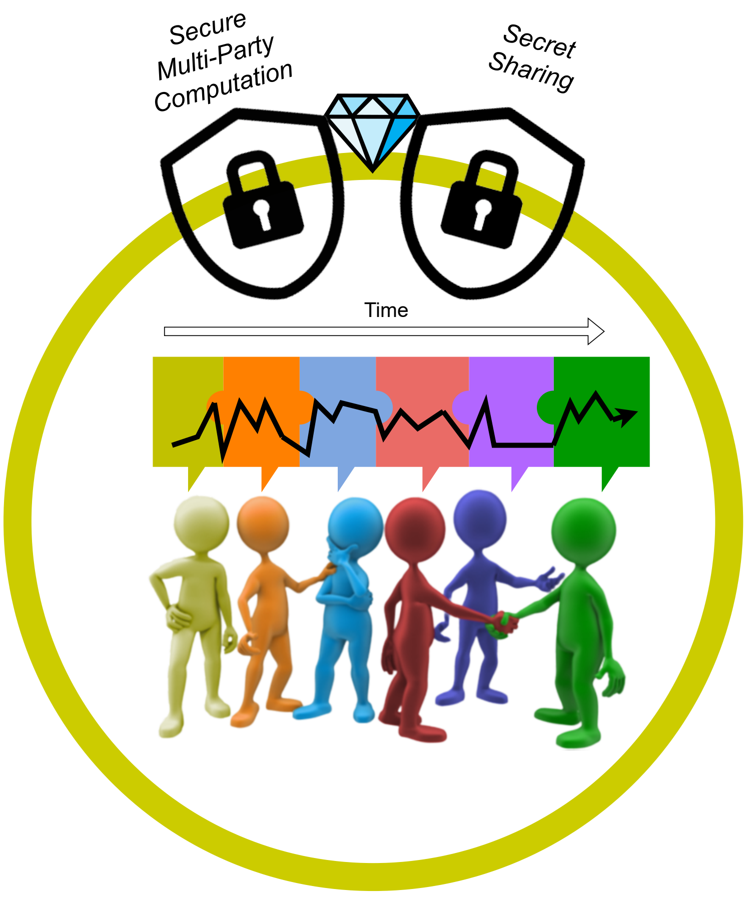
Share your Secrets for Private Forecasting with Vertical Federated Learning

Version of August 14, 2023



Aditya Shankar

Share your Secrets for Private Forecasting with Vertical Federated Learning

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Aditya Shankar

Distributed Systems Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Share your Secrets for Private Forecasting with Vertical Federated Learning

Author:	Aditya Shankar
Student id:	5454360
Email:	`a.ashankar@student.tudelft.nl`

Abstract

Vertical federated learning's (VFL) immense potential for time series forecasting in industrial applications such as predictive maintenance and machine control remains untapped. Critical challenges to be addressed in the manufacturing industry include small and noisy datasets, model explainability, and stringent privacy requirements for training and inference of forecasting. Additionally, to increase industry adaptability, such forecasting models must scale well with the parties/clients while ensuring strong convergence and low tuning complexity. To this end, we propose and design "**S**ecret-shared **T**ime Series Forecasting with **V**FL" (STV), a novel framework with the following features: i) a privacy-preserving VFL algorithm for time series forecasters such as SARIMAX and autoregressive trees, ii) secret sharing with multi-party computation protocols for aggregating intermediate training data and for privacy-preserving serverless inference, iii) extension of secure two-party matrix operations for direct parameter optimization to multiple parties, giving strong convergence with minimal hyperparameter tuning complexity. We conduct evaluations on six diverse datasets from both public and industry-specific contexts. Our results demonstrate that `STV`'s forecasting accuracy is comparable to those of centralized approaches and that direct optimization can outperform centralized methods by 23.81% on forecasting accuracy, including state-of-the-art diffusion models and long-short-term memory. We also conduct scalability analysis to offer the opportunity for flexible decision-making by examining the communication costs of direct and iterative optimization, allowing navigating between these two approaches.

Thesis Committee:

Chair:	Dr. Lydia Y. Chen, Faculty EEMCS, TU Delft
University supervisor:	Dr. Jérémie Decouchant, Faculty EEMCS, TU Delft
Company supervisor:	Dr. Dimitra Gkorou, ASML
Committee Member:	Dr. Rihan Hai, Faculty EEMCS, TU Delft

Acknowledgements

I would like to express my heartfelt gratitude to my academic supervisors, Dr. Lydia Chen, Dr. Jérémie Decouchant, and Dr. Dimitra Gkorou, for their invaluable guidance, encouragement, and support throughout my thesis journey. Their insightful feedback, constructive criticism, and patience have been instrumental in shaping the quality of this work. I am also deeply grateful to Dr. Satej Khedekar from ASML B.V., for his practical and insightful inputs. Their contributions to this research project have been immeasurable, and I am honored to have had the opportunity to work with such dedicated professionals.

My heartfelt gratitude also goes to my family and friends for their unwavering support, and for always believing in me. Their encouragement and motivation have kept me going during the most challenging times.

Aditya Shankar
Delft, the Netherlands
August 14, 2023

Contents

List of Figures

Chapter 1

Research Paper

This first chapter contains the draft of the research paper. It is a condensed version of the thesis, providing a concise overview of the main points and findings.

The more detailed and comprehensive version of the thesis begins from the following chapters. These delve deeper into the research, presenting detailed analysis, methodology, and conclusions.

Share your Secrets for Private Forecasting with Vertical Federated Learning

Abstract

Vertical federated learning's (VFL) immense potential for time series forecasting in industrial applications such as predictive maintenance and machine control remains untapped. Critical challenges to be addressed in manufacturing include small and noisy datasets, model explainability, and stringent privacy requirements for training and inference of forecasting. Additionally, to increase industry adaptability, such forecasting models must scale well with the parties while ensuring strong convergence and low-tuning complexity. To this end, we propose and design "**S**ecret-shared **T**ime Series Forecasting with **V**FL" (STV), a novel framework with the following features: i) a privacy-preserving VFL algorithm for time series forecasters such as SARIMAX and autoregressive trees, ii) secret sharing with multi-party computation protocols for aggregating intermediate training data and for private, serverless inference, iii) extension of secure two-party matrix operations for direct parameter optimization, giving strong convergence with minimal hyperparameter tuning complexity. We conduct evaluations on six diverse datasets from public and industry-specific contexts. Our results demonstrate that `STV`'s forecasting accuracy is comparable to those of centralized approaches and that direct optimization can outperform centralized methods by 23.81% on forecasting accuracy, including state-of-the-art diffusion models and long-short-term memory. We also conduct scalability analysis to offer flexible decision-making opportunities by examining the communication costs of direct and iterative optimization, allowing navigation between the two.

Introduction

Time series forecasting with vertically-partitioned data is relevant in manufacturing applications like continuous operations, predictive maintenance, and machine control (Lin et al. 2019; Susto and Beghi 2016). For example, let us consider the scenario in Figure 1 with two primary manufacturers and their customer[1]. The manufacturers collect sensory data during production while the customer owns post-production performance data, serving as the outputs or labels. Predicting outputs from sensor values would allow the manufacturers to make pre-shipment corrections, saving valuable time and helping to calibrate equipment in a complex, high-volume production factory. However, confidentiality agreements between customers and manufacturers prevent sharing of sensitive information like outputs and sensor data.

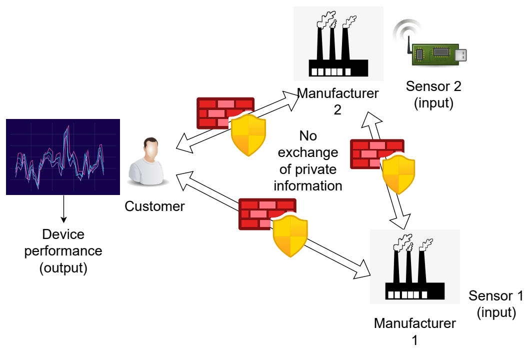


Figure 1: Problem scenario—forecasting device performance needs inputs from multiple parties, all of whom want to protect the confidentiality of their data.

Existing framework. To address these privacy concerns, *federated learning* (FL) has been emerging as a promising solution (Konečnỳ, McMahan, and Ramage 2015). In FL, training follows a model-to-data approach without data leaving the party's premises. *Vertical federated learning* (VFL) falls within this, where each participant owns a different feature set pertaining to the same sample ID (Yang et al. 2019). Our manufacturing example comes under VFL since the sensor inputs and outputs are split between the participants.

Challenges. Despite its relevance, time-series forecasting with VFL has received limited scholarly attention. This underscores the critical need for further exploration and research in this domain, especially considering the following complexities introduced by manufacturing scenarios.

First, neural networks (NNs) tend to overfit on manufacturing datasets since they can be affected by slow collection and noisy measurements, leading to small datasets (Li et al. 2021; Gkorou et al. 2017; Zhu et al. 2023). Moreover, their complex parameter interactions limit their interpretability, which is needed for understanding the reasoning

[1]A secondary manufacturer, but is the "customer" from the perspective of the primary manufacturers

behind predictions to optimize production quality and downtime (Wang et al. 2022; Vollert, Atzmueller, and Theissler 2021).

Second, inferencing privately is very important in manufacturing scenarios involving business competitors. While VFL methods based on homomorphic encryption (HE) (Hardy et al. 2017; Cheng et al. 2021; Fu et al. 2022) and split learning (Vepakomma et al. 2018; Chen et al. 2020; Yan et al. 2022) protect the privacy of features during training, the outputs are assumed to exist with one party alone, i.e., a *server* who is usually the label owner. This is unsuitable as the server observes the inference results of all requesting parties. Other methods, like differential privacy, introduce privacy-performance trade-offs by adding noise to data, which is unsuitable when datasets are already noisy (Sarwate, Chaudhuri, and Monteleoni 2009; Geyer, Klein, and Nabi 2017; Abadi et al. 2016; Yang et al. 2019).

Third, while direct optimization methods like the normal equation (Blais and others 2010) reach globally optimal solutions without requiring any hyperparameter tuning, they are not scalable to large problem instances, unlike iterative methods such as gradient descent. However, industries require both scalable and convenient solutions.

Contributions. To address these challenges, we develop a novel framework, *Secret-Shared Time Series Forecasting with Vertical Federated Learning* (`STV`), with the following contributions.

VFL forecasting framework. `STV` enables forecasting for VFL using *Secret sharing* (SS) (Shamir 1979) and *secure multi-party computation* (SMPC) (Cramer, Damgård, and others 2015; Lindell 2005). These do not involve privacy-performance trade-offs and offer cryptographic privacy guarantees. We propose $\texttt{STV}_\texttt{L}$ for linear models such as SARIMAX (Korstanje 2021; Perktold 2023; Hamilton 2020), and $\texttt{STV}_\texttt{T}$ for autoregressive trees (ARTs) (Meek, Chickering, and Heckerman 2002).

Private, serverless inference. `STV` ensures that the party requiring inference obtains the final predictions directly, without an intermediary. This is achieved by maintaining values as secret shares throughout the process. The requesting party can collect/aggregate these shares to get the predictions first-hand without needing a server.

Adaptable optimization with Least Squares. $\texttt{STV}_\texttt{L}$ uses a two-step approach with least squares (LS). This lends adaptability, as LS can be optimized using both iterative and direct methods. While iterative approaches like gradient descent scale better, direct optimization methods do not require hyperparameter tuning and guarantee global convergence. Therefore, we offer both options for diverse forecasting scenarios.

We thoroughly evaluate `STV` on multiple fronts. First, we compare the forecasting accuracy of $\texttt{STV}_\texttt{L}$ and $\texttt{STV}_\texttt{T}$ with centralized state-of-the-art forecasters based on diffusion models (Alcaraz and Strodthoff 2022), Long Short Term Memory (LSTM), and SARIMAX with Maximum Likelihood Estimation (MLE) (Hamilton 2020). Second, we compare the communication costs of iterative and direct optimization of linear forecasters under different scaling scenarios, highlighting their trade-offs. We use a wide range of datasets: five public datasets (Air quality, flight passengers, SML 2010, PV Power, Rossman Sales) and one real dataset from semiconductor manufacturing.

Background and Related Work

In this section, we provide background knowledge on time series models and SMPC and highlight the shortcomings of related works.

Time series forecasting

`STV` uses a popular linear forecaster, SARIMAX (**S**easonal **A**uto**R**egressive **I**ntegrated **M**oving **A**verage with e**X**ogenous variables) (Korstanje 2021; Perktold 2023), that generalizes other forecasters such as ARIMAX, ARMAX, and ARX (Hamilton 2020; Montgomery, Jennings, and Kulahci 2015). It forecasts future values by combining past values (autoregressive), past residuals (moving average), and exogenous variables linearly (Perktold 2023). For example, output Y at time t can be represented using a polynomial H, containing errors $\epsilon(t-i)$, past values $Y(t-i)$, and the currently observed exogenous features $X_j(t)$:

$$\begin{aligned} H: Y(t) = \alpha_1 Y(t-1) + \alpha_2 Y(t-2) + \beta_1 \varepsilon(t-1) + \\ \gamma_1 X_1(t) + \gamma_2 X_2(t) + \varepsilon(t) \end{aligned} \tag{1}$$

The coefficients (α, β, γ) are estimated using methods like MLE (Hamilton 2020) or LS (Hannan and Kavalieris 1984; Lütkepohl 2007; Liu et al. 2016; Tarsitano and Amerise 2017). However, MLE with SMPC is limited to likelihood functions like the exponential or multivariate normal distributions (Snoke et al. 2017; Lin and Karr 2010), since computations with SMPC are challenging due to the limited number of supported mathematical operations.

Under normally distributed errors, LS approaches can be used, by transforming the datasets, (X, Y), into time-lagged design matrices, (ϕ_X, ϕ_Y), to represent Equation 1:

$$\underbrace{\begin{bmatrix} Y(3) \\ Y(4) \\ .. \\ Y(t) \end{bmatrix}}_{\phi_Y} = \underbrace{\begin{bmatrix} Y(2) & Y(1) & \varepsilon(2) & X_1(3) & X_2(3) \\ Y(3) & Y(2) & \varepsilon(3) & X_1(2) & X_2(2) \\ .. & .. & .. & .. & .. \\ Y(t-1) & Y(t-2) & \varepsilon(t-1) & X_1(t) & X_2(t) \end{bmatrix}}_{\phi_X} \times \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \gamma_1 \\ \gamma_2 \end{bmatrix}}_{A} + \underbrace{\begin{bmatrix} \varepsilon(3) \\ \varepsilon(4) \\ .. \\ \varepsilon(t) \end{bmatrix}}_{\varepsilon} \tag{2}$$

Optimizing A can be viewed as a linear regression problem if not for its circular dependency on the residuals. A two-step approach is used to resolve this (Tarsitano and Amerise 2017; Lütkepohl 2007; Hannan and Kavalieris 1984; Liu et al. 2016). First, the residuals are estimated by modeling using only autoregressive (AR) and exogenous terms. Then all the coefficients in A are jointly optimized by setting the residuals in ϕ_X to the estimates . An example of applying the method is shown in Appendix E.

With LS, A can be estimated using iterative approaches like gradient descent (GD) or direct methods like the normal equation (NE) (Blais and others 2010):

$$A_{optimal} = ((\phi_X)^T \phi_X)^{-1} ((\phi_X)^T \phi_Y) \tag{3}$$

Method	Type			TS	IP.	Optim.	
	Trees	LR	NN			it.	dir.
(Yan et al. 2022)	✗	✗	✓	✓	✗	✓	✗
(Hardy et al. 2017)	✗	✓	✗	✗	✗	✓	✗
(Han et al. 2009)	✗	✓	✗	✗	✓	✓	✓
(Cheng et al. 2021)	✓	✗	✗	✗	✗	✓	✗
(Xie et al. 2022)	✓	✗	✗	✗	✓	✓	✗
(Shi et al. 2022)	✗	✓	✗	✗	✓	✓	✗
STV (this work)	✓	✓	✗	✓	✓	✓	✓

Table 1: Comparison of related works in VFL. The aspects compared are Model type, Time Series (TS), Inference privacy (IP.) and Optimization methods (optim.), that can be either iterative (it.) or direct (dir.).

Autoregression can also be utilized for tree-based models like XGBoost (Chen et al. 2015), leading to the formulation of autoregressive trees (ARTs) (Meek, Chickering, and Heckerman 2002). Extending VFL methods for XGBoost (Xie et al. 2022; Cheng et al. 2021; Fang et al. 2021) to ARTs can be done by transforming the datasets into design matrices using a polynomial like Equation 1. While ARTs do not use residual terms, they can model non-linear dependencies between AR and exogenous features.

SMPC

Secure Multi-Party Computation methods use the principle of secret sharing (Shamir 1979) for privacy by scattering a value into random shares among parties. With K parties, $C_{i\in[1,K]}$, if C_i wants to share data V with others, it generates $K-1$ random shares, denoted $\langle V\rangle^{i'}$ $\forall i' \in [1,K]; i' \neq i$. These are sent to the corresponding party $C_{i'}$. C_i's own share is computed as $\langle V\rangle^{i} = V - \sum_{i'\neq i}^{K}\langle V\rangle^{i'}$. The whole ensemble of K shares representing the shared state of V, is denoted as $\langle V\rangle$.

Parties cannot infer others' data from their shares alone; however, the value can be recovered by combining all shares. By analogy, when applied to VFL, local features and outputs are distributed into shares across parties to preserve their privacy. All parties jointly follow decentralized training protocols using the secretly shared data and end up with local models. Inference is made similarly by distributing features into secret shares and computing the prediction as a distributed share.

Current SMPC-based works in VFL exist for performing matrix operations (Han et al. 2009), gradient descent (Shi et al. 2022), and training XGBoost (Fang et al. 2021; Xie et al. 2022), making it convenient for training linear forecasters and ARTs that utilize these frameworks.

Related work on VFL

On the spectrum of models learned, research in VFL is diverse, including neural networks, trees, and regression models, as shown in studies (Khan, ten Thij, and Wilbik 2023; Liu et al. 2022; Wei et al. 2022). In this context, we direct our attention to the selected works presented in Table 1, as they collectively provide a comprehensive representation of research in VFL. We compare the methods on their model type, their applicability to time series forecasting, whether inference privacy can be easily achieved through decentralization, and their adaptability to alternative optimization choices.

Model type and time-series forecasting. Due to the industrial requirement for explainable models, we focus on linear/logistic regression (LR) (Hardy et al. 2017; Han et al. 2009; Shi et al. 2022), and tree-based models (Xie et al. 2022; Cheng et al. 2021), with transparent structures that are considered inherently interpretable (Samek and Müller 2019; Zhang et al. 2021). Yan et al. (Yan et al. 2022) employ a modification of the split learning architecture using Gated Recurrent Units (GRUs) with a shared upper model for predictions. But its large parameter complexity limits interpretability. Nevertheless, to the best of our knowledge, this remains the only other method for time-series forecasting with VFL. While STV does not include NNs, it adheres to industrial requirements for explainability.

Inference privacy. While privacy during training is implicit in all selected works, we consider schemes adopting SS as potential candidates for private inference requirements, as these can be seamlessly integrated with a serverless/decentralized approach, akin to ours (Xie et al. 2022; Shi et al. 2022; Han et al. 2009). HE-based schemes for logistic regression, such as Hardy et al. (Hardy et al. 2017), assume that predictions are decrypted on a particular party before sending them to the party requesting inference, which is not privacy-preserving. Additionally, Yan et al. (Yan et al. 2022) use a shared upper model in their split architecture, rendering the predictions accessible to all parties, compromising inference privacy.

