SecureFedYJ: a safe feature Gaussianization protocol for Federated Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

The Yeo-Johnson (YJ) transformation is a standard parametrized per-feature unidimensional transformation often used to Gaussianize features in machine learning. In this paper, we investigate the problem of applying the YJ transformation in a cross-silo Federated Learning setting under privacy constraints. For the first time, we prove that the YJ negative log-likelihood is in fact convex, which allows us to optimize it with exponential search. We numerically show that the resulting algorithm is more stable than the state-of-the-art approach based on the Brent minimization method. Building on this simple algorithm and Secure Multiparty Computation routines, we propose SECUREFEDYJ, a federated algorithm that performs a pooled-equivalent YJ transformation without leaking more information than the final fitted parameters do. Quantitative experiments on real data demonstrate that, in addition to being secure, our approach reliably normalizes features across silos as well as if data were pooled, making it a viable approach for safe federated feature Gaussianization.

1 Introduction

Federated Learning (FL) [43, 30] is an approach that was recently proposed to train machine learning (ML) models across multiple data holders, or *clients*, without centralizing data points, notably for privacy reasons. While many FL applications have been proposed, two main settings have emerged [21]: cross-device FL, involving a large number of small edge devices, and cross-silo FL, dealing with a smaller number of clients, with larger computational capabilities. Due to the sensitivity and relative local scarcity of medical data, healthcare is a promising application of cross-silo FL [38], e.g. to train a biomedical ML model between different hospitals as if all the datasets were pooled in a central server. In this paper, we focus on the cross-silo setting.

The constraints of cross-silo FL Although cross-silo FL resembles standard distributed learning, it faces at least two important distinct challenges: privacy and heterogeneity. Due to data sensitivity, clients might impose stringent security and privacy constraints on FL collaborations. This arises in *coopetitive* FL projects, where models are jointly trained on industrial competitors' datasets [53], as well as medical FL applications, where conservative data regulations might apply. In this setting, using standard FL algorithms such as FEDAVG [30] might not provide enough privacy guarantees, as privacy attacks such as data reconstruction can be carried out based on the clients' gradients [54, 52]. Various protocols based on Secure Multiparty Computation (SMC) (see Section 2 for more details), such as Secure Aggregation [3], can mitigate this shortcoming by disclosing only the sum of the gradients from all clients to the server, without disclosing each gradient individually.

An additional constraint is that data might present statistical heterogeneity across clients, i.e. the local clients' data distributions may not be identical. In the case of medical applications, such

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

heterogeneity may be caused e.g. by environmental variations or differences in the material that was used for acquisition [41, 45, 1]. While different ways of adapting federated training algorithms have been proposed to automatically tackle heterogeneity [26, 27, 22], these solutions do not address data harmonization and normalization prior to FL training.

Preprocessing in ML Data preprocessing is a crucial step in many ML applications, leading to important performance gains. Among others, common preprocessing methods include data whitening, principal component analysis (PCA) [20] or zero component analysis [25, 18, 44]. However, linear normalization methods might not suffice when the original data distribution is highly non-Gaussian. For tabular and time series data, a popular approach to Gaussianize the marginal distributions is to apply feature-wise non-linear transformations. Two commonly-used parametric methods are the Box-Cox [4] transformation and its extension, the Yeo-Johnson (YJ) transformation [50]. Both have been used in multiple applications, such as climate and weather forecast [51, 48, 49], economics [12] and genomic studies [6, 56, 8].

Problem and contributions In this paper, we investigate the problem of data normalization in the cross-silo FL setting, by exploring how to apply the YJ transformation to a distributed dataset. This problem arises frequently in medical cross-silo FL, e.g. when trying to jointly train models on genetic data (see e.g. [17, 55]). Due to data heterogeneity, no single client can act as a reference client: indeed, there is no guarantee that transformation parameters fitted on a single client would be relevant for other clients' data. Hence, it is necessary to fit normalization methods on the full federated dataset. Moreover, in this setting, data privacy is of paramount importance, and therefore FL protocols should be carefully designed. Our main contributions to this problem are as follows:

1. We prove that the negative YJ log-likelihood is convex (Section 3), which is a novel result, to the best of our knowledge.
2. Building on this property, we introduce ExpYJ, a method to fit the YJ transformation based on exponential search (Section 3). We numerically show that this method is more stable than standard approaches for fitting the YJ transformation based on the Brent minimization method [5].
3. We propose SecureFedYJ (Section 4), a secure way to extend ExpYJ in the cross-silo FL setting using SMC. We show that SecureFedYJ does not leak any information on the datasets apart from what is leaked by the parameters minimizing the YJ negative log-likelihood (Section 4 and Proposition 4.1). By construction, SecureFedYJ provides the same results as the pooled-equivalent ExpYJ, regardless of how the data is split across the clients. We check this property in numerical experiments (Section 4). The core ideas behind the resulting algorithm, SecureFedYJ, are summarised in Figure 4.

Finally, we illustrate our contributions in numerical applications on synthetic and genomic data in Section 5.

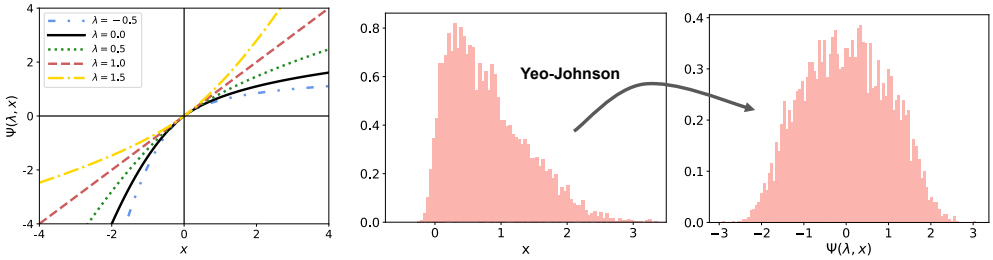
2 Background

The Yeo-Johnson transformation The YJ transformation [50] was introduced in order to Gaussianize data that can be either positive or negative. It was proposed as a generalization of the Box-Cox transformation [4], that only applies to non-negative data. The YJ transformation consists in applying to each feature a monotonic function $\Psi(\lambda, \cdot)$ parametrized by a scalar λ, independently of the other features. Thus, there are as many λ's as there are features. For a real number x, $\Psi(\lambda, x)$ is defined as:

$$\Psi(\lambda, x) = \begin{cases} [(x+1)^{\lambda} - 1]/\lambda, & \text{if } x \geq 0, \lambda \neq 0, \\ \ln(x+1), & \text{if } x \geq 0, \lambda = 0, \\ -[(-x+1)^{2-\lambda} - 1]/(2-\lambda), & \text{if } x < 0, \lambda \neq 2, \\ -\ln(-x+1), & \text{if } x < 0, \lambda = 2. \end{cases}$$

Figure 1a shows the shape of the YJ function for various values of λ.

