Scalable Privacy-Preserving Distributed Learning

David Froelicher, Juan R. Troncoso-Pastoriza, Apostolos Pyrgelis, Sinem Sav,
Joao Sa Sousa, Jean-Philippe Bossuat and Jean-Pierre Hubaux

Abstract

In this paper, we address the problem of privacy-preserving distributed learning and the evaluation of machine-learning models by analyzing it in the widespread MapReduce abstraction that we extend with privacy constraints. We design SPINDLE (Scalable Privacy-preservINg Distributed LEarning), the first distributed and privacy-preserving system that covers the complete ML workflow by enabling the execution of a cooperative gradient-descent and the evaluation of the obtained model and by preserving data and model confidentiality in a passive-adversary model with up to $N-1$ colluding parties. SPINDLE uses multiparty homomorphic encryption to execute parallel high-depth computations on encrypted data without significant overhead. We instantiate SPINDLE for the training and evaluation of generalized linear models on distributed datasets and show that it is able to accurately (on par with non-secure centrally-trained models) and efficiently (due to a multi-level parallelization of the computations) train models that require a high number of iterations on large input data with thousands of features, distributed among hundreds of data providers. For instance, it trains a logistic-regression model on a dataset of one million samples with 32 features distributed among 160 data providers in less than three minutes.In this paper, we address the problem of privacy-preserving distributed learning and the evaluation of machine-learning models by analyzing it in the widespread MapReduce abstraction that we extend with privacy constraints. We design SPINDLE (Scalable Privacy-preservINg Distributed LEarning), the first distributed and privacy-preserving system that covers the complete ML workflow by enabling the execution of a cooperative gradient-descent and the evaluation of the obtained model and by preserving data and model confidentiality in a passive-adversary model with up to $N-1$ colluding parties. SPINDLE uses multiparty homomorphic encryption to execute parallel high-depth computations on encrypted data without significant overhead. We instantiate SPINDLE for the training and evaluation of generalized linear models on distributed datasets and show that it is able to accurately (on par with non-secure centrally-trained models) and efficiently (due to a multi-level parallelization of the computations) train models that require a high number of iterations on large input data with thousands of features, distributed among hundreds of data providers. For instance, it trains a logistic-regression model on a dataset of one million samples with 32 features distributed among 160 data providers in less than three minutes.

I. Introduction

The training of machine-learning (ML) models usually requires large and diverse datasets [134]. In many domains, such as medicine and finance, assembling sufficiently large datasets has proven difficult [129] and often requires the sharing of data among multiple data-providers. This is particularly true in medicine, where patients' data are spread among multiple entities: For example, for rare diseases, one hospital might have only a few patients, whereas a medical study requires hundreds of them to produce significant results. Data sharing among many entities, which can be located in multiple countries, is hence required. However, when the data are sensitive and/or personal, they are particularly difficult to share. Data sharing is highly restricted by legal regulations, such as GDPR [43] in Europe. The financial and reputational consequences of a data breach often make the risk of data sharing higher than its potential benefits. Hence, it is often impossible to obtain sufficient data to train ML models that are key enablers in medical research [90], financial analysis [116], and many other domains.

To address this issue, privacy-preserving solutions are gaining interest as they can be key-enablers for ML with sensitive data. Many solutions have been proposed for secure predictions that use pre-trained models [12], [17], [45], [61], [78], [106], [105], [107]. However, the secure training of ML models, which is much more computationally demanding, has been less studied. Some centralized solutions [7], [15], [23], [29], [50], [60], [63], [66] that rely on homomorphic encryption (HE) were proposed. They have the advantage of being straightforward to implement but require individual records to be transferred out of the control of their owners, which is often not possible, e.g., due to data protection legislation [77], [62]. Also, the data are moved to a central repository, which can become a single point of failure. Secure multiparty computation solutions (SMC) proposed for this scenario [3], [28], [42], [44], [57], [88], [96], often assume that a limited number of computing parties are honest-but-curious and non-colluding. These assumptions might not hold when the data are sensitive and/or when the parties have competing interests. In contrast, homomorphic encryption-based (HE) or hybrid (HE and SMC) solutions [41], [132] that assume a malicious threat model (e.g., Anytrust model [124]) focus on limited ML operations (e.g., the training of regularized linear models with low number of features) and are not quantum secure. Recent advances in quantum computing [56], [47], [93], [115], [128] have made this technology a potential threat, in a not-so-far future, for existing cryptographic solutions [89]. Google recently announced that they have reached "quantum-supremacy" [49]. Even though quantum computers are still far from being able to break state-of-the-art cryptoschemes, we note that certain data (e.g., genomics) remain sensitive over a long period and will be at risk in the future.

Finally, *federated learning*, a non-cryptographic approach for privacy-preserving training of ML models, has recently gained interest. The data remain under the control of their owners and a server coordinates the training by sending the model directly to the data owners, which then update the model with their data. The updated models from multiple participants are averaged to obtain the global model [82], [68]. Recent works have shown that sharing intermediate models with a coordinating server, or among the participants, can lead to various privacy attacks, e.g., extracting participants' inputs [54], [123], [133] or membership inference [84], [92]. To address these problems,

This work was partially supported by the grant #2017-201 of the Strategic Focal Area "Personalized Health and Related Technologies (PHRT)" of the ETH Domain.

D. Froelicher is with the Laboratory for Data Security and DeDiS Laboratory, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland, e-mail: david.froelicher@epfl.ch. Juan R. Troncoso-Pastoriza, Apostolos Pyrgelis, Sinem Sav, Joao Sa Sousa, Jean-Philippe Bossuat and Jean-Pierre Hubaux are with the Laboratory for Data Security, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland, e-mail: name.surname@epfl.ch.

multiple works [72], [112], [83] rely on a differentially private mechanism to obfuscate the intermediate values. However, this obfuscation decreases the data and model utility, whereas the training of accurate models requires high privacy budgets and the achieved privacy level remains unclear [58].

Existing cryptographic distributed solutions are practical with only a small number of parties and most of the aforementioned solutions focus either on training or on prediction. They neither consider the complete ML workflow nor enable the training of a model that remains secret and enables oblivious prediction on confidential data. In many cases, the trained model is as sensitive as the data on which it is trained, and the use of the model after the training has to be tightly controlled. ML is used in very competitive domains and a balance has to be found between collaboration and competition [90], [114], [116]. For example, entities that collaborate to train a ML model should equally benefit from the resulting model.

In this paper, we address the problem of privacy-preserving learning and prediction among multiple parties, i.e., data providers (DPs), that do not trust each other. To address this issue, we design a solution that uses the MapReduce abstraction [31] that is often used to define distributed ML tasks [27], [119]. MapReduce defines parallel and distributed algorithms in a simple and well-known abstraction: PREPARE (data preparation), MAP (distributed computations executed independently by multiple nodes or machines), COMBINE (combination of the MAP results, e.g., aggregation) and REDUCE (computation on the combined results). We build on and extend this abstraction to determine and delimit which information, e.g., MAP outputs, have to be protected to design a decentralized **privacy-preserving** system for ML training and prediction. The model is locally trained by the DPs (MAP) and the results are iteratively combined (COMBINE) to update the global model (REDUCE). We exploit the partitioned (distributed) data to enable DPs to keep control of their respective data, and we distribute the computation to provide an efficient solution for the training of ML models on confidential data. After the training, the model is kept secret from all entities and is obliviously and collectively used to provide predictions on confidential data that are known only to the entity requesting the prediction. We remark that differential-privacy-based federated-learning solutions [2], [22], [34], [55], [72], [112], [64], [100], [59], [118] follow the same model, i.e., they can be defined according to the MapReduce abstraction. However, most solutions introduce a trade-off between accuracy and privacy [58], and do not provide data and model confidentiality simultaneously. On the contrary, our solution uses a different paradigm in which, similarly to non-secure solutions, the accuracy is traded off with the performance (e.g., number of iterations), but not with privacy.

We propose SPINDLE (Scalable Privacy-preservINg Distributed LEarning), a system that enables the privacy-preserving, distributed (cooperative) execution of the widespread stochastic mini-batch gradient-descent (SGD) on data that are stored and controlled by multiple DPs. SPINDLE builds on a state-of-the-art multiparty, lattice-based, quantum-resistant cryptographic scheme to ensure data and model confidentiality, in the passive-adversary model in which all-but-one DPs can be dishonest. SPINDLE is meant to be a generic and widely-applicable system that supports the SGD-based training of many different ML models. This includes, but is not limited to, support vector machines, graphical models, generalized linear-models and neural networks [33], [48], [69], [117], [131]. For concreteness and comparison with existing works, we instantiate SPINDLE for the training of and prediction on generalized linear models (GLMs) [94], (e.g., linear, logistic and multinomial logistic regressions). GLMs are easily interpretable, capture complex non-linear relations (e.g., logistic regression), and are widely-used in many domains such as finance, engineering, environmental studies and healthcare [76].

In a realistic scenario where a dataset of 11,500 samples and 90 features is distributed among 10 DPs, SPINDLE efficiently trains a logistic regression model in less than 54 seconds, achieving an accuracy of 83.9%, equivalent to a non-secure centralized solution. The distribution of the workload enables SPINDLE to efficiently cope with a large number of DPs (parties), as its execution time is practically independent of it. SPINDLE handles a large number of features, by optimizing the use of the cryptosystem's packing capabilities, and by exploiting *single-instruction multiple-data (SIMD)* operations. It is able to perform demanding training tasks, with high number of iterations and thus high-depth computations, by relying on the multiparty cryptoscheme's ability to collectively refresh a ciphertext with no significant overhead. As shown by our evaluation, these properties enable SPINDLE to support training on large and complex data such as imaging or medical datasets. Moreover, SPINDLE scalability over multiple dimensions (features, DPs, data) represents a notable improvement with respect to state-of-the-art secure solutions [132], [41].

In this work, we make the following contributions:

- We analyze the problem of privacy-preserving distributed training and of the evaluation of ML models by extending the widespread MapReduce abstraction with privacy constraints. Following this abstraction, we instantiate SPINDLE, the first operational and efficient distributed system that enables the privacy-preserving execution of a complete machine-learning workflow through the use of a cooperative gradient descent on a dataset distributed among many data providers.
- We propose multiple optimizations that enable the efficient use of a quantum-resistant multiparty (N-party) cryptographic scheme by relying on parallel computations, SIMD operations, efficient collective operations and optimized polynomial approximations of the models' activation functions, e.g., sigmoid and softmax.
- We propose a method for the parameterization of SPINDLE by capturing the relations among the security and the learning parameters in a graphical model.
- We evaluate SPINDLE against centralized and decentralized secure solutions and demonstrate its scalability and accuracy.

To the best of our knowledge, SPINDLE is the first operational system that provides the aforementioned features and security guarantees.

II. Related Work

Privacy-Preserving Training of Machine Learning Models. Some works have focused on *securely outsourcing* the training of linear ML models to the cloud, typically by using homomorphic encryption (HE) techniques [7], [15], [29], [50], [63], [66], [101]. For instance, Graepel et al. [50] outsource the training of a linear classifier by employing somewhat HE [38], whereas Aono et al. [7] approximate logistic regression, and outsource its computation to the cloud by using additive HE [98]. Jiang et al. [60] present a framework for outsourcing logistic regression training to public clouds by combining HE with hardware-based security techniques (i.e., Software Guard Extensions). In SPINDLE, we consider a substantially different setting where the sensitive data are distributed among multiple (untrusted) data providers.

Along the research direction of *privacy-preserving distributed learning*, most works operate on the two-server model, where data owners encrypt or secret-share their data among two non-colluding servers that are responsible for the computations. For instance, Nikolaenko et al. [96] combine additive homomorphic encryption (AHE) and Yao's garbled circuits [126] to enable ridge regression on data that are horizontally partitioned among multiple data providers. Gascon et al. [42] extend Nikolaenko et al. work [96] to the case of vertically partitioned datasets and improve its computation time by employing a novel conjugate gradient descend (GD) method, whereas Giacomelli et al. [44] further reduce computation and communication overheads by using only AHE. Akavia et al. [3] improve the performance of Giacomelli et al. protocols [44] by performing linear regression on packed encrypted data. Mohassel and Zhang [88] develop techniques to handle secure arithmetic operations on decimal numbers, and employ stochastic GD, which, along with multi-party-computation-friendly alternatives for non-linear activation functions, supports the training of logistic regression and neural network models. Schoppmann et al. [109] propose data structures that exploit data sparsity to develop secure computation protocols for nearest neighbors, naive Bayes, and logistic regression classification. SPINDLE differs from these approaches as it does not restrict to the two non-colluding server model, and focuses instead on N-party systems, with $N \geqslant 2$.