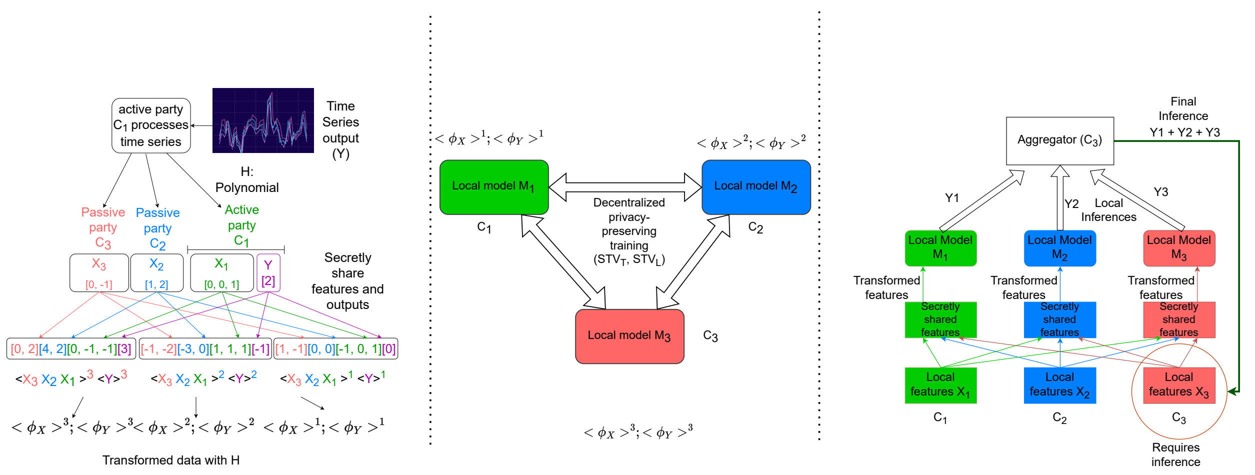
Optimization. When it comes to optimization techniques, all selected works, except for Han et al. (Han et al. 2009), employ solely iterative approaches. Notably, Han et al. (Han et al. 2009) offer iterative and matrix-based methods for direct optimization using Equation 3. This requires computing matrix multiplications and inverses, which are only implemented for the two-party case. We extend both operations to a generic N-party setup.

STV Framework

This section introduces the adversarial model and problem statement, followed by the general design of STV and the detailed implementation for linear and tree forecasters.

Adversarial Models We assume all parties are *honest-but-curious/semi-honest* (Hardy et al. 2017; Yang et al. 2019), i.e., they adhere to protocol but will try to infer knowledge of other parties' private data using their own local data and whatever is communicated to them. Also, it is assumed that parties do not collude, a standard assumption in VFL as all parties are incentivized to collaborate due to their mutual dependence on one another for training and inference (Yang et al. 2019; Hardy et al. 2017). In addition, we also assume that communication between parties is encrypted.

Problem Statement We assume a setup with K parties, C_1 to C_K, grouped into two types: $active$ and $passive$. The active party, C_1, owns the true values of the time series output, $Y(t)$, and exogenous features, $X_1(t)$, for timestep, t. The passive parties have exogenous features, $X_i, \forall i \in$



(a) Preprocessing, Secret sharing, and transformation

(b) Training

(c) First-hand inference with requesting party as aggregator

Figure 2: STV framework

$[2, K]$. The common samples between parties are assumed to be already identified using privacy-preserving entity alignment approaches (Hardy et al. 2017; Scannapieco et al. 2007). Our goal is to forecast future values using exogenous and autoregressive features without sharing them with others in plaintext.

We further assume there is a *coordinator*, a trusted third party that oversees the training process and is responsible for generating randomness, such as Beaver's triples for element-wise multiplication with SMPC (Beaver 1992; Fang et al. 2021). The coordinator cannot access private data and intermediate results, so it does not pose a security threat, as mentioned in Xie et al. (Xie et al. 2022).

Protocol Overview

An overview of STV is provided in Algorithm 1. The working of the individual steps of STV, with pre-processing, training, and inference, is illustrated in Figure 2.

Training. To start, the active party initiates by pre-processing the output and determines parameters like the auto-correlations and partial auto-correlations of the series (line 2). These are then used to generate a polynomial for SARIMAX or ARTs (line 3). Features and outputs are then secretly shared and transformed into lagged design matrices (line 7), like Equation 2. All parties then follow decentralized Algorithm 2 or Algorithm 3 to train a distributed model.

Privacy-preserving inference. During inference, the final prediction exists as a distributed share (line 14), which is aggregated on the requesting party (line 16). Since only the party making the inference acquires the aggregate value of the secret shares, the final prediction is with that party alone.

$\mathrm{STV_T}$

By transforming the original datasets, X and Y, into time-lagged design matrices, the training framework in Xie et al. (Xie et al. 2022) can be used. The modification requires the transformed matrices to be passed as inputs rather than the secretly shared X, Y datasets. The details are provided in Algorithm 2.

Training proceeds iteratively, finally resulting in the generation of T trees on each party. At each step, every party learns a new tree and makes a local prediction (lines 4-5). The tree-building function, ***SecureFit***, from the framework in Xie et al. (Xie et al. 2022), uses secret sharing primitives to train XGBoost by computing the first and second-order gradients. Details on this are in Appendix D.

During training, individual predictions are aggregated on the active party (C_1) since gradients are computed by C_1 (see Appendix D). For inference, aggregation can be performed on any party since gradients are not calculated, thus enabling private inferencing.

$\mathrm{STV_L}$

With $\mathrm{STV_L}$, the objective is optimizing the coefficients, A, in Equation 2, which can be done directly or iteratively using two-step regression. The first step uses only the autoregressive and exogenous terms to estimate the residuals. These are then substituted for the moving average terms in the second step to optimize all coefficients jointly. Details are provided in Algorithm 3.

Several steps in Algorithm 3 require matrix operations like multiplications (lines 8-9; 13-19) and inverse (line 14) on secretly shared data. Han et al. (Han et al. 2009) provide algorithms only for the two-party case; we extend to multiple parties.

N-party matrix multiplication. To extend matrix multiplication to multiple parties (algorithm 4), we view the computation of every output element $W_{i,j}$ as a scalar product of row and column vectors (line 3), which can be implemented using the N-party element-wise product using Beaver's triples (see Appendix C).

N-party matrix inverse. For matrix inverses (Algorithm

Algorithm 1: General protocol STV

Data: X_k on party C_k $\forall k \in [1, K]$, and Y on C_1
Accepted Parameters: Task: *Training/Inference*, Model *type*, number of trees T, Optimization method O, learning rate α, iterations, e, Trained distributed $Model$, Requesting party C_j
Output:Trained model distributed across K parties or final inference result on party C_j

```
1: if $Training$ and active party $C_1$ then
2:     params = ProcessSeries(Y)
3:     H = GenPoly (params, type)
4:     Broadcast $H$ to $C_i$ $\forall i \in [1, K]$
5: end if
6: Share local features $\langle X_k \rangle$ and (or) outputs $\langle Y \rangle$
7: $\langle \phi_X \rangle$,$\langle \phi_Y \rangle$ = TransformData (H){ Equation 2}
8: if type == Tree and Training then
9:     return $Model$ = $STV_T$ ($\langle \phi_X \rangle$, $\langle Y \rangle$,T)
10: else if type == Linear and Training then
11:     return $Model$ = $STV_L$ ($\langle \phi_X \rangle$,$\langle \phi_Y \rangle$, O, $\alpha$, e)
12: end if
13: if Inference then
14:     $\langle Result \rangle$ = $Model.Predict(\langle \phi_X \rangle)$
15:     if requesting party $C_j$ then
16:         $Result = \sum_{k=1}^{K} \langle Result \rangle^k$ {Aggregate predictions}
17:     end if
18: end if
```

5), we compute the inverse of a secretly-shared matrix, U, using a non-singular perturbation matrix, P, generated by the active party (line 2). Subsequently, the aggregation of product UP on a passive party does not leak U as the party does not know P (line 5). $(UP)^{-1}$ can then be computed locally and secretly shared, followed by a matrix multiplication with P, i.e., $P \times (UP)^{-1} = U^{-1}$, giving the result as a secret share (lines 7-10).

Experiments

We evaluate the forecasting accuracy of STV against centralized approaches. We also compare the scalability of iterative and direct optimization for linear forecasters using the total communication cost.

Algorithm 2: $\mathrm{STV_T}$

Data: Secretly shared transformed matrices $\langle \phi_X \rangle$, $\langle \phi_Y \rangle$
Parameter: number of trees T
Output: Distributed autoregressive XGBoost tree

```
1: Initialize predictions $\langle \hat{\phi_Y} \rangle^k$= 0 on all parties $C_k$
2: Initialize $Trees_k$ = [ ] on all parties $C_k$
3: for $t \in [1, T]$ do
4:     $tree_{t_k}$ = SecureFit($\langle \phi_X \rangle^k$,$\langle \phi_Y \rangle^k$,$\langle \hat{\phi_Y} \rangle^k$)
5:     $\langle \hat{\phi_{Y_t}} \rangle^k$ = $tree_{t_k}$.Predict($\langle \phi_X \rangle^k$)
6:     if active party $C_1$ then
7:         $\hat{\phi_{Y_t}} = \sum_{i=1}^{K} \langle \hat{\phi_{Y_t}} \rangle^k$ {Aggregate predictions}
8:         $\hat{\phi_Y} = \hat{\phi_Y} + \hat{\phi_{Y_t}}$ {Add to final predictions}
9:     end if
10:    $Trees_k.append(tree_{t_k})$ on every party $C_k$
11: end for
12: return $Trees_k$ on party $C_k$
```

Algorithm 3: $\mathrm{STV_L}$

Data: Secretly shared transformed matrices $\langle \phi_X \rangle$, $\langle \phi_Y \rangle$
Accepted Parameters: O, α, e
Output: Shared optimized coefficients $\langle A \rangle$

```
1: for step $\in [1, 2]$ do
2:     if step $== 1$ then
3:         Initialize residuals to zero in $\langle \phi_X \rangle^k$ for all $C_k$
4:     end if
5:     if $O ==$ "iterative" then
6:         Randomly initialize $\langle A \rangle^k$ for all $C_k$
7:         for $e$ iterations do
8:             Get $\langle \hat{\phi_Y} \rangle$ = $\langle \phi_X \rangle \times \langle A \rangle$ using Alg. 4
9:             $\langle \frac{dl}{dA} \rangle$ = 2$\times$($\langle \phi_X \rangle$)$^T$$\times$($\langle \hat{\phi_Y} \rangle$ - $\langle \phi_Y \rangle$) (Alg. 4)
10:            Perform update: $\langle A \rangle$ := $\langle A \rangle$ - $\alpha \langle \frac{dl}{dA} \rangle$
11:        end for
12:    else if $O ==$ "direct" then
13:        $\langle Z \rangle$ = ($\langle \phi_X \rangle$)$^T$$\times$$\langle \phi_X \rangle$ using Alg. 4
14:        $\langle W \rangle$ = $\langle Z^{-1} \rangle$ using Alg. 5
15:        $\langle V \rangle$ = ($\langle \phi_X \rangle$)$^T$$\times$$\langle \phi_Y \rangle$ using Alg. 4
16:        $\langle A \rangle$ = $\langle W \rangle$ $\times$ $\langle V \rangle$
17:    end if
18:    if step $== 1$ then
19:        Predict: $\langle \hat{\phi_Y} \rangle$ = $\langle \phi_X \rangle$ $\times$ $\langle A \rangle$ using Alg. 4
20:        Estimate residuals $\langle \varepsilon \rangle$ = $\langle \phi_Y \rangle$ - $\langle \hat{\phi_Y} \rangle$
21:        Set $\langle \phi_X \rangle^k$ using residuals from $\langle \varepsilon \rangle^k$ for all $C_k$
22:    end if
23: end for
24: return $\langle A \rangle^k$ on all parties $C_k$
```

Forecasting accuracy

We compare the performance of $\mathrm{STV_L}$ (direct optimization) and $\mathrm{STV_T}$ with other centralized methods: Long-Short-Term Memory (LSTM), SARIMAX with MLE[2], and diffusion models for forecasting (Alcaraz and Strodthoff 2022) (*SSSD*^{S4}).

The LSTM has two layers of 64 LSTM units each, followed by two dense layers of size 32 and 1. We train the model for 500 epochs and use a batch size 32 with a default learning rate of 0.001. The diffusion configuration has the

Algorithm 4: Secure Matrix Multiplication

Data: Secretly shared matrices $\langle U \rangle$ and $\langle V \rangle$ across K parties, $C_1, C_2, .., C_K$
Output: $\langle W \rangle$, i.e., product $W = U \times V$ as shares across K parties

```
1: for each row index $i$ do
2:     for each column index $j$ do
3:         $\langle \vec{T} \rangle$ = $\langle U[\vec{i}, :] \rangle$ * $\langle V[\vec{:}, j] \rangle$ {Element-wise product}
           $\langle W_{i,j} \rangle^k$ = sum{$\langle \vec{T} \rangle^k$}
4:     end for
5: end for
6: return $\langle W \rangle^k$ on party $C_k$
```

[2]https://www.statsmodels.org/devel/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html

Algorithm 5: Secure Matrix Inverse

Data: Secretly shared matrix $\langle U\rangle$ across K parties, $C_1, C_2, .., C_K$
Output: $\langle V\rangle$, i.e., inverse, $V = U^{-1}$, as a distributed share

```
1:  if active party ($P_1$) then
2:      Generate random non-singular perturbation matrix $P$ and
        secretly share as $\langle P\rangle$
3:  end if
4:  Get $\langle Q\rangle$ = $\langle U\rangle \times \langle P\rangle$ using Alg. 4
5:  Aggregate $Q = \sum_{k=1}^{K} \langle Q\rangle^k$ on passive party $C_j; j > 1$
6:  if Passive party $C_j$ then
7:      Compute $R = Q^{-1} = (UP)^{-1} = P^{-1}U^{-1}$
8:      Generate shares $\langle R\rangle$
9:  end if
10: Compute $\langle T\rangle$ = $\langle U^{-1}\rangle$ = $\langle P\rangle \times \langle R\rangle$ using Alg. 4
11: return $\langle T\rangle^k$ on party $C_k$
```

following settings[3]: $T = 200, \beta_0 = 0.0001, \beta_T = 0.02$. For the wavenet configuration, we use two residual layers, four residual and skip channels, with three diffusion embedding layers of dimensions 8×16. Training is done for 4000 iterations with a learning rate of 0.002.

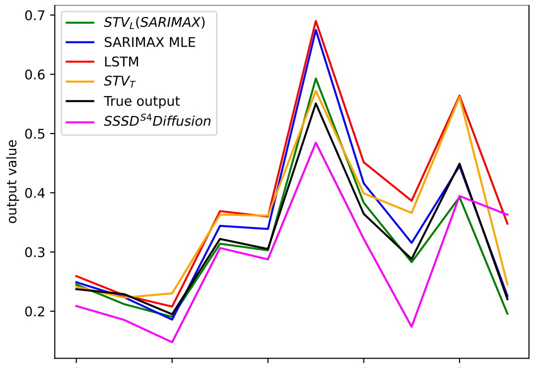
Five public forecasting datasets are used: Airline passengers (Kothari 2018), Air quality data (Vito 2016), PV Power (Kannal 2020), SML 2010 (Romeu-Guallart and Zamora-Martinez 2014), and Rossman Sales (ros 2015). An industry-specific dataset to estimate a performance parameter from inline sensor values in semiconductor manufacturing is also included (Gkorou et al. 2017; Gkorou et al. 2020). Additional details on the datasets, evaluation, and pre-processing are given in Appendix A and Appendix B.

A variant of prequential window testing (Cerqueira, Torgo, and Mozetič 2020) is used since industrial time series data can significantly change after intervals due to machine changes/repairs. The dataset is partitioned into multiple windows of a given size, each further divided into an 80-20 train-test split. After forecasting the test split, the model is retrained on the next window.

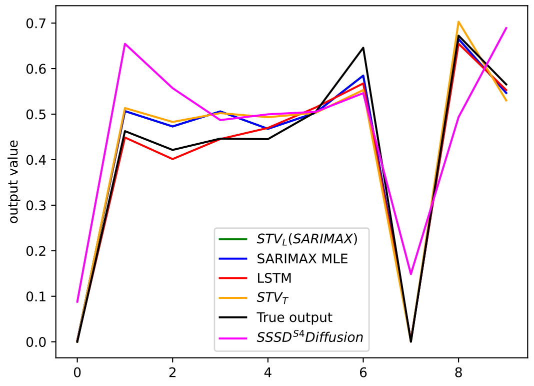
For a consistent comparison, all features and outputs are scaled between 0 and 1. We thus present the normalized mean-squared errors (n-MSE) of the predictions. Ground truths on the test set are measured and averaged across multiple windows. We average the n-MSE scores across different window sizes to generalize performance scores across varying forecasting ranges, the results of which are in Table 2.

Table 2 shows that $\mathtt{STV_L}$ is usually the best-performing method. We use direct optimization since iterative gradient descent eventually converges to the same value in the long run. Due to its guaranteed convergence, direct optimization improves against centralized methods by up to 23.81%. Though it does worse than SARIMAX MLE on the passenger data, it is still among the top two methods.

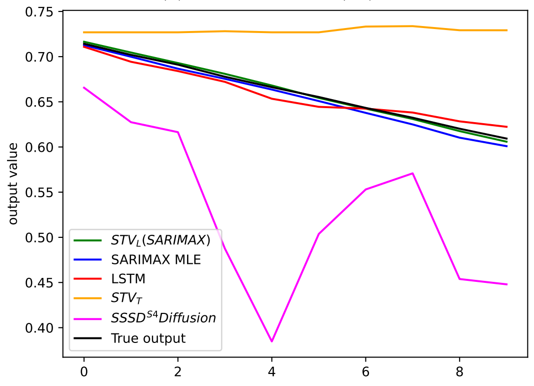
Furthermore, we also show three regression plots from a prequential window of size 50 from three datasets in Figure 3. In general, all the methods can capture patterns in the time series, like in Figure 3a and Figure 3b. However,



(a) Airquality (50)



(b) Rossman Sales (50)



(c) SML 2010 (50)

Figure 3: Forecasts from one of the windows of total size 50 (40 train, 10 test) from three datasets.

highly complex models such as *SSSD*^{S4} may occasionally overfit, showing the drawback of NNs when dealing with small training windows (see Figure 3c).

Scalability

We measure the total communication costs of direct optimization using the normal equation (NE) and iterative batch gradient descent (GD) to analyze the scalability of the two methods under increasing parties, features, and samples.

We vary the parties between 2, 4, and 8; features between

[3]Using https://github.com/AI4HealthUOL/SSSD.git

Dataset	STV_L (SARIMAX, VFL)	STV_T (ART, VFL)	*SARIMAX* (MLE, Centralized)	*LSTM* (Centralized)	*SSSD*^{S4} (Centralized)	Rel. imp. (%)
Airquality	**0.00069** (0.00008)	0.00100 (0.00056)	0.00088 (0.00031)	0.00114 (0.00076)	0.00150 (0.00073)	21.59
Airline passengers	0.00304 (0.00086)	0.00808 (0.00379)	**0.00222** (0.00148)	0.04130 (0.04086)	0.00392 (0.00226)	-36.94
PV Power	**0.00138** (0.00136)	0.00249 (0.00254)	0.00159 (0.00095)	0.14333 (0.23033)	0.00167 (0.00058)	15.22
SML 2010	**0.00787** (0.00711)	0.01875 (0.01458)	0.01033 (0.01061)	0.01716 (0.01086)	0.01085 (0.00871)	23.81
Rossman Sales	**0.00074** (0.00012)	0.00243 (0.00153)	0.00077 (0.00006)	0.00639 (0.00280)	0.00331 (0.00151)	3.89
Industrial data	**0.00602** (0.00049)	0.04118 (0.04051)	0.00969 (0.00449)	0.00617 (0.00203)	0.01875 (0.00313)	2.43

Table 2: Average normalized MSE and (standard deviation) results for different datasets and methods. Relative improvement of the best VFL method with the best centralized one is also shown (rel. imp). Lowest MSE values are highlighted in bold. SML 2010, Air quality, and Rossman Sales have prequential window sizes 50, 100, 200, and 400. PV Power uses 25, 50, 100, and 200. Airline passengers uses 60, 80, 100, 120, and 140. Finally, the industrial data uses 25, 50, and 100.

10 and 100; and samples between 10, 100, and 1000. Since communication costs depend only on the dataset dimensions and the number of parties, we generate random data matrices for all valid combinations of features and samples on each party, i.e., #features $\leq$ #samples. We then optimize the coefficients using either the direct approach (Algorithm 3, lines (13-19)), or batch gradient descent (lines (6-10)), for a different number of iterations (10, 100, 1000). For party-scaling, we average the total communication costs across various feature and sample combinations for a given number of parties. Similarly, we measure feature and sample scaling by averaging the total costs across different (sample, party) and (feature, party) combinations respectively. Results are shown in Table 3, Table 4, and Table 5.

parties	NE	GD iterations		
		10	100	1000
2	2.54E+08	2.33E+07	2.33E+08	2.33E+09
4	5.85E+08	4.65E+07	4.65E+08	4.65E+09
8	1.48E+09	9.31E+07	9.31E+08	9.31E+09

Table 3: Average of total communication sizes (bytes) with varying parties.

Features	NE	GD iterations		
		10	100	1000
10	9.77E+06	8.34E+06	8.34E+07	8.34E+08
100	1.92E+09	1.23E+08	1.23E+09	1.23E+10

Table 4: Average of total communication sizes (bytes) with varying feature counts.