The Yeo-Johnson likelihood Let us consider real-valued samples $\{x_i\}_{i=1,\cdots,n}$, and let us apply the YJ transformation $\Psi(\lambda, \cdot)$ to these samples to Gaussianize their distribution. The log-likelihood



(a) $\Psi(\lambda, \cdot)$ for various λ (b) The YJ transformation applied to a skew-normal distribution

Figure 1: The Yeo-Johnson transformation applies a 1-D univariate transform to Gaussianize data.

that $\{\Psi(\lambda, x_i)\}_{i=1,\cdots,n}$ comes from a Gaussian with mean μ and variance σ^2 is given by:

$$\log \mathcal{L}_{\mathrm{YJ}}(\lambda, \sigma^2, \mu) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left[\Psi(\lambda, x_i) - \mu\right]^2 + (\lambda - 1)\sum_{i=1}^{n}\mathrm{sgn}(x_i)\log(|x_i| + 1).$$

For a given λ, the log-likelihood is maximized for $\mu_* = \frac{1}{n}\sum_{i=1}^{n}\Psi(\lambda, x_i)$ and $\sigma_*^2 = \frac{1}{n}\sum_{i=1}^{n}(\Psi(x_i, \lambda) - \mu_*)^2$. Once we replace μ and σ^2 by μ_* and σ_*^2, it becomes:

$$\log \mathcal{L}_{\mathrm{YJ}}(\lambda) = -\frac{n}{2}\log(\sigma^2_{\Psi(\lambda,\{x_i\})}) + (\lambda - 1)\sum_{i=1}^{n}\mathrm{sgn}(x_i)\log(|x_i| + 1) - \frac{n}{2}\log(2\pi), \qquad (1)$$

see [50]. Maximizing the YJ log-likelihood is therefore a 1-dimensional problem for each feature. Once the optimal λ_* is found, the transformed data $\Psi(\lambda, x_i)$ is usually renormalized by subtracting its empirical mean μ_* and dividing by the square root of its empirical variance σ_*^2. Figure 1b shows an example of the YJ transformation applied to a skew-normal distribution. Note that in a typical application, the triplet $(\lambda_*, \mu_*, \sigma_*^2)$ is fitted on the training data only, and is then used to Gaussianize the test dataset during inference.

Minimization methods in dimension 1 As seen above, fitting a YJ transformation can be reduced to a 1D optimization problem. To tackle this problem, we introduce two standard 1D minimization methods: (i) Brent minimization [5] and (ii) exponential search [2].

Brent minimization [5] (not to be confused with the Brent-Dekker method, see [5], chapters 3 and 4) is a widely used method for 1D optimization. It is based on golden section search and successive parabolic interpolations, and does not require evaluating any derivatives. This algorithm is guaranteed to converge to a local minimum with superlinear convergence of order at least 1.3247. Standard implementations of the YJ transformation, in particular the *scikit-learn* implementation [34], are based on the Brent minimization method to minimize the negative log-likelihood provided by Equation (1).

Exponential search [2] is a dichotomic algorithm designed for unbounded search spaces. The idea is to first find bounds, and then to perform a classic binary search within these bounds. This algorithm can be used to find the minimum of convex differentiable functions with linear convergence, as explained in Appendix B. In this work, we build on exponential search to propose a federated version of the YJ transform, for two main reasons: (i) it is more numerically stable than Brent minimization, as shown in Section 3 and Figure 2, (ii), it may conveniently be adapted to a federated setting, as shown in Section 4, and (iii), this latter federated adaptation offers strong privacy garantees, as shown by Proposition 4.1.

Secure Multiparty Computation As illustrated by various privacy gradient attacks [54, 52], sensitive information on the clients' datasets can be leaked to the central server during an FL training. One way to mitigate this risk is to use Secure Multiparty Computation (SMC) protocols to hide individual contributions to the server. SMC enables one to evaluate functions with inputs distributed across different users without revealing intermediate results and is often based on secret sharing. SMC protocols tailored for ML use-cases have been recently proposed [11, 13, 32, 37, 46, 31, 47, 39]. These protocols are either designed to enhance the privacy of FL trainings, or to perform secure inference, i.e. to enable the evaluation of model trained privately on a server without revealing the data nor the model.

A popular FL algorithm relying on SMC is Secure Aggregation (SA) [3]. Schematically, in SA each client adds a random mask to their model update before sending it to the central server. These masks have been tailored in such a way that they all together sum to zero. Therefore, the central server cannot see the individual updates of the clients, but it can recover the sum of these updates by adding all the masked quantities sent from them.

More generally, an SMC routine schematically works as follows (we refer to Appendix D for further details). Let us consider the setting where K parties $k = 1, \dots, K$ want to compute $g = f(h^{(1)}, \dots, h^{(K)})$ for a known function f, where $(h^{(1)}, \dots, h^{(K)})$ denote private inputs. Each party k knows $h^{(k)}$ and is not willing to share it. During the first step, *secret sharing*, each party splits its private input $h^{(k)}$ into K secret shares $h_1^{(k)}, \dots, h_K^{(k)}$, and sends the shares $h_{k'}^{(k)}$ to the party k'. These secret shares are constructed in such a way that (i) knowing $h_{k'}^{(k)}$ does not provide any information on the value of $h^{(k)}$, and (ii) $h^{(k)}$ can be reconstructed from the vector $(h_1^{(k)}, \dots, h_K^{(k)})$. For simplicity, we denote $[\![h^{(k)}]\!] = (h_1^{(k)}, \dots, h_K^{(k)})$ the vector of share secrets. In a second step, *computation*, each party k' performs local operations involving only the secret shares they know to compute $g_{k'} = \tilde{f}(h_{k'}^{(1)}, \dots, h_{k'}^{(K)})$. The function $\tilde{f}$ depends on f and on the SMC protocol that is used, and is chosen so that $g = f(h^{(1)}, \dots, h^{(K)})$ can be reconstructed from $(g_1, \dots g_K)$. Said otherwise, $g_{k'}$ are secret shares of g: $[\![g]\!] = (g_1, \dots g_K)$. Finally, during the *reveal* step, each party k reveals g_k to all other parties, and each party can reconstruct g from $(g_1, \dots g_K)$.