Other distributed and privacy-preserving ML approaches employ a three-server model and rely on secret-sharing techniques to train linear regressions [13], logistic regressions [26], and neural networks [87], [120]. However, such solutions are tailored to the three-party server model and assume an honest majority among the computing parties. An honest majority is also required in the recent work of Rachuri and Suresh [104], who improve on Mohassel and Rindal [87] performance by extending its techniques to the four-party setting. Other works focus on the training of ML models among N-parties ($N \geqslant 4$), with stronger security assumptions, i.e., each party trusting itself. For instance, Corrigan-Gibbs and Boneh [28] present Prio, which relies on secret-sharing to enable the training of linear models, and Zheng et al. [132] propose Helen, a system that uses HE [98] and verifiable secret sharing [30] to execute ADMM [19] (alternating direction method of multipliers, a convex optimization approach for distributed data), which supports regularized linear models. Similarly, Froelicher et al. [41] employ HE [35], along with encoding techniques, to enable the training of basic regression models and provide auditability with the use of zero-knowledge proofs. SPINDLE enables better scalability in terms of the number of model's features, size of the dataset and number of data providers, and it offers richer functionalities by relying on the generic and widely-applicable SGD.

Another line of research considers the use of differential privacy for training ML models. Early works [2], [22] focus on a centralized setting where a trusted party holds the data, trains the ML model, and performs the noise addition. Differential privacy has also been envisioned in distributed settings, where to collectively train a model, multiple parties exchange or send differentially private model parameters to a central server [34], [55], [72], [112]. However, the training of an accurate collective model requires very high privacy budgets and, as such, it is unclear what privacy protection is achieved in practice [58], [123], [54]. To this end, some works consider hybrid approaches where differential privacy is combined with HE [64], [100], or multi-party computation techniques [59], [118]. We consider differential privacy as an orthogonal approach; these techniques can be combined with our solution to protect the resulting models and their predictions from inference attacks [39], [113], see Section IX-A.

Privacy-Preserving Prediction on ML Models. Another line of work is focused on privacy-preserving ML prediction, where a party (e.g., a cloud provider) holds an already trained ML model on which another party (e.g., a client) wants to evaluate its private input. In this setting, Bost et al. [17] use additive HE techniques to evaluate naive Bayes and decision tree classifiers, whereas Gilad-Bachrach et al. [45] employ fully homomorphic encryption (FHE) [16] to perform prediction on a small neural network. The computation overhead of these approaches has been further optimized by using multi-party computation (MPC) techniques [105], [107], or by combining HE and MPC [61], [78], [103]. Riazi et al. [106] evaluate deep neural networks by employing garbled circuits and oblivious transfer, in combination with binary neural networks. Boemer et al. [12] propose nGraph-HE2, a compiler that enables service providers to deploy their trained ML models in a privacy-preserving manner. Their method uses HE, or a hybrid scheme that combines HE with MPC, to compile ML models that are trained with well-known frameworks such as TensorFlow [1] and PyTorch [99]. The scope of our work is broader than these approaches, as SPINDLE accounts not only for the private evaluation of machine-learning models but also for their privacy-preserving training in the distributed setting.

III. Secure Federated Training and Evaluation

We first introduce the problem of privacy-preserving distributed training and evaluation of machine-learning (ML) models. Then, we present a high-level overview and architecture of a solution that satisfies the security requirements of the presented problem. In Section IV, we present SPINDLE, a system that enables the privacy preserving and distributed execution of a stochastic gradient-descent. We instantiate our solution for the training and evaluation of the widely-used Generalized Linear Models [94]. In the rest of this paper, matrices are denoted by upper-case-bold characters and vectors by lowercase-bold characters; the i-th row of a matrix $\boldsymbol{X}$ is depicted as $\boldsymbol{X}[i,\cdot]$, and its i-th column as $\boldsymbol{X}[\cdot,i]$. Similarly, the i-th element of a vector $\boldsymbol{y}$ is denoted by $\boldsymbol{y}[i]$. We provide a list of recurrent symbols in Table VI (see Appendix E).

A. Problem Statement

We consider a setting where a dataset $(\boldsymbol{X}_{n\times c},\boldsymbol{y}_n)$, with $\boldsymbol{X}_{n\times c}$ a matrix of n records and c features, and $\boldsymbol{y}_n$ a vector of n labels, is distributed among a set of data providers, i.e., $S=\{DP_1,...,DP_{|S|}\}$. The dataset is horizontally partitioned, i.e., each data provider DP_i

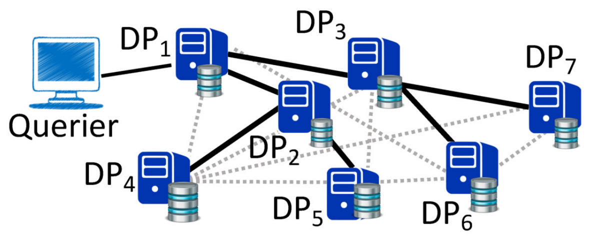


Fig. 1: SPINDLE's Model. Thick arrows represent a possible (efficient) query-execution flow.

holds a partition of n_i samples $(\boldsymbol{X}^{(i)},\boldsymbol{y}^{(i)})$, with $\sum_{i=1}^{|S|} n_i = n$. A querier, which can also be a data provider (DP), requests the training of a ML model on the distributed dataset $(\boldsymbol{X}_{n\times c},\boldsymbol{y}_n)$ or the evaluation of an already trained model on its input $(\boldsymbol{X'},\cdot)$.

We assume that the DPs are willing to contribute their respective data to train and to evaluate ML models on the distributed dataset. To this end, DPs are all interconnected and organized in a topology that enables efficient execution of the computations, e.g., in a tree structure as depicted in Figure 1. Even though the DPs wish to collaborate for the execution of ML workflows, they do not trust each other. As a result, they seek to protect the confidentiality of their data (used for training and evaluation) and of the collectively learned model. More formally, we require that the following privacy properties hold in a passive-adversary model in which all-but-one DPs can collude, i.e., the DPs follow the protocol, but up to $|S|-1$ DPs might share among them the information they observe during the execution, to extract information about the other DPs' inputs.

(a) Data Confidentiality: The training data of each data provider DP_i, i.e., $(\boldsymbol{X}^{(i)},\boldsymbol{y}^{(i)})$ and the querier's evaluation data $(\boldsymbol{X'},\cdot)$ should remain only known to their respective owners. To this end, data confidentiality is satisfied as long as the involved parties (DPs and querier) do not obtain any information about other parties' inputs other than what can be deduced from the output of the process of training or evaluating a model.

(b) Model Confidentiality: During the training process, no data provider DP_i should gain more information about the model that is being trained than what it can learn from its own input data $(\boldsymbol{X}^{(i)},\boldsymbol{y}^{(i)})$. During prediction, the querier should not learn anything more about the model than what it can infer from its input data $(\boldsymbol{X'},\cdot)$ and the corresponding predictions $\boldsymbol{y'}$.

We remark here that input correctness and computation correctness are not part of the problem requirements, i.e., we assume that DPs input correct data and do not perform wrong computations. We discuss possible countermeasures against malicious DPs in Section IX-A.

B. Solution Overview

To address the problem of privacy-preserving distributed learning, we leverage the MapReduce abstraction, which is often used to capture the parallel and repetitive nature of distributed learning tasks [27], [119]. We complement this abstraction with a protection mechanism $P(\cdot)$; $P(x)$ denotes that value x has to be protected to satisfy data and model confidentiality (Section III-A). We present the extended MapReduce abstraction in Protocol 1. In PREPARE, the data providers $(DP_i \in S)$ pre-process their data $(\boldsymbol{X}^{(i)},\boldsymbol{y}^{(i)})$, they agree on the learning parameters and on one data provider that plays the role of DP_R and is then responsible for the execution of REDUCE. As explained later, DP_R only manipulates protected data and is subject to the same security constraints as any other DP. We discuss the choice of DP_R and its availability in Section IX. Each DP_i then iteratively (g iterations) trains its local model ($P(\boldsymbol{W}^{(i,j)})$ at iteration j) on its data in MAP. They combine their local models in COMBINE (through an application-dependent function $C(\cdot)$), and update the global model $P(\boldsymbol{W}_G^{(\cdot,j)})$ in REDUCE. To capture the complete ML workflow, we extend the MapReduce architecture with a PREDICTION phase in which predictions $P(\boldsymbol{y'})$ are computed from the querier's protected evaluation data $P(\boldsymbol{X'})$ by using the (protected) global model $P({\boldsymbol{W}_G}^{(\cdot,g)})$ obtained during the training.

Protocol 1 Extended MapReduce Abstraction.

```
TRAINING: $S$ receives query from Querier and outputs $P(\boldsymbol{W}_G^{(\cdot,g)})$
 1: Each $DP_i$ has $(\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)})$
 2: DPs appoint $DP_R$ and agree on learning params.                         – PREPARE
 3: Each $DP_i \in S$ initializes its local model $\boldsymbol{W}^{(i,0)}$
 4: for $j=1,...,g$ do
 5:    Each $DP_i \in S$ computes:                                              – MAP
         $P(\boldsymbol{W}^{(i,j)}) \leftarrow$ Map($(\boldsymbol{X}^{(i)},\boldsymbol{y}^{(i)}),P(\boldsymbol{W}_G^{(\cdot,j-1)}),P(\boldsymbol{W}^{(i,j-1)})$)
 6:    Each $DP_i$ sends $P(\boldsymbol{W}^{(i,j)})$ to $DP_R$                                – COMBINE
 7:    $DP_R$ computes: $P(\boldsymbol{W}^{(\cdot,j)}) \leftarrow C(P(\boldsymbol{W}^{(i,j)})), \forall\, DP_i \in S$
                                                                             – REDUCE
 8:    $DP_R$ computes: $P({\boldsymbol{W}_G}^{(\cdot,j)}) \leftarrow$ Red($P({\boldsymbol{W}_G}^{(\cdot,j-1)}),P(\boldsymbol{W}^{(\cdot,j)})$)
   PREDICTION: $DP_R$ receives $P(\boldsymbol{X'})$ from Querier and uses $P({\boldsymbol{W}_G}^{(\cdot,g)})$ to compute $P(\boldsymbol{y'})$ that is sent back to the Querier
```

IV. SPINDLE Design

Following the extended MapReduce abstraction described in Section III-B, we design a system, named SPINDLE, that enables the privacy-preserving execution of the widely applicable cooperative gradient descent [121], [122] – which is used to minimize many cost functions in machine-learning [69], [117], [131]. We instantiate this system for the training of and prediction on Generalized Linear Models [94]. To implement the protection mechanism $P(\cdot)$, it builds on a multiparty fully homomorphic encryption scheme. We introduce these concepts in Section IV-A. Then, in Section IV-B, we describe how SPINDLE instantiates the phases of the extended MapReduce abstraction and how we address the collective data-processing on the distributed dataset through secure and interactive protocols. We also demonstrate how training is performed, notably by executing the gradient descent operations under homomorphic encryption, and how predictions are executed on encrypted models. The detailed cryptographic operations are presented in Section V.

A. Background

Cooperative Gradient-Descent. We rely on a distributed version of the popular mini-batch stochastic gradient-descent (SGD) [69], [117], [131]. In the standard version of SGD, the goal is to minimize $\min_{\boldsymbol{w}}[F(\boldsymbol{w}):=(1/n)\sum_{\phi=1}^{n} f(\boldsymbol{w};\, \boldsymbol{X}[\phi,\cdot])]$, where $f(\cdot)$ is the loss function defined by the learning model, $\boldsymbol{w}\in\mathbb{R}^c$ are the model parameters, and $\boldsymbol{X}[\phi,\cdot]$ is the ϕ^{th} data sample (row) of $\boldsymbol{X}$. The model is then updated by m iterations $\boldsymbol{w}^{(l)}=\boldsymbol{w}^{(l-1)}-\alpha[\zeta(\boldsymbol{w}^{(l-1)};\, \boldsymbol{B}^{(l)})]$, for $l=1,\ldots,m$, with α the learning rate, $\boldsymbol{B}^{(l)}$ a randomly sampled sub-matrix of $\boldsymbol{X}$ of size $b\times c$, and $\zeta(\boldsymbol{w};\boldsymbol{B})=\boldsymbol{B}^T(\sigma(\boldsymbol{B}\boldsymbol{w})-I(\boldsymbol{z}))$, where $\boldsymbol{z}$ is the vector of labels corresponding to the batch $\boldsymbol{B}$. The activation function σ and $I(\cdot)$ are both model-dependent, e.g., for a logistic regression σ is the sigmoid and $I(\cdot)$ is the identity.