Samples	NE	GD iterations		
		10	100	1000
10	1.16E+06	2.76E+05	2.76E+06	2.76E+07
100	4.53E+08	1.24E+07	1.24E+08	1.24E+09
1000	1.48E+09	1.23E+08	1.23E+09	1.23E+10

Table 5: Average of total communication sizes (bytes) with varying samples.

What we see from these three tables is that when the number of parties/samples/features is small, direct optimization's cost is comparable to an iterative version with a larger number of iterations. For example, when the number of parties is 2, we see that the cost of NE is close to GD with 100 iterations in Table 3. However, when we increase to 8 parties, the cost of NE is greater than GD with 100 iterations.

Similarly, in Table 4 and Table 5, we see that NE has a lower cost than GD with 100 iterations for a small number of features and samples. This is no longer the case upon increasing the features and samples.

However, the cost of GD increases proportionally with the iterations, and at some point, it becomes more expensive than NE, as we see with 1000 iterations. In practice, hyperparameters like the learning rate affect the steps required for convergence, which may be hard to tune in a distributed setup. So choosing between iterative or direct optimization depends on several factors, requiring adaptability in frameworks.

Conclusion

This work presents a novel privacy-preserving forecasting framework for VFL using SS and SMPC, tackling challenges the manufacturing industry faces. A key part of our work is distributing predictions' ownership among parties, addressing requirements for inference privacy, which challenges conventional VFL assumptions that predictions are always available to a specific party. Our results show that VFL methods are competitive against centralized methods, and scalability analyses bring out the nuanced dynamics between iterative and direct optimization, highlighting the need for an adaptable framework.

For the future, several paths of exploration beckon. First, the models used in this study are poor at capturing the patterns present in long-range time series data, for which LSTMs and diffusion models are better suited. We aim to enable these models for VFL. Second, current VFL cases assume a static set of participants. Exploring a hybrid combination of horizontal FL and VFL elements by independently training clusters of VFL models, which are aggregated later, can enhance their collective forecasting power. Lastly, fortifying our VFL framework against adversarial attacks is essential. While our current work assumes semi-honest participants, malicious adversaries cannot be ignored.

References

[Abadi et al. 2016] Abadi, M.; Chu, A.; Goodfellow, I.; McMahan, H. B.; Mironov, I.; Talwar, K.; and Zhang, L. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 308–318.

[Alcaraz and Strodthoff 2022] Alcaraz, J. M. L., and Strodthoff, N. 2022. Diffusion-based time series imputation and forecasting with structured state space models. *arXiv preprint arXiv:2208.09399*.

[Beaver 1992] Beaver, D. 1992. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology—CRYPTO'91: Proceedings 11*, 420–432. Springer.

[Blais and others 2010] Blais, J., et al. 2010. Least squares for practitioners. *Mathematical Problems in Engineering* 2010.

[Cerqueira, Torgo, and Mozetič 2020] Cerqueira, V.; Torgo, L.; and Mozetič, I. 2020. Evaluating time series forecasting models: An empirical study on performance estimation methods. *Machine Learning* 109:1997–2028.

[Chen et al. 2015] Chen, T.; He, T.; Benesty, M.; Khotilovich, V.; Tang, Y.; Cho, H.; Chen, K.; Mitchell, R.; Cano, I.; Zhou, T.; et al. 2015. Xgboost: extreme gradient boosting. *R package version 0.4-2* 1(4):1–4.

[Chen et al. 2020] Chen, T.; Jin, X.; Sun, Y.; and Yin, W. 2020. Vafl: a method of vertical asynchronous federated learning. *arXiv preprint arXiv:2007.06081*.

[Cheng et al. 2021] Cheng, K.; Fan, T.; Jin, Y.; Liu, Y.; Chen, T.; Papadopoulos, D.; and Yang, Q. 2021. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems* 36(6):87–98.

[Cramer, Damgård, and others 2015] Cramer, R.; Damgård, I. B.; et al. 2015. *Secure multiparty computation*. Cambridge University Press.

[Fang et al. 2021] Fang, W.; Zhao, D.; Tan, J.; Chen, C.; Yu, C.; Wang, L.; Wang, L.; Zhou, J.; and Zhang, B. 2021. Large-scale secure xgb for vertical federated learning. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 443–452.

[Fu et al. 2022] Fu, F.; Xue, H.; Cheng, Y.; Tao, Y.; and Cui, B. 2022. Blindfl: Vertical federated machine learning without peeking into your data. In *Proceedings of the 2022 International Conference on Management of Data*, 1316–1330.

[Geyer, Klein, and Nabi 2017] Geyer, R. C.; Klein, T.; and Nabi, M. 2017. Differentially private federated learning: A client level perspective. *CoRR* abs/1712.07557.

[Gkorou et al. 2017] Gkorou, D.; Ypma, A.; Tsirogiannis, G.; Giollo, M.; Sonntag, D.; Vinken, G.; van Haren, R.; van Wijk, R. J.; Nije, J.; and Hoogenboom, T. 2017. Towards big data visualization for monitoring and diagnostics of high volume semiconductor manufacturing. In *Proceedings of the Computing Frontiers Conference*, CF'17, 338–342. New York, NY, USA: Association for Computing Machinery.

[Gkorou et al. 2020] Gkorou, D.; Larranaga, M.; Ypma, A.; Hasibi, F.; and van Wijk, R. J. 2020. Get a human-in-the-loop: Feature engineering via interactive visualizations.

[Hamilton 2020] Hamilton, J. D. 2020. *Time series analysis*. Princeton university press.

[Han et al. 2009] Han, S.; Ng, W. K.; Wan, L.; and Lee, V. C. 2009. Privacy-preserving gradient-descent methods. *IEEE transactions on knowledge and data engineering* 22(6):884–899.

[Hannan and Kavalieris 1984] Hannan, E. J., and Kavalieris, L. 1984. A method for autoregressive-moving average estimation. *Biometrika* 71(2):273–280.

[Hardy et al. 2017] Hardy, S.; Henecka, W.; Ivey-Law, H.; Nock, R.; Patrini, G.; Smith, G.; and Thorne, B. 2017. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*.

[Kannal 2020] Kannal, A. 2020. Solar power generation.

[Khan, ten Thij, and Wilbik 2023] Khan, A.; ten Thij, M.; and Wilbik, A. 2023. Vertical federated learning: A structured literature review.

[Konečný, McMahan, and Ramage 2015] Konečný, J.; McMahan, B.; and Ramage, D. 2015. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*.

[Korstanje 2021] Korstanje, J. 2021. *The SARIMAX Model*. Berkeley, CA: Apress. 125–131.

[Kothari 2018] Kothari, C. 2018. Airline passengers.

[Li et al. 2021] Li, L.; Damarla, S. K.; Wang, Y.; and Huang, B. 2021. A gaussian mixture model based virtual sample generation approach for small datasets in industrial processes. *Information Sciences* 581:262–277.

[Lin and Karr 2010] Lin, X., and Karr, A. F. 2010. Privacy-preserving maximum likelihood estimation for distributed data. *Journal of Privacy and Confidentiality* 1(2).

[Lin et al. 2019] Lin, C.-Y.; Hsieh, Y.-M.; Cheng, F.-T.; Huang, H.-C.; and Adnan, M. 2019. Time series prediction algorithm for intelligent predictive maintenance. *IEEE Robotics and Automation Letters* 4(3):2807–2814.

[Lindell 2005] Lindell, Y. 2005. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*. IGI global. 1005–1009.

[Liu et al. 2016] Liu, C.; Hoi, S. C.; Zhao, P.; and Sun, J. 2016. Online arima algorithms for time series prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

[Liu et al. 2022] Liu, Y.; Kang, Y.; Zou, T.; Pu, Y.; He, Y.; Ye, X.; Ouyang, Y.; Zhang, Y.-Q.; and Yang, Q. 2022. Vertical federated learning.

[Lütkepohl 2007] Lütkepohl, H. 2007. General-to-specific or specific-to-general modelling? an opinion on current econometric terminology. *Journal of Econometrics* 136(1):319–324.

[Meek, Chickering, and Heckerman 2002] Meek, C.; Chickering, D. M.; and Heckerman, D. 2002. Autoregressive tree models for time-series analysis. In *Proceedings of the 2002 SIAM International Conference on Data Mining*, 229–244. SIAM.

[Montgomery, Jennings, and Kulahci 2015] Montgomery, D. C.; Jennings, C. L.; and Kulahci, M. 2015. *Introduction to time series analysis and forecasting*. John Wiley & Sons.

[Perktold 2023] Perktold, J. 2023. Sarimax.

[Romeu-Guallart and Zamora-Martinez 2014] Romeu-Guallart, P., and Zamora-Martinez, F. 2014. SML2010. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5RS3S.

[ros 2015] 2015. Rossman store sales.

[Samek and Müller 2019] Samek, W., and Müller, K.-R. 2019. Towards explainable artificial intelligence. *Explainable AI: interpreting, explaining and visualizing deep learning* 5–22.

[Sarwate, Chaudhuri, and Monteleoni 2009] Sarwate, A. D.; Chaudhuri, K.; and Monteleoni, C. 2009. Differentially private support vector machines. *CoRR* abs/0912.0071.

[Scannapieco et al. 2007] Scannapieco, M.; Figotin, I.; Bertino, E.; and Elmagarmid, A. K. 2007. Privacy preserving schema and data matching. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 653–664.

[Shamir 1979] Shamir, A. 1979. How to share a secret communications of the acm, 22.

[Shi et al. 2022] Shi, H.; Jiang, Y.; Yu, H.; Xu, Y.; and Cui, L. 2022. Mvfls: multi-participant vertical federated learning based on secret sharing. *The Federate Learning* 1–9.

[Snoke et al. 2017] Snoke, J.; Brick, T. R.; Slavkovic, A.; and Hunter, M. D. 2017. Providing accurate models across private partitioned data: Secure maximum likelihood estimation.

[Susto and Beghi 2016] Susto, G. A., and Beghi, A. 2016. Dealing with time-series data in predictive maintenance problems. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–4. IEEE.

[Tarsitano and Amerise 2017] Tarsitano, A., and Amerise, I. L. 2017. Short-term load forecasting using a two-stage sarimax model. *Energy* 133:108–114.

[Vepakomma et al. 2018] Vepakomma, P.; Gupta, O.; Swedish, T.; and Raskar, R. 2018. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*.

[Vito 2016] Vito, S. 2016. Air Quality. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C59K5F.

[Vollert, Atzmueller, and Theissler 2021] Vollert, S.; Atzmueller, M.; and Theissler, A. 2021. Interpretable machine learning: A brief survey from the predictive maintenance perspective. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 01–08.

[Wang et al. 2022] Wang, J.; Li, Y.; Gao, R. X.; and Zhang, F. 2022. Hybrid physics-based and data-driven models for smart manufacturing: Modelling, simulation, and explainability. *Journal of Manufacturing Systems* 63:381–391.

[Wei et al. 2022] Wei, K.; Li, J.; Ma, C.; Ding, M.; Wei, S.; Wu, F.; Chen, G.; and Ranbaduge, T. 2022. Vertical federated learning: Challenges, methodologies and experiments.

[Xie et al. 2022] Xie, L.; Liu, J.; Lu, S.; Chang, T.-H.; and Shi, Q. 2022. An efficient learning framework for federated xgboost using secret sharing and distributed optimization. *ACM Transactions on Intelligent Systems and Technology (TIST)* 13(5):1–28.

[Yan et al. 2022] Yan, Y.; Yang, G.; Gao, Y.; Zang, C.; Chen, J.; and Wang, Q. 2022. Multi-participant vertical federated learning based time series prediction. In *Proceedings of the 8th International Conference on Computing and Artificial Intelligence*, 165–171.

[Yang et al. 2019] Yang, Q.; Liu, Y.; Chen, T.; and Tong, Y. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10(2):1–19.

[Zhang et al. 2021] Zhang, Y.; Tiňo, P.; Leonardis, A.; and Tang, K. 2021. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence* 5(5):726–742.

[Zhu et al. 2023] Zhu, Q.-X.; Zhang, H.-T.; Tian, Y.; Zhang, N.; Xu, Y.; and He, Y.-L. 2023. Co-training based virtual sample generation for solving the small sample size problem in process industry. *ISA transactions* 134:290–301.

Appendix A: Datasets and Pre-processing

We make use of the following public datasets in our work: Air Quality (Vito 2016), SML 2010 (Romeu-Guallart and Zamora-Martinez 2014), PV Power (Kannal 2020), Airline Passengers (Kothari 2018), and Rossman Sales (ros 2015). We do not provide the details on the industrial dataset due to confidentiality agreements. Here we provide a brief description of each dataset and the pre-processing steps applied to the dataset itself. These steps are unconnected with the series pre-processing steps which are part of the `STV` framework. For all datasets, we scale all features and output values between the range 0 to 1 using a MinMax scaler to ensure that all datasets have the same range of values for comparing the MSE losses.

Air Quality

The Air Quality dataset contains approximately 9300 samples of multivariate time-series data, with 15 attributes. Five of these are true output values on five gases: Carbon Monoxide (CO), Non Metanic Hydrocarbons (NMHC)), Benzene (C6H6)), Total Nitrogen Oxides (NOx), and Nitrogen Dioxide (NO2). Exogenous features such as the temperature, ozone levels, and humidity are provided, along with strongly correlated sensor data for each of the five gases. The dataset contains missing values and duplicates, and contains hourly data for each of the five gases.

We preprocess the data by discarding all rows with any missing information and remove duplicate rows. We predict the ground truth values of CO using the other sensor values and information such as temperature and humidity as the exogenous regressors.

SML 2010

The SML 2010 dataset contains infomation from a monitor system in a domotic house. It contains approximately 4100 samples with 24 attributes in total, corresponding to 40 days of monitoring data. The attributes contain values such as the indoor and outdoor temperature, lighting levels, Carbon Dioxide levels, relative humidity, rain, windspeed, etc. We predict the indoor habitation temperature using the others as exogenous features.

Airline Passengers

Airline Passengers is a small dataset of 145 samples containing the number of international airline passengers (in thousands) on a monthly basis. The exogenous features are also just two: the year and the month. We predict the number of passengers using the year and month as exogenous regressors.

PV Power

The PV Power dataset contains around 3100 samples of solar power generation data from each of two power plants over a 34-day period. Attributes include features such as the DC power, AC power, yield, ambient temperature, irradiation levels, and the data and time.

We drop identifiers, empty, and duplicate data. We also drop the DC power attribute, and total yield as these features are very strongly correlated with the AC output. As outputs, we predict the AC power generation using the remaining features as exogenous regressors.

Rossman Sales

The Rossman Sales dataset contains sales data for 1115 store outlets. The attributes consists of features such as holidays, store type, competitor distance, number of customers, and promotional details among others. We predict the sales of the store with ID 1, using the other features as exogenous regressors.

Appendix B: Experimental Evaluation

As mentioned in the main text, we use a variation of prequential window testing (Cerqueira, Torgo, and Mozetič 2020), whereby the entire data is broken into windows of a defined length, each one internally split in an 80-20 train-test ratio. This is illustrated in Figure 4.

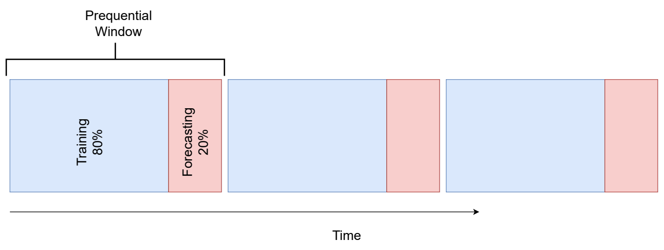


Figure 4: Prequential window evaluation with re-training in every window

For each window, we train on the portion allotted for training and forecast the remaining. Within a given training window, we first generate the polynomial by processing the time series as in line 3 of Algorithm 1. Identifying the parameters for generating the polynomial can be automated using implementations such as auto arima[4].

For each window size, such as 50, 100, 200, 400, we average the MSE loss between the forecasts and the true values across all windows. The average MSE per-window size is given in Table 6, which is an expanded version of Table 2.

Appendix C: SMPC primitives

We elaborate on the computation details for basic arithmetic operations, such as addition, subtraction, and multiplication, using the SMPC framework below.

1. ***Addition and subtraction***: If X and Y exist as secret shares, $\langle X\rangle$ and $\langle Y\rangle$, each party performs a local addition or subtraction, i.e., $\langle Z\rangle^k = \langle X\rangle^k +(-) \langle Y\rangle^k$. To obtain $Z = X + (-)Y$, the shares are aggregated: $Z = \sum_k^K \langle Z\rangle^k$.

[4]https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html

Dataset	Window size	$\texttt{STV}_\texttt{L}$	SARIMAX MLE	LSTM	$\texttt{STV}_\texttt{T}$	$SSSD^{S4}$
Air quality	50	0.00071	0.00110	0.00244	0.00198	0.00270
	100	0.00059	0.00048	0.00055	0.00075	0.00086
	200	0.00066	0.00124	0.00069	0.00067	0.00100
	400	0.00080	0.00068	0.00087	0.00061	0.00144
	Avg.	**0.00069**	0.00088	0.00114	0.00100	0.00150
SML 2010	50	0.00645	0.00621	0.00343	0.00755	0.01373
	100	0.01958	0.02819	0.02778	0.04305	0.02384
	200	0.00500	0.00662	0.02781	0.01709	0.00336
	400	0.00045	0.00030	0.00960	0.00729	0.00249
	Avg.	**0.00787**	0.01033	0.01716	0.01875	0.01085
Rossman Sales	50	0.00070	0.00079	0.00464	0.00183	0.00589
	100	0.00064	0.00071	0.00340	0.00496	0.00281
	200	0.00068	0.00086	0.00674	0.00088	0.00233
	400	0.00095	0.00072	0.01079	0.00206	0.00220
	Avg.	**0.00074**	0.00077	0.00639	0.00243	0.00331
PV Power	25	0.00133	0.00093	0.00950	0.00660	0.00131
	50	0.00360	0.00307	0.01037	0.00258	0.00143
	100	0.00004	0.00063	0.01115	0.00010	0.00267
	200	0.00054	0.00175	0.54227	0.00068	0.00128
	Avg.	**0.00138**	0.00159	0.14333	0.00249	0.00167
Airline Passengers	60	0.00261	0.00113	0.11651	0.00673	0.00110
	80	0.00450	0.00444	0.00339	0.00628	0.00403
	100	0.00308	0.00066	0.01982	0.00270	0.00798
	120	0.00316	0.00136	0.05164	0.01134	0.00356
	140	0.00185	0.00351	0.01512	0.01332	0.00293
	Avg.	0.00304	**0.00222**	0.04130	0.00808	0.00392

Table 6: Average normalized MSE values for different public datasets, with different prequential window sizes

Knowing the value of $\langle Z\rangle^k$ makes it impossible to infer the private values X or Y, as each participant only owns a share of the whole secret. Moreover, the individual values of the shares, $\langle X\rangle^k$ and $\langle Y\rangle^k$, are also masked by adding them.

2. ***Multiplication (using Beaver's triples)*** (Beaver 1992; Xie et al. 2022): To get $Z = X * Y$, where $*$ denotes element-wise multiplication, and X and Y, are secretly shared, the coordinator first generates three numbers a, b, c such that $c = a * b$. These are then secretly shared, i.e., C_k, receives $\langle a\rangle^k$, $\langle b\rangle^k$, and $\langle c\rangle^k$. C_k computes $\langle e\rangle^k$ = $\langle X\rangle^k$ - $\langle a\rangle^k$ and $\langle f\rangle^k$ = $\langle Y\rangle^k$ - $\langle b\rangle^k$, and sends it to C_1. C_1 then aggregates these shares to recover e and f and broadcasts them to all parties. C_1 then computes $\langle Z\rangle^1$ = $e * f$+f*$\langle a\rangle^1$+ e*$\langle b\rangle^1$+$\langle c\rangle^1$, and the others calculate $\langle Z\rangle^k$ = f*$\langle a\rangle^k$+e*$\langle b\rangle^k$+$\langle c\rangle^k$. It is easy to see that aggregation of the individual shares gives the product Z.

   Despite knowing e and f, the actual values of X and Y are hidden because none of the parties knows the values of a, b, and c. Also, like in the case with addition, the individual shares, $\langle Z\rangle^k$, do not reveal anything about the local share values, i.e., $\langle X\rangle^k$, $\langle Y\rangle^k$, $\langle a\rangle^k$, $\langle b\rangle^k$, and $\langle c\rangle^k$.

Additional primitives such as division, argmax, and sigmoid can also be computed using secret sharing, the details of which are provided in Fang et al. (Fang et al. 2021) and Xie et al. (Xie et al. 2022).

