15 (2020) 3709–3721.

Shiyu Liu is a Ph.D. Candidate at the University of Electronic Science and Technology of China, he received a B.S. degree from Central South University in 2016. His research interests include Bayesian inference, probabilistic machine learning, and federated learning.

Jinglong Luo is a joint Ph.D. student at Harbin Institute of Technology (Shenzhen) and Pengcheng Lab. His research interests include privacy-preserving machine learning, trustworthy federated learning, and secure multi-party computation.



Yehong Zhang is a researcher at Peng Cheng Lab. She received a doctoral degree in computer science from the National University of Singapore, and a B.S. degree from the Harbin Institute of Technology. Her research interests include Bayesian optimization, probabilistic machine learning, and multi-task learning. She has published papers in international journals and conference proceedings.

Hui Wang received an M.Sc. degree in information systems engineering from the National University of Defense Technology, Changsha, Hunan, China, in 1998, and a Ph.D. degree in systems engineering from the National University of Defense Technology in 2005. He is currently a researcher at Peng Cheng Laboratory, Shenzhen, Guangdong, China. His main research interests include distributed machine learning, federated learning, computing power networks, NLP, and application. He has published more than 80 publications including three books.

Yue Yu (Member, IEEE) is a researcher with Peng Cheng Lab, an associate professor with the College of Computer, National University of Defense Technology, and a technical committee member of the OpenI community. His research findings have been published on IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology, CHI, CSCW, ICSE, FSE, ACL, etc. His current research interests include software engineering and artificial Intelligence.

Zenglin Xu (Senior Member, IEEE) received a Ph.D. degree in computer science and engineering from the Chinese University of Hong Kong. He is currently a full professor at the Harbin Institute of Technology, Shenzhen. He has been working with Michigan State University, Cluster of Excellence with Saarland University and Max Planck Institute for Informatics, and later Purdue University. His research interests include machine learning and its applications in information retrieval, health informatics, and social network analysis.