We rely on the **cooperative SGD** (CSGD) proposed by Wang and Joshi [121], [122], due to its properties; in particular: (i) modularity, as it can be synchronous or asynchronous, and can be combined with classic gradient-descent convergence optimizations such as Nesterov accelerated SGD [95]; (ii) applicability, as it accommodates any ML model that can be trained with SGD and enables the distribution of any SGD based solution; (iii) it guarantees a bound on the error-convergence depending on the distributed parameters; e.g., the number of iterations and the update function for the global weights [121], [122], [18], [130]; and (iv) it has been shown to work well even in the case of non-independent-and-identically-distributed (non-i.i.d.) data partitions [81], [121], [122]. The data providers (DPs), each of which owns a part of the dataset, locally perform multiple iterations of the SGD before aggregating their model weights into the global model weights. The global weights are included in subsequent local DP computations to avoid that they learn, or descend, in the wrong direction. For simplicity, we present SPINDLE with the synchronous CSGD version, where the DPs perform local model updates simultaneously. For each DP_i, the local update rule at global iteration j and local iteration l is:

$$\boldsymbol{w}^{(i,j,l)}=\boldsymbol{w}^{(i,j,l\text{-}1)}-\alpha\zeta(\boldsymbol{w}^{(i,j,l\text{-}1)};\boldsymbol{B}^{(l)})-\alpha\rho(\boldsymbol{w}^{(i,j,l\text{-}1)}-\boldsymbol{w}_G^{(\cdot,j\text{-}1)}), \tag{1}$$

where $\boldsymbol{w}_G^{(\cdot,j-1)}$ are the global weights from the last global update iteration $j-1$, α is the learning rate and ρ, the elastic rate, is the parameter that controls how much the data providers can diverge from the global model. The set of DPs S performs m local iterations between each update of the global model that is updated at global iteration j with a moving average by:

$$\boldsymbol{w}_G^{(\cdot,j)}=(1-|S|\alpha\rho)\boldsymbol{w}_G^{(\cdot,j-1)}+\alpha\rho\textstyle\sum_{i=0}^{|S|}\boldsymbol{w}^{(i,j,m)}. \tag{2}$$

Generalized Linear Models (GLMs). GLMs [94] are a generalization of linear models where the linear predictor, i.e., the combination $\boldsymbol{X}\boldsymbol{w}$ of the feature matrix $\boldsymbol{X}$ and weights vector $\boldsymbol{w}$, is related to a vector of class labels $\boldsymbol{y}$ by an activation function σ such that $E(\boldsymbol{y})=\sigma^{-1}(\boldsymbol{X}\boldsymbol{w})$, where $E(\boldsymbol{y})$ is the mean of $\boldsymbol{y}$. In this work, we consider the widely-used linear (i.e., $\sigma(\boldsymbol{X}\boldsymbol{w})=\boldsymbol{X}\boldsymbol{w}$), logistic (i.e., $\sigma(\boldsymbol{X}\boldsymbol{w})=1/(1+e^{-\boldsymbol{X}\boldsymbol{w}})$) and multinomial (i.e., $\sigma(\boldsymbol{X}\boldsymbol{w}_\lambda)=e^{\boldsymbol{X}\boldsymbol{w}_\lambda}/(\sum_{j\in cl}e^{\boldsymbol{X}\boldsymbol{w}_j})$, for $\lambda\in cl$) regression models. We remark that for multinomial regression, the weights are represented as a matrix $\boldsymbol{W}_{c\times|cl|}$, where c is the number of features, cl is the set of class labels and $|cl|$ its cardinality. In the rest of the paper, unless otherwise stated, we define the operations on a single vector of weights $\boldsymbol{w}$ and we note that in the case of multinomial regression, they are replicated on the $|cl|$ vectors of weights, i.e., each column of $\boldsymbol{W}_{c\times|cl|}$.

Multiparty Homomorphic Encryption. For the protection mechanism of SPINDLE, we rely on a multiparty (or distributed) fully-homomorphic encryption scheme [91] in which the secret key is distributed among the parties, while the corresponding collective public key pk is known to all of them. Thus, each party can independently compute on ciphertexts encrypted under pk but all parties have to collaborate to decrypt a ciphertext. In SPINDLE, this enables the data providers (DPs) to train a collectively encrypted model, that cannot be decrypted as long as one DP is honest and refuses to participate in the decryption. As we show later, this multiparty scheme also enables DPs to collectively switch the encryption key of a ciphertext from pk to another public key without decrypting. In SPINDLE, a collectively encrypted prediction result can thus be switched to the querier's public key, so that only the querier can decrypt the result.

Mouchet et al. [91] propose a multiparty version of the Brakerski Fan-Vercauteren (BFV) lattice-based homomorphic cryptosystem [38] and introduce interactive (distributed) protocols for key generation DKeyGen$(\cdot)$, decryption DDec$(\cdot)$, and bootstrapping DBootstrap$(\cdot)$. We use an adaptation of this multiparty scheme to the Cheon-Kim-Kim-Song cryptosystem (CKKS) [25] that enables approximate arithmetic, and whose security is based on the ring learning with errors (RLWE) problem [80]. CKKS (See Appendix A) enables arithmetic over $\mathbb{C}^{N/2}$; the plaintext and ciphertext spaces share the same domain $R_Q=\mathbb{Z}_Q[X]/(X^N+1)$, with N a power of 2. Both plaintexts and ciphertexts are represented by polynomials of N coefficients (degree $N-1$) in this domain. A plaintext/ciphertext encodes a vector of up to $N/2$ values.

Parameters: The CKKS parameters are denoted by the tuple (N,Δ,η,mc), where N is the ring dimension, Δ is the plaintext scale, or precision, by which any value is multiplied before being quantized and encrypted/encoded, η is the standard deviation of the noise distribution, and

mc represents a chain of moduli $\{q_0,...,q_L\}$ such that $\Pi_{\iota\in\{0,...,\tau\}} q_\iota = Q_\tau$ is the ciphertext modulus at level τ, with $Q_L = Q$, the modulus of fresh ciphertexts. Operations on a level-τ ciphertext $\langle \boldsymbol{v} \rangle$ are performed modulo Q_τ, with Δ always lower than the current Q_τ. Ciphertexts at level τ are simply vectors of polynomials in R_{Q_τ}, that we represent as $\langle \boldsymbol{v} \rangle$ when there is no ambiguity about their level, and use $\{\langle \boldsymbol{v} \rangle, \tau, \Delta\}$ otherwise. After performing operations that increase the noise and the plaintext scale, $\{\langle \boldsymbol{v} \rangle, \tau, \Delta\}$ has to be rescaled (see the ReScale$(\cdot)$ procedure defined in Appendix A) and the next operations are performed modulo $Q_{\tau-1}$. When reaching level 0, $\langle \boldsymbol{v} \rangle$ has to be bootstrapped. The security of the cryptosystem depends on the choice of N, Q and η, which in this work are parameterized to achieve at least 128-bits of security.

(Distributed) Operations: A vector $\boldsymbol{v}$ of cleartext values can be encrypted with the public collective key pk and can be decrypted with the collaboration of all DPs (DDec$(\cdot)$ protocol, in which each DP_i uses its secret key sk_i). The DPs can also change a ciphertext encryption from the public key pk to another public key pk' without decrypting the ciphertext, by relying on the DKeySwitch$(\cdot)$ protocol. Each DP can independently add, multiply, rotate (i.e., inner-rotation of $\boldsymbol{v}$), rescale Rescale$(\cdot)$ or relinearize Relin$(\cdot)$ a vector encrypted with pk. When two ciphertexts are multiplied together, the result has to be relinearized Relin$(\cdot)$ to preserve the ciphertext size. After multiple Rescale$(\cdot)$ operations, $\langle \boldsymbol{v} \rangle$ has to be refreshed by a collective protocol, i.e., DBootstrap$(\cdot)$, which returns a ciphertext at level L. The dot product DM$(\cdot)$ of two encrypted vectors of size a can be executed by a multiplication followed by $log_2(a)$ inner-left rotations and additions. We list all the operations used in SPINDLE and their properties in Appendix A.

B. *SPINDLE Protocols*

We first describe SPINDLE's operations for training a Generalized Linear Model following Protocol 1. In this case, the model $\boldsymbol{W}$ is a vector of weights that we denote by $\boldsymbol{w}$, and MAP corresponds to multiple local iterations of the gradient descent. Recall that in the case of multinomial regression, all operations are repeated for each label class $\lambda \in cl$.

1) TRAINING: **PREPARE.** The data providers (DPs) collectively agree on the training parameters: the maximum number of global g and local m iterations, and the learning parameters $lp = \{\alpha, \rho, b\}$, where α is the learning rate, ρ the elastic rate, and b the batch size. The DPs also collectively initialize the cryptographic keys for the distributed CKKS scheme by executing DKeyGen$(\cdot)$ (see Appendix A). Then, the DPs initialize their local weights and pre-compute operations that involve only their input data ($\alpha \boldsymbol{X}^{(i)} I(\boldsymbol{y}^{(i)})$ and $\alpha \boldsymbol{X}^{(i)T}$). We discuss in Section IX how the DPs can collaborate to standardize or normalize the distributed dataset (if needed) and check that their respective inputs are consistent, e.g., they have data distribution homogeneity.

MAP. As depicted in Protocol 2, the DPs execute m iterations of the cooperative gradient-descent local update (Section IV-A). The local weights of DP_i (i.e., $\langle \boldsymbol{w}^{(i,j,l-1)} \rangle$) are updated at a global iteration j and a local iteration l by computing the gradient (Protocol 2, lines 4, 5, and 6) that is then combined with the current global weights $\langle \boldsymbol{w}_G^{(\cdot,j-1)} \rangle$ (Protocol 2, line 7) following Equation 1. These computations are performed on batches of b samples and c features. To ensure that the update of DP_i's local weights, i.e., the link between the ciphertexts $\langle \boldsymbol{w}^{(i,j-1)} \rangle = \langle \boldsymbol{w}^{(i,j,0)} \rangle$ and $\langle \boldsymbol{w}^{(i,j,m)} \rangle$, does not leak information about the DP's local data, $\langle \boldsymbol{w}^{(i,j,m)} \rangle$ is re-randomized RR$(\cdot)$ at the end of MAP, i.e., DP_i adds to it a fresh encryption of 0.

Protocol 2 MAP.

```
Each $DP_i$ outputs $\langle w^{(i,j)} \rangle \leftarrow$ Map$((X^{(i)}, y^{(i)}), \langle w_G^{(\cdot,j-1)} \rangle, \langle w^{(i,j-1)} \rangle)$
1: $\langle w^{(i,j,0)} \rangle = \langle w^{(i,j-1)} \rangle$
2: for $l = 1,...,m$ :
3:     Select batch $(B, z)$ of $b$ rows in $(X^{(i)}, y^{(i)})$
4:     $\langle u[k] \rangle =$ DM$(B[k,\cdot], \langle w^{(i,j,l-1)} \rangle)$, for $k = 1, ..., b$
5:     $\langle v[e] \rangle =$ DM$(\alpha B[\cdot,e]^T, \sigma(\langle u \rangle))$,  for $e = 1,...,c$
6:     $\mu[e] = \sum_{k=1}^{b} \alpha B[\cdot,e]^T I(z[k])$,    for $e = 1,...,c$
7:     $\langle w^{(i,j,l)} \rangle = \langle w^{(i,j,l-1)} \rangle + \mu - \langle v \rangle - \alpha\rho(\langle w^{(i,j,l-1)} \rangle - \langle w_G{}^{(\cdot,j-1)} \rangle)$
8: $\langle w^{(i,j)} \rangle =$ RR$(\langle w^{(i,j,m)} \rangle)$
```

Note that in Protocol 2, line 5 the activation function $\sigma(\cdot)$ is computed on the encrypted vector $\langle \boldsymbol{u} \rangle$ (or a matrix $\langle \boldsymbol{U} \rangle$ in the case of multinomial). The exponential activation functions for logistic (i.e., sigmoid) and multinomial (i.e., softmax) regressions have to be approximated to polynomial functions to be evaluated on encrypted data by using the homomorphic properties of CKKS. We rely on a least-square polynomial approximation (LSPA) for the sigmoid, as it provides an optimal average mean-square error for uniform inputs in a specific interval, which is a reasonable assumption when the input distribution is not known. For softmax, we rely on Chebyshev approximation (CA) to minimize the maximum approximation error and thus avoid that the function diverges on specific inputs. The approximation intervals can be empirically determined by using synthetic datasets with distribution similar to the real ones, by computing the minimum and maximum input values over all DPs and features, or by relying on estimations based on the data distribution [53]. Protocol 3 takes as input an encrypted vector/matrix $\langle \boldsymbol{u} \rangle$ or $\langle \boldsymbol{U} \rangle$ and the type of the regression t (i.e., linear, logistic or multinomial). If t is linear, the protocol simply returns $\langle \boldsymbol{u} \rangle$. Otherwise, if t is logistic, it computes the activated vector $\langle \sigma(\boldsymbol{u}) \rangle$ by using the sigmoid's LSPA (apSigmoid$(\cdot)$). If t is multinomial, it computes the activated matrix $\langle \sigma(\boldsymbol{U}) \rangle$ using the softmax approximation that is computed by the multiplication of two CAs, one for the nominator e^x (apSoftN$(\cdot)$) and one for the denominator $\frac{1}{\sum e^{x_j}}$ (apSoftD$(\cdot)$), each computed on different intervals. The polynomial approximation computation is detailed in Protocol 6 (Appendix B). To avoid an explosion of the exponential values in the softmax, a vector

