AITIA: Efficient Secure Computation of Bivariate Causal Discovery

Truong Son Nguyen
Arizona State University
snguye63@asu.edu

Lun Wang
UC Berkeley
wanglun@berkeley.edu

Evgenios M. Kornaropoulos
George Mason University
evgenios@gmu.edu

Ni Trieu
Arizona State University
nitrieu@asu.edu

ABSTRACT

Researchers across various fields seek to understand causal relationships but often find controlled experiments impractical. To address this, statistical tools for causal discovery from naturally observed data have become crucial. Non-linear regression models, such as Gaussian process regression, are commonly used in causal inference but have limitations due to high costs when adapted for secure computation. Support vector regression (SVR) offers an alternative but remains costly in an Multi-party computation context due to conditional branches and support vector updates.

In this paper, we propose AITIA, the first two-party secure computation protocol for bivariate causal discovery. The protocol is based on optimized multi-party computation design choices and is secure in the semi-honest setting. At the core of our approach is BSGD-SVR, a new non-linear regression algorithm designed for MPC applications, achieving both high accuracy and low computation and communication costs. Specifically, we reduce the training complexity of the non-linear regression model from approximately from $O(N^3)$ to $O(N^2)$ where N is the number of training samples. We implement AITIA using `CrypTen` and assess its performance across various datasets. Empirical evaluations show a significant speedup of 3.6× to 340× compared to the baseline approach.

1 INTRODUCTION

Many researches in social, medical and natural sciences aim to answer questions with the following format: "What is the *cause* of X?" or "What is the *effect* of X?" Although some of these questions can be answered by controlled experiments, these trials are usually expensive or even impossible to conduct. To address this challenge, people turn to developing statistical tools to discover causal relationships between variables from naturally observed data. This category of tools is referred to as *causal inference*.

Causal Inference is intrinsically data-hungry. The more data is fed to the algorithm, the more accurate the result will be. As a result, causal inference usually requires rich datasets collected from different parties. However, this raises privacy concerns. For example, suppose that two hospitals want to collaboratively run a causal inference algorithm on their combined dataset so as to determine if a new medicine is effective on diabetes. However, due to HIPAA regulations, they are reluctant to share medical records with one another.

Multi-party computation (MPC) is a promising approach to addressing the above privacy concern. Generally, MPC allows multiple parties to jointly compute a function on shared inputs without revealing more information than the function output. However, general-purpose MPC protocols, such as garbled circuit [50, 51] or GMW [13], typically introduce large performance overhead due to extra computation and communication. Therefore, one would like to develop *a customized and efficient protocol for privacy-preserving causal inference* between the combined dataset of multiple parties.

Non-linear Regression Approaches and Their Limitations. The core component of bivariate causal inference is a non-linear regression algorithm. In the community of causal inference, one of the most widely used non-linear regression is *Gaussian process regression* [35]. However, as pointed out in [35], Gaussian process regression suffers from slow training time and the disadvantage is further exacerbated as the number of training samples increases. Besides, Gaussian process regression is an one-off regression algorithm. Its training and testing happen at the same time and once there is a small change in the dataset the whole process needs to be re-run, which makes it less preferred in a dynamic setting where data changes rapidly. To address the issue, we turn to another standard non-linear regression model namely *support vector regression* (*abbrev.* SVR). There are many variants of SVR and in this paper we mainly focus on ϵ-SVR [7]. Although SVR partially overcome the defects of Gaussian process regression, it contains many if-branches which is costly (i.e., not friendly) in the MPC setting. Besides, the model of SVR stores some training examples for future prediction, known as *support vectors*. For obliviousness in the MPC setting, if trivially adapted, the SVR model needs to contain all the training samples, which leads to slow training and huge memory consumption.

Our Proposed MPC-friendly Approach. To overcome the above challenges, we propose a new training algorithm for SVR based on the well-known *stochastic gradient descent (SGD)*. There have been systematic efforts to apply SGD on support vector machine (SVM) [44, 55], a close relative of SVR but there are no such efforts for SVR. Hence, we propose the first SGD-based training for SVR, following the same design pattern of P-packSVM [55], a SGD-based SVM training algorithm. To suppress the number of support vectors, we adopt the idea of *budgeting* [48] to impose an upper bound on the number of support vectors. Putting it all together, our new training algorithm employs a *Budgeted Stochastic Gradient Descent* approach for *Support Vector Regression* (BSGD-SVR).

From SVR to Secure Protocol. Although our BSGD-SVR algorithm is designed to be MPC-friendly, it is far from trivial to adapt the algorithm to the privacy-preserving context. The main reason is that the BSGD-SVR contains many conditional branching operators while modifying the training model. The secure version of the BSGD-SVR

Work presented at CCS'24.

algorithm should not reveal which branch was evaluated. Thus, it is inefficient to directly apply generic secure computation techniques (such as garbled circuit [14, 52]) to the non-secure algorithm without customized optimizations. To build a secure and efficient version of BSGD-SVR that we call Aitia[1], we propose a series of optimizations that: speed up the initialization process, make the budgeting technique oblivious, eliminate conditional branches, and vectorize the algorithm.

Our Contribution. In summary, we make the following contributions in this paper.

- We propose BSGD-SVR, an MPC-friendly non-linear regression model and design an efficient secure protocol for it. Compared to the baselines, BSGD-SVR has lower computation and communication complexity.
- We propose Aitia, the first secure bivariate causal inference protocol, designed for the semi-honest setting, with a straightforward extension to the malicious setting.
- We implement Aitia in the Crypten framework [9] and evaluate it empirically. The results show that Aitia achieves a $3.6 - 340\times$ speedup compared to the baseline. Our implementation can be found at https://github.com/asu-crypto/Aitia

2 BACKGROUND AND RELATED WORK

In this section, we discuss the concepts and relevant literature from causal discovery and cryptography used in our protocol Aitia.

2.1 Causal Inference

Let X, Y be two random variables, and $\Pr(X, Y)$ be the joint probability distribution, i.e., the observational distribution after measuring both quantities *without any intervention*. In a causality study, e.g., drug effects, the designer applies an external intervention that forces variable X to take value x; this action is denoted as "$\text{do}(x)$". If this intervention has an effect, it is reflected in the *interventional distribution*, i.e., $\Pr(Y|\text{do}(x))$. Variable X *causes* Y, denoted as $X \to Y$, holds when $\Pr(Y|\text{do}(x)) \neq \Pr(Y|\text{do}(x'))$ for $x \neq x'$.

Even though there are several possible outcomes when analyzing the causal relation between X and Y, in this work, we focus on the following well-studied case: we assume that (1) X and Y are dependent and that (2) there is no selection bias, no confounding, no feedback relation between them. Under these assumptions, the study of bivariate causal discovery reduces to deciding whether $X \to Y$ or $Y \to X$, i.e., the direction of the causal relation.

Bivariate Causal Inference. In this work, we are interested in the setting where the direction of causality must be inferred *purely from observational data*. That is, we assume that the data analyst does not have access to intervention data and does not have the resources to run a new intervention to test a hypothesis, which, in the medical field, has high costs in recruiting new participants, designing the experiment, and getting IRB approval. Our goal is to focus on the most practical and realistic scenario for causal discovery, which is a *data-driven approach*. Due to its practicality, this approach has attracted a lot of attention in the machine learning community [11, 17, 18, 23, 28, 33, 38, 45, 57].

[1]pronounced `e-tee'-a`: the Greek word for *cause* or *reason*.

Unfortunately, it is not always possible to decide the direction of the causality from observational data. On a high level, there has to be a certain asymmetry between the two variables so that the causality can be inferred purely through observational data. A series of works analyzed potential relations between the variables X, Y that permit such a causal discovery. Hoyer *et al.* [17] proved that it is possible to discover causality when the relation between X, Y is non-linear as long as the latent causes of the system can be modeled as an *additive* noise, i.e., the Additive Noise Model. A rigorous definition of the relation among the random variables is presented in Definition 2.1, and an illustration of the relation between the conditional and joint distributions is depicted in Figure 1.

Definition 2.1. We define as *Additive Noise Model* (ANM) with causal relation $X \to Y$ the model in which (*i*) the r.v. X follows the density p_X, (*ii*) the noise is captured by r.v. N and follows the density p_N, (*iii*) X and N are independent, and (*iv*) there is a (potentially non-linear) f_Y such that $Y = f_Y(X) + N$.

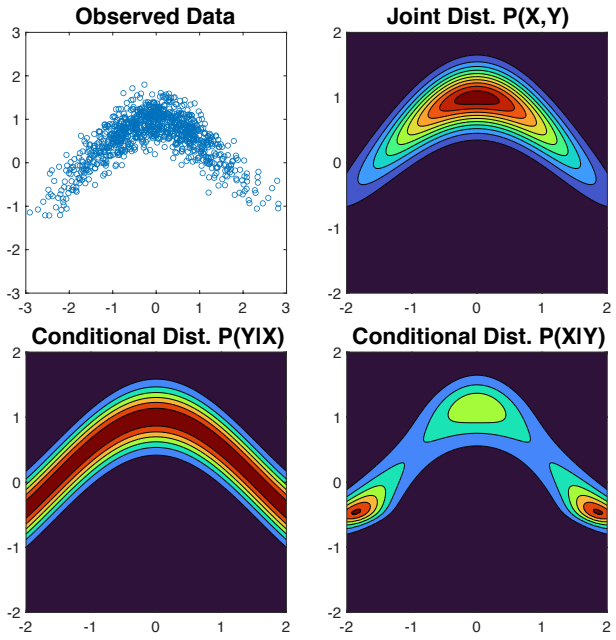


Figure 1: Illustration where causality $X \to Y$ in the ANM is identifiable. X-axis (resp. Y-axis) represents the domain of the random variable. We have $Y = \cos(X) + Z$ where $X \sim N(0, 1)$ and the noise is $Z \sim N(0, 0.3^2)$. The upper-left subplot shows that data sampled from the joint distribution. The rest show the contour lines for the joint and conditional distributions.

Specifically, Figure 1 illustrates why the direction of causality $X \to Y$ is identifiable in ANM. Interestingly, when $X \to Y$ in ANM, the mean of the conditional distribution $\Pr(Y|X)$ depends on X, i.e., the contour lines of $\Pr(Y|X)$ shift as X changes (lower-left subplot). Whereas the conditional distribution $\Pr(X|Y)$ depends on Y in a less obvious way. This asymmetry enables us to discover causality.

In a realistic scenario we only have access to a *sample* from the joint distribution, i.e., the observational data. Mooji *et al.* [33] proposed an approach (see Algorithm 1) uses the insights from Figure 1 to discover causality using *only observational data*. The

observational data is split into two datasets D_1 and D_2. Dataset D_1 is used to estimate the two regression functions $\hat{f} : x \to E(Y|X = x)$ and $\hat{g} : y \to E(X|Y = y)$. After finalizing the regression functions $\hat{f}$ and $\hat{g}$, the dataset D_2 is used to calculate the *dependence score* between the residual and the corresponding input variable. If the dependence score of pair $(Y - \hat{f}_Y(X))$ and X is higher than the score of $(X - \hat{f}_X(Y))$ and Y then the direction of causality is $Y \to X$, otherwise the direction is $X \to Y$.

Algorithm 1: Causal Discovery in ANM.

Input: Observational data D_1: $\{x_i, y_i\}_{i=1}^n$, D_2: $\{x'_i, y'_i\}_{i=1}^m$, chosen of regression model, chosen dependence score $s(\cdot, \cdot)$.

1 Fit non-linear regression models $\hat{f}, \hat{g}$ such that $\hat{f}(\mathbf{x}) \approx \mathbf{y}$ and $\hat{g}(\mathbf{y}) \approx \mathbf{x}$, where $(x, y) \in D_1$

2 Define the dependence scores $s_{X\to Y} = s(\mathbf{x'}, \mathbf{y'} - \hat{f}(\mathbf{x'}))$, and $s_{Y\to X} = s(\mathbf{y'}, \mathbf{x'} - \hat{g}(\mathbf{y'}))$, where $(x', y') \in D_2$

3 **if** $s_{X\to Y} < s_{Y\to X}$ **then return** $X \to Y$

4 **else return** $Y \to X$

Jumping ahead, the superior performance of our approach comes from co-designing (*i*) the machine learning models, i.e., the training of the regression model, and the dependence score, as well as (*ii*) the cryptographic protocols, so that the overall protocol is orders of magnitude faster than a direct implementation of the state-of-the-art causal discovery algorithm [33] using a standard and efficient secure computation library such as CrypTen [9].

Causality Dataset. We evaluate our new MPC-friendly training method as well as the proposed secure protocol on dataset `CauseEffectPairs` (`CEP`), which is a standard dataset [35] for causal discovery. `CEP` contains pairs of random variables that are statistically dependent, where one variable is known to cause the other. The `CEP` collection, version 1.0, consists of pairs from 37 different datasets across various domains and is available at [34]. For more details about the chosen datasets, see Appendix D.