Threat model In this work, we consider a setting where all participants are honest-but-curious [33]. Neither the clients nor the central server will deviate from the agreed protocol, but each participant can potentially try to infer as much information as possible using data they see during the protocol.

3 A novel method to optimize the Yeo-Johnson log-likelihood: ExpYJ

In this section, we leverage the convexity of the negative log-likelihood of the YJ transformation (see Proposition 3.1) to propose a new method to find the optimal λ_* using exponential search. While this method only offers linear convergence, compared to the super-linear convergence of Brent minimization method, we demonstrate two of its advantages: (i) it is more numerically stable, and (ii) it is easily amenable to an FL setting with strong privacy guarantees. The method proposed in this section is based on the following result.

Proposition 3.1. *The negative log-likehood* $\lambda \mapsto -\log \mathcal{L}_{\mathrm{YJ}}(\lambda)$ *(1) is strictly convex.*

Algorithm 1 ExpYJ

Input: data x_i, total data size n, number of steps $t_{\max}$
Initialize $\lambda_{t=0} \leftarrow 1$, $\lambda_{t=0}^+ \leftarrow \infty$, $\lambda_{t=0}^- \leftarrow -\infty$
Compute S_φ
for $t = 1$ **to** $t_{\max}$
  for $g \in \{\Psi(\lambda, \cdot), \Psi(\lambda, \cdot)^2, \partial_\lambda \Psi(\lambda, \cdot), \partial_\lambda \Psi(\lambda, \cdot)^2\}$
    Compute S_g
  end for
  $\Delta_t = \mathrm{sgn}\left[n S_{\partial \Psi^2} - 2 S_\Psi S_{\partial \Psi} - 2 S_\varphi \left(S_{\Psi^2} - \frac{S_\Psi^2}{n}\right)\right]$
  $\lambda_t, \lambda_t^-, \lambda_t^+ \leftarrow \mathrm{ExpUpdate}(\lambda_{t-1}, \lambda_{t-1}^-, \lambda_{t-1}^+, \Delta_t)$
end for
$\lambda_* \leftarrow \lambda_{t_{\max}}$
Compute $\mu_* = S_\Psi / n$ and $\sigma_*^2 = S_{\Psi^2}/n - \mu_*^2$
Output: The fitted triplet $(\lambda_*, \mu_*, \sigma_*^2)$

The proof of Proposition 3.1 builds upon the work of [24] which shows that the negative log-likelihood of the Box-Cox transformation [4] is convex. The complete proof is deferred to Appendix C.

The exponential YJ algorithm The pseudo-code of the proposed algorithm is presented in Algorithm 1, and relies on the exponential search presented in Algorithm 2 (cf Appendix B for more details on exponential search). An illustration of ExpYJ is shown in Figure 7 in Appendix B. Due to the strict convexity of the negative log-likelihood of the YJ transformation, we may perform the exponential search described in Section 2 and Appendix B. To do so, it is enough to obtain the sign of the derivative. Let $\partial_\lambda \Psi(\lambda, \cdot)^2 = 2\Psi(\lambda, \cdot)\partial_\lambda \Psi(\lambda, \cdot)$ and $\varphi(x) = \mathrm{sgn}(x) + \log(|x| + 1)$. Further, for $g \in \{\Psi(\lambda, \cdot), \partial_\lambda \Psi(\lambda, \cdot), \Psi(\lambda, \cdot)^2, \partial_\lambda \Psi(\lambda, \cdot)^2, \varphi\}$, let us define $S_g \stackrel{\text{def}}{=} \sum_{i=1}^n g(x_i)$. The derivative of the log-likelihood is available in closed form (see Appendix A.2):

$$\partial_\lambda \log \mathcal{L}_{\mathrm{YJ}} = \frac{n}{2} \frac{S_{\partial \Psi^2} - 2(S_\Psi S_{\partial \Psi})/n}{S_{\Psi^2} - S_\Psi^2/n} - S_\varphi.$$

Notice that $S_{\Psi^2} - S_\Psi^2/n$ can be expressed as a variance, hence is non-negative. We may therefore obtain $\operatorname{sgn}[\partial_\lambda \log \mathcal{L}_{\mathrm{YJ}}]$ while avoiding performing division by computing

$$\operatorname{sgn}[\partial_\lambda \log \mathcal{L}_{\mathrm{YJ}}] = \operatorname{sgn}\left[nS_{\partial\Psi^2} - 2S_\Psi S_{\partial\Psi} - 2S_\varphi(S_{\Psi^2} - S_\Psi^2/n)\right]. \tag{2}$$

Avoiding this division is crucial to make the overall procedure more numerical stable, as explained below, and eases the use of SMC routines.