Appendix D: SMPC-based XGBoost with VFL

XGBoost (Chen et al. 2015) is a tree-based gradient-boosting algorithm, that iteratively generates an ensemble of trees by greedily learning a new tree at every step to improve on the earlier one. Each tree has weights assigned to its leaf nodes. When making a prediction for a sample, the weights corresponding to the leaf to which the sample was assigned to are summed to give the final prediction score.

To generate a tree at every iteration, it uses first and second order gradients of the latest predictions, i.e., from the previous tree, in order to set the optimal weights for the new one.

For each sample with index i, the first and second order gradients are denoted as g_i and h_i, respectively. The sum of the gradients of all instances on a particular node is used to set the new weights for it. For example, for node j and corresponding instance set I_j, the accumulated values of g and h are computed as follows: $G_j = \sum_{i\in I_j} g_i$, and $H_j = \sum_{i\in I_j} h_i$. Based on this, for a tree with T nodes, the weights and objective are calculated as follows:

$$w_j = -G_j/(H_j + \lambda) \tag{4}$$

$$obj = -0.5 \times \sum_{j=1}^{T}((G_j)^2/(H_j + \lambda)) + \gamma T \tag{5}$$

, where γ, λ are regularizers. While Equation 4 sets the new

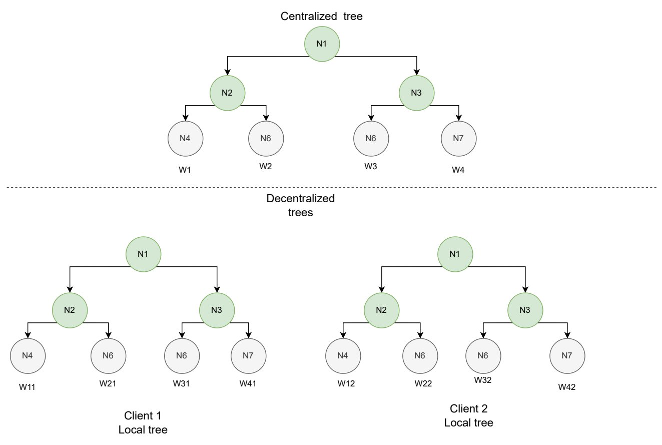


Figure 5: Centralized vs Secretly shared XGBoost trees

weights, Equation 5 is used to identify how to split nodes at each iteration.

With this in mind, using the secret sharing primitives it is possible to compute these functions to extend XGBoost to VFL, which we show in Algorithm 6.

When XGBoost is trained for VFL using secret sharing, each client obtains a local tree with their own weights as shown in Figure 5. In the figure, the weights for clients 1 and 2 are distributed such that their local weights are shares of the weights of the centralized version, i.e., $wi = wi1 + wi2$ $\forall i \in [1, 4]$

The indicator vector s on line 1 of algorithm 6, is a binary vector that is used to point out the location of instances on nodes. We explain this with the help of the example in Figure 6. To calculate the updated weight of the node with instances that have an age greater than 30 (bottom left), we need to find the sum of $\sum_{i \in I_j} g_i$, where $I_j = \{2, 4\}$(Equation 4). The indicator vector in this case would be $s = [0, 1, 0, 1]$, meaning that nodes 2 and 4 are part of node j. If we have a vector of the gradients, $g = [g1, g2, g3, g4]$, we can compute $g2 + g4$ as $s \odot g$, i.e., the inner product. Under VFL, both the gradients g, and the indicator vector s, exist as secret shares across clients, i.e., $s = \sum_k^K \langle s_k \rangle^k$, and $g = \sum_k^K \langle g_k \rangle^k$. Therefore, to compute the product, we can do it using secret shared primitives for matrix multiplication.

The process of split-finding and setting weights is done using the ***SecureBuild*** function, which makes use of secret sharing primitives to compute the functions in Equation 5 and Equation 4. We defer readers to the implementation in Xie et al. (Xie et al. 2022) for additional details on this.

Appendix E: Two-step regression

Consider the example from the main text:

$$\underbrace{\begin{bmatrix} Y(3) \\ Y(4) \\ .. \\ Y(t) \end{bmatrix}}_{\phi_Y} = \underbrace{\begin{bmatrix} Y(2) & Y(1) & \varepsilon(2) & X_1(3) & X_2(3) \\ Y(3) & Y(2) & \varepsilon(3) & X_1(2) & X_2(2) \\ .. & .. & .. & .. & .. \\ Y(t-1) & Y(t-2) & \varepsilon(t-1) & X_1(t) & X_2(t) \end{bmatrix}}_{\phi_X} \times \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \gamma_1 \\ \gamma_2 \end{bmatrix}}_{A} + \underbrace{\begin{bmatrix} \varepsilon(3) \\ \varepsilon(4) \\ .. \\ \varepsilon(t) \end{bmatrix}}_{\varepsilon} \tag{6}$$

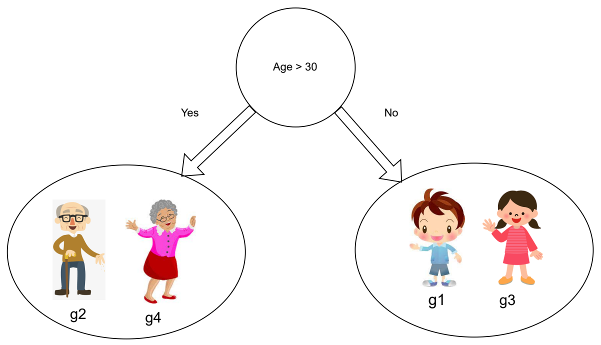


Figure 6: Mapped instances to a node in XGBoost

Algorithm 6: Fit XGBoost (Xie et al. 2022)

Data: Secretly shared matrices: $\langle X \rangle$, $\langle Y \rangle$, $\langle \hat{Y} \rangle$ across K clients, $C_1, C_2, .., C_K$
Output: Learned tree for the current iteration, one on each client.

```
1: Initialize indicator vector $s \leftarrow 1$ on all $C_k$
2: **if** active client $C_1$ **then**
3:     Compute derivatives $g, h$
4:     Generate shares $\langle g \rangle$, $\langle h \rangle$, $\langle s \rangle$
5: **end if**
6: $Tree_k$ = ***SecureBuild***($\langle g \rangle^k$,$\langle h \rangle^k$, $\langle s \rangle^k$) on each $C_k$
7: **return** $Tree_k$ on $C_k$
```

The application of the two steps to this is shown below.

1. **First step:** As the residual terms in ϕ_X are unknown, they are initialized to zero to give $\hat{\phi_X}$. The estimated coefficients for this step are denoted as $\hat{A}$:

$$\underbrace{\begin{bmatrix} Y(3) \\ Y(4) \\ .. \\ Y(t) \end{bmatrix}}_{\phi_Y} = \underbrace{\begin{bmatrix} Y(2) & Y(1) & 0 & X_1(3) & X_2(3) \\ Y(3) & Y(2) & 0 & X_1(2) & X_2(2) \\ .. & .. & .. & .. & .. \\ Y(t-1) & Y(t-2) & 0 & X_1(t) & X_2(t) \end{bmatrix}}_{\hat{\phi_X}} \times \underbrace{\begin{bmatrix} \hat{\alpha_1} \\ \hat{\alpha_2} \\ \hat{\beta_1} \\ \hat{\gamma_1} \\ \hat{\gamma_2} \end{bmatrix}}_{\hat{A}} + \underbrace{\begin{bmatrix} \varepsilon(3) \\ \varepsilon(4) \\ .. \\ \varepsilon(t) \end{bmatrix}}_{\varepsilon} \tag{7}$$

$\hat{A}$ is optimized using the normal equation or gradient descent. The residuals, ε, are then estimated as:

$$\varepsilon = \phi_Y - \hat{\phi_X} \times \hat{A} \tag{8}$$

These are then re-substituted within ϕ_X before the second optimization. Residual terms that are still unavailable, like $\varepsilon(2)$, can be set to zero:

$$\tilde{\phi_X} = \begin{bmatrix} Y(2) & Y(1) & 0 & X_1(3) & X_2(3) \\ Y(3) & Y(2) & \varepsilon(3) & X_1(2) & X_2(2) \\ .. & .. & .. & .. & .. \\ Y(t-1) & Y(t-2) & \varepsilon(t-1) & X_1(t) & X_2(t) \end{bmatrix} \tag{9}$$

2. **Second step:** All coefficients in A can be jointly optimized using the normal equation or an iterative approach by replacing ϕ_X in Equation 7 with $\tilde{\phi_X}$.

With iterative methods like gradient descent, the update rule for optimizing A is shown in Equation 10 under mean-squared-error (MSE) loss:

$$A := A - 2\alpha \times (\phi_X)^T \times (\hat{\phi_Y} - \phi_Y) \text{ (for } e \text{ iterations)} \quad (10)$$

; where $\hat{\phi_Y}$ are the predictions at a particular step and α is the learning rate.

For direct optimization, the normal equation can be used as shown below: Equation 11.

$$A = ((\phi_X)^T \phi_X)^{-1} ((\phi_X)^T \phi_Y) \quad (11)$$

Enabling the training of linear forecasting models with VFL requires the design matrices, ϕ_X and ϕ_Y to exist as secret shares. Following this, SMPC versions of the operations in Equation 11 and Equation 10 can be applied to optimize the parameters.

Chapter 2

Introduction

In this chapter, we approach the problem from the viewpoint of its application in manufacturing, followed by the challenges in existing methodologies and our contributions.

2.1 Problem Scenario

In today's manufacturing landscape, time series forecasting has emerged as a critical factor in various applications, particularly within machine control, predictive maintenance, and diagnostics [29, 46]. An illustration of this significance can be found in the scenario depicted in Figure 2.1, involving two primary manufacturers and their shared customer[1]. The manufacturers collect sensory data during production. In contrast, the customer owns post-production performance data, serving as the outputs or labels. Predicting outputs from sensor values would allow the manufacturers to make pre-shipment corrections, saving valuable time and helping to calibrate equipment in a complex, high-volume production factory.

The relevance of this scenario reflects in semiconductor manufacturing processes such as photolithography [48], where performance measurements can be predicted from inline sensor data [17]. The conventional approach to developing models combines the input features and outputs into a centralized dataset, upon which models are trained. However, this approach confronts a significant hurdle regarding scenarios involving such feature/vertically-partitioned datasets, as business confidentiality agreements prevent the involved parties from sharing their features and outputs in plaintext. This challenge pivots around the delicate domain of privacy-preserving machine learning, necessitating the need for strategies to circumvent the data-sharing problem.

2.2 Existing Solution Framework

A promising solution that has emerged to address privacy concerns within the context of distributed machine learning is federated learning [25]. This approach operates under a distinctive model-to-data paradigm, where data remains securely confined to each party's

[1] A secondary manufacturer who is the "customer" from the perspective of the other two.

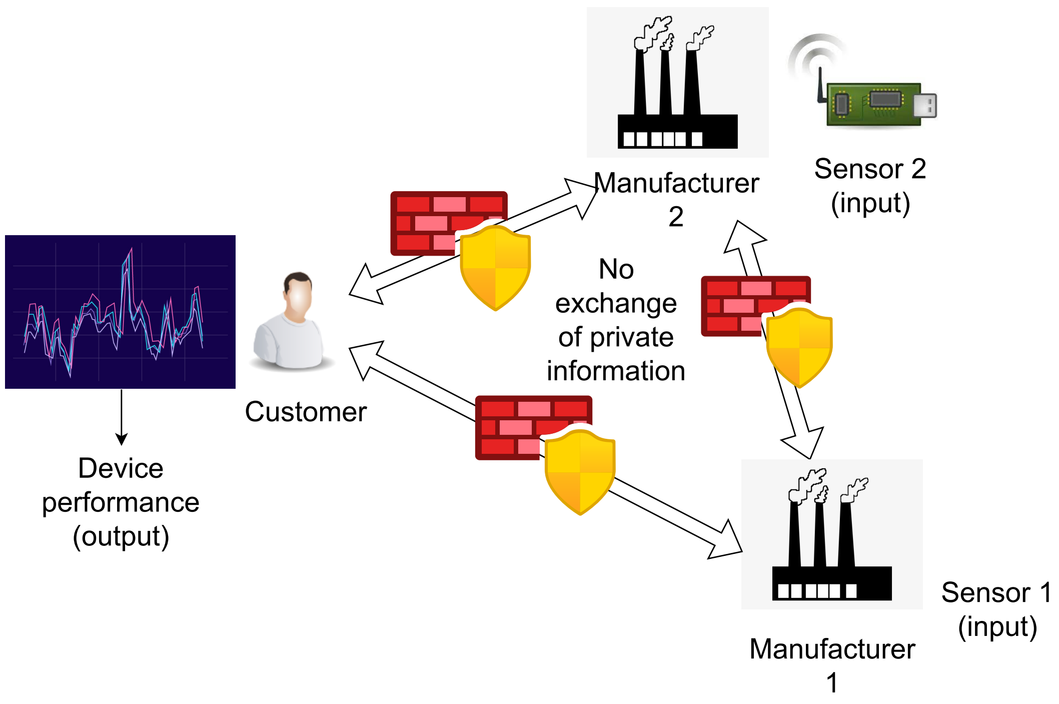


Figure 2.1: Problem scenario—forecasting device performance needs inputs from multiple parties who want to protect the confidentiality of the data.

premises. Every party undertakes the task of training a localized model using their private data. Subsequently, knowledge across these models is combined via decentralized or distributed protocols while ensuring data privacy. This culminates in creating models capable of generalizing across the diverse datasets of multiple parties.

Central to federated learning are the two ways in which datasets on each party can be organized: horizontally or vertically partitioned datasets. In the case of horizontal partitioning, every party's local dataset has an identical set of attributes, though the specific samples might vary, as depicted in Figure 2.2. In contrast, vertical partitioning entails splitting a dataset's features or columns across different parties (see Figure 2.3). All parties retain access to a shared attribute, such as a sample ID or timestamp which serves as a cohesive link, allowing the disparate samples across parties to be effectively interconnected.

Both modes of partitioning require a different modeling approach within federated learning. In our problem context, the sensor measurements and outputs can be viewed as a vertically partitioned dataset, which falls within the realm of *Vertical Federated Learning* (VFL) [56, 8, 24, 53, 33].

2.3 Challenges

Despite its relevance, time-series forecasting with VFL has received limited scholarly attention. This underscores the critical need for further exploration and research in this domain,

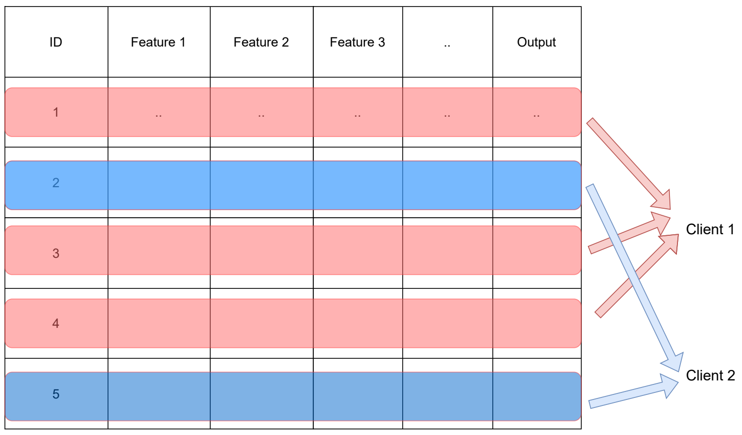


Figure 2.2: Horizontally partitioned data

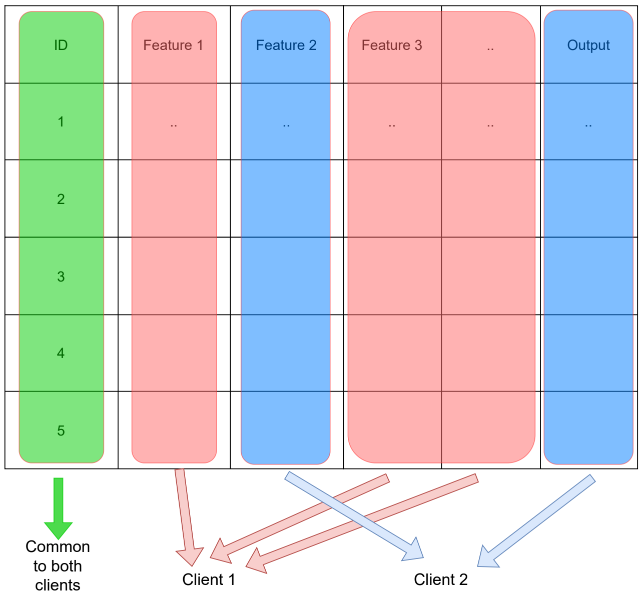


Figure 2.3: Vertically partitioned data

especially considering the following complexities introduced by manufacturing scenarios.

1. **Small and noisy datasets** Manufacturing data can be slow and resource-intensive to gather, resulting in small datasets, causing complex models like neural networks (NNs) to overfit. Moreover, the complex parameter interactions of NNs limit their interpretability, which is an important factor in manufacturing applications, to understand the reasoning behind predictions for optimizing production quality and downtime [52, 51].

   Datasets can also be noisy since they might originate from noisy sensors. This makes modeling even more challenging [28, 17, 58], and may not be appropriate with privacy measures like differential privacy [2, 11, 12, 13], which would involve adding noise to already noisy data. Differential privacy also introduces a trade-off between privacy and performance, necessitating careful consideration [41, 16, 2, 56].

2. **Training and inference privacy requirements.** On top of the default assumption of privacy during training, parties in VFL also need to perform inference privately, i.e., obtain first-hand predictions, given the competitive business environments in manufacturing. For the example shown in Figure 2.1, if manufacturer 1 wants to make a prediction, the customer (True output/label holder) should not be able to observe these predictions since the manufacturer would use them for their private use such as optimizing the factory pipeline. However, many existing works in VFL assume that predictions are obtained on one party (usually the label owner), which may be different from the party requiring inference. For instance, in split learning architectures [49], where an NN is divided into a bottom and top half, the predictions are yielded on the party owning the top half [8, 55, 15]. Similarly, homomorphic encryption (HE) schemes [22, 9] may require the predictions to be first decrypted by a specific party holding the decryption key, such as a coordinator/output holder.

   Since the manufacturers would like to obtain the predictions first-hand without having another party observe them first, there is a need to make the inferencing *serverless*, i.e., ownership of the predictions is not with a particular party like the output holder, but is distributed across multiple parties. Here, we make it a point to distinguish between the *predictions*, which are obtained during inference, and the *true* outputs/labels. While the latter is still the output holder's property, it does not include ownership of the predictions.

3. **Navigating optimization tradeoffs.** Parameter optimization for models can be achieved using direct or iterative methods. While direct optimization methods like the normal equation [5] reach globally optimal solutions without requiring any hyperparameter tuning, they are not scalable to large problem instances. On the other hand, iterative methods like gradient descent scale better but require additional tuning, such as setting the learning rate or batch size, which may be inconvenient in a distributed setting. However, industries require both scalable and convenient solutions with little tuning, as this would save time and money. Therefore, navigating the trade-off between these factors requires a solution adaptable to both optimization approaches.

2.4 Our Contributions

Towards achieving the goal of a forecasting framework for VFL while addressing industrial challenges and requirements, we develop a novel framework, ***S**ecret-Shared **T**ime Series Forecasting with **V**ertical Federated Learning*, (`STV`), with the following features.

1. **VFL forecasting framework.** `STV` extends centralized forecasting models such as autoregressive trees (ARTs) [35], and classical linear forecasters such as SARIMAX and ARIMAX [26, 38, 19] to VFL. These models are less prone to overfitting on small datasets due to their low parameter complexity. They also have transparent structures that are inherently interpretable [57, 40], which adheres to the industrial requirement for model explainability. Within `STV`, ARTs and SARIMAX are trained using the $\texttt{STV}_{\texttt{T}}$ and $\texttt{STV}_{\texttt{L}}$ protocols, respectively.

2. **Privacy-preserving training and inference.** `STV` is built on the foundations of *Secure Multi-Party Computation* (SMPC) [10, 31] and *Secret Sharing* (SS) [43], which are cryptographic methods offering strong privacy guarantees. With secret sharing, privacy is preserved by distributing sensitive data into random shares across multiple parties. All computations are performed on local shares, causing the final result to also exist as a distributed share. These shares of the result can be collected/aggregated to yield the final prediction on the requesting party first-hand, which preserves inference privacy. SMPC and SS also have the additional advantage of their computations being lossless, which overcomes the drawback of differential privacy.