Protocol 3 Activation Function $\sigma(\cdot)$.

```
Func. $\sigma(\langle \boldsymbol{u} \rangle$ or $\langle \boldsymbol{U} \rangle, t)$ returns the activated $\langle \sigma(\boldsymbol{u}) \rangle$ or $\langle \sigma(\boldsymbol{U}) \rangle$
1: if $t$ is Linear then $\langle \sigma(\boldsymbol{u}) \rangle = \langle \boldsymbol{u} \rangle$
2: else if $t$ is Logistic then
3:   $\langle \sigma(\boldsymbol{u}) \rangle =$ apSigmoid($\boldsymbol{u}$)
4: else if $t$ is Multinomial, input is a matrix $\langle \boldsymbol{U}_{c\times|cl|} \rangle$ then
5:   $\langle \boldsymbol{m} \rangle =$ apMax($\langle \boldsymbol{U} \rangle$)
6:   for $\lambda \in cl$:
7:   $\langle \boldsymbol{U'}[\lambda,\cdot] \rangle = \langle \boldsymbol{U}[\lambda,\cdot] \rangle - \langle \boldsymbol{m} \rangle$
8:   $\langle \sigma(\boldsymbol{U}[\lambda,\cdot]) \rangle$=M(apSoftN($\langle \boldsymbol{U'}[\lambda,\cdot] \rangle$),apSoftD($\langle \boldsymbol{U'}[\lambda,\cdot] \rangle$))
```

$\langle \boldsymbol{m} \rangle$ that contains the approximated max (apMax($\cdot$)) value of each column of $\langle \boldsymbol{U} \rangle$ is subtracted from all input values, i.e., from each $\langle \boldsymbol{U}[\lambda,:] \rangle$ with $\lambda = 0,...,|cl|$. Similar to softmax, the approximation of the max function requires two CAs, and is detailed in Appendix B.

COMBINE. The MAP outputs of each DP_i, i.e., $\langle \boldsymbol{w}^{(i,j)} \rangle$, are homomorphically combined ascending a tree structure, such that each DP_i aggregates its encrypted updated local weights with those of its children and sends the result to its parent. In this case, the combination function $C(\cdot)$ is the homomorphic addition operation. At the end of this phase, the DP at the root of the tree DP_R obtains the encrypted combined weights $\langle \boldsymbol{w}^{(\cdot,j)} \rangle$.

REDUCE. DP_R updates the encrypted global weights $\langle \boldsymbol{w}_G^{(\cdot,j)} \rangle$, as shown in Protocol 4. More precisely, it computes Equation 2 by using the encrypted sum of the DPs' updated local weights $\langle \boldsymbol{w}^{(\cdot,j)} \rangle$ (obtained from COMBINE), the previous global weights $\langle \boldsymbol{w}_G^{(\cdot,j-1)} \rangle$, the pre-defined elastic rate ρ and the learning rate α. After g iterations of the MAP, COMBINE, and REDUCE, DP_R obtains the encrypted global model $\langle \boldsymbol{w}_G^{(\cdot,g)} \rangle$ and broadcasts it to the rest of the DPs.

Protocol 4 REDUCE.

```
$DP_R$ computes $\langle \boldsymbol{w}_G^{(\cdot,j)} \rangle \leftarrow$Red($\langle \boldsymbol{w}_G^{(\cdot,j-1)} \rangle$,$\langle \boldsymbol{w}^{(\cdot,j)} \rangle$,$\rho$,$\alpha$)
1: $\langle \boldsymbol{w}_G^{(\cdot,j)} \rangle = (1-\alpha\rho|S|)\langle \boldsymbol{w}_G^{(\cdot,j-1)} \rangle + \alpha\rho\langle \boldsymbol{w}^{(\cdot,j)} \rangle$
```

2) PREDICTION: The querier's input data $(\boldsymbol{X'},\cdot)$ is encrypted with the collective public key pk. Then, $\langle \boldsymbol{X'} \rangle_{pk}$ is multiplied ($DM(\cdot,\cdot)$ with the weights of the trained model $\langle \boldsymbol{w}_G^{(\cdot,g)} \rangle$ and processed through the activation function $\sigma(\cdot)$ to obtain the encrypted prediction values $\langle \boldsymbol{y'} \rangle$ (one prediction per row of $\boldsymbol{X'}$). The prediction results encrypted under pk are then collectively switched by the DPs to the querier public key pk' using DKeySwitch($\cdot$), so that only the querier can decrypt $\langle \boldsymbol{y'}_{pk'} \rangle$.

Protocol 5 PREDICTION.

```
$DP_R$ gets $\langle \boldsymbol{X'}_{n'\times c} \rangle$ from Querier and computes $\langle \boldsymbol{y'}_{n'} \rangle$ using $\langle \boldsymbol{w}_G^{(\cdot,g)} \rangle$
1: $\langle \boldsymbol{y'}[p] \rangle = \sigma$(DM($\langle \boldsymbol{X'}[p,\cdot] \rangle$,$\langle \boldsymbol{w}_G^{(\cdot,g)} \rangle$)), for $p = 0,...,n'$
2: $\langle \boldsymbol{y'} \rangle_{pk'} =$ DKeySwitch($\langle \boldsymbol{y'} \rangle$,$pk'$,$\{sk_i\}$)
```

V. System Operations

We describe how SPINDLE relies on the properties of the distributed version of CKKS to efficiently address the problem of privacy-preserving distributed learning. We first describe how we optimize the protocols of Section IV-B by choosing when to execute cryptographic operations such as rescaling and (distributed) bootstrapping. Then, we discuss how to efficiently perform the MAP protocol that involves a sequence of vector-matrix-multiplications and the evaluation of the activation function, in the encrypted domain.

A. Cryptographic Operations

As explained in Section IV-A (and Appendix A), ciphertext multiplications incur the execution of other cryptographic operations hence increase SPINDLE's computation overhead. This overhead can rapidly increase when the same ciphertext is involved in sequential operations, i.e., when the operations' multiplicative depth is high. As we will describe in Section VIII, SPINDLE relies on the Lattigo [85] lattice-based cryptographic library, where a ciphertext addition or multiplication requires a few ms, whereas Rescale($\cdot$), Relin($\cdot$), and DBootstrap($\cdot$), are 1-order, 2-orders, and 1.5-orders of magnitude slower than the addition, respectively. These operations can be computationally heavy, hence their execution in the protocols should be optimized. Note that we avoid the use of the centralized traditional bootstrapping, as it would require a much more conservative parameterization for the same security level, resulting in higher computational overheads (see Section VIII).

Lazy Rescaling. To maintain the precision of the encrypted values and for efficiency we rescale a ciphertext $\{\langle \boldsymbol{v} \rangle,\tau,\Delta\}$ only when Δ is close to q_τ. Hence, we perform a ReScale($\cdot$) only when this condition is met after a series of consecutive operations.

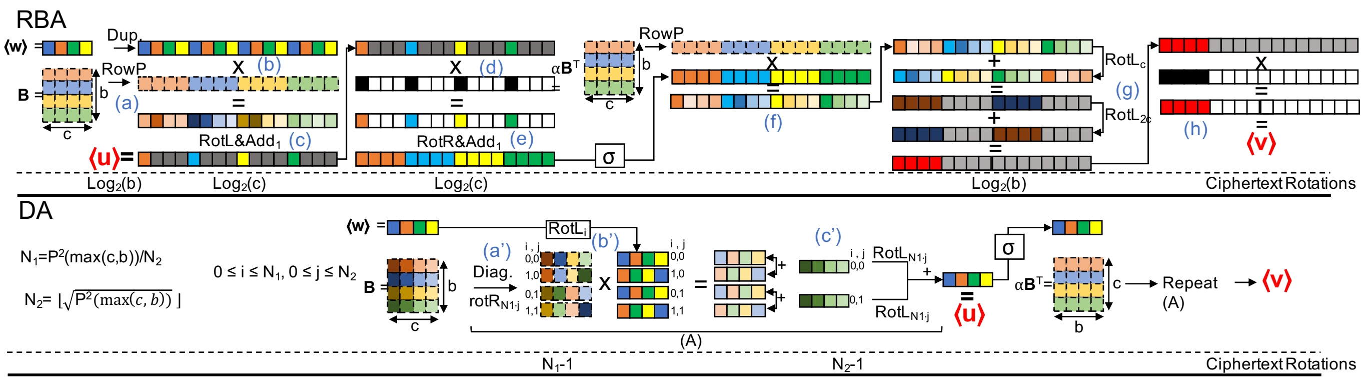


Fig. 2: Packing approaches for executing Protocol 2, lines 4 and 5. We assume that $c{\cdot}b < N/2$ and show an example with $c=b=4$. Dash elements are plaintext values, everything else is encrypted. Dup duplicates and adds, rowP packs the rows in one ciphertext, RotL(/R)&Add$_i$ rotates the encrypted vector by i,$2i$,$4i$,... to the left(/right) and at each step, aggregates the result with the previous ciphertext, RotL(/R)$_j$ rotates a vector left(/right) by j positions. P$^2(x)$ returns the next power of 2 larger than x.

Relinearization. Letting the ciphertext size increase after every multiplication would add to the subsequent operations an overhead that is higher than the relinearization. Hence, to maintain the ciphertext size and degree constant, a Relin$(\cdot)$ operation is performed after each ciphertext-ciphertext multiplication. We here note that a Relin$(\cdot)$ operation can be deferred if doing so incurs in lower computational complexity (e.g., if additions performed after the ciphertext-ciphertext multiplications reduce the number of ciphertexts to relinearize).

Bootstrapping. In the protocols of Section IV-B, we observe that the data providers' local weights and the model global weights ($\langle \boldsymbol{w} \rangle$ and $\langle \boldsymbol{w}_G \rangle$, resp.) are the only persistent ciphertexts over multiple computations and iterations. They are therefore the only ciphertexts that need to be bootstrapped, and we consider three approaches for this. With **Local bootstrap (LB)**, each data provider (DP) bootstraps (calling a DBootstrap$(\cdot)$ protocol) its local weights, every time they reach level τ_b during the MAP local iterations and before the COMBINE. As a result, the global weights are always combined with fresh encryptions of the local weights and only need to be bootstrapped after multiple REDUCE. Indeed, REDUCE involves a multiplication by a constant hence a Rescale$(\cdot)$. With **Global bootstrap (GB)**, we use the interdependency between the local and global weights, and we bootstrap only the global weights and assign them directly to the local weights. The bootstrapping is performed on the global weights during REDUCE. Thus, we modify TRAINING so that MAP operates on the (bootstrapped) global weights, i.e., $\langle \boldsymbol{w}^{(i,j-1)} \rangle = \langle \boldsymbol{w}_G{}^{(\cdot,j-1)} \rangle$, for a DP_i at global iteration j. By following this approach, the number of bootstrap operations is reduced, with respect to the local approach, because it is performed by only one DP and depends only on the number of global iterations. However, it modifies the learning method, and it offers less flexibility, as the number of local iterations in MAP is constrained by the number of ciphertext multiplications required in each iteration and by the available ciphertext levels. With **Hybrid bootstrap (HB)**, both GB and LB approaches are combined to reduce the total number of bootstrapping operations. The global weights are bootstrapped at each global iteration (GB) and the DPs can still perform many local iterations by relying on the LB. In our experiments (Section VIII-B), we observed that the effect on the trained model's accuracy depends mainly on the data and that, in most cases, enabling DPs to perform more local iterations (LB and HB) between two global updates yields better accuracy. Even though LB incurs at least $|S|$ more executions of the DBootstrap$(\cdot)$, the DPs execute them in parallel and thus amortize the overhead on SPINDLE's execution time. However, if the training of a dataset requires frequent global updates, then GB (or HB) achieves a better trade-off, see Section VIII-B. Taking into account these cryptographic transformations and the strategy to optimize their use in SPINDLE, we explain how to optimize the required number of ciphertext operations.