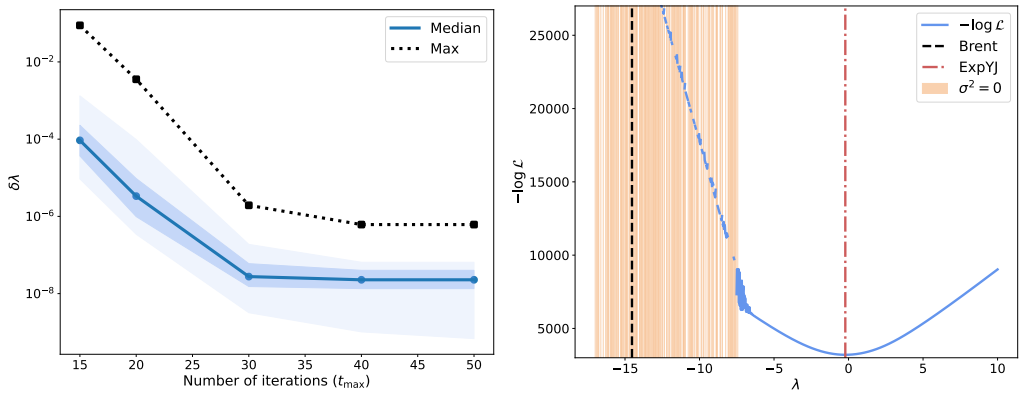


Figure 2: Comparison of ExpYJ and *scikit-learn*. **Left**: For each of the 106 features (see Appendix E.1), we compute the relative difference $\delta\lambda = |\lambda_{\mathrm{ExpYJ}} - \lambda_{\mathrm{sk}}|/|\lambda_{\mathrm{sk}}|$ and plot its median, maximum and 25%-75% and 10%-90% percentiles across the 106 features. **Right**: Negative log-likelihood of the YJ transformation for the mean area of the cell of each sample of the *Breast Cancer* dataset. Full orange bars correspond to values of λ for which the likelihood computing using scikit-learn returns ∞ as $\sigma_\lambda^2(\{x_i\})$ is equal to 0 up to float-64 machine precision. Dotted lines correspond to the λ_* found using Brent minimization or ExpYJ with one client.

Accuracy of ExpYJ We check the accuracy of ExpYJ on the datasets presented in Appendix E.1. In particular, we compare the results provided by ExpYJ with the outputs of the *scikit-learn* algorithm based on Brent minimization.

For 2 of the 108 features present in the datasets, the *scikit-learn* implementation leads to numerical instabilities discussed hereafter. Therefore, we focus our comparison on the 106 remaining features, that we aggregated regardless of the dataset. Figure 2 reports the relative difference $\delta\lambda$ between the results obtained by ExpYJ and by the *scikit-learn* implementation as a function of the number of iteration $t_{\max}$ (as defined in Algorithm 1). These results show that this relative difference is of order less than 10^{-6} when $t_{\max} = 40$.

Algorithm 2 ExpUpdate

Input: $\lambda, \lambda^+, \lambda^-, \Delta \in \{-1, 1\}$
  if $\Delta = 1$ **then**
    $\lambda^- \leftarrow \lambda$
    $\lambda \leftarrow (\lambda^+ + \lambda)/2$ **if** $\lambda^+ < \infty$
    else $\lambda \leftarrow \max(2\lambda, 1)$
  else
    $\lambda^+ \leftarrow \lambda$
    $\lambda \leftarrow (\lambda^- + \lambda)/2$ **if** $\lambda^- > -\infty$
    else $\lambda \leftarrow \min(2\lambda, -1)$
  end if
Output: Updated $\lambda, \lambda^+, \lambda^-$

Numerical stability of ExpYJ Our experiments demonstrate that ExpYJ is numerically more stable than Brent minimization. Indeed, for some values of λ and some datasets $\{x_i\}$, the transformation $\Psi(\lambda, \cdot)$ concentrates all data points in a small interval such that the values of $\Psi(\lambda, \{x_i\})$ are all equal up to machine precision. In that case, the log-likelihood is not well-defined and the term $\log \sigma_{\Psi_\lambda}^2$ takes the value $-\infty$, which prevents Brent minimization from converging. This phenomenon does not affect the ExpYJ routine as we do not compute directly the sign of $\partial_\lambda \mathcal{L} = \partial_\lambda \sigma_{\Psi_\lambda}^2 / \sigma_{\Psi_\lambda}^2 - \sum_i \varphi(x_i)$, but rather the sign of $\sigma_{\Psi_\lambda}^2 \partial_\lambda \mathcal{L} = -\partial_\lambda \sigma_{\Psi_\lambda}^2 - \sigma_{\Psi_\lambda}^2 \sum_i \varphi(x_i)$, see Eq. (2).

Figure 2 illustrates this in the case of a feature of the *Breast Cancer Dataset*. The λ_* returned by the Brent minimization method of *scikit-learn* is -14.53 while the minimizer of the negative log-likelihood found by the ExpYJ is -0.21. In particular, Figure 2 shows the values of the negative log-likelihood as a function of λ computed using 64-bit float precision. The orange vertical full bands correspond to values for which $\sigma_{\Psi_\lambda}^2$ is zero within the machine precision, resulting to a negative log-likelihood of ∞. This instability happens for 2 of the 108 features used in numerical experiments,

where blindly applying the Brent-based YJ transformation leads to all data points collapsing to zero, while EXPYJ succeeds in transforming the data distributions to more Gaussian-like ones.

4 Applying the Yeo-Johnson transformation in FL

So far, we only considered the centralized setting, where data is accessible from a single server. Yet, as mentioned in Sections 1 and 2, many real-world situations require working with heterogeneous data split between different centers, and to take privacy constraints into account. When the data is split across centers $k = 1, \dots, K$ and the function to optimize is separable, i.e. of the form $\mathcal{F}(\lambda) = \sum_{k=1}^{K} f_k(\lambda)$ where each f_k can be computed from data present in the center k exclusively, Federated Learning techniques were recently proposed. In short, they consist in repeatedly performing a few rounds of local optimization in each center, before aggregating local parameters in the server. We refer to [21] for an overview of recent advances in FL. In our case, however, the YJ negative log-likelihood (1) is not separable, due to the log-variance term. Indeed, turning the variance into a separable term would require sharing the global YJ mean μ_λ to all centers at each iteration. Compared to the method we propose in this section, this would lead to more privacy leakage.