3. **Adaptable optimization.** Our linear forecasting algorithm, $\texttt{STV}_{\texttt{L}}$, is designed with a two-step approach that harnesses the power of least squares (LS) for optimizing the parameters of linear forecasting models. LS optimization can be done using both iterative and direct methods, adding adaptability to our algorithm. However, enabling direct optimization within $\texttt{STV}_{\texttt{L}}$ requires a fundamental expansion: extending the existing two-party schemes for matrix operations to accommodate multiple parties.

We evaluate `STV` on multiple fronts. First, we compare the forecasting accuracy of $\texttt{STV}_{\texttt{L}}$ and $\texttt{STV}_{\texttt{T}}$ with centralized forecasters like state-of-the-art diffusion models [3], Long Short Term Memory (LSTM), and SARIMAX with Maximum Likelihood Estimation (MLE) [19]. Second, we compare the communication costs of iterative and direct methods for optimizing linear forecasters under different scaling scenarios, highlighting their trade-offs. We experiment using a wide range of datasets: five public datasets (Air quality, flight passengers, SML 2010, PV Power, Rossman Sales) and one real dataset from semiconductor manufacturing to validate our findings.

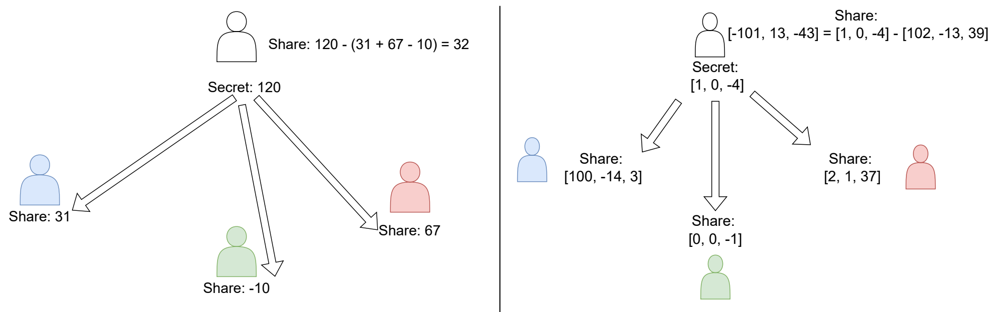
Chapter 3

Background and Related Work

In this chapter, we provide background knowledge on Secret Sharing (SS) and Secure Multi-Party Computation (SMPC), forecasting models, and VFL. Then, we highlight the shortcomings of existing related works in VFL.

3.1 Secret Sharing and Secure Multi-Party Computation

Secret sharing [43] serves as a crucial privacy-enhancing technique by dispersing a private value into randomized shares distributed among parties, as depicted in Figure 3.1. The original value can only be reconstructed through aggregating these individual shares, achieved by summing the contributions from all the parties. This concept forms a foundational component within Secure Multi-Party Computation (SMPC) methodologies [10, 31], which operate on these distributed data shares. As a result of this distribution, the computed result also persists in the form of a secret share, maintaining confidentiality. Furthermore, these operations exhibit consistency, ensuring the final aggregated result aligns with what would have been obtained through centralized computation. Although basic mathematical operations, such as simple arithmetic, can be executed on distributed shares, it is important to note that tackling more complex or non-linear functions remains a challenging endeavor within this framework.



(a) Sharing a scalar (b) Sharing a vector

Figure 3.1: Secret sharing principle

Formally, in a setup involving K parties, denoted as $C_{i\in[1,K]}$, when a party C_i intends to share data V with the other parties, it generates $K-1$ random shares, denoted as $\langle V\rangle^{i'}$ for all $i' \in [1,K]$ where $i' \neq i$. These shares are then transmitted to their respective parties, i.e., $C_{i'}$. C_i computes its own share as $\langle V\rangle^{i}$ = V - $\sum_{i'\neq i}^{K}\langle V\rangle^{i'}$. This process results in an ensemble of K shares, collectively representing the shared state of V, denoted as $\langle V\rangle$.

We elaborate on the computation details for basic arithmetic operations, such as addition, subtraction, and multiplication, using the SMPC framework below.

1. ***Addition and subtraction***: If X and Y exist as secret shares, $\langle X\rangle$ and $\langle Y\rangle$, each party performs a local addition or subtraction, i.e., $\langle Z\rangle^k$ = $\langle X\rangle^k$ $+(-)$ $\langle Y\rangle^k$. To obtain $Z = X + (-)Y$, the shares are aggregated: $Z = \sum_k^K \langle Z\rangle^k$.

   Knowing the value of $\langle Z\rangle^k$ makes it impossible to infer the private values X or Y, as each participant only owns a share of the whole secret. Moreover, the individual values of the shares, $\langle X\rangle^k$ and $\langle Y\rangle^k$, are also masked by adding them.

2. ***Multiplication (using Beaver's triples)*** [4, 54]: To get $Z = X * Y$, where $*$ denotes element-wise multiplication, and X and Y are secretly shared, the coordinator first generates three numbers a,b,c such that $c = a*b$. These are then secretly shared, i.e., C_k, receives $\langle a\rangle^k$, $\langle b\rangle^k$, and $\langle c\rangle^k$. C_k computes $\langle e\rangle^k$ = $\langle X\rangle^k$ - $\langle a\rangle^k$ and $\langle f\rangle^k$ = $\langle Y\rangle^k$ - $\langle b\rangle^k$, and sends it to C_1. C_1 then aggregates these shares to recover e and f and broadcasts them to all parties. C_1 then computes $\langle Z\rangle^1$ = $e*f$+$f*\langle a\rangle^1$+ $e*\langle b\rangle^1$+$\langle c\rangle^1$, and the others calculate $\langle Z\rangle^k$ = $f*\langle a\rangle^k$+$e*\langle b\rangle^k$+$\langle c\rangle^k$. It is easy to see that aggregation of the individual shares gives the product Z.

   Despite knowing e and f, the actual values of X and Y are hidden because none of the parties knows the values of a,b, and c. Also, like in the case with addition, the individual shares, $\langle Z\rangle^k$, do not reveal anything about the local share values, i.e., $\langle X\rangle^k$, $\langle Y\rangle^k$, $\langle a\rangle^k$, $\langle b\rangle^k$, and $\langle c\rangle^k$.

Additional primitives such as division, argmax, and sigmoid can also be computed using secret sharing, the details of which are provided in Fang et al. [14] and Xie et al. [54].

SMPC and SS can also be used for training and inferencing with VFL models due to their strong privacy guarantees. For training, all parties' local features or outputs are secretly shared to preserve their privacy. All parties then jointly follow decentralized training protocols using the secretly shared data and end up with local models. Inference is made similarly by distributing features into secret shares and computing the prediction as a distributed share.

As mentioned earlier, due to the limited mathematical operations under SMPC, applying it to complex models such as neural networks is challenging because of their non-linear operations such as sigmoid or ReLU. However, for models such as XGBoost trees [7], and linear regressors, these methods can still be applied as shown in several VFL works [20, 44, 54, 14].

3.2 Linear Forecasting Models

Linear forecasters fall under the autoregressive family of models, which use historical values of the output variables to predict the future [19]. In addition to historical data, these models may incorporate other factors to enhance their predictive capabilities, such as exogenous features (external variables that impact the target variable) and past errors (residuals).

The most general of these forecasting models is **S**easonal **A**uto**R**egressive **I**ntegrated **M**oving **A**verage with e**X**ogenous variables [26, 38], a.k.a SARIMAX. By incorporating seasonality, it generalizes other forecasters such as ARIMAX, ARMAX, and ARX [19, 36]. The outputs can be modeled by combining all the factors linearly. For example, the output, Y, at time t can be represented using a polynomial, H, in the following way using the past errors, $\varepsilon(t-i)$, past values $Y(t-i)$, and the currently observed exogenous features, $X_j(t)$:

$$\begin{aligned} H : Y(t) = \alpha_1 Y(t-1) + \alpha_2 Y(t-2) + \beta_1 \varepsilon(t-1) + \\ + \gamma_1 X_1(t) + \gamma_2 X_2(t) + \varepsilon(t) \end{aligned} \tag{3.1}$$

Estimation of the coefficients (α, β, γ) is done using methods like MLE [19] or LS [21, 34, 32, 47]. However, MLE with SMPC is limited to certain families of likelihood functions like the exponential or multivariate normal distributions [45, 30], since computing non-linear functions is challenging due to the limited mathematical operations with SMPC.

Under normally distributed errors, LS approaches can be used, where the original datasets are transformed into time-lagged design matrices, (ϕ_X, ϕ_Y), starting at $t = 1$, to represent Equation 3.1:

$$\underbrace{\begin{bmatrix} Y(3) \\ Y(4) \\ .. \\ Y(t) \end{bmatrix}}_{\phi_Y} = \underbrace{\begin{bmatrix} Y(2) & Y(1) & \varepsilon(2) & X_1(3) & X_2(3) \\ Y(3) & Y(2) & \varepsilon(3) & X_1(2) & X_2(2) \\ .. & .. & .. & .. & .. \\ Y(t-1) & Y(t-2) & \varepsilon(t-1) & X_1(t) & X_2(t) \end{bmatrix}}_{\phi_X} \times \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \gamma_1 \\ \gamma_2 \end{bmatrix}}_{A} + \underbrace{\begin{bmatrix} \varepsilon(3) \\ \varepsilon(4) \\ .. \\ \varepsilon(t) \end{bmatrix}}_{\varepsilon} \tag{3.2}$$

Optimizing A can be viewed as a linear regression problem if not for its circular dependency on the residuals. Specifically, the value of the residuals in ϕ_X, $\varepsilon(t-k)$ for some k, requires first knowing the value of A and vice-versa. A two-step approach is used to resolve this [47, 34, 21, 32]. First, the predictions are modeled using only the autoregressive and exogenous terms by substituting the residuals in ϕ_X to zero and then estimating them by subtracting the actual values, ϕ_Y, with the predictions. Finally, joint optimization of all the coefficients in A is done by setting the residuals in ϕ_X to the estimates. The application of these two steps is shown below.

1. **First step:** As the residual terms in ϕ_X are unknown, they are initialized to zero to give $\hat{\phi_X}$. The estimated coefficients for this step are denoted as $\hat{A}$:

$$\underbrace{\begin{bmatrix} Y(3) \\ Y(4) \\ .. \\ Y(t) \end{bmatrix}}_{\phi_Y} = \underbrace{\begin{bmatrix} Y(2) & Y(1) & 0 & X_1(3) & X_2(3) \\ Y(3) & Y(2) & 0 & X_1(2) & X_2(2) \\ .. & .. & .. & .. & .. \\ Y(t-1) & Y(t-2) & 0 & X_1(t) & X_2(t) \end{bmatrix}}_{\hat{\phi_X}} \times \underbrace{\begin{bmatrix} \hat{\alpha_1} \\ \hat{\alpha_2} \\ \hat{\beta_1} \\ \hat{\gamma_1} \\ \hat{\gamma_2} \end{bmatrix}}_{\hat{A}} + \underbrace{\begin{bmatrix} \varepsilon(3) \\ \varepsilon(4) \\ .. \\ \varepsilon(t) \end{bmatrix}}_{\varepsilon} \tag{3.3}$$

$\hat{A}$ is optimized using the normal equation or gradient descent. The residuals, ε, are then estimated as:

$$\varepsilon = \phi_Y - \hat{\phi_X} \times \hat{A} \tag{3.4}$$

These are then re-substituted within ϕ_X before the second optimization. Residual terms that are still unavailable, like $\varepsilon(2)$, can be set to zero:

$$\tilde{\phi_X} = \begin{bmatrix} Y(2) & Y(1) & 0 & X_1(3) & X_2(3) \\ Y(3) & Y(2) & \varepsilon(3) & X_1(2) & X_2(2) \\ .. & .. & .. & .. & .. \\ Y(t-1) & Y(t-2) & \varepsilon(t-1) & X_1(t) & X_2(t) \end{bmatrix} \tag{3.5}$$

2. **Second step:** All coefficients in A can be jointly optimized using the normal equation or an iterative approach by replacing ϕ_X in Equation 3.3 with $\tilde{\phi_X}$.

Viewing parameter estimation as a regression problem is a crucial step to enable the training of SARIMAX and its like under VFL. It also makes it adaptable to use both iterative and closed-form optimization approaches. With iterative methods like gradient descent, the update rule is shown in Equation 3.6 under mean-squared-error (MSE) loss:

$$A := A - 2\alpha \times (\phi_X)^T \times (\hat{\phi_Y} - \phi_Y) \text{ (for } e \text{ iterations)} \tag{3.6}$$

; where $\hat{\phi_Y}$ are the predictions at a particular step and α is the learning rate.

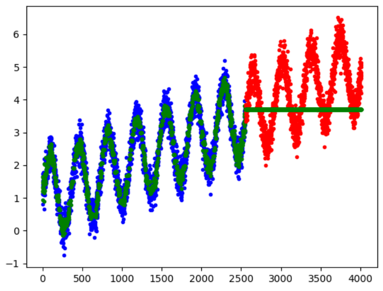
For direct optimization, the normal equation can be used as shown below: Equation 3.7.

$$A = ((\phi_X)^T \phi_X)^{-1}((\phi_X)^T \phi_Y) \tag{3.7}$$

Enabling the training of linear forecasting models with VFL requires the design matrices, ϕ_X and ϕ_Y to exist as secret shares. Following this, SMPC versions of the operations in Equation 3.7 and Equation 3.6 can be applied to optimize the parameters.

3.3 Autoregressive Trees (ARTs)

Standard gradient-boosted tree methods like XGBoost [7] are not catered to time-series data as they do not account for temporal lags and autocorrelations, which are critical for


(a) Standard XGBoost forecasts using only X value as an input.

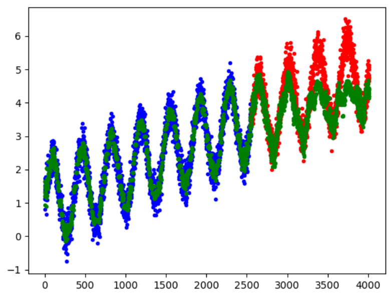

(b) ART using the both X value and lagged outputs as features.

Figure 3.2: Forecasts made by standard XGBoost and ARTs

forecasting. They lack memory of past observations, making it challenging to recognize the sequential patterns inherent in time series data.

In this context, ARTs [35] offer a promising solution by incorporating the concept of autoregression into the tree-based framework. By considering past observations as predictors, ARTs can capture temporal dependencies like in the case with linear forecasters.

As an illustrative example, we consider Figure 3.2, which shows the forecasts made by an XGBoost tree and an ART on generated time series data with a linear trend component. For both models, training is done on the samples from the blue region while forecasts are made on the red region. The green part shows the forecasts made by the trained models.

While both models are able to predict values quite accurately in the training region, we see that the standard XGBoost completely collapses when it is given inputs from outside its training domain. The autoregressive tree on the other hand shows a clear improvement on the forecasts because of regressing on the lagged output values.

Extending tree-based VFL methods [54, 9, 14] to ARTs can be done by transforming the datasets into design matrices using a polynomial like Equation 3.1. While ARTs do not use residual terms unlike SARIMAX, they can model non-linear dependencies between autoregressive and exogenous features.

3.4 Background on VFL

In this section, we provide a general overview of the steps involved when modeling in VFL. We also cover some popular model architectures as they provide more context on the related works.

3.4.1 General overview

The general procedure for modeling with VFL consists of two steps: entity alignment and model training (see Figure 3.3), which we explain as follows.

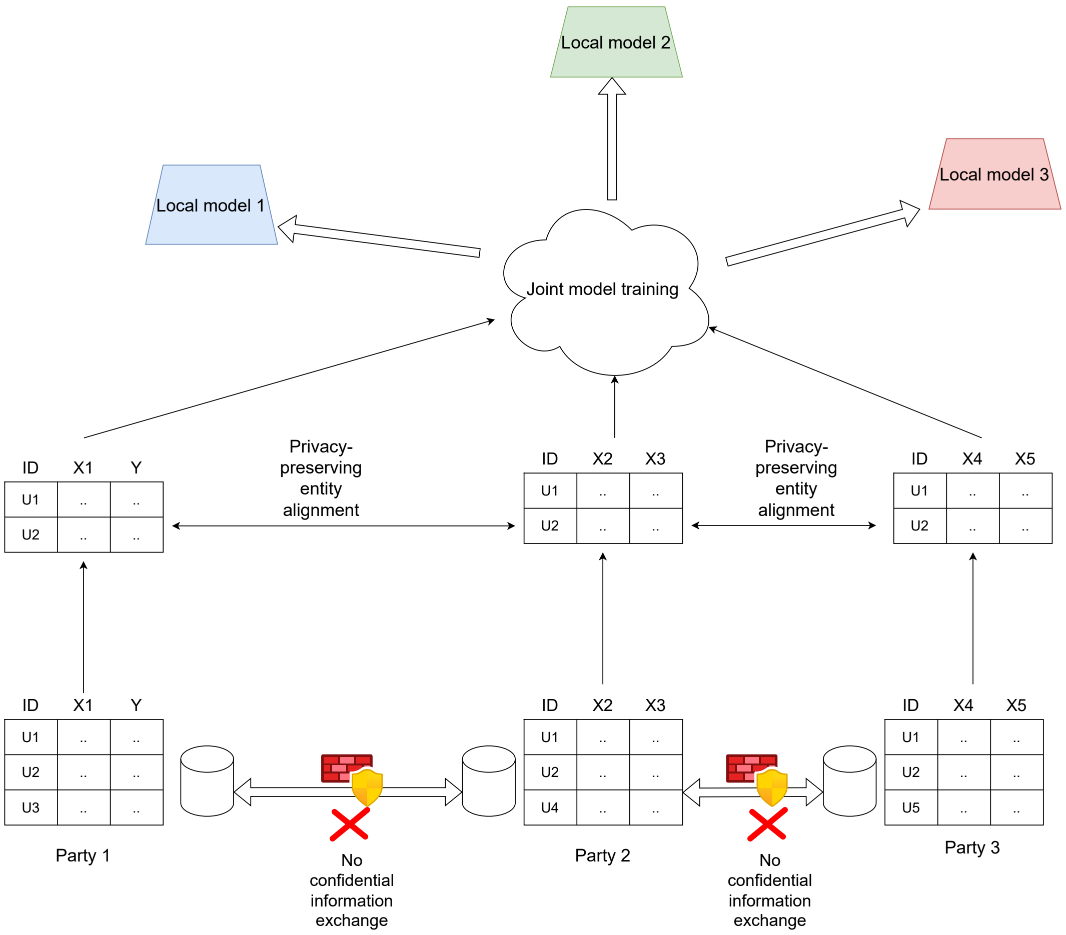
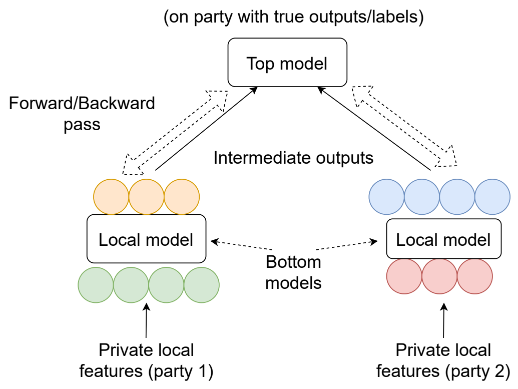


Figure 3.3: Overview of VFL. Entity alignment and joint model training

1. **Entity alignment:** In the context of Figure 2.3, it was noted that in vertical federated learning, each party possesses a shared column, often referred to as a sample ID, which can be used to establish links between samples across different parties. In a scenario without privacy restrictions, accomplishing this task would be straightforward. Parties could exchange their lists of IDs, allowing them to identify matching samples directly. However, this approach poses privacy concerns within the vertical federated learning setting, as the sample IDs are considered sensitive information. This is where privacy-preserving entity alignment approaches come into play, protecting the confidentiality of the shared IDs [42, 22]. One such approach involves encrypting the sample IDs from multiple parties and then performing the matching process at a trusted third party. This third party plays a key role by sending a mask to the individual parties, indicating the locations of the common samples without revealing the actual sample IDs. This mechanism ensures that the parties can align their data while maintaining the privacy of the sensitive information, as described in Hardy et al. [22].


(on party with true outputs/labels)
Top model
Forward/Backward pass
Intermediate outputs
Local model
Bottom models
Local model
Private local features (party 1)
Private local features (party 2)
Figure 3.4: Split neural network

2. **Privacy-preserving distributed training:** Once the common samples are identified, models are trained using different protocols. Three such approaches are split learning [49], homomorphic encryption based [22, 15, 9], and SMPC-based [14, 54, 44]

3.4.2 Modeling approaches in VFL

Here we provide a brief description of split learning, HE-based methods, and SMPC based methods.