B. *MAP Vector-Matrix Multiplications*

As described in Section IV-A, each CKKS ciphertext encrypts (or packs) a vector of values, e.g., 8,192 elements if the ring dimension is $N = 2^{14}$. This packing enables us to simultaneously perform operations on all the vector values, by using a Single-Instruction Multiple Data (SIMD) approach for parallelization. To execute computations among values stored in different slots of the same ciphertext, e.g., an inner sum, we rely on ciphertext rotations that have a computation cost similar to a relinearization (Relin$(\cdot)$). Recall that for the execution of stochastic gradient-descent, each local iteration in MAP involves two sequential multiplications between encrypted vectors and cleartext matrices (Protocol 2, lines 4 and 5). As a result, packing is useful for reducing the number of vector multiplications and rotations needed to perform these operations. To this end, SPINDLE integrates two packing approaches and automatically selects the most appropriate approach at each DP during the training. We now describe these two approaches and how to choose between them, depending on the settings, i.e., the learning parameters, the number of features, and the DP computation capabilities. Figure 2 depicts SPINDLE's packing approaches for a toy example of the computation of $\langle \boldsymbol{u} \rangle$ (Protocol 2, line 4) whose result is activated (i.e., $\sigma(\langle \boldsymbol{u} \rangle)$) before used in the computation of $\langle \boldsymbol{v} \rangle$ (Protocol 2, line 5), for a setting with $c=b=4$. For clarity, we assume that a vector of c (number of features) or b (batch size) elements can be encoded in one ciphertext (or plaintext), i.e., $\max(c{,}b) \leq N/2$.

Row-Based Approach (RBA). This approach was proposed by Kim et al. [63]. The input matrices ($\boldsymbol{B}$ and $\alpha \boldsymbol{B}^T$) are packed row-wise, and multiple rows are packed in one plaintext ((a) in the upper part of Figure 2), i.e., the number of plaintexts required to encode the input matrix is $\lceil \frac{c \cdot b \cdot 2}{N} \rceil$. Each plaintext is then multiplied with a ciphertext containing the replicated weights' vector (b), such that the number of replicas is equal to the number of rows in $\boldsymbol{B}$. To obtain the results of the dot products between each weights' vector and row of $\boldsymbol{B}$, a partial inner sum is performed by adding the resulting ciphertext with rotated versions of itself (c). The values in between the dot product results are eliminated (i.e., masked) through a multiplication with a binary vector (d), and the dot product results are duplicated in the ciphertext (e) such that it can be activated ($\sigma(\cdot)$) and used directly for the multiplication with $\alpha \boldsymbol{X}^T$ (f). The result is then rotated and added to itself (g) such that it can be masked (h) to obtain $\langle \boldsymbol{v} \rangle$. As shown in Figure 2, the total number of vector multiplications is $\lceil \frac{c \cdot b \cdot 2}{N} \rceil \cdot 4$, whereas the number of ciphertext rotations is $\lceil \frac{c \cdot b \cdot 2}{N} \rceil \cdot 2 \cdot (log(b) + log(c))$. This approach has a multiplicative depth of $a_m + 4$, where a_m denotes the depth of the activation function $\sigma(\cdot)$.

Diagonal Approach (DA). This approach was presented by Halevi and Shoup [51] as an optimized homomorphic vector-matrix-multiplication evaluation. It optimizes the number of ciphertext rotations by transforming the input plaintext matrix $\boldsymbol{B}$. In particular, $\boldsymbol{B}$ is diagonalized, and each line is rotated ((a') in lower part of Figure 2) so that they can be independently multiplied with the (rotated) weights' vector (b'). The resulting ciphertexts are aggregated and rotated to obtain $\langle \boldsymbol{u} \rangle$ (c'), and a similar approach is used to compute $\langle \boldsymbol{v} \rangle$ after the activation. As shown in Figure 2, DA only executes $2 \cdot ((N_1 - 1) + (N_2 - 1))$ rotations on the encrypted vector, with $N_1 = P^2(\max(c,b))/N_2$ and $N_2 = \lfloor \sqrt{P^2(\max(c,b))} \rfloor$, where $P^2(x)$ returns the next power of 2 larger than x. This approach involves $N_1 \cdot N_2$ plaintext-ciphertext multiplications on independent ciphertexts and does not require any masking, which results in a multiplicative depth of $a_m + 2$. Therefore, this approach consumes fewer levels than RBA.

In both approaches, the number of rotations and multiplications depends on the batch size b and the number of features c. DA almost always requires more multiplications than RBA and uses more rotations after a certain c (e.g., if $b = 8$, the break-even happens at $c = 64$). However, as DA is *embarrassingly parallelizable* for both multiplications and rotations (with rotations being the most time-consuming operations), the computations can be amortized on multiple threads. Taking this into account, SPINDLE automatically chooses, based on c, b, and the number of available threads, the best approach at each DP. We analyze these trade-offs in Section VIII.

C. *Optimized Activation Function*

As described in Section IV-B, to enable their execution under FHE, we approximate the sigmoid (apSigmoid($\cdot$)) and softmax (apMax($\cdot$), apSoftN($\cdot$), apSoftD($\cdot$)) activation functions with least-squares and Chebyshev polynomial approximations (PA), respectively. We adapt the baby-step giant-step algorithm introduced by Han and Ki [52] to enable the minimum-complexity computation of degree-d polynomials (multiplicative depth of $\lceil log(d) \rceil$ for $d \leq 7$, and with depth $\lceil log(d) + 1 \rceil$ otherwise). Protocol 6 in Appendix B inductively computes the (element-wise) exponentiation of the encrypted input vector before recursively computing the polynomial approximation.

VI. System Configuration

We discuss how to parameterize SPINDLE by taking into account the interdependencies between the input data, and the learning and cryptographic parameters. We then discuss two modular functionalities of SPINDLE, namely *data outsourcing* and *model release*.

Parameter Selection. SPINDLE relies on the configuration of (a) cryptographic parameters that determine its security level, and (b) learning parameters that affect the accuracy of the training and evaluation of the models. Both are tightly linked, and we capture these relations in a graph-based model, displayed in Figure 3, where vertices and edges represent the parameters and their interdependence, respectively. For simplicity, we present a directed graph that depicts our empirical method for choosing the parameters (see Appendix E, Table VI for notation symbols). We highlight that the corresponding non-directed graph is more generic and simply captures the main relations among the parameters. We observe two main clusters: the cryptographic parameters on the upper part of the graph (dotted circles), and the learning parameters (circles) on the lower one. The input data and their intrinsic characteristics, i.e., the number of features c or precision (bits of precision required to represent the data), are connected with both clusters that are also interconnected through the plaintext scale Δ. As such, there are various ways to configure the overall system parameters.

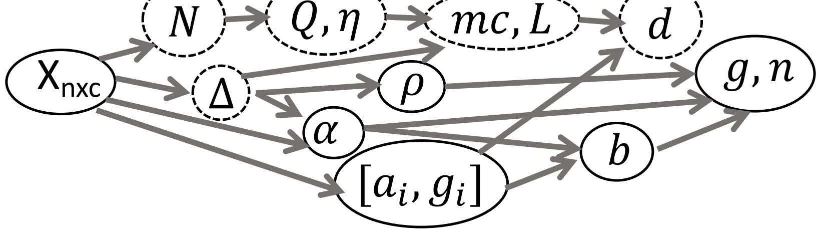


Fig. 3: System parameters graph. Circles and dotted circles represent learning and cryptographic parameters, respectively.

In our case, we decide to first choose N (ciphertext polynomial degree), such that at least c elements can be packed in one ciphertext. Q (ciphertext modulus) and η (fresh encryption noise) are then fixed to ensure a sufficient level of security (e.g., 128-bits) following the accepted parameterization from the homomorphic encryption standard whitepaper [4]. The scale Δ is configured to provide enough precision for the input data $\boldsymbol{X}$, and mc (moduli chain) and L (number of levels) are set accordingly. The intervals $[a_i, g_i]$ used for the approximations of the activation functions are defined according to $\boldsymbol{X}$. The approximation degrees $\boldsymbol{d}$ are then set depending on these intervals and the available number of levels L. The remaining learning parameters (α, ρ, b, g, m) are agreed upon by the data providers based on their observation of their part of the dataset. Note that the minimum values for the learning rate α and elastic rate ρ are limited by the scale Δ, and if they are too small the system might not have enough precision to handle their multiplication with the input data.

Data Outsourcing. SPINDLE's protocols (Section IV-B) seamlessly work with data providers (DPs) that either have their input data $\boldsymbol{X}$ in cleartext, or that obtain data $\langle \boldsymbol{X} \rangle_{pk}$ encrypted under the public collective key from their respective owners. In the latter case, SPINDLE enables both secure data storage and computation outsourcing to always-available untrusted cloud providers. It distributes the workload among multiple data providers and is still able to rely on efficient multiparty homomorphic-encryption operations, e.g., DBootstrap$(\cdot)$. We note that operating on encrypted input data affects the complexity of MAP, as all the multiplication operations (Protocol 2) would happen between ciphertexts, instead of between the cleartext inputs and ciphertexts.

Model Release. By default, the trained model in SPINDLE is kept secret from any entity, enabling privacy-preserving predictions on (private) evaluation-data input by the querier and offering end-to-end *model confidentiality*. If required by the application setting, SPINDLE can also reveal the trained model to the querier or to a third party. This is collectively enabled by the DPs, who perform a DKeySwitch$(\cdot)$.

VII. Security Analysis

We demonstrate that SPINDLE achieves the data and model confidentiality requirements defined in Section III-A by relying on the real/ideal simulation paradigm [74] and showing that a computationally-bounded adversary that controls up to ($|S|-1$)-out-of-$|S|$ DPs cannot distinguish a *real* world experiment, in which the adversary is given actual data (sent by honest DP(s)), and an *ideal* world experiment, in which the adversary is given random data generated by a simulator.

The semantic security of the CKKS scheme is based on the hardness of the decisional RLWE problem [25], [80], [75]. The achieved practical bit-security against state-of-the-art attacks can be computed using Albrecht's LWE-Estimator [4], [5]. Mouchet et al. [91] prove that their distributed protocols, i.e., Collective Encryption-Key Generation, Collective Relinearization-Key Generation (DKeyGen$(\cdot)$) and Collective Key Switching (DKeySwitch$(\cdot)$ and DDec$(\cdot)$) are secure under the simulator paradigm. They show that the distribution of the cryptoscheme preserves its security in the passive adversary model with all-but-one dishonest DPs, as long as the decisional-RLWE problem is hard. Their proofs, which are constructed using the BFV scheme, generalize to our adaptation of their protocols to CKKS, as they preserve the same computational assumptions, and the security of CKKS is based on the same hard problem as BFV. The security of DBootstrap$(\cdot)$ is based on Lemma 1 which we state and prove in Appendix C.

Proposition 1. *Assume that* SPINDLE *uses CKKS encryptions with parameters* (N,Δ,η,mc) *ensuring a post-quantum security level* λ. *Given a passive adversary corrupting at most* $|S|-1$ *parties,* SPINDLE *achieves* data and model confidentiality *for training and prediction.*

Sketch of the Proof. We consider a real-world simulator $\mathcal{S}$ that simulates the view of a computationally-bounded adversary corrupting $|S|-1$ parties, i.e., it has access to the inputs and outputs of $|S|-1$ parties. In PREPARE and MAP, the data providers (DPs) locally compute on their data and only exchange encrypted information with each other to perform DKeyGen$(\cdot)$ and DBootstrap$(\cdot)$. In COMBINE, the DPs' MAP outputs, encrypted under the public collective key, are aggregated. These outputs are the encrypted results of multiple local iterations in which elements derived from each DP_i's local private data $(\boldsymbol{X}^{(i)},\boldsymbol{y}^{(i)})$ are combined with its encrypted local model $\langle \boldsymbol{w}^{(i,\cdot)} \rangle$ and the current encrypted global model $\langle \boldsymbol{w}_G \rangle$. This result is re-randomized (i.e., added to a fresh encryption of **0**) to ensure that the outputs of successive MAP (i.e., the inputs to COMBINE) do not leak any information about the DPs' data. In REDUCE, the global model is updated by combining encrypted data, and a DBootstrap$(\cdot)$ is executed. All the information exchanged by the DPs is encrypted and the DPs rely on the aforementioned CPA-secure-proven protocols. We show in Appendix C that DBootstrap$(\cdot)$ is also simulatable. In PREDICTION, only encrypted information is exchanged and the security-proven DKeyswitch$(\cdot)$ protocol is used. We consider two cases: (a) the adversary controls $|S|-1$ DPs and (b) it controls the querier and $|S|-2$ DPs. In (a), the encryption of the querier's input data (with the DPs common public key pk) can be simulated by $\mathcal{S}$ and SPINDLE ensures *Data Confidentiality* of the querier. In (b) the confidentiality of the adversary-controlled-querier's data is trivial. The simulator has access to the prediction result and can produce all the intermediate (indistinguishable) encryptions that the adversary sees. Hence, $\mathcal{S}$ can simulate all the values communicated during the TRAINING and PREDICTION by generating random ciphertexts using the parameters (N,Δ,η,mc), such that the real outputs cannot be distinguished from the ideal ones. The sequential composition of all cryptographic functions remains simulatable by $\mathcal{S}$, as different random values are used in each step, and the exchanged ciphertexts are re-randomized, i.e., there is no dependency between the random values that an adversary can leverage on. Also, the adversary cannot decrypt collectively encrypted data unless all DPs collude, which would contradict the considered threat model. Following this, SPINDLE ensures the data and model confidentiality of the honest party/ies.