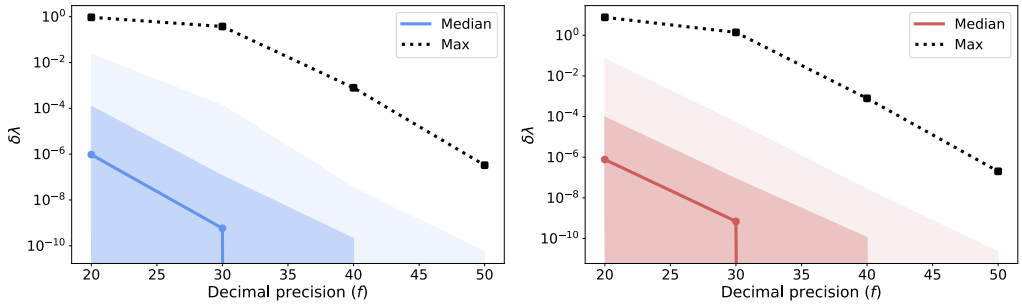
We now introduce SECUREFEDYJ, a secure federated algorithm that builds upon EXPYJ to apply YJ transformations. This algorithm satisfies the two following properties: (i) it is *pooled-equivalent*, i.e. it yields the same results as if the data were freely accessible from a single server, and (ii) it leaks as little information as possible about the underlying datasets, as shown by Proposition 4.1. Finally, it converges in a limited number of iterations, thanks to the linear convergence of the underlying exponential search.

SECUREFEDYJ SECUREFEDYJ is a federated adaptation of EXPYJ presented in Section 3 to find the best parameters $(\lambda_*, \mu_*, \sigma_*^2)$ of the YJ transformation when training datasets are split across different clients. It relies on SMC to ensure that only the final triplet $(\lambda_*, \mu_*, \sigma_*^2)$ fitted on the training datasets is revealed, without leaking any other information apart from the overall total number of training samples n. Indeed, at each intermediate step, only the sign of $\partial_\lambda \log \mathcal{L}_{\mathrm{YJ}}$ is revealed, and the mean and variance of the transformed data is only revealed at the last step. The pseudo-code of the resulting algorithm is presented in Algorithm 3, and relies on the exponential search presented in Algorithm 2. A functional representation of SECUREFEDYJ is displayed Figure 4.

In Algorithm 3,we label the clients by $k = 1, \dots, K$ and each client k holds data $\{x_{k,i} : i = 1, \dots, n_k\}$. We suppose that the total number of samples $n = \sum_{k=1}^{K} n_k$ is public and shared to all clients. For a given function g, we denote $S_{k,g}$ the sum $S_{k,g} \stackrel{\text{def}}{=} \sum_{i=1}^{n_k} g(x_{k,i})$. As introduced in Section 2, we use double brackets $[\![\cdot]\!]$ to indicate an SMC secret shared across the clients (see Appendix D for more details).

Algorithm 3 SECUREFEDYJ

Input: Data $\{x_{k,i}\}$, total data size n, number of steps $t_{\max}$
  Notations: $[\![\cdot]\!]$ indicates a SMC secret shared across the clients (see Appendix D) .
  Initialize $\lambda_{t=0} \leftarrow 0$, $\lambda^+ \leftarrow \infty$, $\lambda^- \leftarrow -\infty$
  Compute $[\![S_\varphi]\!] = \sum_k [\![S_{k,\varphi}]\!]$
  for $t = 1$ **to** $t_{\max}$
    for $g \in \{\Psi(\lambda, \cdot), \Psi(\lambda, \cdot)^2, \partial_\lambda \Psi(\lambda, \cdot), \partial \Psi(\lambda, \cdot)^2\}$
      Compute $[\![S_g]\!] = \sum_k [\![S_{k,g}]\!]$,
    end for
    $[\![\Delta_t]\!] = \operatorname{sgn}\left[n[\![S_{\partial\Psi^2}]\!] - 2[\![S_\Psi]\!][\![S_{\partial\Psi}]\!] - 2[\![S_\varphi]\!]([\![S_{\Psi^2}]\!] - [\![S_\Psi]\!]^2/n)\right]$
    Reveal Δ_t to each client
    $\lambda_t, \lambda_t^-, \lambda_t^+ \leftarrow$ EXPUPDATE$(\lambda_{t-1}, \lambda_{t-1}^-, \lambda_{t-1}^+, \Delta_t)$ independently on each client
  end for
  $\lambda_* \leftarrow \lambda_{t_{\max}}$
  Compute $[\![\mu]\!] = \sum_k [\![S_{k,\Psi}]\!]/n$ and $[\![\sigma^2]\!] = \sum_k [\![S_{k,\Psi^2}]\!]/n - [\![\mu^2]\!]$
  Reveal $\mu_* \leftarrow \mu$ and $\sigma_*^2 \leftarrow \sigma^2$
Output: The fitted triplet $(\lambda_*, \mu_*, \sigma_*^2)$



(a) Homogeneous (b) Heterogeneous

Figure 3: Comparison of SECUREFEDYJ and EXPYJ for various fixed-point floating precisions f used in SMC, with $l = f + 50$ and $t_{\max} = 40$. The data is distributed across 10 clients, either randomly (homogeneous, *left*), or per decile (heterogeneous, *right*, i.e. each client gets one decile of the data). We report the maximum, median, 25%-75% and 10%-90% percentiles of the relative error $\delta\lambda = |\lambda_{\text{SECFEDYJ}} - \lambda_{\text{EXPYJ}}|/|\lambda_{\text{EXPYJ}}|$ across the 108 features described in Appendix E.1.

Privacy leakage In Proposition 4.1 we show that Algorithm 3 only reveals information already contained in the fitted triplet $(\lambda_*, \mu_*, \sigma_*^2)$. In comparison, turning the YJ negative log-likelihood (1) into a log-separable function before using off-the-shelf FL methods would require sharing μ and its gradient and centrally computing σ^2 for intermediate values of λ at each iteration. This could potentially lead to uncontrolled privacy leakage.