2.2 Secure Computation

Secret Sharing. Our Aitia construction makes usage of secret sharing schemes for computing on private data. Values of the computation are split into two randomly looking values that are held by a two non-colluding servers. To additively share an ℓ-bit value x, the data owner chooses two random values $x_1, x_2 \leftarrow \{0,1\}^\ell$ such that $x_1 + x_2 = x \mod \{0,1\}^\ell$. For simplicity, we omit the subscript of the share and the mod operation and denote the share by $[\![.]\!]$. To reconstruct shared value $[\![x]\!]$, one party sends its share to the other, who reconstructs the secret $x = x_1 + x_2$ locally.

Addition, subtraction, and multiplication-by-constant can be directly applied to the shares as they can be done locally by the parties without communication, for example, $[\![x + y]\!] = [\![x]\!] + [\![y]\!]$. For secure multiplication between two ℓ-bit values, we use the Beaver triple [3] approach. The main idea of Beaver triple is to shift most of the communication and computation cost into a preprocessing phase which can be done offline since it does not require knowledge of the inputs. The offline phase outputs the secret shared values $([\![a]\!], [\![b]\!], [\![c]\!])$ such that $c = ab$. In the online phase, parties compute locally the values $[\![\alpha]\!] = [\![x]\!] - [\![a]\!]$ and $[\![\beta]\!] = [\![y]\!] - [\![b]\!]$, where x and y indicate the sensitive inputs. As a next step the two parties jointly reconstruct α and β by exchanging the shares $[\![\alpha]\!], [\![\beta]\!]$. The secret shared product $[\![xy]\!]$ is equal to $[\![c]\!] + \alpha[\![b]\!] + \beta[\![a]\!] + \alpha\beta$, which can be locally evaluated by each party. Boolean sharing can be seen as additive sharing in the field $\mathbb{Z}_2$. The addition operation is replaced by XOR, and multiplication is replaced by AND.

Garbled Circuits. Garbled Circuits (GC) [14, 52] is currently the most common generic technique for practical two-party secure computation. The ideal functionality of GC is to take the parties' inputs x and y, respectively, and compute f on them without revealing the secret parties' inputs. In our design for Aitia, we use "less than" and "equal" GC where inputs are secretly shared amongst two parties (i.e., each party holds the shares $[\![x]\!]$ and $[\![y]\!]$). We denote this garbled circuit by $[\![z]\!] \leftarrow \mathcal{GC}([\![x]\!], [\![y]\!], f)$. To evaluate a function f on *shared values*, GC first reconstructs the shares, performs f on the top of obtained values, and then secret shares the result $f(x, y)$ to parties. The garbled circuit technique has seen dramatic improvements in recent years. The most notable optimized techniques are point-and-permute [5], Free-XOR [22], the half-gate [53], and fixed-key AES garbling optimizations [4].

2.3 Privacy-Preserving Machine Learning

The emerging MPC-based privacy-preserving machine learning (PPML) paradigm [8, 12, 16, 26, 27, 30, 32, 40, 43] enables different entities to jointly and privately train and evaluate various ML models over their joint data. Existing literature on PPML mainly focused on linear regression, logistic regression, neural network (NN), and transformer [8, 12, 16, 26, 32, 43]. Most PPML schemes follow a server-aided setting where data owners outsource the computation to a small number of non-trusted and non-colluding servers. Mohassel and Zhang [32] introduced the first practical PPML systems based on a two-server setting. Three-server [30] and four-server [40] designs achieve a weaker security guarantee in which collusion between any pair of these servers reveals the private data of the data owners. Therefore, the two-server PPML model is still preferable in many applications.

While causal inference plays a crucial role in modern data analysis, particularly in healthcare applications, there is relatively less emphasis on developing a secure protocol for causal inference, despite the extensive body of work in the broader field of PPML. Recent works applied differential privacy (DP) on top of the causal inference algorithm [25, 37]. Applying DP-noise significantly reduces model accuracy, an important factor in causal discovery. More significantly, noise-driven methods reveal sensitive information that a cryptographic approach like Aitia can completely hide.

2.4 Server-Aided Architecture

In this work, we follow the server-aided framework using two non-trusted and non-colluding servers. The goal is to train a causal discovery model on a joint dataset shared by multiple data owners. To achieve this, the data owners secret-share their sensitive data among two servers which then train models directly on the secret-shared data. This approach offers several benefits. Firstly, it involves minimal participation by the data owners, who only distribute their inputs once in the setup phase and are not involved

Algorithm	Training						Testing
	Time-Complexity	# Exponentiations	#Sqrt	# Divisions	# Comparisons	# Multiplications	Asym
GP Regression [49]	$O(N^3)$	$N(N+1)/2+nN$	N	$N(N+1)$	0	$N^3/2+N^2(n+m/2+3/2)+N(nm+m/2+n+7/3)$	N/A
SMO-SVR [7]	$O(T_{\text{SMO}}N^2)$	$N(N+1)/2$	0	T_{SMO}	$T_{\text{SMO}}(5N+7)$	$N(N+1)m/2+T_{\text{SMO}}(2N^2+N)$	$O(nN)$
BSGD-SVR	$O(T_{\text{BSGD}}B+N^2)$	$N(N+1)/2$	0	0	$T_{\text{BSGD}}(Bm+B+2)$	$N(N+1)m/2+T_{\text{BSGD}}B$	$O(nB)$

Table 1: Detailed counting of fundamental operations for training/testing non-linear regression models. We assume the Radial Basis Function (RBF) kernel is used in all three algorithms and Cholesky decomposition is used for matrix inversion. We have listed the operations in decreasing order of cost for MPC implementation.

in any future computation. Secondly, it utilizes efficient two-party secure computation techniques that require less communication and computation when deployed between a small number of participants. In Aitia, all intermediate values, such as the output of causal inference, are secret-shared between the two servers.

3 A NEW MODEL-TRAINING APPROACH FOR EFFICIENT SECURE COMPUTATION

In this section, we rethink the causal discovery model originally proposed in the influential work from Hoyer *et al.* [17]. We revisit the choice of the model and training algorithm with the goal of significantly accelerating performance when translated to a 2PC protocol. As we show in this section, an efficient ML model on plaintext data does not always translate to an efficient 2PC protocol. In the next section, we develop additional cryptographic optimizations tailored to the newly proposed training algorithm.

Notation	Explanation
N	Number of training samples
n	Number of testing samples
m	Number of features
B	The support vector budget (See Section 3.3)
$T_{\texttt{alg}}$	Number of iterations of algorithm `alg`, e.g., BSGD, SMO.

Table 2: Notation Table.

3.1 Notations and Problem Setup

We use bold lowercase letters to denote vectors (e.g., $\mathbf{x}$) and bold uppercase letters to denote matrices (e.g., $\mathbf{X}$). The sequence $1,\ldots,n$ is denoted as $[n]$. Suppose we have a training set ($\mathbf{X}^{\text{train}} \in \mathbb{R}^{N\times m}$, $\mathbf{y}^{\text{train}} \in \mathbb{R}^N$) with N samples. Each sample has m features and a target value. For a testing dataset $\mathbf{X}^{\text{test}} \in \mathbb{R}^{n\times m}$ with true target $\mathbf{y}^{\text{test}} \in \mathbb{R}^n$, we consider a non-linear regression model $\mathcal{A}$ that makes a prediction $\hat{\mathbf{y}}$. The accuracy of the prediction by the regression model is measured by the mean squared error (MSE): $\sum_i(\mathbf{y}_i^{\text{test}}-\hat{\mathbf{y}}_i)^2$. With the term $\mathbf{x}_i^*$ (resp. y_i^*) to denote the i^{th} sample (resp. target) in the $*$ dataset. For simplicity, we omit the superscript when the dataset being used is clear from the context. An overview of the notation is presented in Table 2. A non-linear regression model $\mathcal{M}$ can call two algorithms:

- `Train`: takes as input $(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$ and outputs a set of parameters for a non-linear regression model $\mathcal{M}$.
- `Predict`: takes as input $\mathbf{X}_{\text{test}}$ as well as $\mathcal{M}$ and outputs the prediction $\hat{\mathbf{y}}$.

3.2 On Choosing Non-linear Regression

According to Hoyer *et al.* [17], any non-linear regression model can be used for bivariate causal discovery (see Algorithm 1). In the following, we review two existing options for the ANM model.

On Gaussian Process Regression. The work of Mooij *et al.* [35] uses *Gaussian Process (GP) Regression* [41, 42] as the non-parametric regression model. Typically, Gaussian processes are presented either through the weight-space view or function-space view (see Chapter 2 in [42]). According to the function-space view, a Gaussian process is a collection of random variables (each associated with a function) where any finite number of the variables define a joint Gaussian distribution. GP is characterized by its mean function and its covariance function defined among pairs of random variables. In practice, to train a GP regression, one needs to choose a covariance function, e.g., squared exponential covariance, and use this function to compute the inversion (or the Cholesky decomposition) of an N-by-N matrix. A detailed version of the algorithm is presented in Algorithm 4 in the Appendix.

Overhead in the 2PC Setting. Given that matrix inversion takes $O(N^3)$ time, translating the above computation in a 2PC setting between two servers requires $O(N^3)$ rounds of interaction. Furthermore, matrix inversion can lead to numerical stability challenges that need to be addressed within the 2PC protocol. This illustrates that computational methods efficient in non-secure settings may not necessarily be efficient in secure computation settings due to the need for interaction in the 2PC setting.

It is known in the PPML community that different fundamental computation steps, e.g., exponentiation, division, comparison, addition, multiplication, incur different cost when translated to a secure protocol. Table 1 presents a detailed breakdown of the number of fundamental operations in GP. The operation that introduces the highest overhead in an MPC protocol is exponentiation, typically simplified to multiplication through approximate computation, albeit at the expense of accuracy loss.

On Support Vector Regression (SVR). In the following, take a different approach than [35] and propose a new non-linear regression model for ANM. Our candidate substitute for GP regression is called *support vector regression (SVR)* [2], see Algorithm 5 in the Appendix . SVR expands the support vector machine method to handle regression. A *kernel* SVR is parameterized by a weight vector $\mathbf{w}$ and a bias term b. To make a prediction for an unseen data point $\mathbf{x}$, kernel SVR first maps the feature vector $\mathbf{x}$ to a high-dimensional space using function $\phi(\cdot)$ and then outputs the inner product added with a bias term: $\langle\mathbf{w},\phi(\mathbf{x})\rangle+b$. Note that the range of mapping $\phi(\cdot)$ has infinite dimensions. Thus, the inner product cannot be trivially calculated. The above computational challenge can be solved efficiently by the kernel trick. Intuitively, we can rewrite the weight vector as $\mathbf{w}=\sum_{i=1}^N \alpha_i\phi(\mathbf{x}_i)$ where $\mathbf{x}_i, i\in[N]$ are training samples and α_is are weights for the training samples. Hence, the prediction formula for input $\mathbf{x}$ becomes $b+\sum_{i=1}^N \alpha_i\langle\phi(\mathbf{x}_i),\phi(\mathbf{x})\rangle$. Then the inner product can be calculated using the kernel trick

$\langle \phi(\mathbf{x}'), \phi(\mathbf{x}) \rangle = \mathcal{K}(\mathbf{x}', \mathbf{x})$ where $\mathcal{K}(\cdot, \cdot)$ is a tractable kernel function. Only a small number of training points will have $\alpha_i \neq 0$, , which are referred to as the *support vectors*.

Challenges of Training SVR via 2PC. The textbook approach [54] for training a kernel SVR is called sequential minimal optimization (SMO). SMO solves the dual of the regularized SVR optimization problem. It works by iteratively choosing a pair of α parameters, deciding whether they satisfy optimality conditions namely Karush-Kuhn-Tucker (KKT conditions) [24], and if not, updating the two α. The above SMO approach does not translate to an efficient 2PC protocol. First, it has an asymptotic complexity of $O(T_{\text{SGD}}N^2)$ where T_{SGD} is the upper bound on the number of iterations (typically at least N). Second, it is unclear how many training points will become support vectors after training. Consequently, while translating this approach to a secure 2PC protocol, one has to introduce oblivious computation to hide not only the identity but also the number of support vectors. Treating *all* training points as potential support vectors incurs significant computation and communication cost.

3.3 Rethinking SVR with MPC training in Mind

In the following, we sidestep the inefficiencies of GP and SMO-SVR by designing a new SVR training algorithm that we call *Budgeted Stochastic Gradient Descent SVR* or simply BSGD-SVR. Our objective is to modify SVR so that we simultaneously (*i*) maintain comparable accuracy of previously proposed training approaches, and (*ii*) ensure that the resulting model translates to an efficient training and testing as a 2PC protocol. We use two algorithmic adjustments to create a novel variation of SVR that has not been previously explored, potentially making it of independent interest:

- The first algorithmic insight is to train SVR with Stochastic Gradient Descent (SGD) instead of SMO, an approach inspired by Zeyuan *et al.* [55] and Shalev *et al.* [44] where SGD was used for Support Vector Machines. This adjustments allows a simpler update rule for the parameters of the model (as opposed to optimality testing via the KKT conditions).
- The second algorithmic insight to control the number of support vectors using *budgeting*. A similar technique was used by Wang *et al.* [48] to suppress the number of support vectors for SVM.