1. **Split learning:** Such a scheme is popular with NN-based methods. As shown in Figure 3.4, the model is divided into two halves. Each party uses a local network to perform forward propagation using the private inputs (bottom half), and the intermediate outputs are sent to the party owning the top half. The rest of the forward propagation is performed on this party, and the gradients are sent to the individual parties at the split section so that the parties can update their local models. The same procedure follows during inference, and the predictions are obtained on the top half of the model. Privacy in such methods is typically achieved through differential privacy [11, 13], by adding noise to the intermediate values communicated.

2. **Homomorphic encryption-based solutions:** Such methods use homomorphic encryption schemes like Paillier encryption [37]. The advantage of such a scheme is that mathematical operations on encrypted data yield the same result in the encrypted space, as would be the case by performing the computation in plaintext and then encrypting it. For example, the sum of two numbers, $U+V$, in encrypted space can be written as $[\![U+V]\!] = [\![U]\!] + [\![V]\!]$, where $[\![.]\!]$ denotes the encrypted state of a value. Such operations extend to other mathematical operations as well, enabling parties to perform computations on data in the encrypted space without knowing their true plaintext values. This is valuable in VFL as computations like the loss and gradients can be computed on the label/output-holding party using encrypted intermediate computations received from the other parties. Such schemes are popular with linear/logistic regression methods [15, 22], or tree-based methods [9].

3. **SMPC-based solutions:** Like HE-based solutions, these methods perform computations using secretly shared data rather than encrypted values. Like with HE-schemes, SMPC methods exist for tree-based solutions [54, 14], and also for linear regressors [44, 20], both of which are within the scope of this thesis.

3.5 Related work on VFL

Method	Type			Time Series	Inference Privacy	Optimization	
	Trees	LR	NN			iterative	direct
Yan et al. [55]	✗	✗	✓	✓	✗	✓	✗
Hardy et al. [22]	✗	✓	✗	✗	✗	✓	✗
Han et al. [20]	✗	✓	✗	✗	✓	✓	✓
Cheng et al. [9]	✓	✗	✗	✗	✗	✓	✗
Xie et al.[54]	✓	✗	✗	✗	✓	✓	✗
Shi et al. [44]	✗	✓	✗	✗	✓	✓	✗
`STV` (this work)	✓	✓	✗	✓	✓	✓	✓

Table 3.1: Comparison of related works in VFL. The aspects compared are model type, compatability with time series data, inference privacy, and optimization methods offerred.

On the spectrum of models learned, research in VFL is diverse, including neural networks, trees, and regression models, as shown in studies [24, 33, 53]. In this context, we direct our attention to the selected works presented in Table 3.1, as they collectively provide a comprehensive representation of research in VFL. We compare the methods on their model type, their applicability to time series forecasting, whether inference privacy can be easily achieved through decentralization, and their adaptability to alternative optimization choices.

Model type and time-series forecasting. Due to the industrial requirement for explainable models, we focus on linear/logistic regression (LR) [22, 20, 44], and tree-based models [54, 9], with transparent structures that are considered inherently interpretable [40, 57]. Yan et al. [55] employ a modification of the split learning architecture using Gated Recurrent Units (GRUs) with a shared upper model for predictions. But its large parameter complexity limits interpretability. Nevertheless, to the best of our knowledge, this remains the only other method for time-series forecasting with VFL. While `STV` is not developed for neural networks, its use of lightweight forecasting models adheres to industrial requirements.

Inference privacy. While privacy during training is implicit in all selected works, we consider schemes adopting SS as potential candidates for private inference requirements, as these can be seamlessly integrated with a serverless/decentralized approach, akin to ours [54, 44, 20]. HE-based schemes for logistic regression, such as Hardy et al. [22], assume that predictions are decrypted on a particular party before sending them to the party requesting inference, which is not privacy-preserving. Additionally, Yan et al. [55] use a shared upper model in their split architecture, rendering the predictions accessible to all parties, compromising inference privacy.

Optimization. When it comes to optimization techniques, all selected works, except for Han et al. [20], employ solely iterative approaches. Notably, Han et al. [20] offer iterative and matrix-based methods for direct optimization using Equation 3.7. However, this requires computing matrix multiplications and inverses, which are only implemented for the two-party case in Han et al. [20]. We extend both operations to a generic N-party setup.

Chapter 4

Methodology

This section introduces the adversarial model and problem statement, followed by the general design of `STV` and the detailed implementation for linear and tree forecasters.

4.1 Adversarial Models

We assume that all parties are *honest-but-curious/semi-honest* [22, 56]. This means that they adhere to protocol, but will try to infer knowledge of other parties' private data using their own local data and whatever is communicated to them. Also, it is assumed that parties do not collude, a standard assumption in VFL as all parties involved are incentivized to collaborate. This is because of their mutual reliance on one another for training and inference [56, 22]. In addition, we also assume that communication between parties is encrypted.

4.2 Problem Statement

We assume a setup with K parties, C_1 to C_K, grouped into two types: *active* and *passive*. The active party, C_1, owns the true values of the time series output, $Y(t)$, and exogenous features, $X_1(t)$, for timestep t. The passive parties only have exogenous features, $X_i, \forall i \in [2, K]$. The common samples between parties are assumed to be already identified using privacy-preserving entity alignment approaches, as explained earlier in subsection 3.4.1. Our goal is to forecast future values of $Y(t)$ using exogenous and autoregressive features, without sharing them with others in plaintext.

We further assume there is a *coordinator*, a trusted third party that oversees the training process and is responsible for generating randomness, such as Beaver's triples for element-wise multiplication with SMPC [4, 14]. The coordinator cannot access private data and intermediate results, so it does not pose a security threat, as mentioned in Xie et al. [54].

4.3 Protocol Overview

An overview of `STV` is provided in Algorithm 1. A high level overview of the individual steps of `STV`, with pre-processing, training, and inference, is illustrated in Figure 4.1.

Training. To start, the active party initiates by pre-processing the output and determines parameters like the auto-correlations and partial auto-correlations of the series (line 2). These are then used to generate a polynomial for SARIMAX or ARTs (line 3). Features and outputs are then secretly shared and transformed into lagged design matrices (line 7), like in Equation 3.2. All parties then follow decentralized Algorithm 2 or Algorithm 3 to train a distributed model across all parties.

Inference privacy. During inference (lines 14 - 16), the final prediction exists as a distributed share across all parties (line 14), which is aggregated on the requesting party (line 16). Since only the party making the inference acquires the aggregate value of the secret shares, the final prediction is with that party alone.

Algorithm 1 General protocol `STV`

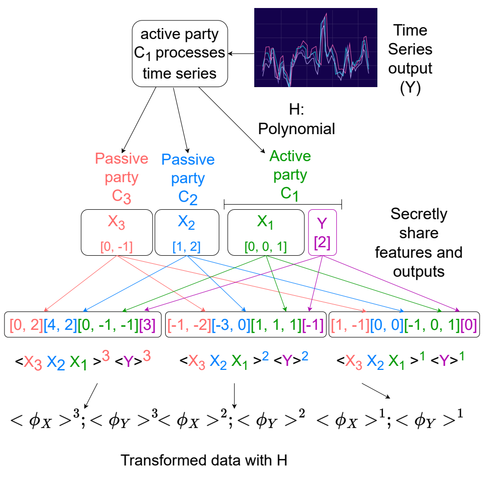
Data: X_k on party C_k $\forall k \in [1,K]$, and Y on C_1
Accepted Parameters: Task: *Training/Inference*, Model *type*, number of trees T, Optimization method O, learning rate α, iterations, e, Trained distributed $Model$, Requesting party C_j
Output:Trained model distributed across K parties or final inference result on party C_j

```
 1: if $Training$ and active party $C_1$ then
 2:     $params = ProcessSeries(Y)$
 3:     $H = GenPoly\ (params, type)$
 4:     Broadcast $H$ to $C_i$ $\forall i \in [1,K]$
 5: end if
 6: Share local features $\langle X_k \rangle$ and (or) outputs $\langle Y \rangle$
 7: $\langle \phi_X \rangle, \langle \phi_Y \rangle = TransformData\ (H)$ {Equation 3.2}
 8: if $type == Tree$ and $Training$ then
 9:     return $Model = \mathtt{STV}_T\ (\langle \phi_X \rangle, \langle \phi_Y \rangle, T)$
10: else if $type == Linear$ and $Training$ then
11:     return $Model = \mathtt{STV}_L\ (\langle \phi_X \rangle, \langle \phi_Y \rangle, O, \alpha, e)$
12: end if
13: if $Inference$ then
14:     $\langle Result \rangle = Model.Predict(\langle \phi_X \rangle)$
15:     if requesting party $C_j$ then
16:         $Result = \sum_{k=1}^{K} \langle Result \rangle^k$ {Aggregate predictions}
17:     end if
18: end if
```

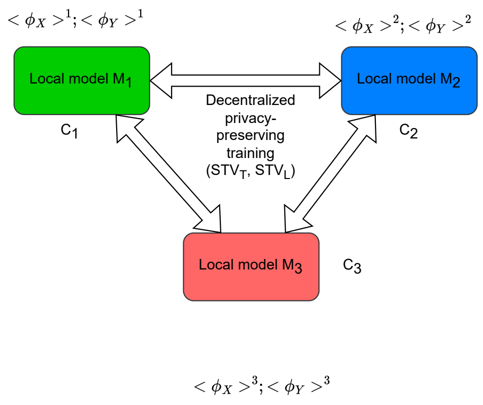
4.4 $\mathtt{STV_T}$

Training ARTs within `STV` is done using $\mathtt{STV_T}$. This approach is rooted in our explanation from section 3.2, where we illuminated how time series forecasting can be framed as a regression problem by transforming the original data into design matrices. This strategic alteration allows us to leverage tree-training methodologies, such as the approach introduced by Xie et al. [54], in the training of ARTs. A detailed description of $\mathtt{STV_T}$ is provided in Algorithm 2.

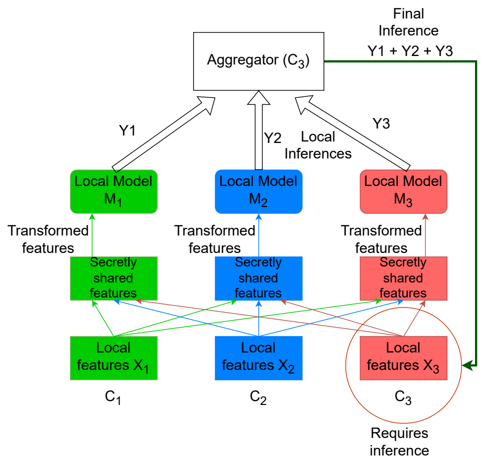
In Algorithm 2, the training process unfolds iteratively, resulting in the generation of T trees on each party. In every iteration, each participating party learns a new tree, akin to the conventional approach in standard XGBoost, and subsequently makes a local prediction



(a) Pre-processing, secret sharing, and transformation



(b) Privacy-preserving training on secretly shared and transformed features



(c) First-hand, serverless inference

Figure 4.1: STV framework

(lines 4-5). Since data is secretly shared, the final prediction is derived by aggregating the shares of local predictions from all clients (line 7).

The key component in this process is the tree-building function, ***SecureFit***, derived from the work of Xie et al. This function employs secret sharing primitives to train XGBoost, involving computations of first and second-order gradients. Given that XGBoost is traditionally a centralized algorithm, its VFL-based variant, with secret sharing, replaces centralized operations with SMPC-based alternatives. Additional details of this algorithm can be found in Appendix B.

During the training phase, individual predictions are aggregated on the active party (C_1) since the gradients are computed by C_1. However, for the inference process, aggregation can be carried out on any party since gradients are not computed, enabling serverless, first-hand inferencing capability.

Algorithm 2 $\mathtt{STV_T}$

Data: Secretly shared transformed matrices $\langle \phi_X \rangle$, $\langle \phi_Y \rangle$
Parameter: number of trees T
Output: Distributed autoregressive XGBoost tree

```
 1: Initialize predictions $\langle \hat{\phi_Y} \rangle^k = 0$ on all parties $C_k$
 2: Initialize $Trees_k$ = [ ] on all parties $C_k$
 3: for $t \in [1, T]$ do
 4:     $tree_{t_k}$ = SecureFit($\langle \phi_X \rangle^k$, $\langle \phi_Y \rangle^k$, $\langle \hat{\phi_Y} \rangle^k$)
 5:     $\langle \hat{\phi_{Y_t}} \rangle^k = tree_{t_k}.Predict(\langle \phi_X \rangle^k)$
 6:     if active party $C_1$ then
 7:         $\hat{\phi_{Y_t}} = \sum_{i=1}^{K} \langle \hat{\phi_{Y_t}} \rangle^k$ {Aggregate predictions}
 8:         $\hat{\phi_Y} = \hat{\phi_Y} + \hat{\phi_{Y_t}}$ {Add to final predictions}
 9:     end if
10:     $Trees_k.append(tree_{t_k})$ on every party $C_k$
11: end for
12: return $Trees_k$ on party $C_k$
```

4.5 $\mathtt{STV_L}$

In the context of $\mathtt{STV_L}$, our primary objective is optimizing the coefficients, denoted as A, in Equation 3.2. This optimization can be approached directly or iteratively, mirroring the two-step regression process. The key distinction here is that the traditional centralized operations are replaced with their SMPC equivalents to enable compatibility for VFL. Details are outlined in Algorithm 3. In the centralized version of this process, as we previously explored, the optimization of the coefficients can be accomplished using either iterative methods or direct optimization techniques, like in Equation 3.6 and Equation 3.7, which use matrix operations. This is reflected in several steps within Algorithm 3, where matrix operations such as multiplications (lines 8-9; 13-19) and inverses (line 14) are conducted on secretly shared data. While Han et al. [20] have provided algorithms for the two-party scenario, we have extended these techniques to accommodate a multi-party context.

N-party matrix multiplication. To extend matrix multiplication to multiple parties (algorithm 4), we view the computation of every output element $W_{i,j}$ as a scalar product of row and column vectors (line 3). A scalar product between two vectors involves an element-wise multiplication of the elements followed by a summation of the products. Using Beaver's triples method and addition on secretly-shared values (see section 3.1), these operations can be securely computed.

N-party matrix inverse. For matrix inverses (Algorithm 5), we compute the inverse of a secretly-shared matrix, U, using a non-singular perturbation matrix, P, generated by the active participant (line 2). The idea is to compute the inverse of UP, i.e., $UP^{-1} = P^{-1}U^{-1}$ and then pre-multiply this with P to obtain U^{-1}. The aggregation of product UP on a passive party does not leak U as the party does not know P (line 5). $(UP)^{-1}$ can then be computed locally and secretly shared, followed by a matrix multiplication with P, i.e., $P \times (UP)^{-1} = U^{-1}$, giving the result as a secret share (lines 7-10).

Algorithm 3 $\texttt{STV}_{\texttt{L}}$

Data: Secretly shared transformed matrices $\langle \phi_X \rangle$, $\langle \phi_Y \rangle$
Accepted Parameters: O, α, e
Output: Shared optimized coefficients $\langle A \rangle$

```
1:  for $step \in [1,2]$ do
2:      if $step == 1$ then
3:          Initialize residuals to zero in $\langle \phi_X \rangle^k$ for all $C_k$
4:      end if
5:      if $O ==$ "$iterative$" then
6:          Randomly initialize $\langle A \rangle^k$ for all $C_k$
7:          for $e$ iterations do
8:              Get $\langle \hat{\phi_Y} \rangle = \langle \phi_X \rangle \times \langle A \rangle$ using Alg. 4
9:              $\langle \frac{dl}{dA} \rangle = 2 \times (\langle \phi_X \rangle)^T \times (\langle \hat{\phi_Y} \rangle - \langle \phi_Y \rangle)$ (Alg. 4)
10:             Perform update: $\langle A \rangle := \langle A \rangle - \alpha \langle \frac{dl}{dA} \rangle$
11:         end for
12:     else if $O ==$ "$direct$" then
13:         $\langle Z \rangle = (\langle \phi_X \rangle)^T \times \langle \phi_X \rangle$ using Alg. 4
14:         $\langle W \rangle = \langle Z^{-1} \rangle$ using Alg. 5
15:         $\langle V \rangle = (\langle \phi_X \rangle)^T \times \langle \phi_Y \rangle$ using Alg. 4
16:         $\langle A \rangle = \langle W \rangle \times \langle V \rangle$
17:     end if
18:     if $step == 1$ then
19:         Predict: $\langle \hat{\phi_Y} \rangle = \langle \phi_X \rangle \times \langle A \rangle$ using Alg. 4
20:         Estimate residuals $\langle \varepsilon \rangle = \langle \phi_Y \rangle - \langle \hat{\phi_Y} \rangle$
21:         Set $\langle \phi_X \rangle^k$ using residuals from $\langle \varepsilon \rangle^k$ for all $C_k$
22:     end if
23: end for
24: return $\langle A \rangle^k$ on all parties $C_k$
```

Algorithm 4 Secure Matrix Multiplication

Data: Secretly shared matrices $\langle U \rangle$ and $\langle V \rangle$ across K parties, $C_1, C_2, .., C_K$
Output: $\langle W \rangle$, i.e., product $W = U \times V$ as shares across K parties

```
1: for each row index $i$ do
2:     for each column index $j$ do
3:         $\langle \vec{T} \rangle = \langle U[\vec{i},:] \rangle * \langle V[\vec{:},j] \rangle$ {Element-wise product} $\langle W_{i,j} \rangle^k = sum\{\langle \vec{T} \rangle^k\}$
4:     end for
5: end for
6: return $\langle W \rangle^k$ on party $C_k$
```

Algorithm 5 Secure Matrix Inverse

Data: Secretly shared matrix $\langle U \rangle$ across K parties, $C_1, C_2, .., C_K$
Output: $\langle V \rangle$, i.e., inverse, $V = U^{-1}$, as a distributed share

```
 1: if active party ($P_1$) then
 2:     Generate random non-singular perturbation matrix $P$ and secretly share as $\langle P \rangle$
 3: end if
 4: Get $\langle Q \rangle = \langle U \rangle \times \langle P \rangle$ using Alg. 4
 5: Aggregate $Q = \sum_{k=1}^{K} \langle Q \rangle^k$ on passive party $C_j; j > 1$
 6: if Passive party $C_j$ then
 7:     Compute $R = Q^{-1} = (UP)^{-1} = P^{-1}U^{-1}$
 8:     Generate shares $\langle R \rangle$
 9: end if
10: Compute $\langle T \rangle = \langle U^{-1} \rangle = \langle P \rangle \times \langle R \rangle$ using Alg. 4
11: return $\langle T \rangle^k$ on party $C_k$
```

Chapter 5

Experiments and Analysis

We assess the predictive accuracy of `STV` in comparison to centralized approaches. Additionally, we analyze the scalability of both iterative and direct optimization methods for linear forecasters with respect to the overall communication cost.

5.0.1 Forecasting accuracy

We compare the performance of $\texttt{STV}_\texttt{L}$ (direct optimization) and $\texttt{STV}_\texttt{T}$ with other centralized methods: Long-Short-Term Memory (LSTM), SARIMAX with MLE[1], and diffusion models for forecasting (*SSSD*^{S4}) [3].

The LSTM has two layers of 64 LSTM units each, followed by two dense layers of size 32 and 1. We train the model for 500 epochs and use a batch size 32 with a default learning rate of 0.001. The diffusion configuration has the following settings[2]: $T = 200, \beta_0 = 0.0001, \beta_T = 0.02$. For the wavenet configuration, we use two residual layers, four residual and skip channels, with three diffusion embedding layers of dimensions 8×16. Training is done for 4000 iterations with a learning rate of 0.002.

Five public forecasting datasets are used: Airline passengers [27], Airquality data [50], PV Power [23], SML 2010 [39], and Rossman Sales [1]. An industry-specific dataset to estimate a performance parameter from inline sensor values in semiconductor manufacturing is also included [17, 18]. Additional details on the datasets, evaluation, and pre-processing are provided in Appendix C and Appendix D.

A variant of prequential window testing [6] is used since industrial time series data can significantly change after intervals due to machine changes/repairs. The dataset is partitioned into multiple windows of a given size, each further divided into an 80-20 train-test split. After forecasting the test split, the model is retrained on the next window (see Appendix D).