Proposition 4.1. *The fitted parameter λ_* contains all the information revealed during the intermediate steps of* SECUREFEDYJ. *More precisely, there exists a deterministic function $\mathcal{F}$ such that for any set of datasets $\{x_{k,i}\}$, if $\lambda_*(\{x_{k,i}\})$ is the result of* SECUREFEDYJ *on $\{x_{k,i}\}$, then $\{\lambda_t, \lambda_t^+, \lambda_t^-, \Delta_t\}_{t=1,\cdots,t_{\max}} = \mathcal{F}\left[\lambda_*(\{x_{k,i}\})\right]$*

Proof. This proposition comes from the fact that all gradient signs Δ_t revealed during the algorithm can be retrospectively inferred from λ_*. Indeed, $\partial_\lambda \log \mathcal{L}_{\text{YJ}} < 0$ for $\lambda > \lambda_*$ and $\partial_\lambda \log \mathcal{L}_{\text{YJ}} > 0$ for $\lambda < \lambda_*$. Besides, the successive values of λ_t explored at each step t can be deterministically inferred from the initial value $\lambda_{t=0}$ and and the final fitted value λ_*. ■

Performance of SECUREFEDYJ We implement SECUREFEDYJ in Python, using the MPyC library [40] based on Shamir Secret Sharing [42]. We refer to Appendix D for more details on our implementation. To represent signed real-valued numbers in an SMC protocol, we use a fixed-point representation (see Appendix D.2) using l bits, among which f bits are used for the decimal parts. This means that we consider floats ranging from -2^{l-f} to 2^{l-f} and that we have an absolution precision of 2^{-f} in our computations.

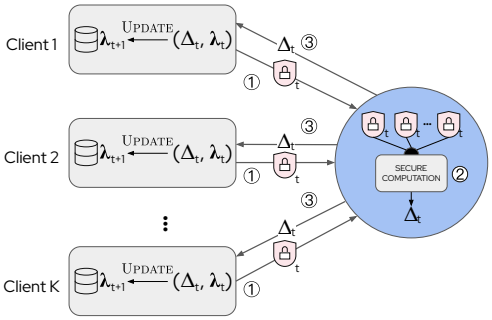


Figure 4: Simplified view of one round of SECUREFEDYJ. ①: All clients compute local, data-dependent quantities. ②: Δ_t is computed using SMC. Data-dependent quantities computed by each client are not disclosed during the process. ③: Δ_t is disclosed to all the clients, a new value λ_{t+1} is computed using an exponential update.

In order to ensure the accuracy of SECUREFEDYJ results, we need to make sure that l and f are large enough. Figure 3 shows the accuracy of SECUREFEDYJ when compared to EXPYJ for various values of f. According to these numerical experiments, taking $f = 50$ and $l = 100$ provides reasonably accurate results. Moreover, by construction, the outputs of SECUREFEDYJ do not depend on how the data is split across the clients, up to rounding numerical errors. Therefore this algorithm is resilient to data heterogeneity, as long as the numerical decimal precision f is large enough, as shown in Figure 3.

Performing SECUREFEDYJ with $t_{\max} = 40$ takes 726 rounds of communication (see Appendix D.4). During these communication rounds, each client sends overall about 8 Mb per feature to every other

client (see also Appendix D.4). SECUREFEDYJ can be applied independently and in parallel to each feature. Therefore, the overall number of rounds does not depend on the number of features being considered, and the communication costs grow proportionally to the number of features. In a realistic cross-silo FL setting as described in [17], the bandwidth of the network is 1 Gb per second with a delay of 20 ms between every two clients. In this context, the execution of SECUREFEDYJ with $t_{\max} = 40$ on p features would take about 726×20 ms $\simeq 15$ s due to the communication overhead, in addition to $p \times 8$ Mb/1 Gbps $\simeq 8p$ ms due to the bandwidth. This shows that SECUREFEDYJ is indeed a viable algorithm in a real-world scenario.

5 Applications

Genomic data: TCGA We start by showing the benefits of YJ preprocessing in survival analysis experiments on lung (LUAD+LUSC), pancreas (PAAD), and colorectal (CRC) cancers. Given gene expression raw counts (features) and censored survival data (responses) from patients having either of those three cancers, we aim to fit a Cox Proportional Hazards (CoxPH) model [9] with the highest possible concordance index (C-index) [23], which measures how well patients are ranked with respect to their survival times. We refer to [23] for a more thorough introduction to survival analysis. In Figure 5, we compare three different preprocessing methods: (i) whitening, (ii) log normalization, and (iii) YJ, each followed by a PCA dimensionality reduction step. More precisely, whitening (i) consists in centering and reducing to unit variance the total read counts of all genes across all samples, log normalization consists in applying $u \mapsto \log(1+u)$ to raw read counts before applying global whitening, and YJ is a global YJ transform on the total read counts. We then evaluate each strategy using 5-fold cross-validation with 5 different seeds. We refer to Appendix E.2 for experimental details. While this experiment is performed in a pooled environment, note that, importantly, each step has a federated pooled-equivalent version: apart from the proposed SECUREFEDYJ for YJ, see e.g. [19] for PCA, and Webdisco [29] for Cox model fitting. This simplified setting allows us to understand the importance of the Yeo-Johnson transformation in an ideal setting, independently of other potential downstream federated learning artifacts.

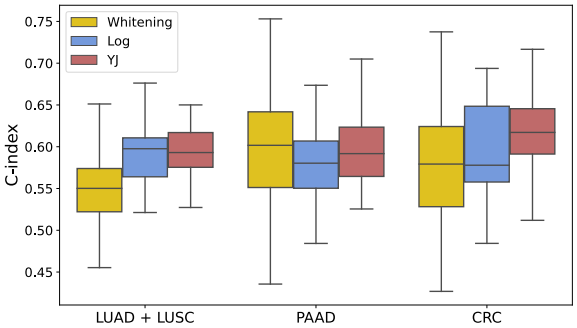


Figure 5: Cross-validation survival analysis performance (higher is better) of a CoxPH model with different normalization methods. The YJ transformation yields either a better or on par performance, and further stabilizes results compared to other approaches.

In Figure 5, we see that YJ is better or on par with the best method for each cancer: YJ improves prediction results for colorectal cancer, while yielding results which are on par with the best results for lung and pancreas cancers, with a smaller variance.