We perform a detailed experimental analysis under standard benchmarks for causal inference (see Table 3) and confirm that our approach has comparable accuracy to GP and SMO-SVR for every tested dataset. In Figure 5 in Appendix C, we show additional illustrative examples (similar to those in [10]) demonstrating the close performance of our newly proposed model to previous approaches.

Training SVR via Stochastic Gradient Descent. First, we propose replacing SMO with SGD to train SVR. To the best of our knowledge, the only application of SGD to SVR considers non-kernel based formulation [44]. Trivially applying SGD on SVR will lead to a gradient vector of infinite dimensions. To control the length of the gradient, we follow the intuition from [55] which trains kernel SVMs using SGD by dynamically maintaining a set of support vectors as shown in Algorithm 2 in Appendix.

The Role of Budgeting in Managing Support Vectors. The SGD training strategy lowers the time complexity of training from $O(T_{\text{SMO}}N^2)$ to $O(T_{\text{BSGD}}B + N^2)$. However, much like SMO, we do not know beforehand how many training points will act as support vectors. To address the issue, we replicate the adjustment from [48] and explicitly enforce a *budget B* for the number of support vectors. Roughly speaking, when the number of support vectors is smaller than the budget B , we maintain a set of fake support vectors so as to always have B members; when the number exceeds the threshold, we remove the appropriate number of support vectors.

Specifically, when training, we initialize a dictionary $\mathcal{D}$ to store the support vectors, their weights (which are denoted as $\mathbf{x}_i$ and α_i, respectively, in SVR algorithm) and the bias value (line 8-9).

Then in each training round, we pick a random sample indexed by i and run the prediction function on it (line 11-12). If the prediction is accurate enough, we continue to the next loop (line 13). Otherwise, we either insert the sample into the dictionary or update its weight depending on whether it is already in the dictionary (line 14-23). Last, if the dictionary is overflowed, we remove the support vector with the smallest weight (line 24-25). This way, we manage to control the number of support vectors and obtain a SVR protocol with an efficient training under MPC setting.

Algorithm 2: BSGD-SVR.

1 **Function** `Predict`($\boldsymbol{X}^{test}$, $\mathcal{D}$, b)**:**
2 Initialize $\hat{\mathbf{y}}[i] = b, \forall i \in [n_{test}]$
3 **for** $i \in 1, 2, \cdots, n_{test}$ **do**
4 **for** $\boldsymbol{x}_j \in \mathcal{D}.keys()$ **do**
5 $\hat{\mathbf{y}}[i] = \hat{\mathbf{y}}[i] + \mathcal{D}[\mathbf{x}_j] \cdot \mathcal{K}(\mathbf{x}_j, \mathbf{x}_i)$
6 **return** $\hat{\mathbf{y}}$
7 **Function** `Train`($\boldsymbol{X}^{train}$, $\boldsymbol{y}^{train}$, `params` = $\{B, T_{BSGD}, \eta, \xi\}$)**:**
8 Initialize a dictionary $\mathcal{D}$ for support vectors and their corresponding weights
9 Initialize a bias value $b = 0$
10 **for** $t = 1, 2, \cdots, T_{BSGD}$ **do**
11 Randomly pick a training index $i \in [N]$
12 Run the prediction procedure $\hat{y}_i \leftarrow$ `Predict`($\boldsymbol{x}_i$,$\mathcal{D}$, b)
13 **if** *prediction and true value differ too much:* $|y_i - \hat{y}_i| > \xi$ **then**
14 **if** *the sampled point is already in* $\mathcal{D}$*:* $\boldsymbol{x}_i$ *is in* $\mathcal{D}$*.keys()* **then**
15 **if** $\hat{y}_i > y_i$ **then**
16 $\mathcal{D}[\mathbf{x}_i] = \mathcal{D}[\mathbf{x}_i] - \eta\mathcal{K}(x_i, x_i)$
17 $b = b - \eta$
18 **else if** $\hat{y}_i < y_i$ **then**
19 $\mathcal{D}[\mathbf{x}_i] = \mathcal{D}[\mathbf{x}_i] + \eta\mathcal{K}(x_i, x_i)$
20 $b = b + \eta$
21 **else**
22 **if** $\hat{y}_i > y_i$ **then** $\mathcal{D}[\mathbf{x}_i] = -\eta\mathcal{K}(x_i, x_i)$
23 **else if** $\hat{y}_i < y_i$ **then** $\mathcal{D}[\mathbf{x}_i] = \eta\mathcal{K}(x_i, x_i)$
24 **if** *there are more than B support vectors in* $\mathcal{D}$ **then**
25 Remove the minimum weight absolute value from $\mathcal{D}$
26 **return** $\mathcal{D}$

SVR with Budgeted Stochastic Gradient Descent. Combining the above ideas, we obtain BSGD-SVR, an 2PC-friendly non-linear regression model shown in Algorithm 2. The training function takes data as input and has four hyperparameters: B for budget size, T_{BSGD} for number of iterations, η for the learning rate to update the weight, and ξ as the prediction-threshold. When predicting, we

Dataset	Feature→Target	Mean Squared Error (MSE)			Causal Direction via BSGD-SVR
		GP	SMO-SVR	BSGD-SVR	
Liver Disorder(345)	pair0033: alcohol→corpuscular volume	2.31×10^{-2}	1.98×10^{-2}	$(2.40 \pm 0.06) \times 10^{-2}$	✓
	pair0034: alcohol→alkaline phosphotase	2.72×10^{-2}	2.16×10^{-2}	$(2.45 \pm 0.26) \times 10^{-2}$	✓
	pair0035: alcohol→alanine aminotransferase	2.90×10^{-2}	2.36×10^{-2}	$(2.70 \pm 0.21) \times 10^{-2}$	✓
	pair0036: alcohol→aspartate aminotransferase	3.27×10^{-2}	2.10×10^{-2}	$(2.17 \pm 0.10) \times 10^{-2}$	✓
	pair0037: alcohol→gamma-glutamyl transpeptidase	3.78×10^{-2}	1.99×10^{-2}	$(2.21 \pm 0.06) \times 10^{-2}$	✓
Arrhythmia(452)	pair0022: age→height	1.35×10^{-4}	1.67×10^{-4}	$(2.44 \pm 0.50) \times 10^{-4}$	✓
	pair0023: age→weight	5.98×10^{-3}	4.62×10^{-3}	$(5.32 \pm 0.50) \times 10^{-3}$	✓
	pair0024: age→heart rate	10.4×10^{-3}	9.19×10^{-3}	$(10.3 \pm 0.9) \times 10^{-3}$	✓
Income(3000)	pair0012: age→wage per hour	6.10×10^{-4}	6.17×10^{-4}	$(6.09 \pm 0.18) \times 10^{-4}$	✓
	pair0017: age→dividends from stocks	13.6×10^{-5}	14.8×10^{-5}	$(8.70 \pm 1.54) \times 10^{-5}$	✓
NCEP-NCAR(3000)	pair0043: temperature (t)→temperature (t+1)	6.48×10^{-4}	6.52×10^{-4}	$(7.50 \pm 0.52) \times 10^{-4}$	✓
	pair0044: pressure (t)→pressure (t+1)	6.76×10^{-5}	6.77×10^{-5}	$(10.1 \pm 1.0) \times 10^{-5}$	✓
	pair0045: sea level pressure (t)→sea level pressure (t+1)	4.80×10^{-3}	4.89×10^{-3}	$(4.92 \pm 0.19) \times 10^{-3}$	✓
	pair0046: rel. humidity (t)→rel. humidity (t+1)	1.24×10^{-2}	1.48×10^{-2}	$(1.51 \pm 0.08) \times 10^{-2}$	✓
Abalone(4177)	pair0005: age→length	1.20×10^{-2}	1.32×10^{-2}	$(1.58 \pm 0.10) \times 10^{-2}$	✓
	pair0006: age→shell weight	1.24×10^{-2}	1.35×10^{-2}	$(1.66 \pm 0.20) \times 10^{-2}$	✓
	pair0007: age→diameter	5.56×10^{-4}	5.68×10^{-4}	$(7.82 \pm 1.16) \times 10^{-4}$	✓
	pair0008: age→height	1.61×10^{-2}	1.74×10^{-2}	$(1.90 \pm 0.05) \times 10^{-2}$	✓
	pair0009: age→whole weight	1.35×10^{-2}	1.47×10^{-2}	$(1.63 \pm 0.08) \times 10^{-2}$	✓
	pair0010: age→shucked weight	1.17×10^{-2}	1.26×10^{-2}	$(1.35 \pm 0.04) \times 10^{-2}$	✓
	pair0011: age→viscera weight	9.63×10^{-3}	10.1×10^{-3}	$(11.1 \pm 0.7) \times 10^{-3}$	✓

Table 3: Mean Square Error under different causality inference datasets for models trained with (1) Gaussian Process (GP) Regression; (2) SMO-SVR; (3) BSGD-SVR. Deviation is calculated over 10 independent runs. As the GP and SMO-SVR implementations are deterministic, we only show the standard deviation for BSGD-SVR. The "Causal Direction" column compares whether the predicted causality direction of the model trained with BSGD-SVR matches the ground truth for each pair.

calculate the weighted average of the kernel between each support vector and the testing data, much like standard SVR.

3.4 On Choosing Dependence Score

As our last step, we need to choose a dependence score that is suitable for the 2PC setting. According to Mooij *et al.* [35], the main candidates for dependence score are (1) HSIC-based scores; (2) entropy-based scores; (3) Gaussian scores; (4) empirical-Bayes scores; (5) minimum message length scores. We choose Gaussian score (see Definition 3.1) because it only requires two variance computation and two logarithm operations. This choice would competitive performance in a 2PC setting due to its simplicity.

Definition 3.1 (Gaussian Score). The Gaussian score between two vectors $\mathbf{u}$ and $\mathbf{v}$ is defined as

$$\log \mathrm{Var}(\mathbf{u}) + \log \mathrm{Var}(\mathbf{v})$$

3.5 Evaluation of BSGD-SVR

In this section, we would like to answer the following questions:

- How should we choose the support vector budget in BSGD-SVR?
- What is the trade-off between accuracy loss and performance improvement in BSGD-SVR?
- How do the number of fundamental operations in BSGD-SVR compare to those in other approaches?

We emphasize that the evaluation in this section concerns the accuracy of the SVR model trained with the newly proposed BSGD-SVR algorithm on *plaintext/unencrypted data*. Section 5 presents the performance of the (optimized) privacy preserving version.

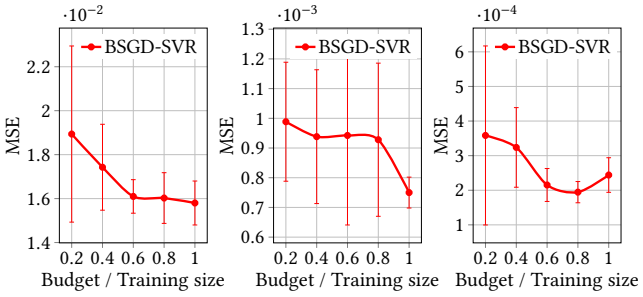
Setup. To answer these questions, we use five datasets from the causality dataset `CEB` [34] to test the proposed non-linear regression model and compare its accuracy to GP and SMO-SVR. For Gaussian process regression, we use the implementation from `scikit-learn` [39] with the default hyper-parameters. For SMO-SVR, we use the implementation from LibSVM [7] with the default hyper-parameters. We implement BSGD-SVR in slightly above 100 lines of Python code. We use the Gaussian score as the dependence score for all the causal inference experiments.

(1) - Parameter Selection. When selecting the budget size B and the number of training iterations T_{BSGD}, four aspects should be taken into account:

- Memory cost: Higher budget means we need to store more data support vectors in dictionary $\mathcal{D}$
- Generalization: If the budget size B is too small, the support vectors stored in dictionary $\mathcal{D}$ might need to be more representative to cover all data distribution.
- Accuracy: If T_{BSGD} is too small, the protocol has not "learned enough" about the dataset to correctly predict future inputs.
- Runtime: If T_{BSGD} is too large, the protocol runs longer.

In Figure 2, we study the budget size v.s. accuracy trade-off. The mean squared error (MSE) is being compared against various budget sizes for different numbers of training data points. The MSE decreases rapidly as we increase the budget size from $0.2N$ to $0.4N$ in all datasets. However, when we increase the budget size to $0.6N$, the MSE either decreases at a slower rate or remains the same. Finally, when we increase the budget size from $0.6N$ to $1.0N$, the trend shows that MSE decreases in some datasets, while it increases in others. Overall, although the budget that provides the lowest MSE varies based on the dataset, the assignment of $B = 0.5N$ balances our

objectives for high accuracy and a small budget. Additionally, our preliminary experiments showed that the assignment $T_{\text{BSGD}} = 2N$ is small enough while providing a competitive accuracy. We choose learning rate $\eta = 0.01$ and $\xi = 0.01$ for all datasets. More importantly, we verified (last column of Table 3) that our parameterization led to the correct *direction of causality* in all datasets.