For a consistent comparison, all features and outputs are scaled between 0 and 1. We thus present the normalized mean-squared errors (n-MSE) of the predictions, and ground truths on the test set are measured and averaged across multiple windows. We then average

[1]https://www.statsmodels.org/devel/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html
[2]Using https://github.com/AI4HealthUOL/SSSD.git

Dataset	STV_L (SARIMAX, VFL)	STV_T (ART, VFL)	*SARIMAX* (MLE, Centralized)	*LSTM* (Centralized)	*SSSD*^{S4} (Centralized)	Rel. imp. (%)
Airquality	**0.00069** (0.00008)	0.00100 (0.00056)	0.00088 (0.00031)	0.00114 (0.00076)	0.00150 (0.00073)	21.59
Airline passengers	0.00304 (0.00086)	0.00808 (0.00379)	**0.00222** (0.00148)	0.04130 (0.04086)	0.00392 (0.00226)	-36.94
PV Power	**0.00138** (0.00136)	0.00249 (0.00254)	0.00159 (0.00095)	0.14333 (0.23033)	0.00167 (0.00058)	15.22
SML 2010	**0.00787** (0.00711)	0.01875 (0.01458)	0.01033 (0.01061)	0.01716 (0.01086)	0.01085 (0.00871)	23.81
Rossman Sales	**0.00074** (0.00012)	0.00243 (0.00153)	0.00077 (0.00006)	0.00639 (0.00280)	0.00331 (0.00151)	3.89
Industrial data	**0.00602** (0.00049)	0.04118 (0.04051)	0.00969 (0.00449)	0.00617 (0.00203)	0.01875 (0.00313)	2.43

Table 5.1: Average normalized MSE and (standard deviation) results for different datasets and methods. Relative improvement of the best VFL method with the best centralized one is also shown (rel. imp). Lowest MSE values are highlighted in bold. SML 2010, Air quality, and Rossman Sales have prequential window sizes 50, 100, 200, and 400. PV Power uses 25, 50, 100, and 200. Airline passengers uses 60, 80, 100, 120, and 140. Finally, the industrial data uses 25, 50, and 100.

the n-MSE scores across different window sizes to generalize performance scores across varying forecasting ranges, the results of which are in Table 5.1.

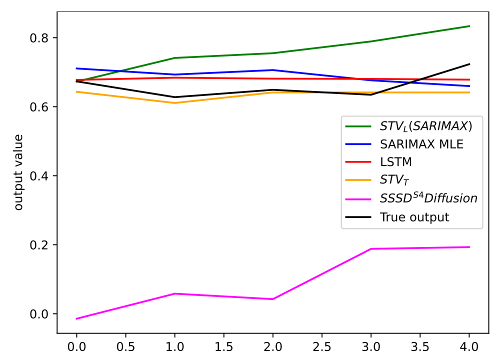
Table 5.1 demonstrates that, in general, $\mathtt{STV_L}$ stands out as the best-performing method across various datasets. We opt for direct optimization due to its convergence properties, as iterative gradient descent eventually reaches the same value over time. This strategic choice results in a remarkable improvement of up to 23.81%

For a deeper understanding, we present regression plots for a window from both the largest and smallest prequential window sizes for each dataset in Figure 5.1 and Figure 5.2. Notably, all the methods demonstrate the ability to capture patterns in the time series, as observed in Air Quality data and Rossman Sales (see figures 4.1c, 4.1d, 4.2c, 4.2d). However, it's worth noting that highly complex models, such as *SSSD*^{S4}, may occasionally exhibit overfitting, highlighting the limitation of neural networks when dealing with small datasets and short-term forecasting (see figures 4.1e, 4.2a, 4.2b).

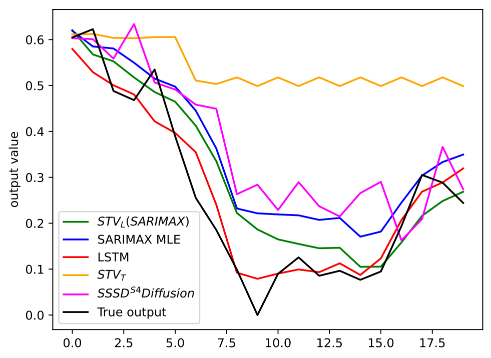
5.0.2 Scalability

We quantify the total communication costs of optimization using the normal equation (NE) and iterative batch gradient descent (GD), aiming to analyze the scalability of these two methods as the number of parties, features, and samples increase.

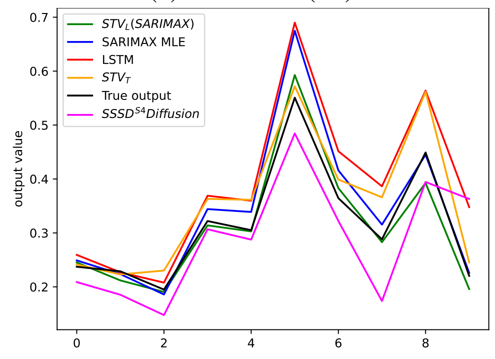
We systematically vary the number of parties across 2, 4, and 8; features between 10 and 100; and samples with values of 10, 100, and 1000. To ensure that communication costs depend solely on the dataset dimensions and the number of parties, we generate random data matrices for all valid combinations of features and samples on each client, adhering to the constraint that the number of features should be less than or equal to the number of samples (#features $\leq$ #samples). The coefficient optimization is performed using either the direct approach (Algorithm 3, lines (13-19)), or batch gradient descent (Algorithm 3, lines (6-10)), with varying numbers of iterations (10, 100, 1000).



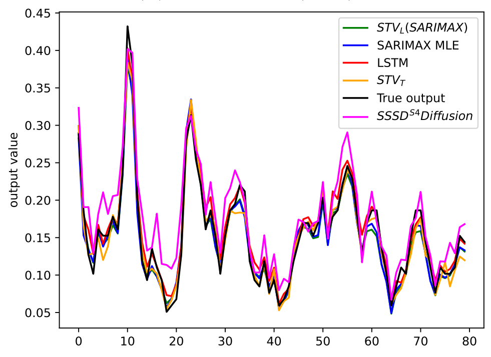
(a) Industrial (25)



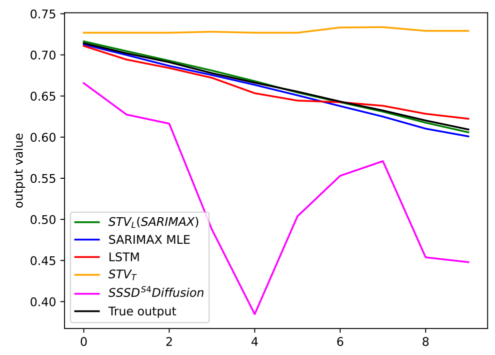
(b) Industrial (100)



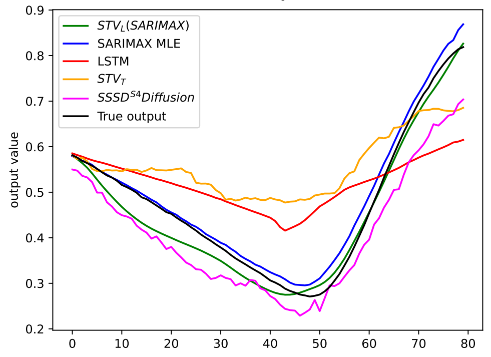
(c) Air Quality (50)



(d) Air Quality (400)

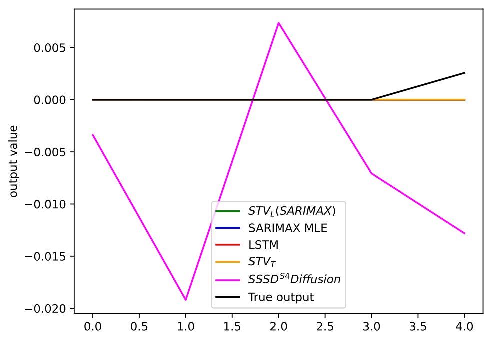


(e) SML 2010 (50)

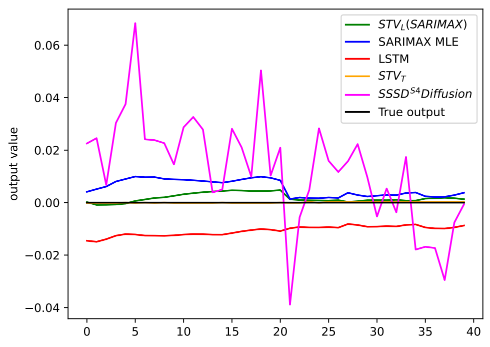


(f) SML 2010 (400)

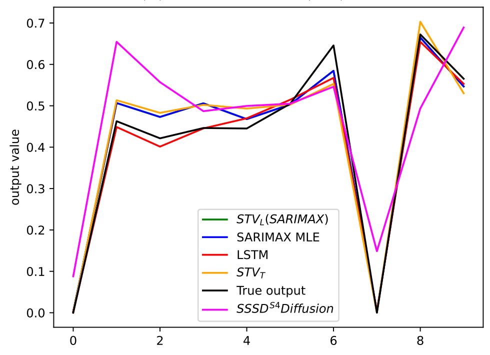
Figure 5.1: Regression plots showing the forecasts of various methods for the largest and smallest prequential window sizes (in brackets). Datasets present are Industrial data, Air Qualtiy, and SML 2010.



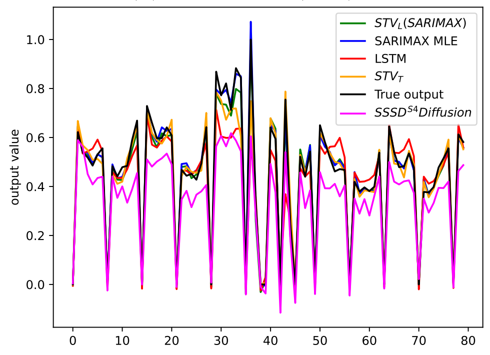
(a) PV Power (25)



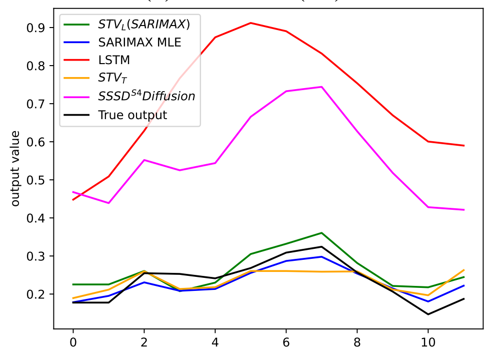
(b) PV Power (200)



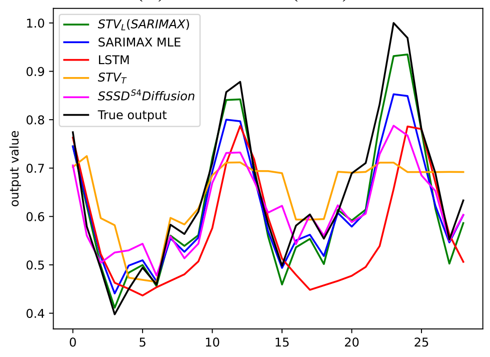
(c) Rossman (50)



(d) Rossman (400)



(e) Flight Passengers (60)



(f) Flight Passengers (140)

Figure 5.2: Regression plots showing the forecasts of various methods for the largest and smallest prequential window sizes (in brackets). Datasets present are PV Power, Rossman Sales, and Flight Passengers.

For client scaling, we average the total communication costs across different combinations of features and samples for each specified number of parties. Similarly, we assess feature and sample scaling by averaging the total costs across different (sample, client) and (feature, client) combinations. The results of these analyses are presented in Table 5.2, Table 5.3, and Table 5.4. The complete version of these tables is provided in Appendix E.

parties	NE	GD iterations		
		10	100	1000
2	2.54E+08	2.33E+07	2.33E+08	2.33E+09
4	5.85E+08	4.65E+07	4.65E+08	4.65E+09
8	1.48E+09	9.31E+07	9.31E+08	9.31E+09

Table 5.2: Average of total communication sizes (bytes) with varying parties.

Features	NE	GD iterations		
		10	100	1000
10	9.77E+06	8.34E+06	8.34E+07	8.34E+08
100	1.92E+09	1.23E+08	1.23E+09	1.23E+10

Table 5.3: Average of total communication sizes (bytes) with varying feature counts.

Samples	NE	GD iterations		
		10	100	1000
10	1.16E+06	2.76E+05	2.76E+06	2.76E+07
100	4.53E+08	1.24E+07	1.24E+08	1.24E+09
1000	1.48E+09	1.23E+08	1.23E+09	1.23E+10

Table 5.4: Average of total communication sizes (bytes) with varying samples.

From these three tables, we discern that when the number of parties, samples, or features is relatively small, the cost of direct optimization (NE) is comparable to an iterative version with a higher number of iterations. For example, with 2 parties, the cost of NE closely aligns with GD with 100 iterations, as evident in Table 5.2. However, as we increase the number of parties to 8, the cost of NE surpasses that of GD with 100 iterations.

Similarly, in Table 5.3 and Table 5.4, we observe that NE exhibits lower costs than GD with 100 iterations when the number of features or samples is limited. However, this balance shifts when we increase the feature and sample counts.

Notably, the cost of GD grows proportionally with the number of iterations, eventually surpassing NE, particularly when 1000 iterations are employed. In practical scenarios, hyperparameters such as the learning rate significantly influence the convergence steps, which might be challenging to fine-tune in a distributed setup. If the correct hyperparameters are chosen, the number of iterations for convergence could be small, making gradient descent a better option. However, for the wrong set, the iterations to converge may be significant, making the cost of direct optimization a more viable option. Therefore, the choice between iterative and direct optimization depends on several factors, necessitating adaptability within frameworks.

Chapter 6

Future Work and Conclusions

This chapter gives possible future directions to continue this line of research. We also provide concluding remarks on the scope of this work.

6.1 Future work

While this work marks an essential initial step toward forecasting within the framework of Vertical Federated Learning (VFL), there remain significant areas for improvement and exploration, which we highlight as follows.

1. ***Extension to complex forecasting models.*** The current models employed in this study, although effective for certain scenarios, possess limitations in capturing the intricate patterns of highly complex time series data, especially for long-term forecasting. As we find ourselves in an era of unprecedented data availability due to digitalization, the potential for more advanced models capable of handling this complexity is noteworthy. Recent advancements in forecasting, such as Long Short-Term Memory (LSTM) networks and the emerging state-of-the-art diffusion models, show promise. However, these cutting-edge models are primarily designed for centralized data scenarios, making their adaptation to the vertically partitioned data context a natural next step. By achieving this, we aim to assemble a diverse set of forecasting models tailored for diverse industry scenarios.

2. ***Hybrid horizontal and vertical federated learning.*** From a practical standpoint, facilitating the onboarding of new participants or collaborators into the VFL forecasting environment is a crucial consideration. Currently, the retraining process poses challenges for new entrants, leading us to contemplate a more efficient approach. This involves the possibility of independently training clusters of VFL models, subsequently aggregating them to harness their collective forecasting power. This innovation, which bridges elements of both horizontal and vertical federated learning, holds the potential to significantly reduce costs and computational burdens while maintaining forecast accuracy.

3. ***Stronger privacy.*** Although this work assumes semi-honest participants, extending our research to address malicious adversaries represents a valuable and necessary future endeavor. By fortifying the VFL framework against adversarial attacks, we elevate the robustness and reliability of our collaborative forecasting system.

6.2 Conclusion

This work introduces a novel privacy-preserving forecasting framework for vertical federated learning. By integrating SS and SMPC, we achieve privacy guarantees and address crucial challenges faced by the manufacturing industry.

One of the key aspects of our approach is to distribute predictions' ownership among multiple parties, addressing requirements for inference privacy. This is a shift from the conventional VFL assumptions that predictions are always available to a specific party.

Our performance evaluations show that VFL methods have competitive performance against centralized methods. Furthermore, our scalability analyses highlight the nuanced dynamics between iterative and direct optimization, highlighting the need for an adaptable framework.

For the future, several avenues of exploration beckon. Firstly, the models employed in this study have limitations when capturing the patterns present in highly complex time series data for which LSTMs and diffusion models are better suited. We aim to adapt these models to the vertically partitioned data context. Another area is exploring a hybrid approach combining horizontal FL and VFL elements by independently training clusters of VFL models, later aggregated to harness their collective forecasting power. Lastly, fortifying our VFL framework against adversarial attacks is essential. While our current work assumes semi-honest participants, the reality of malicious adversaries cannot be ignored.

Bibliography

[1] Rossman store sales, 2015. URL `https://www.kaggle.com/c/rossmann-store-sales`.

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

[3] Juan Miguel Lopez Alcaraz and Nils Strodthoff. Diffusion-based time series imputation and forecasting with structured state space models. *arXiv preprint arXiv:2208.09399*, 2022.

[4] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology—CRYPTO'91: Proceedings 11*, pages 420–432. Springer, 1992.

[5] JA Blais et al. Least squares for practitioners. *Mathematical Problems in Engineering*, 2010, 2010.

[6] Vitor Cerqueira, Luis Torgo, and Igor Mozetič. Evaluating time series forecasting models: An empirical study on performance estimation methods. *Machine Learning*, 109:1997–2028, 2020.

[7] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4, 2015.

[8] Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. Vafl: a method of vertical asynchronous federated learning. *arXiv preprint arXiv:2007.06081*, 2020.

[9] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems*, 36(6):87–98, 2021.

[10] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015.

[11] Cynthia Dwork. Differential privacy. In *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II 33*, pages 1–12. Springer, 2006.

[12] Cynthia Dwork. Differential privacy: A survey of results. In *Theory and Applications of Models of Computation: 5th International Conference, TAMC 2008, Xi'an, China, April 25-29, 2008. Proceedings 5*, pages 1–19. Springer, 2008.

[13] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[14] Wenjing Fang, Derun Zhao, Jin Tan, Chaochao Chen, Chaofan Yu, Li Wang, Lei Wang, Jun Zhou, and Benyu Zhang. Large-scale secure xgb for vertical federated learning. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 443–452, 2021.

[15] Fangcheng Fu, Huanran Xue, Yong Cheng, Yangyu Tao, and Bin Cui. Blindfl: Vertical federated machine learning without peeking into your data. In *Proceedings of the 2022 International Conference on Management of Data*, pages 1316–1330, 2022.

[16] Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *CoRR*, abs/1712.07557, 2017. URL `http://arxiv.org/abs/1712.07557`.

[17] Dimitra Gkorou, Alexander Ypma, George Tsirogiannis, Manuel Giollo, Dag Sonntag, Geert Vinken, Richard van Haren, Robert Jan van Wijk, Jelle Nije, and Tom Hoogenboom. Towards big data visualization for monitoring and diagnostics of high volume semiconductor manufacturing. In *Proceedings of the Computing Frontiers Conference*, CF'17, page 338–342, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450344876. doi: 10.1145/3075564.3078883. URL `https://doi.org/10.1145/3075564.3078883`.

[18] Dimitra Gkorou, Maialen Larranaga, Alexander Ypma, Faegheh Hasibi, and Robert Jan van Wijk. Get a human-in-the-loop: Feature engineering via interactive visualizations. 2020.

[19] James Douglas Hamilton. *Time series analysis*. Princeton university press, 2020.

[20] Shuguo Han, Wee Keong Ng, Li Wan, and Vincent CS Lee. Privacy-preserving gradient-descent methods. *IEEE transactions on knowledge and data engineering*, 22(6):884–899, 2009.

[21] Edward J Hannan and Laimonis Kavalieris. A method for autoregressive-moving average estimation. *Biometrika*, 71(2):273–280, 1984.

[22] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[23] Ani Kannal. Solar power generation, 2020. URL `https://www.kaggle.com/datasets/anikannal/solar-power-generation-data`.

[24] Afsana Khan, Marijn ten Thij, and Anna Wilbik. Vertical federated learning: A structured literature review, 2023.

[25] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[26] Joos Korstanje. *The SARIMAX Model*, pages 125–131. Apress, Berkeley, CA, 2021. ISBN 978-1-4842-7150-6. doi: 10.1007/978-1-4842-7150-6_8. URL `https://doi.org/10.1007/978-1-4842-7150-6_8`.

[27] Chirag Kothari. Airline passengers, 2018. URL `https://www.kaggle.com/datasets/chirag19/air-passengers`.

[28] Ling Li, Seshu Kumar Damarla, Yalin Wang, and Biao Huang. A gaussian mixture model based virtual sample generation approach for small datasets in industrial processes. *Information Sciences*, 581:262–277, 2021.

[29] Chin-Yi Lin, Yu-Ming Hsieh, Fan-Tien Cheng, Hsien-Cheng Huang, and Muhammad Adnan. Time series prediction algorithm for intelligent predictive maintenance. *IEEE Robotics and Automation Letters*, 4(3):2807–2814, 2019.

[30] Xiaodong Lin and Alan F Karr. Privacy-preserving maximum likelihood estimation for distributed data. *Journal of Privacy and Confidentiality*, 1(2), 2010.

[31] Yehida Lindell. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*, pages 1005–1009. IGI global, 2005.