Synthetic data We show how applying YJ may help improving performance compared to no or basic preprocessing, and how SECUREFEDYJ yields improvements compared to local YJ transforms in federated linear regression. To do so, we generate covariates $\widetilde{X}$ and responses y as

$$\begin{aligned} \widetilde{X} &= (\exp(x_1), \exp(x_2+2), \sigma(x_3)) \text{ with } X = (x_1, x_2, x_3) \sim \mathcal{N}(0, \mathbf{I}_3), \\ y &= \beta^T X + \varepsilon \text{ with } \varepsilon \sim \mathcal{N}(0, 0.1), \end{aligned} \tag{3}$$

where $\sigma(\cdot)$ is the sigmoid function and $\beta = (-1.3, 2.4, 0.87)$ was randomly chosen. The goal is then to fit a linear model from i.i.d. samples $(\widetilde{X}_i, y_i), i = 1, \dots, n$ following (3), after an optional preprocessing step. Similarly to the previous example, we simulate a cross-device FL setting only for the preprocessing steps, and the linear model is then fitted in a pooled setting for simplicity. All the details of the numerical experiments are provided in Appendix E.3. We suppose that the samples $(\widetilde{X}_i, y_i), i = 1, \dots, n$ are homogeneously split across 10 centers. The responses y_i have a highly nonlinear dependency on the covariates $\widetilde{X}_i$, but depend linearly on the X_i's (up to Gaussian noise), which are not observed. Hence, we expect that applying a suitable preprocessing step before

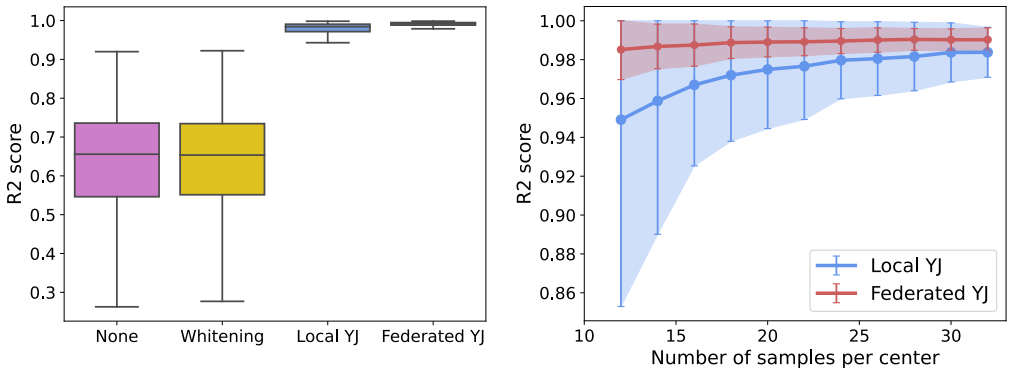


Figure 6: Comparison of different preprocessing methods for linear regression on synthetic federated data. **Left**: performance with 200 samples (20 on each of the 10 centers) on 1000 independent draws, showing the interest of using YJ preprocessing. The R2 score of the models are computed on another dataset of 200 samples not seen during the training. **Right**: Comparison of local and federated YJ over 1000 independent draws. In local YJ, a single center is randomly chosen to fit λ, which is then shared with other centers, to ensure that the same transformation is applied everywhere. Full lines correspond to the means, error bars to $\pm$ std of the R2 scores of the model on an unseen test dataset.

training a linear model will transform back the $\widetilde{X}_i$'s into the normally-distributed X_i's and lead to a high performance, compared to no transformation. The results of our experiments are summarized in Figure 6. The left figure shows that the YJ transformation is indeed capable of roughly inverting the $\widetilde{X}_i$'s into the X_i's, yielding a major improvement compared to no preprocessing or standard centering and reduction to unit variance. Besides, the right figure shows that even in this homogeneous setting where the data is i.i.d. across centers, using a federated version of YJ compared to a local version of YJ leads to better average performance, and reduced variance.

6 Conclusion

Summary of our contributions In this work, we introduce SECUREFEDYJ, a method to fit a YJ transformation on data shared by different clients in a cross-silo setting. SECUREFEDYJ is an SMC version of its pooled equivalent EXPYJ which builds upon the convexity of the negative log-likelihood of the YJ transformation, a novel result introduced by this work, and on the fact that the sign of its derivative can be computed in a stable way. We show that SECUREFEDYJ has the same accuracy as a standard YJ transformation on pooled data. In particular, the results do not depend on how the data is split across the clients, making SECUREFEDYJ resilient to data heterogeneity. Besides, the quantities disclosed by SECUREFEDYJ during the training to the central server do not leak any other information than what is contained in the final parameters $(\mu_*, \lambda_*, \sigma_*^2)$.

Limitations and future work While Brent minimization has a super-linear convergence, our approach only has a linear convergence, as it relies on exponential search. This can be an issue if the communication costs between the clients and the server are high. Acceleration could be achieved by either adapting Brent minimization to a cross-silo setting, or applying a second-order method. We leave the development of a faster accurate SMC method using either of those two approaches to future work.

A further possible improvement would be to make our approach differentially private (DP) [15], by adding Gaussian or Laplacian noise to each sample's features when computing the S_g terms. However it is unclear to what extent the noise would impact the computation of the sign, and therefore the Gaussianization process as a whole.

Finally, we only consider an *honest-but-curious* setting where participants do not deviate from the agreed protocol and only try to infer as much information as possible from the data seen during SECUREFEDYJ. We do not explore the threat of a malicious user (either a malicious client or a malicious central server) that would purposely deviate from the protocol to either gain more information or to jeopardize the convergence.

References

[1] Mathieu Andreux, Jean Ogier du Terrail, Constance Beguier, and Eric W Tramel. Siloed federated learning for multi-centric histopathology datasets. In *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*, pages 129–139. Springer, 2020.

[2] Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information processing letters*, 5(SLAC-PUB-1679), 1976.

[3] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[4] George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

[5] Richard P Brent. *Algorithms for Minimization Without Derivatives*. Prentice Hall, 1973.

[6] Franziska S Brunner, Paul Schmid-Hempel, and Seth M Barribeau. Immune gene expression in bombus terrestris: signatures of infection despite strong variation among populations, colonies, and sister workers. *PloS one*, 8(7):e68181, 2013.

[7] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *International Conference on Financial Cryptography and Data Security*, pages 35–50. Springer, 2010.

[8] Li-Chu Chien. A rank-based normalization method with the fully adjusted full-stage procedure in genetic association studies. *PloS one*, 15(6):e0233847, 2020.