(a) Abalone. **(b) NCEP-NCAR.** **(c) Arrhythmia.**

Figure 2: MSE vs. Budget Divided by Training Set Size (B/N)

To choose the remaining parameters (learning rate η and small threshold ξ), one approach is for a single party (e.g., a hospital) to tune parameters locally on their private dataset before the secure protocol. The party then shares the best hyperparameters (η and ξ) with all others to start BSGD-SVR training.

(2) - Accuracy Evaluation. The results are shown in Table 3. For most tested pairs, the accuracy of the model is very similar to the accuracy of both GP and SMO-SVR. In several datasets, our experiments show that the accuracy of BSGD-SVR is higher than both GP and SMO-SVR. For example, BSGD-SVR reduces the MSEs of GP and SMO-SVR by around ~36% and ~41% respectively for the `age`→`dividends` pair in the Income dataset. Overall, our experiments on 21 real-world datasets show that the newly proposed BSGD-SVR offers a competitive accuracy across all tested data.

(3) - Number of Fundamental Operations. We also analyze the computation cost of the three non-linear regression training algorithms with respect to different fundamental operations so as to extrapolate their performance once integrated with MPC. Our analysis is shown in Table 1. The operations in the first row are ordered in descending order with respect to performance overhead in an MPC setting. One important observation is that BSGD-SVR algorithm doesn't involve any square root or division operation, and it has the smallest number of exponentiations as well as the smallest number of multiplications. Overall, the newly proposed BSGD-SVR performs the smallest number of fundamental operations both in training and testing compared to SMO-SVR.

4 AITIA: SECURE CAUSAL DISCOVERY

In this Section we present a series of optimizations for Algorithm 2 so as to make the computation steps more efficient in an MPC setting. These optimizations neither affect security nor change the final computation result but: (*i*) eliminate conditional branches, and (*ii*) perform oblivious computation when handling the dictionary of support vectors. Finally, we present the 2PC design of the optimized BSGD-SVR that we call Aitia. The protocol is accompanied by a formal security proof and the corresponding threat model.

4.1 Modifying BSGD-SVR for Efficiency, Obliviousness, and Branch Removal

To ensure Obliviousness and become "MPC-friendly," we propose a series of adjustments to our algorithmic steps of BSGD-SVR. In the following, we describe the changes and the rationale behind them, and in Algorithm 3 in Appendix A.

Efficient Initialization of Support Vector Dictionary $\mathcal{D}$. Notice that Line 8 of Algorithm 2 initializes a dictionary denoted by $\mathcal{D}$. For our efficiency-driven optimization, we take into account the context in which the dictionary is used. That is, $\mathcal{D}$ stores the support vectors. A first approach would be to initialize $\mathcal{D}$ with a collection of randomly chosen training datapoints. Such an action introduces the following inefficiencies: (1) Suppose we happen to initialize with a datapoint that has a large-weight absolute value. In that case, because the removal rule (see Line 25 in BSGD-SVR) prioritizes the elimination of low-weight absolute value entries, the model is "stuck" with the large-absolute-value-weight support vector, a consequence that can affect its accuracy and convergence. (2) Suppose we happen to initialize with a significant number of datapoints that need to be removed from $\mathcal{D}$ throughout the first rounds of training. In that case, we introduce a significant overhead because every removal needs to store the new support vector in a temporary buffer, identify the minimum weight absolute value entry, and swap between them. Because of the above inefficiencies (which we identified through micro-benchmarks), we propose initializing the dictionary $\mathcal{D}$ in an empty state. This way, we will not have to remove unlucky large-weight absolute value initializations (point (1)) and will eliminate many unnecessary computational steps from the first B iterations of the algorithm (point (2)). We introduce two matrices **wgt** and **V** to represent the weights and the support vectors respectively, which replace the use of $\mathcal{D}$. **V** contains all the support vectors as its rows while **wgt** contains the weight (that is, α_i in the SVR notation of Section 3.2) of the support vectors in the corresponding position. Formally, **wgt** = $\mathcal{D}$.values() while **V** = $\mathcal{D}$.keys(). Initially, as $\mathcal{D}$ is empty, we initiate both **wgt** and **V** to zero. Overall, the above optimization helps the model to converge faster (so that T_{BSGD} remains low) and speeds up the computation by avoiding unnecessary removals from $\mathcal{D}$.

Oblivious Membership Test in $\mathcal{D}$. Notice that Line 14 of Algorithm 2 checks if the sampled datapoint is already in the dictionary $\mathcal{D}$. To make this computational step oblivious, we introduce vectors **fnd** and **ID**, each of them of length $B+1$. Vector **ID** stores the *index* of the datapoints from $\mathbf{X}^{\text{train}}$ that serve as support vectors in no particular order. Vector **fnd** is a binary vector which is populated as follows: to check if sampled datapoint i (see Line 11 of Algorithm 2) is in $\mathcal{D}$, our oblivious analog performs a linear scan on **ID**; during the j-th iteration we run the following comparison $\mathbf{ID}[j] \stackrel{?}{=} i$ and store the output bit of the comparison to $\mathbf{fnd}[j]$. Thus, after the $(B+1)$-th iteration, if any of the entries of **fnd** is 1, the sampled datapoint is in $\mathcal{D}$. The above modification makes the membership test oblivious by scanning the entire vector. Also, the proposed computation uses only comparison operations, which are efficient in an MPC setting. Finally, the comparisons $\mathbf{ID}[\cdot] \stackrel{?}{=} i$ can be run in parallel for multiple positions of **ID**, i.e., highly parallelizable code. For completeness, we note that other oblivious membership

approaches [47, 56] could work just as well. We opted for a simple approach (the linear scan on B location is fast for the tested datasets) that can also be parallelized.

Conditional Branch & Prediction Threshold. Notice that Line 13 of Algorithm 2 performs a conditional branch. To make this step oblivious, we instead introduce a bit-flag b_{ξ} that takes value 1 if the difference $|y_i - \hat{y}_i| - \xi$ is positive, and value 0 otherwise. This bit-flag b_{ξ} will be used in the computation of the next modification steps and will encode the result of this comparison, e.g., if we multiply a vector with b_{ξ} then it is zeroed in case $|y_i - \hat{y}_i| < \xi$.

Oblivious Insertion & Update in $\mathcal{D}$. Lines 15-25 in Algorithm 2 perform an insertion of a new support vector in dictionary $\mathcal{D}$. Recall that $\mathcal{D}$ operates under the budgeted setting so it can hold at most B support vectors. To accommodate an insertion we introduce the $(B+1)$-th position in $\mathcal{D}$ that is used as a *buffer* to temporarily host the newly inserted support vector. We have a loop invariant that states that at the start of each iteration in Line 10, the buffer position of $\mathcal{D}$ should be empty. Thus, in case we are inserting a new support vector, during the iteration, the buffer must *swap its content* with another entry of $\mathcal{D}$.

Locating the Min-Weight Position. Notice that the insertion takes place when both (1) the membership test fails, i.e., bit-flag $b_{\text{fnd}} = 0$, and (2) the prediction threshold ξ is surpassed, i.e., bit flag $b_{\xi} = 1$. Jumping ahead, we rely on the optimized code of CrypTen to identify the argmin of $|\mathbf{wgt}|$, so in the following we assume that vector **mloc** has 0 everywhere except the location where **wgt** has the minimum (absolute) value. To encode the conditional branching in our computation we introduce the following bit-wise operation

$$\mathbf{mlocCond} = \mathbf{mloc} \cdot b_{\xi}(1 \oplus b_{\text{fnd}}),$$

so that **mlocCond** is 0 everywhere except for when $b_{\text{fnd}} = 0$ and $b_{\xi} = 1$, in which case, **mlocCond** has value 1 in the location of the minimum absolute value of **wgt**. We note that the initialization of **mloc** and the assignment of **mlocCond** need to happen regardless of the values of b_{fnd} and b_{ξ} for the computation to be oblivious.

A Single Position Vector for Swap & Update. In the next step, we create a bit-vector called **editPos**. If we are inserting a new support vector that is temproarily stored in the buffer-location of $\mathcal{D}$ then the entry $\mathbf{editPos}(B+1)$ will be set to 1. On the other hand, if the sampled datapoint i is already a support vector in position k of $\mathcal{D}$, the entry $\mathbf{editPos}[k]$ will be set to 1. Notice that the initialization of **editPos** depends on the datapoint we sampled, to capture this conditional intialization we use again bit operations with the bit-flags b_{fnd} and b_{ξ} to get

$$\mathbf{editPos} = b_{\xi}(\mathbf{fnd} \oplus ((1 \oplus b_{\text{fnd}})\mathbf{1}_{B+1})),$$

where $\mathbf{1}_{B+1}$ is a $(B+1)$-dimensional vector with 0 everywhere except location $(B+1)$ that has value 1.

Weight Adjustment. Next, we use the newly computed **editPos** that identifies the position that needs to be updated (either an existing support vector or the buffer-location) to update the corresponding entry of the weight vector **wgt**. As for the type of weight update listed in Lines 16, 17, 22, and 23 of Algorithm 2, we opt for the RBF kernel K. This kernel has a property that $\mathcal{K}(x_i, x_i) = 1$, so the update for each weight can be computed by simply adding or subtracting the learning rate η. Thus, we define $upd = \text{sign}(\hat{y} - y)\eta$ to capture both the sign and the change of the weight. The operation

$$\mathbf{wgt} = \mathbf{wgt} - upd \cdot \mathbf{editPos}$$

will leave all weights untouched except (1) in case of a new support vector, the $(B+1)$-th weight is updated, and (2) in case of an existing support vector the corresponding weight is updated.

Swap in Case of Insertion. The final step performs a swap between the $(B+1)$-location of **wgt** and the corresponding minimum-weight support vector, *but only if* we are inserting a new support vector. The above requirement is capture already in the way that we computed **mlocCond**. That is, **mlocCond** is all 0s when there is no insertion of a new support vector and it has value 1 in location the support vector that needs to be removed. Thus, the following operation performs a swap to move the newly inserted support vector to its correct location if necessary:

$$\mathbf{wgt} = \mathbf{wgt} - \mathbf{mlocCond}(\mathbf{wgt} - \mathbf{wgt}[B+1]).$$

The swap needs to address the change not only in vector **wgt** but also in vector **ID** that holds the identifiers of support vectors and matrix **V** used to express the support vectors $\mathbf{x}_j$ as a matrix

$$\mathbf{V} = \mathbf{V} - \mathbf{mlocCond}(\mathbf{V} - x_i)$$

$$\mathbf{ID} = \mathbf{ID} - \mathbf{mlocCond}(\mathbf{ID} - i).$$

Efficiency via Vectorization. Another more generic optimization (as opposed to the above customized modifications in which the context was important) is to *vectorize* the computation steps. For example, the `Predict` function in Line 1 of BSGD-SVR, has a double loop; one loop for the set of test vectors and one for the list of support vectors (see Line 3,4 in Algorithm 2). The final closed-form expression of this calculation is

$$b + \sum_{\mathbf{x}_j \in \mathcal{D}.keys()} \mathcal{D}[\mathbf{x}_j] \cdot \mathcal{K}(\mathbf{x}_j, \mathbf{X}_{test}[i])$$

but can be equivalently represented using linear algebra as

$$b + \langle \mathbf{wgt}, \mathcal{K}(\mathbf{V}, \mathbf{X}_{test}[i]) \rangle,$$

where we use **wgt** instead of $\mathcal{D}[\mathbf{x}_j]$ and use **V** as the matrix version of support vectors. Vectorized computation is highly optimized, parallelizable in CrypTen.

