# Appendix: Share Your Secrets for Privacy! Confidential Forecasting with Vertical Federated Learning

## A Datasets and Pre-processing

We make use of the following public datasets in our work: Air Quality [7], SML 2010 [6], PV Power [4], Airline Passengers [5], and Rossman Sales [1]. 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.

### *A.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.

### *A.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.

### *A.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.

### *A.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.

### *A.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.

## B Experimental Evaluation

As mentioned in the main text, we use a variation of prequential window testing [2], 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 1.

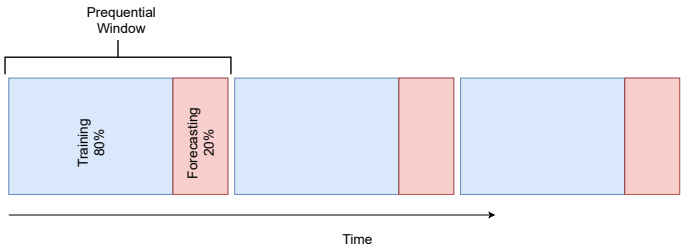


**Figure 1.** Prequential window evaluation with re-training in every window

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 Algorithm 1 from the main text. 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 1, which is an expanded version of Table 1 from the main text.

## C SMPC-based XGBoost with VFL

XGBoost [3] 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:

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

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

, where $\gamma, \lambda$ are regularizers. While Equation 1 sets the new weights, Equation 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 2. 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 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 3. 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 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 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 2 and Equation 1. We defer readers to the implementation in Xie et al. [8] for additional details on this.

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

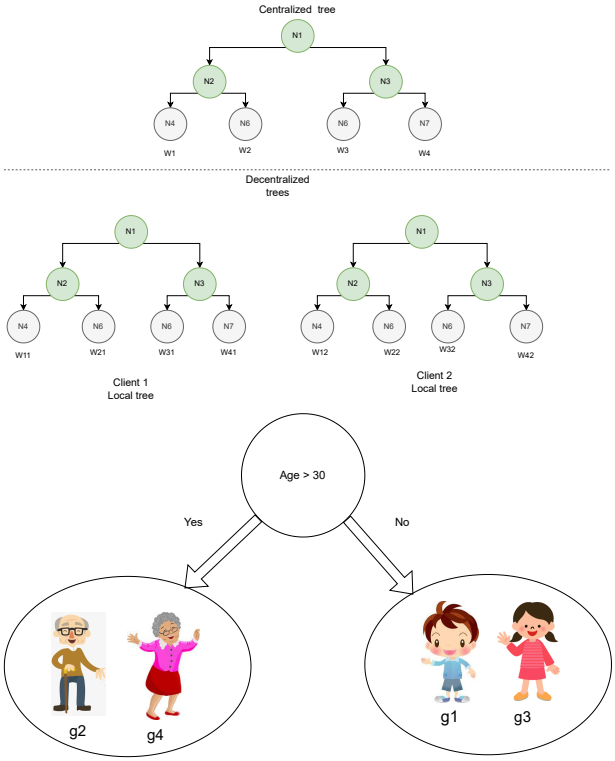


**Figure 3.** Mapped instances to a node in XGBoost

## References


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

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

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

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

[5] C. Kothari. Airline passengers, 2018. URL https://www.kaggle.com/datasets/chirag19/air-passengers.

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

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

[8] L. Xie, J. Liu, S. Lu, T.-H. Chang, and Q. 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.


**Algorithm 1** Fit XGBoost [8]

**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 $C_k$

| 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 1.** Average normalized MSE values for different public datasets, with different prequential window sizes