[9] David R Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 1972.

[10] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *International workshop on public key cryptography*, pages 160–179. Springer, 2009.

[11] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

[12] Thiago Alexandre das Neves Almeida, Luís Cruz, Eduardo Barata, and Isabel-María García-Sánchez. Economic growth and environmental impacts: An analysis based on a composite index of environmental damage. *Ecological Indicators*, 76:119–130, 2017.

[13] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY-A framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[14] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[15] Cynthia Dwork. Differential privacy. In *International Colloquium on Automata, Languages, and Programming*, pages 1–12. Springer, 2006.

[16] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[17] David Froelicher, Juan R Troncoso-Pastoriza, Jean Louis Raisaro, Michel Cuendet, Joao Sa Sousa, Jacques Fellay, and Jean-Pierre Hubaux. Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption. *Nature Communications*, 12 (1):5910, 2021.

[18] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *International Conference on Machine Learning*, pages 1319–1327. PMLR, 2013.

[19] Andreas Grammenos, Rodrigo Mendoza Smith, Jon Crowcroft, and Cecilia Mascolo. Federated principal component analysis. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6453–6464. Curran Associates, Inc., 2020.

[20] Ian Jolliffe. Principal component analysis. *Encyclopedia of statistics in behavioral science*, 2005.

[21] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[22] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143. PMLR, 2020.

[23] David G Kleinbaum and Mitchel Klein. *Survival analysis*, volume 3. Springer, 2010.

[24] Elies Kouider and Hanfeng Chen. Concavity of Box-Cox log-likelihood function. *Statistics and probability letters*, 25(2):171–175, 1995.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[26] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[27] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020.

[28] Thomas Lorünser and Florian Wohner. Performance comparison of two generic MPC-frameworks with symmetric ciphers. In *ICETE (2)*, pages 587–594, 2020.

[29] Chia-Lun Lu, Shuang Wang, Zhanglong Ji, Yuan Wu, Li Xiong, Xiaoqian Jiang, and Lucila Ohno-Machado. Webdisco: a web service for distributed cox model learning without patient-level data sharing. *Journal of the American Medical Informatics Association*, 22(6):1212–1219, 2015.

[30] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[31] Payman Mohassel and Peter Rindal. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 35–52, 2018.

[32] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE, 2017.

[33] Andrew Paverd, Andrew Martin, and Ian Brown. Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep*, 2014.

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

[35] Tord Ingolf Reistad. A general framework for multiparty computations. 2012.

[36] Tord Ingolf Reistad and Tomas Toft. Secret sharing comparison by transformation and rotation. In *International Conference on Information Theoretic Security*, pages 169–180. Springer, 2007.

[37] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 707–721, 2018.

[38] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7, 2020.

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

[40] Berry Schoenmakers. MPyC—python package for secure multiparty computation. In *Workshop on the Theory and Practice of MPC.*, 2018. URL https://github.com/lschoe/mpyc.

[41] Muhammad Shafiq-ul Hassan, Geoffrey G Zhang, Kujtim Latifi, Ghanim Ullah, Dylan C Hunt, Yoganand Balagurunathan, Mahmoud Abrahem Abdalah, Matthew B Schabath, Dmitry G Goldgof, Dennis Mackin, et al. Intrinsic dependencies of CT radiomic features on voxel size and number of gray levels. *Medical physics*, 44(3):1050–1062, 2017.

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

[43] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1310–1321, 2015.

[44] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[45] David Tellez, Geert Litjens, Péter Bándi, Wouter Bulten, John-Melle Bokhorst, Francesco Ciompi, and Jeroen van der Laak. Quantifying the effects of data augmentation and stain color normalization in convolutional neural networks for computational pathology. *Medical image analysis*, 58:101544, 2019.

[46] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: Efficient and private neural network training. *IACR Cryptol. ePrint Arch.*, 2018:442, 2018.

[47] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. FALCON: Honest-majority maliciously secure framework for private deep learning. *arXiv preprint arXiv:2004.02229*, 2020.

[48] QJ Wang and DE Robertson. Multisite probabilistic forecasting of seasonal flows for streams with zero value occurrences. *Water Resources Research*, 47(2), 2011.

[49] QJ Wang, Andrew Schepen, and David E Robertson. Merging seasonal rainfall forecasts from multiple statistical models through bayesian model averaging. *Journal of Climate*, 25(16): 5524–5537, 2012.

[50] In-Kwon Yeo and Richard A Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.

[51] Lina Zhang, Bizheng Wang, and Qingcun Zeng. Impact of the Madden–Julian oscillation on summer rainfall in southeast China. *Journal of Climate*, 22(2):201–216, 2009.

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

[53] Wenting Zheng, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 724–738. IEEE, 2019.

[54] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[55] Olga Zolotareva, Reza Nasirigerdeh, Julian Matschinske, Reihaneh Torkzadehmahani, Mohammad Bakhtiari, Tobias Frisch, Julian Späth, David B Blumenthal, Amir Abbasinejad, Paolo Tieri, et al. Flimma: a federated and privacy-aware tool for differential gene expression analysis. *Genome biology*, 22(1):1–26, 2021.

[56] Isabella Zwiener, Barbara Frisch, and Harald Binder. Transforming RNA-Seq data to improve the performance of prognostic gene signatures. *PloS one*, 9(1):e85150, 2014.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We described them in Section 6
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] The full proof of the main theoretical results, i.e. the convexity of the Yeo-Johnson negative log-likelihood (Proposition 3.1) is provided in Appendix C.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code of the experiments is not provided, but a detailed pseudo-code of the newly proposed algorithms are provided.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We specified all the hyperparameters and the details of the numerical experiment in Appendix E.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Standard deviations or quantiles of the results with respect to the seed are provided (cf plots)
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] The experiment are not heavy and run easily on a personal computer, on a CPU
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] The datasets used are open datasets available online and are systematically cited.
   (b) Did you mention the license of the assets? [Yes] The licence of the datasets used are provided in Appendix E.1
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We are using open datasets available online. The genomic dataset from TCGA have been previously anonymised by its creator before publication
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]