4.2 The Aitia Protocol

Threat Model. As described in Section 2.3, our Aitia follows the server-aided framework using two non-trusted and non-colluding servers P_1 and P_2. Specifically, we assume that there are a set of data owners $U_1, \ldots, U_N$, each holding a private dataset Ψ_j. The data owners securely distribute their sensitive dataset among the two non-colluding servers P_1 and P_2 using a secret sharing scheme described in Section 2.2. At the end of this distribution, each server holds the secret-shared union of the dataset Ψ_j as $[\![\Psi]\!]$ where $\Psi = \{(x_i, y_i) | x_i \in \mathbb{R}^m, y_i \in \mathbb{R}\}_{i \in [n]}$. Our threat model assumes that the servers are semi-honest as we rely on the semi-honest two-party secure computation (2PC) tool, CrypTen [9], for implementation. Thus, the servers follow the protocol description but may attempt to extract sensitive information from the execution transcript. When any server (either P_1 or P_2) colludes with a set of data owners U_j, the coalition of corrupt parties learns nothing about the dataset Ψ_j of other non-corrupt parties (due to the underlying security

PARAMETERS:

- Two parties: P_1 and P_2
- η, T_{BSGD}, ξ, B, security parameter κ
- Kernel function $\mathcal{K}$
- A pseudorandom generator $PRG : \{0,1\}^\kappa \to \{0,1\}^*$

INPUT OF $P_{i\in[2]}$: Secret-shared dataset $[\![\Psi]\!]$ where $\Psi = \{(x_i, y_i) | x_i \in \mathbb{R}^m, y_i \in \mathbb{R}\}_{i\in[n]}$

PROTOCOL:

I. **Initialization:**
1. P_1 chooses a random seed $s \leftarrow \{0,1\}^\kappa$, and sends it to P_2.
2. Each party locally generate shares of 4 matrices $[\![\mathbf{V}]\!] \in \mathbb{R}^{(B+1)\times m}$, $[\![\mathbf{wgt}]\!], \mathbf{ID}, \mathbf{1}_{B+1} \in \mathbb{R}^{B+1}$ such that for party P_i: $[\![\mathbf{V}]\!]_i = 0$, $[\![\mathbf{wgt}]\!]_i = [\![\mathbf{ID}]\!]_i = 0$, and
$$[\![\mathbf{1}_{B+1}]\!]_i[k] = \begin{cases} 1, & \text{if } k = B+1, i = 1 \\ 0 & \text{otherwise} \end{cases}$$
3. Each party locally initiate bias value $[\![b]\!]_1 = [\![b]\!]_2 = 0$

II. **Secure SVR training:** Repeat the following T_{BSGD} times:
1. Each party computes an index $i = PRG(s)$, and define $[\![\mathbf{x}]\!] := [\![\mathbf{x}_i]\!]$
2. The parties jointly compute the prediction $[\![y']\!] : y' \leftarrow b + \langle \mathbf{wgt}, \mathcal{K}(\mathbf{V}, \mathbf{x}) \rangle$
3. Each party locally computes $[\![\Delta]\!] \leftarrow [\![y']\!] - [\![y_i]\!]$
4. The parties jointly computes $[\![b_\xi]\!]$ as:
$$b_\xi = \begin{cases} 1, & \text{if } \Delta > \xi \text{ or } \Delta < -\xi \\ 0 & \text{otherwise} \end{cases}$$
5. For $j \in [B]$, parties jointly computes $[\![\mathbf{fnd}[j]]\!]$ in parallel such that: $\mathbf{fnd}[j] = \begin{cases} 1, & \text{if } \mathbf{ID}[j] = i \\ 0 & \text{otherwise} \end{cases}$
6. Each party locally computes $[\![b_{\text{fnd}}]\!]$ as $b_{\text{fnd}} = \bigoplus_{j=1}^{B} \mathbf{fnd}[j]$
7. The parties jointly compute the updated positions vector $[\![\mathbf{editPos}]\!]$ such that $\mathbf{editPos} = b_\xi(\mathbf{fnd} \oplus ((1 \oplus b_{\text{fnd}})\mathbf{1}_{B+1}))$
8. Each party locally computes $[\![upd]\!] : upd \leftarrow \text{sign}(\Delta)\eta$
9. The parties jointly update $[\![\mathbf{wgt}]\!] : \mathbf{wgt} = \mathbf{wgt} - upd \cdot \mathbf{editPos}$
10. Each party locally update bias value $[\![b]\!]$: $[\![b]\!] = [\![b]\!] - [\![upd]\!]$
11. The parties jointly compute $[\![\mathbf{mloc}]\!]$ such that:
$$\mathbf{mloc}[j] = \begin{cases} 1, & \text{if } j = \arg\min |\mathbf{wgt}| \\ 0 & \text{otherwise} \end{cases}$$
12. The parties jointly compute $\mathbf{mlocCond} = \mathbf{mloc} \cdot b_\xi(1 \oplus b_{\text{fnd}})$
13. The parties jointly update $\mathbf{wgt} = \mathbf{wgt} - \mathbf{mlocCond}(\mathbf{wgt} - \mathbf{wgt}[B])$
14. The parties jointly update $\mathbf{V} = \mathbf{V} - \mathbf{mlocCond}(\mathbf{V} - \mathbf{x})$
15. The parties jointly update $\mathbf{ID} = \mathbf{ID} - \mathbf{mlocCond}(\mathbf{ID} - i)$
16. Each party locally set $\mathbf{wgt}[B+1] = 0, \mathbf{V}[B+1] = 0, \mathbf{ID}[B+1] = 0$

III. **Output:** A party sends its secret-shared causal inference's parameters $[\![\mathbf{V}]\!]$, $[\![\mathbf{wgt}]\!]$ to another party who outputs model by reconstructing the shares locally.

Figure 3: Our AITIA Protocol.

guarantee of secret sharing scheme). This threat model of AITIA has been formalized and used in various PPML scheme [31, 32, 40].

Note that our AITIA protocol can be extended to work with multiple non-colluding and malicious servers if implemented using MPC libraries that are secure against malicious adversaries like SPDZ [20]. Such an extension may allow additional optimizations to scale better in the MPC setting, as opposed to 2PC. We leave this direction as an open problem.

Main Protocol. We now describe the main protocol of AITIA which closely follows the modified BSGD-SVR algorithm presented in Section 4.2. We assume that the training samples Ψ are additively secret-shared amongst two parties. Figure 3 formally presents our AITIA protocol, which consists of two phases: initialization and secure BSGD-SVR training. The first phase is to implement Line 8-9 of Algorithm 2 in a privacy-preserving way. To ensure that both parties chooses similar training samples, the party P_1 can choose an arbitrary random seed s and broadcasts it to P_2. The party P_1 can also generate two shares of each of the 4 matrices $\mathbf{V}, \mathbf{wgt}, \mathbf{ID}, \mathbf{1}_{B+1}$ as well as two shares of the bias b such that they ensure the value indicated in line 2-3 of Figure 3.

The second phase consists of T_{BSGD} iterators. Each iteration starts with choosing a random index $j \leftarrow PRG(s)$ which is the same across both parties, where s is a PRG's seed obtained in the first phase. All the computation is performed on secret-shared versions of the matrices/vectors. After each iterator, the shares of $\mathbf{V}, \mathbf{wgt}, b_\xi, b_{\text{fnd}}, \mathbf{ID}, \mathbf{1}_{B+1}$ are either refreshed by new shares or updated with a new value. Note that XOR (and addition) can be locally computed by the party. We implement multiplication based on Beaver-triple. For equality test (e.g. to learn whether $j = \mathbf{ID}[i]$) we use the Crypten's comparison operator, which evaluate $j \leq \mathbf{ID}[i]$

and $j < \mathbf{ID}[i]$ in parallel. Note that the two comparison operators consists of one arithmetic-to-binary conversion and evaluating the sign of the first bit of $j - \mathbf{ID}[i]$. Hence, the complexity of this operator is linear in the length of the input's value. As the value of index in the dataset is bounded by number of data points in the dataset, the length of the index will not be very large. For example, if the dataset consists of 1 million data points, the length of the index is 20. Thus, this operator is efficient.

Theorem 4.1. *Aitia in Figure 3 securely computes the BSGD-SV described in Algorithm 3 in the semi-honest setting, given the ideal Garbled Circuit (GC) primitive and pseudo-random generator (PRG).*

The security proof of our Aitia construction is presented in Appendix A.1. It follows the security of Aitia's building blocks (e.g., secure comparison) and the fact that all intermediate values are under a secret-shared form.

Additional Optimization for Aitia. In the following, we detail one last optimization not depicted in Figure 3 for the sake of simplicity. In Figure 4, the red-colored bars show the breakdown of the training performance of the original Aitia design (as presented in Figure 3), while the blue-colored bars the breakdown *after* the optimization. The recorded operations are "Weight Update", "Prediction", "Swap", and "ArgMin". The shade of the blue (resp. red) changes to indicate a different operation from the above list.

Based on our experiments the operation "ArgMin", i.e., finding an entry with the smallest absolute weight, which is colored light-red is an expensive step which takes 66% of the total computation cost. To improve efficiency, we only perform the arg min function *after the first B iteration*. At the first B iterations, due to the fact that all **wgt** are initialized at 0, we can explicitly choose the minimum position to be equal to iteration number i and gradually add new vectors to the i-th column of $\mathbf{V}$. By doing this, we achieve 15% speedup since the optimized version performs only $T_{\text{BSGD}} - B$ operations of arg min compared to T_{BSGD} operations in the original version.

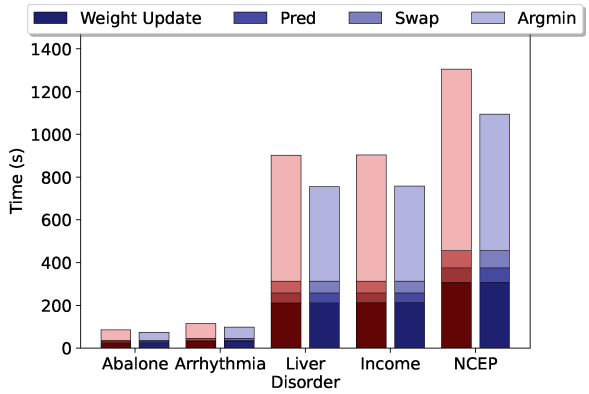


Figure 4: Runtime breakdown in Aitia across datasets. Left and right column refers to runtime before and after removing argmin computation from the first B iterations, respectively.

5 EVALUATION OF AITIA

This section describes the specifics of our implementation of Aitia. We also present an empirical evaluation of its performance in different causality benchmark datasets. The main goal of this Section is to answer the following question:

> *Does Aitia provide faster training and lower communication cost compared to GP and SMO-SVR implemented with PPML libraries?*

5.1 Implementation

We implement our Aitia protocol using CrypTen [9, 21], a framework for privacy-preserving machine learning built on PyTorch. We additionally employ CrypTen to implement GP and SMO-SVR. The outcomes present their secure versions, which we utilize as baselines for comparison. Below, we briefly explain how we use CrypTen in our implementation (see [21] for more details).

Data type conversion. CrypTen works with real numbers by multiplying each of them to a big number B and round the resulted number to the closest integer in the integer group $\mathbb{Z}_q$. In other words, it converts from $x \in \mathbb{R}$ to $\lfloor Bx \rceil \in \mathbb{Z}_q$. Later on, to get back the real number, CrypTen divide by B the integer number:

$$x = \frac{\lfloor Bx \rceil}{B}.$$

Exponentiation. CrypTen has multiple options for doing exponentiation approximation. In our experiment, we use the limit approximation

$$\lim_{n \to \infty} (1 + \frac{x}{2^n})^{2^n}.$$

We set $n = 8$ in our experiment. This value is recommended by CrypTen as it provides a favorable balance between accuracy and efficiency. The exponentiation computation then consists of 8 multiplications (square operation) along with 1 truncation operation.

Comparison. CrypTen calculates secure comparison, i.e., $x > y$ given 2 numbers x, y by securely evaluate the left-most bit of $x - y$, which gives information about sign of the number, leading to evaluation of $[\![(x - y) < 0]\!]$ which is equivalent to $[\![x < y]\!]$.

Argmin. We use tree-reduction with log-reduction algorithm of CrypTen. Given an input list of N elements, the algorithm has a round complexity of $O(\log N)$, communication of $O(N^2)$ bits, and $O(N)$ comparisons.

Reciprocal. CrypTen evaluates $[\![\frac{1}{x}]\!]$ by using Newton-Rhapson iterations. This method uses an initial guess, y_0, for the reciprocal and repeats the following update:

$$y_n = y_n(2 - xy_n)$$

In our experiment, we use $n = 10$, which means we implement 20 secure multiplication operations per secure reciprocal evaluation.

5.2 Evaluation

We utilize the datasets introduced in Section 3.5, employing a train/test split ratio of 8 : 2. We evaluation the performance of our Aitia, secure GP and secure SMO-SVR on a local machine with 11th Gen Intel(R) Core(TM) i9-11900KF Processor with an all-core CPU frequency of 3.50GHz, 16 vCPU, 32GB RAM. Unfortunately, due to extensive computation time, the runtime for secure SMO-SVR is estimated based on the runtime of their first 10 updates.