Finally, we note that by design, SPINDLE thwarts active attacks on federated learning [54], [84], [92], [133] and model inversion attacks [39], as intermediate and final model weights are never revealed during TRAINING.

VIII. System Evaluation

We first analyze the theoretical complexity of SPINDLE before moving to the empirical evaluation of its prototype and its comparison with existing solutions.

A. *Theoretical Analysis*

We refer to Table IVa (Appendix D) for the full complexity analysis of SPINDLE's protocols. We discuss here its main outcomes.

Communication Complexity. SPINDLE's communication complexity depends linearly on the number of data providers $|S|$, iterations (g,m) and the ciphertext size $|ct|$. In MAP, the only communication between the DPs is due to the DBootstrap$(\cdot)$, which requires two rounds of communication of one ciphertext (ct) between the $|S|$ DPs (i.e., $2\cdot(|S|-1)\cdot|ct|$). In COMBINE and REDUCE, the DPs exchange one ciphertext in respectively one and two rounds. Finally, the PREDICTION requires the exchange of one ciphertext between a DP and the querier and one DKeySwitch$(\cdot)$ operation, i.e., 2 ciphertexts are sent per DP.

Computation Complexity. SPINDLE's most intensive computational part is MAP; its complexity depends linearly on the number of DPs $|S|$ and the number of iterations, and logarithmically on the number of features c and batch size b; all these parameters depend also on the dataset size. As shown in Section V-B, the DA packing approach incurs a higher computation complexity but is embarrassingly parallel and can be more time-efficient than RBA depending on the available threads. The activation function is the only operation that requires ciphertext-ciphertext multiplications; its complexity depends logarithmically on the approximation degree. We empirically study the link between the approximation degree and the training accuracy in Section VIII-B. SPINDLE's other steps and protocols only involve lightweight operations, i.e., ciphertexts additions and multiplications with plaintext values.

B. *Empirical Evaluation*

We implemented SPINDLE in Go [46]. Our implementation builds on top of Lattigo [85], an open-source Go library for lattice-based cryptography, and Onet [97], an open-source Go library for building decentralized systems. The communication between data providers (DPs) is done through TCP with secure channels (using TLS). We evaluate our prototype on an emulated realistic network, with a bandwidth of 1 Gbps between every two nodes, using Mininet [86]. We deploy SPINDLE on 5 Linux machines with Intel Xeon E5-2680 v3 CPUs running at 2.5GHz with 24 threads on 12 cores and 256 Giga Bytes RAM, on which we evenly distribute the DPs.

We first provide SPINDLE's cryptographic operations micro-benchmarks before assessing SPINDLE accuracy and performance by testing it on multiple publicly-available datasets: CalCOFI [20] for linear regression, BCW [10], PIMA [102] and ESR [37] for logistic regression, and MNIST [70] for multinomial regression (see Appendix D-B for details on the datasets). We then show SPINDLE's scalability by using randomly generated (larger) datasets with up to 8,192 features and 4 million data samples. Our evaluation shows SPINDLE practicality for large-dimensional datasets, making it suitable for demanding learning tasks such as the training on imaging or genomic datasets [11], [36], [71].

We employ two sets of security parameters (SP), both ensuring 128-bit security: SP1: ($N = 2^{14}$, $Q = 2^{438}$, η=3.2, number of levels L = 9, scale $\Delta = 2^{34}$, degree of the approximated activation function $d = 5$) and SP2: ($N = 2^{13}$, $Q = 2^{218}$, η=3.2, $L = 6$, $\Delta = 2^{30}$, $d = 3$). SP2 is sufficient for linear regression and for specific logistic regression models that accept a low-degree d approximation. To account for a wider-range of solutions, we rely on SP1, unless otherwise stated. We employ the *local bootstrap* approach for all our experiments and for all datasets in *baseline comparison* except for ESR and CalCOFI, for which we use *global bootstrap*, as in most cases, doing multiple m local iterations between two global iterations yields a better accuracy. We further study the choice of bootstrapping strategy later in this section. Unless otherwise stated, we employ the *diagonal approach* (DA) for packing; we compare it with the *row-based approach* (RBA) in Figure 4a. In all our experiments, we consider SPINDLE's total time to train a regression model (including communication) without PREPARE, which is executed once and mostly involves light plaintext operations. E.g., the complete PREPARE takes 16.5s for a dataset of 40,000 samples distributed among 4 DPs. MAP accounts for up to 99.5% of SPINDLE's execution time. As shown in Section V, the DPs perform most of the computations in MAP, which is the only step with multiple local iterations, involving two matrix-vector multiplications, which span most (up to 97%) of its execution time. The remaining time corresponds to the computation of activation function and collective bootstrapping.

Encrypt	0.02
Add	$7\cdot10^{-4}$
Mult	$3\cdot10^{-3}$
Rot	0.08
Relin	0.07
Rescale	0.01

(a) Local Crypto. Ops.

D. Op.	5 DPs	10 DPs	20 DPs	40 DPs
DKeyGen	2.14	3.13	4.20	5.65
DBootstrap	0.26	0.37	0.47	0.61
DKeySwitch	0.26	0.36	0.45	0.57

(b) Distributed Crypto Ops.

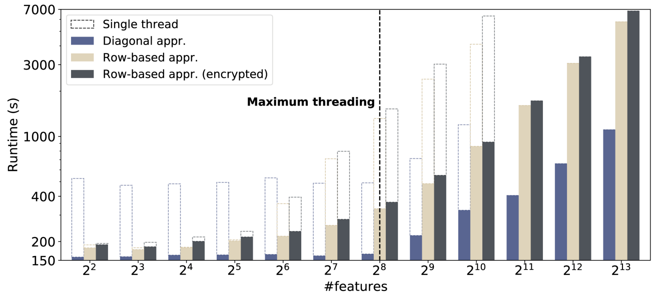
TABLE I: Crypto. micro-benchmarks in seconds with SP1.

Micro-benchmarks. Table I shows the execution time of each cryptographic operation. We observe that, as mentioned before, SPINDLE replaces the usually costly bootstrapping operation by an efficient interactive protocol DBootstrap$(\cdot)$. One of the most recent works on bootstrapping by Han and Ki [52] introduces a solution that only achieves around 108 bits of security (lower than the recommended 128 bits, due to recent attacks [127], [24]) and executes a CKKS bootstrapping in 26 seconds with ciphertexts that can encrypt 2^{13} values, corresponding to SP1, and about 20 seconds for SP2. This is two orders of magnitude slower than our DBootstrap$(\cdot)$, that achieves 128-bit security with execution times of 0.6 and 0.25 seconds for SP1 and SP2 respectively (with 40 DPs).

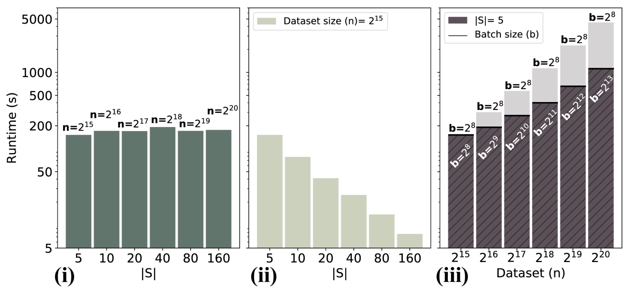
Dataset	Vers.	Acc./MSE	F1/MAE	T.	P.
CalCOFI [812,174x2]	CCS, [IT]	15.157, [408]	3.1, [19.67]	–	–
	DNP	17.679	3.45	6.71	$2 \cdot 10^{-4}$
	SPINDLE	17.938	3.62	65.31	0.23
PIMA [768x8]	CCS, [IT]	0.784, [0.720]	0.680, [0.604]	–	–
	DNP	0.781	0.679	0.038	$9 \cdot 10^{-5}$
	SPINDLE	0.780	0.677	11.28	0.18
BCW [699x9]	CCS, [IT]	0.962, [0.922]	0.947, [0.877]	–	–
	DNP	0.962	0.942	0.034	$5 \cdot 10^{-5}$
	SPINDLE	0.962	0.944	3.25	0.16
ESR [11,500x90]	CCS, [IT]	0.842, [0.838]	0.462, [0.396]	–	–
	DNP	0.840	0.460	2.89	$8 \cdot 10^{-5}$
	SPINDLE	0.839	0.456	53.27	0.35
MNIST [70,000 x 784] (multi.)	CCS, [IT]	0.873, [0.873]	0.871, [0.832]	–	–
	DNP	0.865	0.863	43.95	0.49
	SPINDLE	0.8617	0.86	558	4.33
MNIST [70,000 x 784] (1 vs. a)	CCS, [IT]	0.856, [0.827]	0.859, [0.822]	–	–
	DNP	0.853	0.858	43.98	0.49
	SPINDLE	0.852	0.850	187.8	4.33

TABLE II: Baseline Comparison with K-fold=5. Time to train (T.) and to predict (P.) are in seconds. MSE and MAE are given for the lin. reg. on CalCOFI. Accuracy and F1-score are given for all the others.

Baseline Comparison. To evaluate SPINDLE, we compare its performance (execution time and accuracy) against an *ideal* baseline, i.e., a non-privacy-preserving centralized cleartext solution (CCS) where a DP obtains the full dataset and trains the model on it. We consider the training time on the complete dataset and use the training batch size b as the number of data samples input for the prediction. Moreover, we compare SPINDLE with a distributed non-privacy-preserving (DNP) solution (cleartext values and exact activation functions), to show that our cryptographic approach and activation approximations introduce minimal accuracy degradation. Finally, to demonstrate the benefit of distributed learning approaches, we compare SPINDLE's accuracy with a case where one DP independently trains a model only on its local part of the distributed dataset (Independent Training, IT). In Table II, we show SPINDLE's accuracy (Acc.) (resp., Mean Squared Error, MSE) and F1-score (F1) (resp., Mean Average Error, MAE) for logistic and multinomial (resp., linear) regressions, achieved on the above datasets when they are split among 10 DPs. We refer to Table IVb in Appendix D for the learning parameters description. We observe that SPINDLE's accuracy loss is very low, up to 0.8%, with respect to a non-private centralized (CCS) solution where the model is trained on the full dataset (using standard SGD) with a standard Python library [110]. For instance, on ESR, CCS yields 84.2%, to 83.9% with SPINDLE. Moreover, this loss is mainly due to the data not being centralized, as SPINDLE consistently achieves almost the same accuracy as the decentralized non-private (DNP) equivalent. SPINDLE's total training time (column T. in Table II) is kept between 1 and 2 orders of magnitude higher than DNP, as the costly operations on encrypted data are partially amortized by SIMD operations enabled by the used packing. For instance, the training on the ESR dataset takes almost 3 seconds in DNP and 53.27 seconds in SPINDLE. We do not report the time for the centralized training (CCS and IT), as the settings are too different to be fairly comparable. Multinomial regression requires polynomial approximations of higher degree, i.e., between 15 and 19 (see Appendix IVb); its training on 70,000 records of 784 features (MNIST) is executed in 558 seconds (column T. in Table II). This time can be reduced to 187.8 seconds by performing 10 logistic regressions in parallel, one per label class (one-vs-all), at the cost of a 1% loss in accuracy. In all cases, when a DP independently trains on its part of the dataset (IT), i.e., with $1/10$-th of the data, the achieved accuracy is worse than the one achieved on the entire distributed dataset. As for prediction (P.), SPINDLE's prediction on 10 data samples of 90 features (ESR dataset) requires only 0.35 seconds by packing the input data and executing parallel computations. This time can be further amortized if more predictions are run in parallel.



(a) SPINDLE's performance with the nbr. of features (c).