[32] Chenghao Liu, Steven CH Hoi, Peilin Zhao, and Jianling Sun. Online arima algorithms for time series prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[33] Yang Liu, Yan Kang, Tianyuan Zou, Yanhong Pu, Yuanqin He, Xiaozhou Ye, Ye Ouyang, Ya-Qin Zhang, and Qiang Yang. Vertical federated learning, 2022.

[34] Helmut Lütkepohl. General-to-specific or specific-to-general modelling? an opinion on current econometric terminology. *Journal of Econometrics*, 136(1):319–324, 2007.

[35] Christopher Meek, David Maxwell Chickering, and David Heckerman. Autoregressive tree models for time-series analysis. In *Proceedings of the 2002 SIAM International Conference on Data Mining*, pages 229–244. SIAM, 2002.

[36] Douglas C Montgomery, Cheryl L Jennings, and Murat Kulahci. *Introduction to time series analysis and forecasting*. John Wiley & Sons, 2015.

[37] Pascal Paillier. Paillier encryption and signature schemes., 2005.

[38] Josef Perktold. Sarimax, 2023. URL `https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html`.

[39] Pablo Romeu-Guallart and Francisco Zamora-Martinez. SML2010. UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5RS3S.

[40] Wojciech Samek and Klaus-Robert Müller. Towards explainable artificial intelligence. *Explainable AI: interpreting, explaining and visualizing deep learning*, pages 5–22, 2019.

[41] Anand D. Sarwate, Kamalika Chaudhuri, and Claire Monteleoni. Differentially private support vector machines. *CoRR*, abs/0912.0071, 2009. URL `http://arxiv.org/abs/0912.0071`.

[42] Monica Scannapieco, Ilya Figotin, Elisa Bertino, and Ahmed K Elmagarmid. Privacy preserving schema and data matching. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 653–664, 2007.

[43] A Shamir. How to share a secret communications of the acm, 22, 1979.

[44] Haoran Shi, Yali Jiang, Han Yu, Yonghui Xu, and Lizhen Cui. Mvfls: multi-participant vertical federated learning based on secret sharing. *The Federate Learning*, pages 1–9, 2022.

[45] Joshua Snoke, Timothy R. Brick, Aleksandra Slavkovic, and Michael D. Hunter. Providing accurate models across private partitioned data: Secure maximum likelihood estimation, 2017.

[46] Gian Antonio Susto and Alessandro Beghi. Dealing with time-series data in predictive maintenance problems. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE, 2016.

[47] Agostino Tarsitano and Ilaria L Amerise. Short-term load forecasting using a two-stage sarimax model. *Energy*, 133:108–114, 2017.

[48] Larry F Thompson. An introduction to lithography. ACS Publications, 1983.

[49] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.

[50] Saverio Vito. Air Quality. UCI Machine Learning Repository, 2016. DOI: https://doi.org/10.24432/C59K5F.

[51] Simon Vollert, Martin Atzmueller, and Andreas Theissler. Interpretable machine learning: A brief survey from the predictive maintenance perspective. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 01–08, 2021. doi: 10.1109/ETFA45728.2021.9613467.

[52] Jinjiang Wang, Yilin Li, Robert X Gao, and Fengli Zhang. Hybrid physics-based and data-driven models for smart manufacturing: Modelling, simulation, and explainability. *Journal of Manufacturing Systems*, 63:381–391, 2022.

[53] Kang Wei, Jun Li, Chuan Ma, Ming Ding, Sha Wei, Fan Wu, Guihai Chen, and Thilina Ranbaduge. Vertical federated learning: Challenges, methodologies and experiments, 2022.

[54] Lunchen Xie, Jiaqi Liu, Songtao Lu, Tsung-Hui Chang, and Qingjiang Shi. An efficient learning framework for federated xgboost using secret sharing and distributed optimization. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(5): 1–28, 2022.

[55] Yang Yan, Guozheng Yang, Yan Gao, Cheng Zang, Jiajun Chen, and Qiang Wang. Multi-participant vertical federated learning based time series prediction. In *Proceedings of the 8th International Conference on Computing and Artificial Intelligence*, pages 165–171, 2022.

[56] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[57] Yu Zhang, Peter Tiňo, Aleš Leonardis, and Ke Tang. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):726–742, 2021. doi: 10.1109/TETCI.2021.3100641.

[58] Qun-Xiong Zhu, Hong-Tao Zhang, Ye Tian, Ning Zhang, Yuan Xu, and Yan-Lin He. Co-training based virtual sample generation for solving the small sample size problem in process industry. *ISA transactions*, 134:290–301, 2023.

Appendix A

Glossary

In this appendix we give an overview of frequently used terms and abbreviations.

ART: Autoregressive Tree

GD: Gradient Descent

HE: Homomorphic Encryption

LS: Least Squares

LSTM: Long Short Term Memory

MLE: Maximum Likelihood Estimation

MSE: Mean Squared Error

NE: Normal Equation

NN: Neural Network

SMPC: Secure Multi-Party Computation

SS: Secret Sharing

VFL: Vertical Federated Learning

Appendix B

SMPC-based XGBoost for VFL

XGBoost [7] is a tree-based gradient-boosting algorithm, that iteratively generates an ensemble of trees by greedily learning a new tree at every step to improve on the earlier one. Each tree has weights assigned to its leaf nodes. When making a prediction for a sample, the weights corresponding to the leaf to which the sample was assigned to are summed to give the final prediction score.

To generate a tree at every iteration, it uses first and second order gradients of the latest predictions, i.e., from the previous tree, in order to set the optimal weights for the new one.

For each sample with index i, the first and second order gradients are denoted as g_i and h_i, respectively. The sum of the gradients of all instances on a particular node is used to set the new weights for it. For example, for node j and corresponding instance set I_j, the accumulated values of g and h are computed as follows: $G_j = \sum_{i \in I_j} g_i$, and $H_j = \sum_{i \in I_j} h_i$. Based on this, for a tree with T nodes, the weights and objective are calculated as follows:

$$w_j = -G_j/(H_j + \lambda) \tag{B.1}$$

$$obj = -0.5 \times \sum_{j=1}^{T} ((G_j)^2/(H_j + \lambda)) + \gamma T \tag{B.2}$$

, where γ, λ are regularizers. While Equation B.1 sets the new weights, Equation B.2 is used to identify how to split nodes at each iteration.

With this in mind, using the secret sharing primitives it is possible to compute these functions to extend XGBoost to VFL, which we show in Algorithm 6.

When XGBoost is trained for VFL using secret sharing, each client obtains a local tree with their own weights as shown in Figure B.1. In the figure, the weights for clients 1 and 2 are distributed such that their local weights are shares of the weights of the centralized version, i.e., $wi = wi1 + wi2 \ \forall i \in [1,4]$

The indicator vector s on line 1, is a binary vector that is used to point out the location of instances on nodes. We explain this with the help of the example in Figure B.2. To calculate the updated weight of the node with instances that have an age greater than 30 (bottom left), we need to find the sum of $\sum_{i \in I_j} g_i$, where $I_j = \{2,4\}$(Equation B.1). The indicator vector in this case would be $s = [0,1,0,1]$, meaning that nodes 2 and 4 are part of node j. If we have a vector of the gradients, $g = [g1, g2, g3, g4]$, we can compute $g2 + g4$ as $s \odot g$, i.e., the

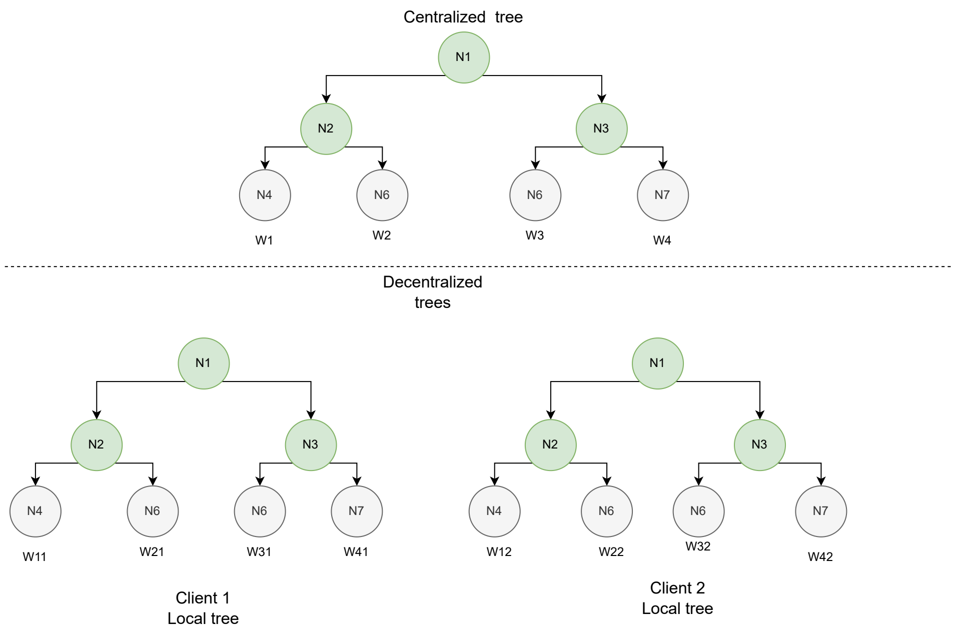


Figure B.1: Centralized vs Secretly shared XGBoost trees

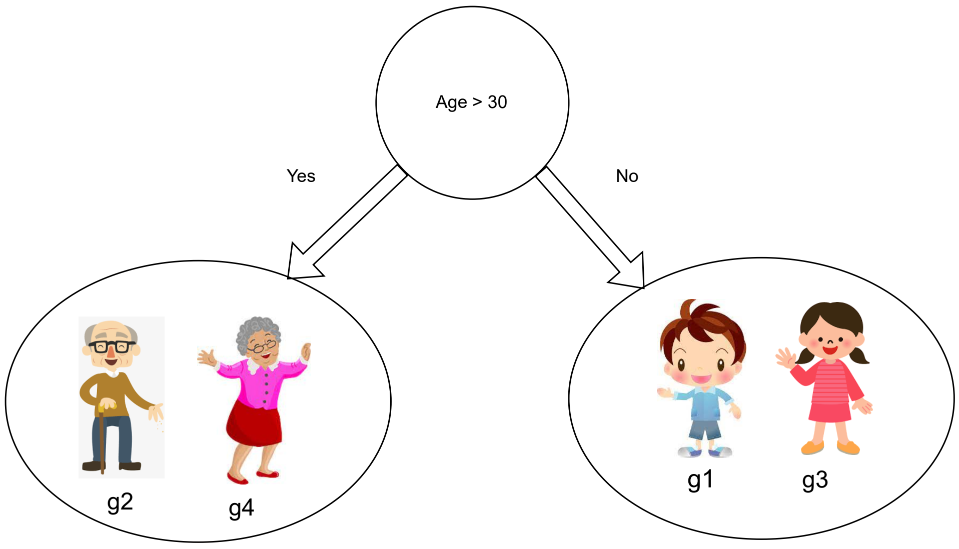


Figure B.2: Mapped instances to a node in XGBoost

inner product. Under VFL, both the gradients g, and the indicator vector s, exist as secret shares across clients, i.e., $s = \sum_k^K \langle s_k \rangle^k$, and $g = \sum_k^K \langle g_k \rangle^k$. Therefore, to compute the product, we can do it using secret shared primitives for matrix multiplication.

The process of split-finding and setting weights is done using the ***SecureBuild*** function, which makes use of secret sharing primitives to compute the functions in Equation B.2 and Equation B.1. We defer readers to the implementation in Xie et al. [54] for additional details

Algorithm 6 Fit XGBoost [54]

Data: Secretly shared matrices: $\langle X \rangle$, $\langle Y \rangle$, $\langle \hat{Y} \rangle$ across K clients, $C_1, C_2, .., C_K$
Output: Learned tree for the current iteration, one on each client.

```
1: Initialize indicator vector $s \leftarrow 1$ on all $C_k$
2: **if** active client $C_1$ **then**
3:     Compute derivatives $g, h$
4:     Generate shares $\langle g \rangle$, $\langle h \rangle$, $\langle s \rangle$
5: **end if**
6: $Tree_k = $ ***SecureBuild***$(\langle g \rangle^k, \langle h \rangle^k, \langle s \rangle^k)$ on each $C_k$
7: **return** $Tree_k$ on $C_k$
```

on this.

Appendix C

Datasets and Pre-processing

We make use of the following public datasets in our work: Air Quality [50], SML 2010 [39], PV Power [23], Airline Passengers [27], and Rossman Sales [1]. Here we provide a brief description of each dataset and the pre-processing steps applied to the dataset itself. These are **not** these steps are unconnected with the series pre-processing steps which are part of the `STV` framework. For all datasets, we scale all features and output values between the range 0 to 1 using a MinMax scaler to ensure that all datasets have the same range of values for comparing the MSE losses.

C.1 Air Quality

The Air Quality dataset contains approximately 9300 samples of multivariate time-series data, with 15 attributes. Five of these are true output values on five gases: Carbon Monoxide (CO), Non Metanic Hydrocarbons (NMHC)), Benzene (C6H6)), Total Nitrogen Oxides (NOx), and Nitrogen Dioxide (NO2). Exogenous features such as the temperature, ozone levels, and humidity are provided, along with strongly correlated sensor data for each of the five gases. The dataset contains missing values and duplicates, and contains hourly data for each of the five gases.

We preprocess the data by discarding all rows with any missing information and remove duplicate rows. We predict the ground truth values of CO using the other sensor values and information such as temperature and humidity as the exogenous regressors.

C.2 SML 2010

The SML 2010 dataset contains infomation from a monitor system in a domotic house. It contains approximately 4100 samples with 24 attributes in total, corresponding to 40 days of monitoring data. The attributes contain values such as the indoor and outdoor temperature, lighting levels, Carbon Dioxide levels, relative humidity, rain, windspeed, etc. We predict the indoor habitation temperature using the others as exogenous features.

C.3 Airline Passengers

Airline Passengers is a small dataset of 145 samples containing the number of international airline passengers (in thousands) on a monthly basis. The exogenous features are also just two: the year and the month. We predict the number of passengers using the year and month as exogenous regressors.

C.4 PV Power

The PV Power dataset contains around 3100 samples of solar power generation data from each of two power plants over a 34-day period. Attributes include features such as the DC power, AC power, yield, ambient temperature, irradiation levels, and the data and time.

We drop identifiers, empty, and duplicate data. We also drop the DC power attribute, and total yield as these features are very strongly correlated with the AC output. As outputs, we predict the AC power generation using the remaining features as exogenous regressors.

C.5 Rossman Sales

The Rossman Sales dataset contains sales data for 1115 store outlets. The attributes consists of features such as holidays, store type, competitor distance, number of customers, and promotional details among others. We predict the sales of the store with ID 1, using the other features as exogenous regressors.

Appendix D

Experimental Evaluation

As mentioned in the main text, we use a variation of prequential window testing [6], whereby the entire data is broken into windows of a defined length, each one internally split in an 80-20 train-test ratio. This is illustrated in Figure D.1.

For each window, we train on the portion allotted for training and forecast the remaining. Within a given training window, we first generate the polynomial by processing the time series as in line 3 of Algorithm 1. Identifying the parameters for generating the polynomial can be automated using implementations such as auto arima[1].

For each window size, such as 50, 100, 200, 400, we average the MSE loss between the forecasts and the true values across all windows. The average MSE per-window size is given in Table D.1, which is an expanded version of Table 5.1.

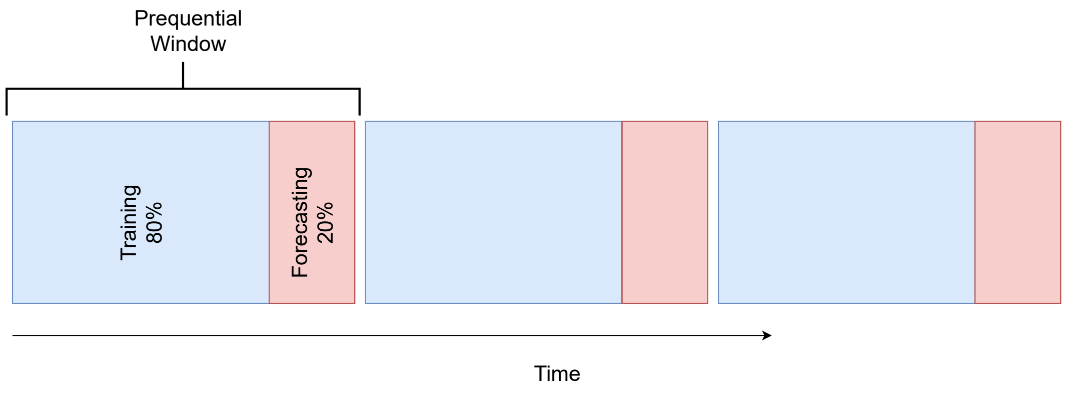


Figure D.1: Prequential window evaluation with re-training in every window

[1]https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html

Dataset	Window size	$\mathtt{STV}_{\mathtt{L}}$	SARIMAX MLE	LSTM	$\mathtt{STV}_{\mathtt{T}}$	*SSSD*^{S4}
Air quality	50	0.00071	0.00110	0.00244	0.00198	0.00270
	100	0.00059	0.00048	0.00055	0.00075	0.00086
	200	0.00066	0.00124	0.00069	0.00067	0.00100
	400	0.00080	0.00068	0.00087	0.00061	0.00144
	Avg.	**0.00069**	0.00088	0.00114	0.00100	0.00150
SML 2010	50	0.00645	0.00621	0.00343	0.00755	0.01373
	100	0.01958	0.02819	0.02778	0.04305	0.02384
	200	0.00500	0.00662	0.02781	0.01709	0.00336
	400	0.00045	0.00030	0.00960	0.00729	0.00249
	Avg.	**0.00787**	0.01033	0.01716	0.01875	0.01085
Rossman Sales	50	0.00070	0.00079	0.00464	0.00183	0.00589
	100	0.00064	0.00071	0.00340	0.00496	0.00281
	200	0.00068	0.00086	0.00674	0.00088	0.00233
	400	0.00095	0.00072	0.01079	0.00206	0.00220
	Avg.	**0.00074**	0.00077	0.00639	0.00243	0.00331
PV Power	25	0.00133	0.00093	0.00950	0.00660	0.00131
	50	0.00360	0.00307	0.01037	0.00258	0.00143
	100	0.00004	0.00063	0.01115	0.00010	0.00267
	200	0.00054	0.00175	0.54227	0.00068	0.00128
	Avg.	**0.00138**	0.00159	0.14333	0.00249	0.00167
Airline Passengers	60	0.00261	0.00113	0.11651	0.00673	0.00110
	80	0.00450	0.00444	0.00339	0.00628	0.00403
	100	0.00308	0.00066	0.01982	0.00270	0.00798
	120	0.00316	0.00136	0.05164	0.01134	0.00356
	140	0.00185	0.00351	0.01512	0.01332	0.00293
	Avg.	0.00304	**0.00222**	0.04130	0.00808	0.00392

Table D.1: Average normalized MSE values for different public datasets, with different prequential window sizes

Appendix E

Scalability Experiments

The complete version of Table 5.2,Table 5.3, and Table 5.4 is shown in Table E.1. As the communication costs don't change with number of trials, the standard deviation is not included.

parties	features	samples	Normal Equation	GD iterations		
				10	100	1000
2	10	10	2.49E+05	1.17E+05	1.17E+06	1.17E+07
		100	1.17E+06	9.81E+05	9.81E+06	9.81E+07
		1000	1.04E+07	9.62E+06	9.62E+07	9.62E+08
	100	100	1.94E+08	9.62E+06	9.62E+07	9.62E+08
		1000	1.06E+09	9.62E+07	9.62E+08	9.62E+09
4	10	10	7.46E+05	2.36E+05	2.36E+06	2.36E+07
		100	2.59E+06	1.96E+06	1.96E+07	1.96E+08
		1000	2.11E+07	1.92E+07	1.92E+08	1.92E+09
	100	100	5.81E+08	1.92E+07	1.92E+08	1.92E+09
		1000	2.32E+09	1.92E+08	1.92E+09	1.92E+010
8	10	10	2.48E+06	4.74E+05	4.74E+06	4.74E+07
		100	6.17E+06	3.93E+06	3.93E+07	3.93E+08
		1000	4.31E+07	3.85E+07	3.85E+08	3.85E+09
	100	100	1.93E+09	3.85E+07	3.85E+08	3.85E+09
		1000	5.41E+09	3.84E+08	3.84E+09	3.84E+10

Table E.1: Total communication sizes (bytes) for varying number of parties features and samples. Compared optimization methods are normal equation and batch gradient descent (GD)