Dataset	Pair	Training Time (s)			Comm (GB)			Comm (M Rounds)		
		GP	SMO-SVR	AITIA	GP	SMO-SVR	AITIA	GP	SMO-SVR	AITIA
Liver Disorder(345)	pair0033	282.53 (3.77×)	2945 (39.26×)	75.01	0.63 (3.50×)	0.87 (4.83×)	0.18	0.16 (2.05×)	3.44 (44.10×)	0.078
	pair0034	271.62 (3.57×)	3100 (40.72×)	76.13						
	pair0035	276.44 (3.61×)	3053 (39.88×)	76.56						
	pair0036	270.96 (3.61×)	3165 (42.12×)	75.15						
	pair0037	275.03 (3.62×)	3025 (39.83×)	75.95						
Arrhythmia(452)	pair0022	474.57 (4.75×)	4928 (49.29×)	99.97	1.40 (4.83×)	1.50 (5.17×)	0.29	0.27 (2.45×)	5.67 (51.55×)	0.11
	pair0023	467.12 (4.69×)	4904(49.44×)	99.60						
	pair0024	467.01 (4.74×)	4924 (49.94×)	98.60						
Income(3000)	pair0012	21171 (27.49×)	188901 (245×)	770	412.07 (32.60×)	63.37 (5.01×)	12.64	11.61 (14.70×)	223.84 (283×)	0.79
	pair0017	21168 (27.35×)	188066 (243×)	774						
NCEP-NCAR(3000)	pair0043	21234 (27.66×)	197850 (258×)	768	412.07 (32.60×)	63.37 (5.01×)	12.64	11.61 (14.70×)	223.84 (283×)	0.79
	pair0044	21243 (27.38×)	197280 (254×)	776						
	pair0045	21204 (27.61×)	196560 (256×)	768						
	pair0046	21328 (27.59×)	197177 (255×)	773						
Abalone(4177)	pair0005	41546 (37.36×)	378603 (340×)	1112	1111.59 (45.20×)	121.03 (4.92×)	24.59	22.45 (20.41×)	431.19 (392×)	1.10
	pair0006	41560 (38.09×)	379104 (347×)	1091						
	pair0007	41846 (38.32×)	361227 (331×)	1092						
	pair0008	41853 (38.29×)	380608 (348×)	1093						
	pair0009	41894 (38.58×)	379606 (350×)	1086						
	pair0010	41710 (38.20×)	379558 (348×)	1092						
	pair0011	41730 (37.90×)	381006 (346×)	1101						

Table 4: System Performance of (1) Privacy-Preserving Gaussian Process (GP) Regression; (2) SMO-SVR; (3) AITIA. Training dataset pair are assigned in Appendix D. Number of rounds are in millions. Training time of SMO-SVR are estimated based on the average of the first 10 updates.

In the evaluation, we choose the parameters as follow: $B = 0.5, T_{\text{BSGD}} = 2N, \eta = 0.01$ for all dataset, and $\xi = 0.01$ for Abalone, Arrhythmia and NCEP-NCAR dataset, $\xi = 0.05$ for liver disorder dataset, and $\xi = 0.001$ for the income dataset.

Table 4 and Table 5 report the training and testing performances of all protocols across various datasets. As expected, our AITIA demonstrates superior running time and communication cost efficiency across all datasets, with the difference becoming more noticeable as the size of the training dataset increases.

Performance of Secure Training. According to our experiments, the training time of our AITIA is *significantly faster* compared to the baselines. In particular, for the Liver Disorder dataset, which consists of 345 data points, our AITIA training is approximately 3.6× faster than GP and 40× faster than SMO-SVR. In terms of concrete numbers, training with GP takes over 4 minutes, and training with SMO-SVR takes nearly an hour, while AITIA achieves the same convergence in just over 1 minute.

As the dataset size increases, the performance gap become more pronounced. Specifically, when training on the Abalone dataset, consisting of 4177 datapoints, GP and SMO-SVR require 37.36× and 340× more time for training than AITIA, respectively. To provide a concrete comparison: GP takes over 41500 seconds (approximately 11.5 hours) for *one pair of variables*, while SMO-SVR takes 361227 seconds (approximately 100 hours) for the same pair. In contrast, AITIA completes training in only about 1100 seconds, equating to less than 20 minutes of computational time. The performance gap between GP and AITIA increases linearly with the number of data points, ranging from 3.6× in datasets with 345 data points (Liver disorder) to 38× in datasets with 4177 data points (Abalone). Similarly, the gap between SMO-SVR and AITIA grows from 39.26× (Liver Disorder) to 350× (Abalone) as the dataset size increases.

The experiment illustrates that AITIA is both practical and scales effectively as the number of data points increases to the thousands, particularly in scenarios where causal inference across multiple variables is necessary. For instance, when all 7 pairs of the Abalone dataset are combined, AITIA requires only 2.13 hours, whereas GP and SMO-SVR demand 81.15 hours and 733 hours, respectively.

When considering communication cost, AITIA demonstrates notably reduced communication size and fewer communication rounds compared to secure GP and SMO-SVR. Like the runtime performance, there is also a noticeable linear trend in the communication gap. Specifically, the difference between secure GP and AITIA ranges from 3.5× more communication size and 2.05× more rounds when training on Liver disorder (345 data points) to 45× more communication size and 20.41× more rounds when training on Abalone (which has 12.11 times more data points than Liver disorder). Similarly, while the gap in communication between SMO-SVR and BSGD-SVR remains around 5×, the difference in the number of rounds increases from 44.10× in Liver disorder to 392× in Abalone.

Performance of Secure Testing. Interstingly, from Table 5, we can see that the gap between AITIA and the baselines decreases as the dataset size increases. For instance, AITIA exhibits a speed improvement of 31.43× compared to GP for Liver Disorder with 345 data points, but this figure reduces to only 8× for Abalone with 4177 data points. According to our micro-benchmarks, this phenomenon is caused by the encoding function of CrypTen's implementation, which encodes small tensors. This explains why AITIA testing time on Liver Disorder and Arrhythmia are much faster compared to

Dataset	Pair	Testing Time (s)			Comm (MB)			Comm (Rounds)		
		GP	SMO-SVR	AITIA	GP	SMO-SVR	AITIA	GP	SMO-SVR	AITIA
Liver Disorder(345)	pair0033	0.44 (31.43×)	0.35 (25.00×)	0.014	17.44 (10.90×)	14.82 (9.26×)	1.60	25 (1.92×)	22 (1.69×)	13
	pair0034	0.45 (32.14×)	0.36 (25.71×)	0.014						
	pair0035	0.41 (29.29×)	0.34 (24.29×)	0.014						
	pair0036	0.43 (33.08×)	0.36 (27.79×)	0.013						
	pair0037	0.45 (32.14×)	0.36 (25.71×)	0.014						
Arrhythmia(452)	pair0022	1.57 (104.67×)	1.46 (97.33×)	0.015	29.89 (10.87×)	25.41 (9.24×)	2.75	25 (1.92×)	22 (1.69×)	13
	pair0023	1.57 (104.67×)	1.28 (85.33×)	0.015						
	pair0024	1.54 (102.67×)	1.32 (88.00×)	0.015						
Income(3000)	pair0012	5.70 (6.79×)	3.81 (4.54×)	0.84	1318.41 (10.91×)	1120.66 (9.27×)	120.89	25 (1.92×)	22 (1.69×)	13
	pair0017	5.70 (8.64×)	3.81 (5.77×)	0.66						
NCEP-NCAR(3000)	pair0043	5.65 (6.42×)	3.83 (4.35×)	0.88	1318.41 (10.91×)	1120.66 (9.27×)	120.89	25 (1.92×)	22 (1.69×)	13
	pair0044	5.68 (7.47×)	3.82 (5.03×)	0.76						
	pair0045	5.61 (7.38×)	3.81 (5.01×)	0.76						
	pair0046	5.66 (8.20×)	3.83 (5.55×)	0.69						
Abalone(4177)	pair0005	9.05 (7.67×)	4.67 (3.96×)	1.18	2555.38 (10.90×)	2172.11 (9.27×)	234.39	25 (1.92×)	22 (1.69×)	13
	pair0006	9.41 (8.40×)	4.71 (4.21×)	1.12						
	pair0007	9.27 (8.06×)	4.73 (4.11×)	1.15						
	pair0008	8.71 (7.38×)	4.68 (3.97×)	1.18						
	pair0009	9.42 (7.79×)	4.69 (3.88×)	1.21						
	pair0010	9.82 (8.77×)	4.69 (4.19×)	1.12						
	pair0011	9.77 (8.01×)	4.68 (3.84×)	1.22						

Table 5: System Performance of (1) Privacy-Preserving Gaussian Process (GP) Regression; (2) SMO-SVR; (3) AITIA. Testing dataset pair are assigned in Table 7 in Appendix D.

secure GP and SMO-SVR. However, when the set of support vectors becomes sufficiently large, the encoding process slows down, resulting in AITIA's testing time being only 3-5× faster compared to secure SMO-SVR and 6-9× faster compared to secure GP.

Regarding communication, the communication gap between secure GP and AITIA remains approximately 10.90× for communication size and 1.92× for the number of rounds. Similarly, the communication gap between secure SMO-SVR and AITIA stays at 9.26× for communication size and 1.69× for communication rounds. When testing the Abalone dataset, AITIA incurs only 234.59 MB in communication costs, whereas secure GP costs 2555.38 MB and SMO-SVR costs 2172.11 MB. This demonstrates that AITIA is capable of significantly more efficient testing even in communication-limited environments.

6 CONCLUSION & DISCUSSION

In this work, we proposed an efficient secure protocol called AITIA for bivariate causal discovery. Our approach involves proposing a new SVR model accompanied by a training algorithm with the potential to accelerate significantly the secure implementation. We proposed a series of optimizations and designed a 2PC protocol that resulted in a training time speedup of up to 346×.

Discussion. Our proposed AITIA design shows significant improvement over textbook approaches implemented in CrypTen. Yet, there are still exciting open problems to address:

- *Alternative Libraries & Speedups*: Our current implementation of AITIA utilizes Crypten, meaning its performance is closely tied to Crypten's performance. For instance, during secure testing, AITIA experiences a slowdown on large datasets, but this could be improved if the encoding function issue in Crypten is resolved.
- *GPU Acceleration*: Utilizing GPUs for accelerating machine learning training is a widespread practice [46], and this trend can also benefit our AITIA design. Fortunately, the algorithmic steps in AITIA are designed for parallelization, enabling direct performance enhancements through GPU acceleration. Additionally, revisiting Algorithm 3 to optimize its execution on GPUs could further improve efficiency, for example, exploring methods to enhance the Oblivious Insertion & Update operation in AITIA.
- *Malicious multiple-party AITIA*: As discussed in Section 4.2, AITIA has the capability to operate with multiple non-colluding and malicious servers if implemented using MPC libraries that offer security against malicious adversaries, such as SPDZ [20]. Future optimization efforts are necessary to accelerate computations in the malicious MPC setting.

ACKNOWLEDGMENTS.

We thanks Qi Pang for contributing helpful suggestions to our protocol implementation. The first and the fourth authors were partially supported by NSF awards #2101052, #2115075, and ARPA-H SP4701-23-C-0074. The third author was partially supported by the NSF award #2154732 and the Meta Security Research Award.

REFERENCES

[1] Website of the u.s. census bureau, 1994.

[2] Mariette Awad and Rahul Khanna. Support vector regression. In *Efficient learning machines*, pages 67–80. Springer, 2015.

[3] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[4] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society Press, May 2013.

[5] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 257–266. ACM Press, October 2008.

[6] C. Williams C.E Rasmussen. *Gaussian Process for Machine Learning*. MIT Press, 2005.

[7] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[8] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 201–210. JMLR.org, 2016.

[9] Facebook. CrypTen: a framework for Privacy Preserving Machine Learning built on PyTorch. https://github.com/facebookresearch/CrypTen.

[10] Gary Flake and Steve Lawrence. Efficient svm regression training with smo. *Machine Learning*, 46, 03 2001.

[11] Nir Friedman and Iftach Nachman. Gaussian process networks. *arXiv preprint arXiv:1301.3857*, 2013.

[12] Radhika Garg, Kang Yang, Jonathan Katz, and Xiao Wang. Scalable mixed-mode mpc. Cryptology ePrint Archive, Paper 2023/1700, 2023. https://eprint.iacr.org/2023/1700.

[13] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. Association for Computing Machinery.

[14] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[15] H Altay Guvenir, Burak Acar, Gulsen Demiroz, and Ayhan Cekin. A supervised machine learning algorithm for arrhythmia analysis. In *Computers in Cardiology 1997*, pages 433–436. IEEE, 1997.

[16] Xiaoyang Hou, Jian Liu, Jingyu Li, Yuhan Li, Wen jie Lu, Cheng Hong, and Kui Ren. Ciphergpt: Secure two-party gpt inference. Cryptology ePrint Archive, Paper 2023/1147, 2023. https://eprint.iacr.org/2023/1147.

[17] Patrik O Hoyer, Dominik Janzing, Joris M Mooij, Jonas Peters, and Bernhard Schölkopf. Nonlinear causal discovery with additive noise models. In *Advances in neural information processing systems*, pages 689–696, 2009.

[18] Dominik Janzing, Joris Mooij, Kun Zhang, Jan Lemeire, Jakob Zscheischler, Povilas Daniušis, Bastian Steudel, and Bernhard Schölkopf. Information-geometric approach to inferring causal directions. *Artificial Intelligence*, 182:1–31, 2012.

[19] Eugenia Kalnay, Masao Kanamitsu, Robert Kistler, William Collins, Dennis Deaven, Lev Gandin, Mark Iredell, Suranjana Saha, Glenn White, John Woollen, et al. The ncep/ncar 40-year reanalysis project. *Bulletin of the American meteorological Society*, 77(3):437–472, 1996.