(b) SPINDLE's performance with the nbr. of DPs ($|S|$) & records (n).

Fig. 4: SPINDLE's Scalability.

Scalability. We study how SPINDLE's execution time evolves when increasing the number of: features (c), data providers ($|S|$), and dataset samples (n). By default, we set $|S| = 5$, each DP having 5,120 data records (synthetically generated) with $c = 32$ features; we use a batch size $b = 256$, with $g = 5$ global iterations, and $m = 20$ local iterations in MAP. When comparing different approaches, we ensure that the number of times that the dataset is fully processed is constant, and we set the learning parameters accordingly. Figure 4a displays

SPINDLE's execution time with an increasing number of features c and shows that it scales logarithmically, in any of the used approaches. In this setting, we also study the influence of the multi-threading, the differences between the two packing approaches (Section V-B) and the impact of having encrypted input data (Section VI). When the computations are single-threaded, the row-based approach (RBA) is more efficient than the diagonal approach (DA) up to $c=128$ features, as RBA incurs fewer multiplications and rotations than DA. In contrast, the diagonal approach (DA) execution time in one or multiple threads is almost constant up to $c=256$ features (with a batch size $b=256$), as its complexity depends mainly on max(c,b) (Section V-B). However, the DA is *embarrassingly parallelizable*, and it is always faster when the computations are executed on 24 threads. As an example, on multiple threads and for 256 features, DA yields an execution time of 165s against 330s for RBA, and 365s when the input data are encrypted and using RBA. For both approaches, the parallelization is efficient up to $c=2^8$, where the maximum thread-utilisation is reached. Afterwards, both approaches scale linearly. When the data providers have encrypted input data (RBA-E), the execution time increases by 7% with respect to RBA.

Figure 4b.i shows that when the number of DPs $|S|$ increases and each DP has a fixed amount of data, SPINDLE's execution time is constant. This means that SPINDLE scales independently of $|S|$. In Figure 4b.ii, where $|S|$ increases but the total amount of data remains constant, SPINDLE's execution time decreases linearly, as the workload is efficiently distributed among the DPs. In Figure 4b.iii, when $|S|$ is constant and the size of the DPs' datasets increases, SPINDLE's execution time increases linearly with the amount of data. If the batch size can be increased when the data providers have more records, then SPINDLE's execution time can be further reduced. In summary, SPINDLE scales independently of the number of data providers, and linearly with the DPs' dataset size. It is able to train models with a high number of features and thus remains practical for real-world sized datasets. Finally, we note that SPINDLE scales similarly to a DNP solution in the three cases. For example, in the case of Figure 4b.ii, DNP ranges from 0.69 seconds with 5 DPs to 0.42 seconds with 10 DPs, whereas SPINDLE's execution time decreases from 150 to 78 seconds.

Bootstrapping & Activation Function Approx. Degree. In Table III, we observe that SPINDLE accuracy slightly improves with higher degree approximations of the sigmoid for the activation function. Relying on LB or HB, which require less global iterations and therefore less communication, and on low-degree approximations improves SPINDLE execution time but, it can lower the achieved accuracy. We remark that HB requires less bootstrapping operations, as the global weights are bootstrapped and assigned to the local weights. However, LB and HB execution times remain similar as in LB, the DP perform the bootstrappings in parallel.

$_{boot.}\backslash^{activ.}$	**3**	**5**	**7**
LB	0.837 ; 120	0.839 ; 125	0.839 ; 128
GB	0.843 ; 140	0.843 ; 143	0.844 ; 146
HB	0.844 ; 120	0.846 ; 124	0.846 ; 128

TABLE III: SPINDLE accuracy and timing *(accuracy;exec. time in sec)* to train on ESR with different bootstrapping strategies and degrees of the activation function.

Communication. With the security parameters SP1, the size of a ciphertext is 2.6MB and each DP receives and sends one ciphertext per global iteration. One ciphertext is also exchanged for each DBootstrap$(\cdot)$ (e.g., every two global iterations).

Comparison with prior art. Here we briefly compare, both qualitatively and quantitatively (when applicable), SPINDLE against (a) centralized cryptographic approaches (e.g., [66], [63], [21]), (b) cryptographic distributed solutions (e.g., [41], [28], [132]) and (c) federated learning solutions (e.g., [34], [55], [72], [112]). See Appendix D-C for an extended analysis.

(a) SPINDLE consistently outperforms centralized HE-based solutions (CES) as SPINDLE distributes the workload among multiple DPs and replaces, as shown before, the costly centralized bootstrapping operation by a lightweight interactive protocol.

(b) None of the existing HE-SMC-based distributed solutions [41], [28], [132] provides both data and model confidentiality, or covers the entire ML workflow (the trained model cannot be kept secret to perform oblivious predictions) or enables the distributed execution of the gradient descent. Moreover, some solutions ([41], [28]) leak more than only the trained model and rely on data encodings (or approximations) that lower the obtained accuracy, whereas in SPINDLE, we approximate only the activation functions. Finally, all previous solutions scale quadratically in at least one dimension, i.e., number of features c, samples n, or DPs $|S|$, whereas SPINDLE's execution time is almost independent of $|S|$, scales logarithmically with the number of features and linearly with the dataset size. Purely secret-sharing-based solutions [13], [26] consider substantially different settings as SPINDLE, as they require the DPs to communicate their data outside their premises and require an honest majority among a limited number of computing servers (typically, 2 to 4, depending on the setting). Whereas SPINDLE also works in this configuration, it is not specifically optimized for 2-4 parties and its execution time would be in the same order of magnitude but slower than secret-sharing-based solutions. This is due to the computation overhead introduced by operations on encrypted data. However, unlike secret-sharing-based solutions, SPINDLE efficiently scales to federated learning settings where many (hundreds of) DPs keep their data locally and can withstand up to N-1 out of N dishonest DPs.

(c) In basic federated learning solutions, data owners train and update the model on their local data and a server aggregates the model updates to obtain the global model [82], [68]. In this setting, the coordinating server has to be fully trusted, as some information can be inferred from the intermediate models, e.g., extracting participants' inputs [54], [123], [133] or membership inference [84], [92]. SPINDLE naturally thwarts federated-learning and model-inversion attacks, as the intermediate and final weights are never revealed. Federated learning approaches based on differential privacy (diffP), e.g., [72], [112], [83], train the model while introducing noise to the intermediate values to mitigate adversarial inferences. These approaches consider a different paradigm by introducing a tradeoff between privacy and accuracy, whereas in SPINDLE security is absolute, and the trade-off (accuracy vs execution time) is the same as for non-secure solutions, e.g., less training iterations can yield a less precise model. DiffP approaches can significantly degrade the data utility, and might require a high privacy budget for which it remains unclear what privacy protection is obtained in practice [58]. In Section IX, we discuss how membership inference and reconstruction attacks from the prediction outputs can also be mitigated in SPINDLE by adding differentially-private noise during the DKeySwitch$(\cdot)$.

IX. EXTENSIONS

We describe here extensions to SPINDLE that can be employed to withstand malicious adversaries, support dynamic DPs, enable quality control and support more complex ML models.

A. Malicious Adversaries

Malicious DPs Interfering with TRAINING. To limit the extent to which a malicious DP could interfere with its TRAINING, SPINDLE can require from the DPs to publish transcripts of their computations [41] and to produce proofs of correct inputs. These features combined would enable SPINDLE to be fully auditable. Mechanisms to avoid model poisoning attacks when the input data are encrypted (and have to remain confidential) are an open research problem. However, SPINDLE can partially mitigate this threat by constraining the DPs' inputs and requiring zero knowledge proofs of range [73], [125], [8], [9] from the DPs. This would substantially limit the extent to which a malicious DP could interfere with SPINDLE's TRAINING. However, we note that this does not thwart all possible attacks, as, for example, poisoning attacks would still be possible with plausible, i.e., in-range, input data.

Malicious DPs Interfering with PREDICTION. As for the TRAINING, computation correctness can be verified through computations' transcripts published by the DPs. To prevent a malicious DP from learning a victim querier's prediction outputs via a replay attack (i.e., reusing the querier's encrypted data in a new query), SPINDLE can require queriers to provide signed proofs of knowledge of the input data [79].

Malicious Querier Inferring Information from PREDICTION's Output. SPINDLE naturally covers federated learning attacks [54], [84], [92] and model inversion attacks [39], as the intermediate and final weights are never revealed. Moreover, SPINDLE can also mitigate inference attacks, e.g., membership inference [113], by limiting the number of prediction requests on the trained model. This solution can be improved by adding noise to the prediction output to achieve differential privacy guarantees. In fact, a mechanism that ensures differential privacy can be used for all the outputs of SPINDLE: on the predictions $\boldsymbol{y'}$ and on the trained model, if it is released after training (Section VI). This would ensure that a passive adversary (e.g., trying to infer information from the system's outputs) or an active adversary controlling a subset of the DPs cannot learn information, e.g., data or local model of honest parties, about a subset of the DPs. To ensure differential privacy, SPINDLE should add some collectively generated noise [40], [65] to the query result before performing DKeySwitch$(\cdot)$. However, the choice of the privacy parameters is not trivial and is an interesting direction for future work. Furthermore, the use of differential privacy in dynamic systems presents serious limitations; minimizing the released non-encrypted information (which also reduces the noise magnitude required to meet a target differential privacy level) is much more effective and practical. This is the approach taken in SPINDLE, contrarily to federated learning systems, where the intermediate outputs of each training iteration are always disclosed.

B. Modular Extensions

We discuss here a set of extensions that can be (optionally) integrated and combined in SPINDLE depending on the application.

Threshold-encryption Scheme. To account for unresponsive DPs, SPINDLE can use a threshold-encryption scheme, where the DPs secret-share [111] their secret keys, thus enabling a subset of the DPs to perform the cryptographic interactive protocols (DBootstrap$(\cdot)$ or DKeySwitch$(\cdot)$).

Dynamic Roles. The role of DP_R played by one DP has no security implications and only incurs small computation overhead for one DP. This role can be dynamically assigned (e.g., round robin) at each global iteration or whenever the DP playing DP_R becomes unavailable.

Asynchronous Learning & Performance Optimizations. We experimentally observed that an uneven distribution of the data across DPs does not affect the training accuracy. However, and as expected, in order to obtain similar accuracy as a centrally trained model, the labels of the DPs' local datasets should be similarly distributed. For this, SPINDLE can integrate optimizations of the stochastic gradient descent (SGD) that can be expressed as a polynomial; in particular, SGD asynchronous variants that account for imbalances in DPs' response times or data distribution, or for sparse networks adaptations (e.g., Koloskova et al. [67]).

To avoid over- or under-fitting, which often happens when the number of training iterations is predefined, SPINDLE can integrate a collective stop-test protocol. This protocol enables the data providers to collectively decrypt the absolute difference between the (global) weights of two subsequent (global) iterations, or a statistic derived from these values. The decrypted value is compared to a chosen threshold to stop the training.

Data Preparation & Quality Control. As mentioned before, the training on a distributed dataset can be optimized according to how the data are distributed among the DPs. This information can also serve for data standardization and quality control, and its leakage can be mitigated by relying on differential privacy or on HE-based interactive protocols [41]. SPINDLE's PREPARE phase can be extended to include these solutions and it is up to the DPs to choose the configuration that achieves the required balance between privacy and performance.

C. More Complex ML Models.

We first remark that the extended, privacy-preserving MapReduce abstraction on which we rely to build SPINDLE can actually capture many of existing solutions for secure distributed ML training [112], [41], [28], [132], [14], [96], [44], [88], [13], [26]. We also remark that, even though we rely on the widely applicable distributed stochastic gradient descent (SGD), other distributed approaches for training ML models such as ADMM [19] could also be expressed in the same abstraction. However, by relying on SGD, we aim at designing a system that can then be extended to other models, as SGD can be used to minimize many cost functions [69], [117], [131]. In particular, it can be extended to more complex models such as neural networks, which are usually trained using SGD [32]. SPINDLE supports any activation function that can be "practically" approximated by a polynomial; hence, the challenges for its extension to more complex models reside in trading-off precision for efficiency when approximating non-polynomial functions, and efficiently packing the data depending on the operations. This is particularly important for neural networks in which the computations are sequentially performed through multiple layers. Thus, each SGD iteration would involve higher multiplicative-depth circuits and their evaluation under encryption.