[20] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[21] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[22] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *Proc. of the 35th ICALP*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[23] Samory Kpotufe, Eleni Sgouritsa, Dominik Janzing, and Bernhard Schölkopf. Consistency of causal inference under the additive noise model. In *International Conference on Machine Learning*, pages 478–486, 2014.

[24] H W Kuhn and A W Tucker. Nonlinear programming proceedings of the 2nd berkeley symposium on mathematical statistics and probability. In *University of California Press, Berkeley*, pages 481–492, 1951.

[25] Matt J. Kusner, Yu Sun, Karthik Sridharan, and Kilian Q. Weinberger. Private causal inference. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1308–1317, Cadiz, Spain, 09–11 May 2016. PMLR.

[26] Dacheng Li, Hongyi Wang, Rulin Shao, Han Guo, Eric Xing, and Hao Zhang. MPCFORMER: FAST, PERFORMANT AND PRIVATE TRANSFORMER INFERENCE WITH MPC. In *The Eleventh International Conference on Learning Representations*, 2023.

[27] Yun Li, Yufei Duan, Zhicong Huang, Cheng Hong, Chao Zhang, and Yifan Song. Efficient 3PC for binary circuits with application to Maliciously-Secure DNN inference. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5377–5394, Anaheim, CA, August 2023. USENIX Association.

[28] David Lopez-Paz, Krikamol Muandet, Bernhard Schölkopf, and Iliya Tolstikhin. Towards a learning theory of cause-effect inference. In *International Conference on Machine Learning*, pages 1452–1461, 2015.

[29] Michael C. Mackey and Leon Glass. Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289, 1977.

[30] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 35–52, New York, NY, USA, 2018. Association for Computing Machinery.

[31] Payman Mohassel, Mike Rosulek, and Ni Trieu. Practical privacy-preserving k-means clustering. The 20th Privacy Enhancing Technologies Symposium, 2020. https://eprint.iacr.org/2019/1158.

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

[33] Joris M Mooij, Dominik Janzing, Tom Heskes, and Bernhard Schölkopf. On causal discovery with cyclic additive noise models. In *Advances in neural information processing systems*, pages 639–647, 2011.

[34] Joris M Mooij, Dominik Janzing, Jakob Zscheischler, and Bernhard Schölkopf. `CauseEffectPairs` repository. http://webdav.tuebingen.mpg.de/cause-effect/, 2014.

[35] Joris M Mooij, Jonas Peters, Dominik Janzing, Jakob Zscheischler, and Bernhard Schölkopf. Distinguishing cause from effect using observational data: methods and benchmarks. *The Journal of Machine Learning Research*, 17(1):1103–1204, 2016.

[36] Warwick J Nash, Tracy L Sellers, Simon R Talbot, Andrew J Cawthorn, and Wes B Ford. The population biology of abalone (haliotis species) in tasmania. i. blacklip abalone (h. rubra) from the north coast and islands of bass strait. *Sea Fisheries Division, Technical Report*, 48:p411, 1994.

[37] Yuki Ohnishi and Jordan Awan. Locally private causal inference for randomized experiments, 2023.

[38] Office on Smoking, Centers for Disease Control, Prevention, et al. How tobacco smoke causes disease: The biology and behavioral basis for smoking-attributable disease: A report of the surgeon general. 2010.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[40] Rahul Rachuri and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. *CoRR*, abs/1912.02631, 2019.

[41] Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (GPML) toolbox. *J. Mach. Learn. Res.*, 11:3011–3015, 2010.

[42] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006.

[43] Deevashwer Rathee, Anwesh Bhattacharya, Divya Gupta, Rahul Sharma, and Dawn Song. Secure floating-point training. Cryptology ePrint Archive, Paper 2023/467, 2023. https://eprint.iacr.org/2023/467.

[44] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[45] Xiaohai Sun, Dominik Janzing, and Bernhard Schölkopf. Causal reasoning by evaluating the complexity of conditional densities with kernel methods. *Neurocomputing*, 71(7-9):1248–1256, 2008.

[46] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. Cryptgpu: Fast privacy-preserving machine learning on the gpu. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1021–1038, 2021.

[47] Xiao Wang, Kartik Nayak, Chang Liu, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. *IACR Cryptol. ePrint Arch.*, page 185, 2014.

[48] Zhuang Wang, Koby Crammer, and Slobodan Vucetic. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. *The Journal of Machine Learning Research*, 13(1):3103–3131, 2012.

[49] Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[50] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[51] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

[52] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[53] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.

[54] Zhi-Qiang Zeng, Hong-Bin Yu, Hua-Rong Xu, Yan-Qi Xie, and Ji Gao. Fast training support vector machines using parallel sequential minimal optimization. In *2008 3rd international conference on intelligent system and knowledge engineering*, volume 1, pages 997–1001. IEEE, 2008.

[55] Allen Zhu Zeyuan, Chen Weizhu, Wang Gang, Zhu Chenguang, and Chen Zheng. P-packsvm: Parallel primal gradient descent kernel svm. In *2009 Ninth IEEE International Conference on Data Mining*, pages 677–686. IEEE, 2009.

[56] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Linear private set union from Multi-Query reverse private membership test. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 337–354, Anaheim, CA, August 2023. USENIX Association.

[57] Kun Zhang and Aapo Hyvarinen. On the identifiability of the post-nonlinear causal model. *arXiv preprint arXiv:1205.2599*, 2012.

Algorithm 3: Oblivious BSGD-SVR without if-branches.

Function `Train`($\boldsymbol{X}^{train}$, $\boldsymbol{y}^{train}$)**:**

Initialize a bias value $b = 0$.

Initialize 4 matrices: $\mathbf{V}$ of size $(B+1) \times m$ to store support vectors, **wgt** to store weights, **ID** to store the indices in the budget, and $\mathbf{1}_{\mathrm{B}+1}$ to indicate buffer position. **wgt**, **ID**, $\mathbf{1}_{\mathrm{B}+1}$ of size $B+1$. $\mathbf{V}$, **wgt**, **ID** are initialized at 0, while $\mathbf{1}_{\mathrm{B}+1}$ are all 0 except for the last position, i.e.

$$\mathbf{1}_{\mathrm{B}+1}[i] = \begin{cases} 1, & \text{if } i = B+1 \\ 0, & \text{otherwise} \end{cases}$$

for $t = 0, 1, \cdots, T_{BSGD}$ **do**

Randomly pick a training index $i \in [N]$

Run the prediction procedure $\hat{y}_i \leftarrow b + \langle \mathbf{wgt}, \mathcal{K}(\mathbf{V}, x_i) \rangle$

Use a boolean variable b_ξ to indicate whether the difference between the prediction and the true target exceeds the threshold:

$$b_\xi = \begin{cases} 1, & \text{if } \hat{y}_i - y_i > \xi \text{ or } \hat{y}_i - y_i < -\xi \\ 0 & \text{otherwise} \end{cases}$$

Use a boolean variable b_{fnd} to denote whether the picked sample is already in the support vectors and a boolean vector **fnd** to locate it if the answer is yes:

for $j = 0, 1, \ldots, B$ **do**

$$\mathbf{fnd}[j] = \begin{cases} 1, & \text{if } i = \mathbf{ID}[j] \\ 0 & \text{otherwise} \end{cases}$$

$b_f = \bigoplus_{j=1}^{B} \mathbf{fnd}[j]$

Get the update position vector **editPos**: $\mathbf{editPos} = b_\xi(\mathbf{fnd} \oplus ((1 \oplus b_{\mathrm{fnd}})\mathbf{1}_{\mathrm{B}+1}))$

Get the update value: $upd = \mathrm{sign}(\hat{y}_j - y_j)\eta$

Update weight **wgt**, bias b, and find the minimum vector **mloc**:

$\mathbf{wgt} = \mathbf{wgt} - upd \cdot \mathbf{editPos}$

$b = b - upd$

$$\mathbf{mloc}[j] = \begin{cases} 1, & \text{if } j = \arg\min |\mathbf{wgt}| \\ 0 & \text{otherwise} \end{cases}$$

Swap the buffered position $B+1$ with the argmin position

$\mathbf{mlocCond} = \mathbf{mloc} \cdot b_\xi(1 \oplus b_{\mathrm{fnd}})$

$\mathbf{wgt} = \mathbf{wgt} - \mathbf{mlocCond}(\mathbf{wgt} - \mathbf{wgt}[B+1])$

Update $\mathbf{V}$, **ID**

$\mathbf{V} = \mathbf{V} - \mathbf{mlocCond}(\mathbf{V} - \mathbf{x}_i)$

$\mathbf{ID} = \mathbf{ID} - \mathbf{mlocCond}(\mathbf{ID} - i)$

Reset buffer position to 0:

$\mathbf{V}[B+1] = \mathbf{wgt}[B+1] = \mathbf{ID}[B+1] = 0$

return $\mathbf{V}$, **wgt**

A OUR BSGD-SVR ALGORITHM

A.1 Security Proof of Aitia

Sketch of proof: In the first step of our Aitia construction, P_1 chooses a random seed s and sends it to P_2. This simulation is elementary as s is random. Excepting Step (I,1), the two parties execute symmetric operations in which they execute the same code with their input and obtain secret shares of the intermediate values and the final output. Thus, the role of both parties is the same, excluding Step (I.1). In the following, we present a simulation for a corrupted P_1. Simulating corrupted P_2 is similar. We first formally describe the behavior of the simulator:

(1) The simulator honestly plays the role of PRG at Steps (II,1). For every query $PRG(s)$ made by the adversary, record outputs in a set O_1.
(2) Using garbled circuit, the simulator executes the below computations honestly.
   - "`if`" conditions at Steps (II,4-5), and (II,11)
   - Kernel function at Step (II,2)
   - Bit operations, additions, and multiplications at Steps (II,7), (II,9), (II,12-15)

(3) Upon receiving the secret-shared parameters of the kernel, the simulator sends them to the ideal functionality of BSGD-SVR. This causes the honest party P_2 to obtain the final output.

Due to the security of the garbled circuit and secret-shared intermediate result, the simulation for (2) and (3) is perfect. We now prove that this simulation and the real interaction are indistinguishable for (1) via the following sequence of hybrids.

Hybrid 0: The real interaction, with P_2 running honestly with input $[\![\Psi]\!]$ and giving its output to the environment according to the protocol description, namely the kernel's parameters K.

Hybrid 1: Same as the previous hybrid, except for how PRG is simulated. A query to $PRG(s)$ is answered with a uniformly random response. Thus, the sets O_1 can be replaced with random.

□

B BASELINE ALGORITHMS

This section provides pseudocode for Gaussian Process (GP) Regression and SMO-SVR algorithms that were used in the main paper. The GP Regression algorithm is taken from [6], and SMO-SVR algorithm is taken from [10]. The algorithms are shown in Algorithm 4 and Algorithm 5 respectively.

Algorithm 4: Gaussian Process Regression.

Input: prior mean $\boldsymbol{\mu}, \boldsymbol{\mu}*$, noise scale σ, training dataset $(\boldsymbol{X}, \boldsymbol{y})$, testing dataset $\boldsymbol{X}*$

1 $\boldsymbol{y}^* = \boldsymbol{\mu}^* + K(\boldsymbol{X}^*, \boldsymbol{X})[K(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}]^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$

C REGRESSION PERFORMANCE

We conducted experiments similar to those described in [10] to evaluate the performance of our BSGD-SVR algorithm in approximating various functions, including both linear and non-linear ones. Our results demonstrate that BSGD-SVR effectively learns to approximate a wide range of complicated functions. The functions considered in this section are:

- Linear function: $y = 2.4x + 1.3$
- Sinus function: $y = sin(x)$
- $y = sin(x * 0.15)/x$
- Mackey Glass function [29] for $\tau = 17, a = 0.2, b = 0.1, \Delta t = 1$

The results of the approximation are depicted in Figure 5. These results indicate that BSGD-SVR performs comparably to conventional SMO-SVR in approximating complex target functions.

D DATASET DETAILS

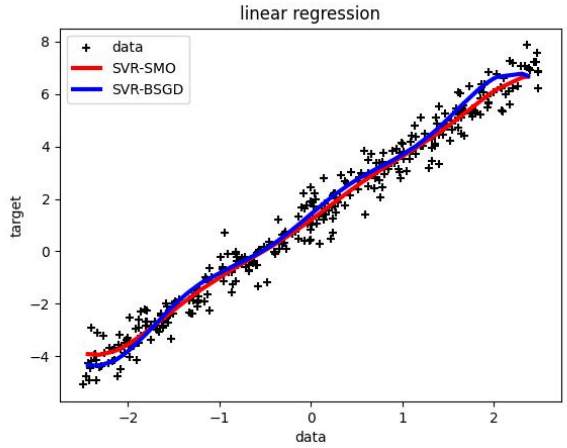
We provide more information about the dataset that we used in our experiments throughout this paper. The dataset's reference and source are presented in Table 6, while the assignments of pair numbers are outlined in Table 7.

Algorithm 5: SMO for ϵ-SVR.

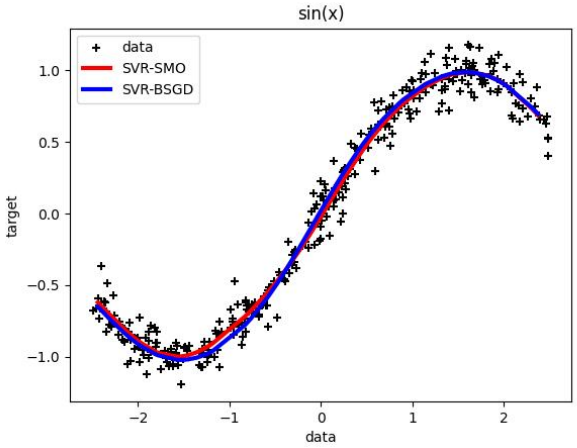
Hyperparameters: learning rate σ, training iteration upper bound T, tolerance threshold ϵ, C, τ
Input: training dataset $\Psi = \{(x_i, y_i) | x_i \in \mathbb{R}^m, y_i \in \mathbb{R}\}_{i \in [n]}$
Function `Train(`Ψ`)`:
  $\boldsymbol{\alpha} \leftarrow 0^n$, $b \leftarrow 0$, $\boldsymbol{K}$ is a matrix, $\boldsymbol{K}_{ij} = K(x_i, x_j)$
  Indicator variable on whether we examine all vectors: γ = true
  for $k \leq T$ **do**
    $\boldsymbol{\alpha}_{old} = \boldsymbol{\alpha}$
    if γ **then**
      $\mathbf{S} = \{1, 2, \ldots, n\}$
    else
      $\mathbf{S} = \{i \in [n] : |\boldsymbol{\alpha}[i]| \notin \{0; C\}\}$
    for $i \in \boldsymbol{S}$ **do**
      for $j \in [n], j \neq i$ **do**
        $s \leftarrow \boldsymbol{\alpha}[i] + \boldsymbol{\alpha}[j]$
        $\eta = \boldsymbol{K}[i, i] + \boldsymbol{K}[j, j] - 2\boldsymbol{K}[i, j]$
        $\Delta = 2\epsilon/\eta$
        $y_i^* = b + \sum_{k=1}^{n} \boldsymbol{\alpha}[k]\boldsymbol{K}[i, k]$
        $y_j^* = b + \sum_{k=1}^{n} \boldsymbol{\alpha}[k]\boldsymbol{K}[j, k]$
        $\boldsymbol{\alpha}[i] = \boldsymbol{\alpha}[i] + \frac{1}{\eta}(y_i + y_j - y_i^* - y_j^*)$
        $\boldsymbol{\alpha}[j] = s - \boldsymbol{\alpha}[i]$
        if $\boldsymbol{\alpha}[i] * \boldsymbol{\alpha}[j] < 0$ **then**
          if $|\boldsymbol{\alpha}[i]| \geq \Delta \&\& |\boldsymbol{\alpha}[j] \geq \Delta|$ **then**
            $\boldsymbol{\alpha}[i] = \boldsymbol{\alpha}[i] - \text{sign}(\boldsymbol{\alpha}[i]) \cdot \Delta$
          else if $|\boldsymbol{\alpha}[i]| > |\boldsymbol{\alpha}[j]|$ **then**
            $\boldsymbol{\alpha}[i] = s$
        $L = \max(s - C, -C)$
        $H = \min(C, s + C)$
        $\boldsymbol{\alpha}[i] = \min(\max(\boldsymbol{\alpha}[i], L), H)$
        $\boldsymbol{\alpha}[j] = s - \boldsymbol{\alpha}[i]$
        $b_i = y_i - y_i^* + (\boldsymbol{\alpha}[i] - \boldsymbol{\alpha}_{old}[i])\boldsymbol{K}[i, i] + (\boldsymbol{\alpha}[j] - \boldsymbol{\alpha}_{old}[j])\boldsymbol{K}[i, j] + b$
        $b_j = y_j - y_j^* + (\boldsymbol{\alpha}[j] - \boldsymbol{\alpha}_{old}[j])\boldsymbol{K}[j, j] + (\boldsymbol{\alpha}[i] - \boldsymbol{\alpha}_{old}[i])\boldsymbol{K}[i, j] + b$
        $b = \frac{b_i + b_j}{2}$
    if $\|\boldsymbol{\alpha}_{old} - \boldsymbol{\alpha}\| < \tau$ && γ **then**
      return $\boldsymbol{\alpha}, b$
    else
      $k = k + 1$
    $\gamma = !\gamma$
  return $\boldsymbol{\alpha}, b$

Dataset	Size	References	URL
Abalone	4177	[35, 36]	https://archive.ics.uci.edu/ml/datasets/Abalone
Arrhythmia	452	[15, 35]	https://archive.ics.uci.edu/ml/datasets/Arrhythmia
Income	3000	[1, 35]	https://archive.ics.uci.edu/ml/datasets/Census-Income+(KDD)
Liver Disorder	345	[35]	https://archive.ics.uci.edu/ml/datasets/Liver+Disorders
NCEP-NCAR	3000	[19, 33]	http://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanalysis.surface.html

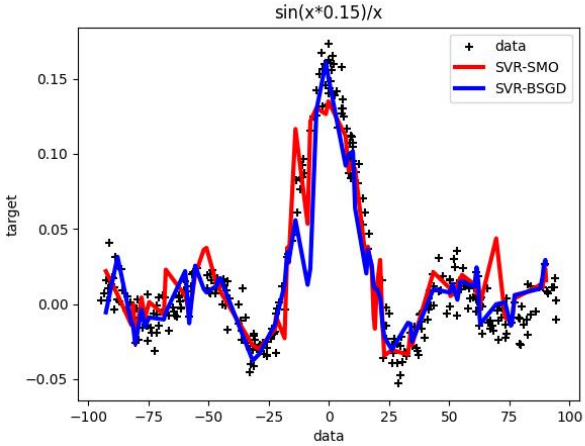
Table 6: Datasets.



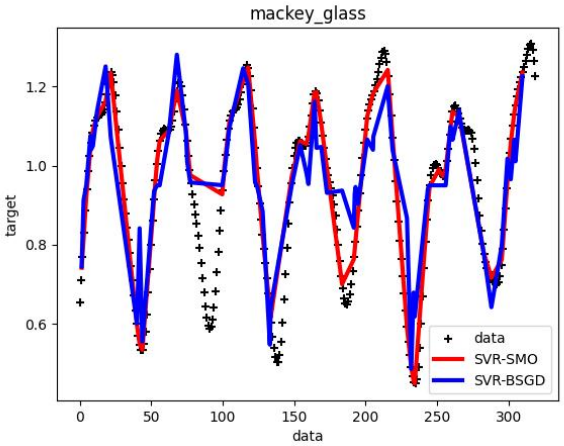
(a) $y = 2.4x + 1.3$



(b) $y = \sin x$



(c) $y = \sin(x * 0.15)/x$



(d) Mackey-Glass

Figure 5: Regression Performance of BSGD-SVR vs. standard SMO-SVR across four different functions.

Dataset	Size	Feature → Label	Pair
Liver Disorder	345	alcohol → mean corpuscular volume	pair0033
		alcohol → alkaline phosphotase	pair0034
		alcohol → alanine aminotransferase	pair0035
		alcohol → aspartate aminotransferase	pair0036
		alcohol → gamma-glutamyl transpeptidase	pair0037
Arrhythmia	452	age → height	pair0022
		age → weight	pair0023
		age → heart rate	pair0024
Income	3000	age → weight per hour	pair0012
		age → dividends from stock	pair0017
NCEP-NCAR	3000	temperature (t) → temperature $(t+1)$	pair0043
		pressure (t) → pressure $(t+1)$	pair0044
		sea level pressure (t) → sea level pressure $(t+1)$	pair0045
		relative humidity (t) → relative humidity $(t+1)$	pair0046
Abalone	4177	age → length	pair0005
		age → shell weight	pair0006
		age → diameter	pair0007
		age → height	pair0008
		age → whole weight	pair0009
		age → shucked weight	pair0010
		age → viscera weight	pair0011

Table 7: Dataset's Feature → Label associated pair number. The pair number are labelled in similar way as in [35]

n_{servers}	Dataset	Pair	Training Time (s)			Comm (GB)			Comm (M Rounds)		
			GP	SMO-SVR	AITIA	GP	SMO-SVR	AITIA	GP	SMO-SVR	AITIA
3	Liver Disorder (345)	0033	495.69 (3.43×)	7121 (49.32×)	144.39	1.93 (3.86×)	3.86 (7.72×)	0.50	0.49 (3.50×)	8.60 (61.43×)	0.14
		0034	505.26 (3.57×)	7476 (52.85×)	141.45						
		0035	503.43 (3.52×)	7441 (52.02×)	143.05						
		0036	497.63 (3.46×)	7475 (52.09×)	143.49						
		0037	502.30 (3.50×)	7431 (52.70×)	143.71						
	Arrhythmia (452)	0022	846.70 (4.20×)	12410 (61.51×)	201.75	4.29 (5.23×)	7.31 (8.91×)	0.82	0.82 (4.32×)	14.39 (75.74×)	0.19
		0023	840.63 (4.17×)	12533 (62.23×)	201.39						
		0024	862.55 (4.30×)	12518 (62.39×)	200.63						
	Income (3000)	0012	39656 (24.89×)	540341 (339×)	1593	1240 (33.86×)	924.94 (25.26×)	36.62	34.81 (24.51×)	386.53 (272×)	1.42
		0017	39206 (24.66×)	536544 (337×)	1590						
	NCEP-NCAR (3000)	0043	39332 (24.80×)	542692 (342×)	1586	1240 (33.86×)	924.94 (25.26×)	36.62	34.81	386.53 (272×)	1.42
		0044	39352 (24.86×)	537471 (340×)	1583						
		0045	39586 (24.98×)	536797 (339×)	1585						
		0046	39377 (24.75×)	538459 (338×)	1591						
	Abalone (4177)	0005	79729 (35.03×)	1071488 (471 ×)	2276	3342 (46.97×)	2605 (36.61×)	71.15	67.32 (34.00×)	1365 (689×)	1.98
		0006	79425 (34.84×)	1073849 (471×)	2280						
		0007	80163 (35.14×)	1073041 (470×)	2281						
		0008	79604 (34.85×)	1076514 (471×)	2284						
		0009	79674 (34.98×)	1076248 (472×)	2278						
		0010	79529 (34.96×)	1075168 (473×)	2274						
		0011	79634 (34.90×)	1075184 (471×)	2282						
4	Liver Disorder (345)	0033	622.27 (3.26×)	9748 (51.13×)	190.65	2.57 (3.52×)	5.45 (7.47×)	0.73	0.49 (3.50×)	8.60 (61.43×)	0.14
		0034	637.54 (3.37×)	9883 (52.29×)	189.02						
		0035	636.23 (3.34×)	9851 (51.84×)	190.04						
		0036	639.16 (3.35×)	9794 (51.34×)	190.76						
		0037	640.96 (3.36×)	9803 (51.37×)	190.84						
	Arrhythmia (452)	0022	1054 (3.96×)	16534 (62.07×)	266.34	5.72 (4.76×)	10.27 (8.56×)	1.20	0.82 (4.32×)	14.39 (75.74×)	0.19
		0023	1099 (4.16×)	16548 (62.65×)	264.13						
		0024	1067 (4.02×)	16571 (62.46×)	265.31						
	Income (3000)	0012	49303 (23.01×)	705245 (329×)	2143	1653 (31.15×)	1251 (23.57×)	53.07	34.81 (24.51×)	386.53 (272×)	1.42
		0017	48930 (22.82×)	710124 (331×)	2144						
	NCEP-NCAR (3000)	0043	49061 (22.88×)	711857 (332×)	2144	1653 (31.15×)	1251 (23.57×)	53.07	34.81 (24.51×)	386.53 (272×)	1.42
		0044	49148 (22.94×)	706524 (330×)	2142						
		0045	49200 (22.96×)	709064 (331×)	2143						
		0046	48926 (22.86×)	709180 (331×)	2140						
	Abalone (4177)	0005	99165 (31.89×)	1438749 (463×)	3110	4455 (43.19×)	3531 (32.49×)	103.13	67.32 (34.00×)	1365 (689×)	1.98
		0006	99252 (31.88×)	1411883 (454×)	3113						
		0007	99378 (31.99×)	1422002 (458×)	3107						
		0008	99475 (32.00×)	1413483 (455×)	3108						
		0009	99090 (31.93×)	1422113 (458×)	3103						
		0010	99147 (31.82×)	1415591 (454×)	3116						
		0011	99004 (31.86×)	1418585 (457×)	3107						

Table 8: Multiparty System Performance of (1) Privacy-Preserving Gaussian Process (GP) Regression; (2) SMO-SVR; (3) AITIA. n_{servers} indicates number of servers.