X. Conclusion

By extending the MapReduce abstraction, we have proposed a generic solution to the problem of privacy-preserving distributed ML model training and prediction. Our abstraction enables us to optimize the application of protection primitives from multiparty homomorphic encryption in a MapReduce workflow. We proposed SPINDLE, a privacy-preserving system that enables the execution of a distributed stochastic gradient descent and we have instantiated our quantum-resistant solution for the training and oblivious prediction on generalized linear models. We have shown that SPINDLE achieves accuracy comparable to non-secure centralized solutions, and it scales independently of the number of DPs and linearly or better with the size of the DPs' local datasets and the number of features. This makes it particularly suitable for difficult and demanding learning tasks that have to be performed on sensitive data that cannot be shared. This is the case in many domains and particularly in medicine, where complex sensitive datasets partitioned across medical institutions need to be regularly analyzed, e.g., Genome Wide Association Studies. SPINDLE achieves better performance than existing centralized and distributed solutions by leveraging the data providers concurrent computation on their local data, and using a multiparty encryption scheme that replaces costly homomorphic operations (e.g., bootstrapping) by efficient collective protocols. To the best of our knowledge, SPINDLE is the first highly scalable system enabling the distributed execution of the gradient descent across hundreds of parties and large datasets in a privacy-preserving, post-quantum, and efficient way.

Acknowledgment

We would like to thank all of those who reviewed the manuscript, in particular: Henry Corrigan-Gibbs, the members of the EPFL Laboratory for Data Security and the anonymous reviewers. We also thank Yupeng Zhang for shepherding the paper. This work was partially supported by the grant #2017-201 of the Strategic Focal Area "Personalized Health and Related Technologies (PHRT)" of the ETH Domain.

References

[1] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
[2] M. Abadi et al. Deep learning with differential privacy. In *ACM CCS*, 2016.
[3] A. Akavia, H. Shaul, M. Weiss, and Z. Yakhini. Linear-Regression on Packed Encrypted Data in the Two-Server Model. In *ACM WAHC*, 2019.
[4] M. Albrecht et al. Homomorphic Encryption Security Standard. Technical report, HomomorphicEncryption.org, 2018.
[5] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. of Mathematical Cryptology*, 2015.
[6] S. V. Algesheimer J., Camenisch J. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *CRYPTO*, 2002.
[7] Y. Aono, T. Hayashi, L. Trieu Phong, and L. Wang. Scalable and secure logistic regression via homomorphic encryption. In *ACM CODASPY*, 2016.
[8] C. Baum, I. Damgård, S. Oechsner, and C. Peikert. Efficient commitments and zero-knowledge protocols from ring-sis with applications to lattice-based threshold cryptosystems. *IACR Cryptol. ePrint Arch.*, 2016.
[9] C. Baum and A. Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In *PKC*, 2020.
[10] Breast Cancer Wisconsin (Original). https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original) (14.02.2020).
[11] A. L. Beam and I. S. Kohane. Big data and machine learning in health care. *Jama*, 2018.
[12] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski. nGraph-HE2: A High-Throughput Framework for Neural Network Inference on Encrypted Data. In *ACM WAHC*, 2019.
[13] D. Bogdanov, L. Kamm, S. Laur, and V. Sokk. Rmind: a tool for cryptographically secure statistical analysis. *IEEE TDSC*, 2016.
[14] K. Bonawitz et al. Towards federated learning at scale: System design. In *SysML*, 2019.
[15] C. Bonte and F. Vercauteren. Privacy-preserving logistic regression training. *BMC medical genomics*, 2018.
[16] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *IMACC*, 2013.
[17] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.
[18] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 2018.
[19] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine learning*, 2011.
[20] CalCOFI, over 60 years of oceanographic data. https://www.kaggle.com/sohier/calcofi (05.03.2020).
[21] S. Carpov, N. Gama, M. Georgieva, and J. R. Troncoso-Pastoriza. Privacy-preserving semi-parallel logistic regression training with fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2019.
[22] K. Chaudhuri and C. Monteleoni. Privacy-preserving logistic regression. In *NIPS*, 2009.
[23] H. Chen, R. Gilad-Bachrach, K. Han, Z. Huang, A. Jalali, K. Laine, and K. Lauter. Logistic regression over encrypted data from fully homomorphic encryption. *BMC medical genomics*, 2018.
[24] J. H. Cheon, M. Hhan, S. Hong, and Y. Son. A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret LWE. *IEEE Access*, 2019.
[25] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*, 2017.
[26] H. Cho, D. Wu, and B. Berger. Secure genome-wide association analysis using multiparty computation. *Nature Biotech.*, 2018.

[27] C.-T. Chu et al. Map-reduce for machine learning on multicore. In *NIPS*, 2007.
[28] H. Corrigan-Gibbs and D. Boneh. Prio: Private, Robust, and Computation of Aggregate Statistics. In *NSDI*, 2017.
[29] J. L. Crawford, C. Gentry, S. Halevi, D. Platt, and V. Shoup. Doing real work with FHE: The case of logistic regression. In *ACM WAHC*, 2018.
[30] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.
[31] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 2008.
[32] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai. Gradient descent finds global minima of deep neural networks. *CoRR*, abs/1811.03804, 2018.
[33] S. S. Du, X. Zhai, B. Poczos, and A. Singh. Gradient descent provably optimizes over-parameterized neural networks. *arXiv preprint arXiv:1810.02054*, 2018.
[34] W. Du, A. Li, and Q. Li. Privacy-Preserving Multiparty Learning For Logistic Regression. In *SecureComm*, 2018.
[35] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans-IT*, 1985.
[36] B. J. Erickson, P. Korfiatis, Z. Akkus, and T. L. Kline. Machine learning for medical imaging. *Radiographics*, 2017.
[37] Epileptic Seizure Recognition Dataset. https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition (14.02.2020).
[38] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012.
[39] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM CCS*, 2015.
[40] D. Froelicher, P. Egger, J. S. Sousa, J. L. Raisaro, Z. Huang, C. V. Mouchet, B. Ford, and J.-P. Hubaux. Unlynx: A decentralized system for privacy-conscious data sharing. *PETS*, 2017.
[41] D. Froelicher, J. R. Troncoso-Pastoriza, J. S. Sousa, and J. Hubaux. Drynx: Decentralized, secure, verifiable system for statistical queries and machine learning on distributed datasets. *IEEE TIFS*, 2020.
[42] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans. Privacy-preserving distributed linear regression on high-dimensional data. *PETS*, 2017.
[43] The EU General Data Protection Regulation. https://eugdpr.org/(10.11.2019).
[44] I. Giacomelli, S. Jha, M. Joye, C. D. Page, and K. Yoon. Privacy-preserving ridge regression with only linearly-homomorphic encryption. In *ACNS*, 2018.
[45] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*, 2016.
[46] Go Programming Language. https://golang.org,(10.11.2019).
[47] L. Gomes. Quantum computing: Both here and not here. *IEEE Spectrum*, 2018.
[48] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
[49] Google CEO Sundar Pichai on achieving quantum supremacy. https://tinyurl.com/y5rnowlc (07.11.2019).
[50] T. Graepel, K. Lauter, and M. Naehrig. ML confidential: Machine learning on encrypted data. In *ICISC*, 2012.
[51] S. Halevi and V. Shoup. Algorithms in HElib. In *CRYPTO*, 2014.
[52] K. Han and D. Ki. Better bootstrapping for approximate homomorphic encryption. In *CT-RSA*, 2020.
[53] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright. Privacy-preserving machine learning as a service. *PETS*, 2018.
[54] B. Hitaj, G. Ateniese, and F. Perez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. In *ACM CCS*, 2017.
[55] Z. Huang, R. Hu, Y. Guo, E. Chan-Tin, and Y. Gong. DP-ADMM: ADMM-based distributed learning with differential privacy. *IEEE TIFS*, 2019.
[56] Quantum Computing is "no longer science fiction," says IBM. https://tinyurl.com/y4zvlsll, (10.02.2020).
[57] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, D. Boneh, and G. Bejerano. Deriving genomic diagnoses without revealing patient genomes. *Science*, 2017.
[58] B. Jayaraman and D. Evans. Evaluating differentially private machine learning in practice. In *USENIX Security*, 2019.
[59] B. Jayaraman, L. Wang, D. Evans, and Q. Gu. Distributed learning without distress: Privacy-preserving empirical risk minimization. In *NIPS*, 2018.
[60] Y. Jiang et al. SecureLR: Secure logistic regression model via a hybrid cryptographic protocol. *IEEE TCB*, 2019.
[61] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security*, 2018.
[62] Why we shouldn't disregard the nda. tinyurl.com/y4hdr42d.
[63] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC genomics*, 2018.
[64] M. Kim, J. Lee, L. Ohno-Machado, and X. Jiang. Secure and differentially private logistic regression for horizontally distributed data. *IEEE TIFS*, 2019.
[65] M. Kim, J. Lee, L. Ohno-Machado, and X. Jiang. Secure and differentially private logistic regression for horizontally distributed data. *IEEE TIFS*, 2020.
[66] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR medical informatics*, 2018.
[67] A. Koloskova, S. U. Stich, and M. Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication. *CoRR*, abs/1902.00340, 2019.
[68] J. Konečný, H. McMahan, D. Ramage, and P. Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.
[69] A. Kumar, J. Naughton, and J. M. Patel. Learning generalized linear models over normalized data. In *ACM SIGMOD*, 2015.
[70] Y. LeCun and C. Cortes. MNIST handwritten digit database. *http://yann.lecun.com/exdb/mnist/*, 2010.
[71] M. K. Leung, A. Delong, B. Alipanahi, and B. J. Frey. Machine learning in genomic medicine: a review of computational problems and data sets. *Proceedings of the IEEE*, 2015.
[72] W. Li et al. Privacy-preserving federated brain tumour segmentation. In *MLMI*, 2019.
[73] B. Libert, S. Ling, K. Nguyen, and H. Wang. Lattice-based zero-knowledge arguments for integer relations. In *CRYPTO*, 2018.
[74] Y. Lindell. How to simulate it–a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*. 2017.
[75] R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In *CT-RSA*, 2011.
[76] J. K. Lindsey. *Applying generalized linear models*. Springer Science & Business Media, 2000.
[77] Why NDAs often don't work when expected to do so and what to do about it. https://tinyurl.com/y64qlzs9.
[78] J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *ACM CCS*, 2017.
[79] V. Lyubashevsky, N. K. Nguyen, and G. Seiler. Practical lattice-based zero-knowledge proofs for integer relations. In *ACM CCS*, 2020.
[80] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, 2010.
[81] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.
[82] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016.
[83] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. In *ICLR*, 2018.
[84] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *IEEE S&P*, 2019.
[85] Lattigo: A library for lattice-based homomorphic encryption in go. https://github.com/ldsec/lattigo (14.02.2019).
[86] Mininet. http://mininet.org (13.12.2019).
[87] P. Mohassel and P. Rindal. ABY 3: a mixed protocol framework for machine learning. In *ACM CCS*, 2018.
[88] P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE S&P*, 2017.
[89] M. Mosca. Cybersecurity in an era with quantum computers: Will we be ready? *IEEE S&P*, 2018.
[90] M. Mostert, A. Bredenoord, M. Biesaart, and J. Delden. Big data in medical research and EU data protection law: challenges to the consent or anonymise approach. *European Journal of Human Genetics*, 2016.
[91] C. Mouchet, J. R. Troncoso-pastoriza, J.-P. Bossuat, and J. P. Hubaux. Multiparty homomorphic encryption: From theory to practice. In *Tech. Report https://eprint.iacr.org/2020/304*, 2019.
[92] M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *IEEE S&P*, 2019.
[93] C. Neill et al. A blueprint for demonstrating quantum supremacy with superconducting qubits. *Science*, 2018.

[94] J. A. Nelder and R. W. M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society*, 1972.
[95] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical programming*, 2005.
[96] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE S&P*, 2013.
[97] Cothority network library. https://github.com/dedis/onet,.
[98] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.
[99] A. Paszke et al. Automatic differentiation in PyTorch. 2017.
[100] M. Pathak, S. Rane, and B. Raj. Multiparty differential privacy via aggregation of locally trained classifiers. In *NIPS*, 2010.
[101] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE TIFS*, 2018.
[102] Pima Indians Diabetes Dataset. https://tinyurl.com/y8o3x8me (14.04.2018).
[103] M. Pratyush, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa. Delphi: A cryptographic inference service for neural networks. In *USENIX Security*, 2020.
[104] R. Rachuri and A. Suresh. Trident: Efficient 4PC framework for privacy preserving machine learning. In *NDSS*, 2020.
[105] M. S. Riazi et al. Chameleon: A hybrid secure computation framework for machine learning applications. In *ASIACCS*, 2018.
[106] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar. XONN: XNOR-based oblivious deep neural network inference. In *USENIX Security*, 2019.
[107] B. D. Rouhani, M. S. Riazi, and F. Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *ACM DAC*, 2018.
[108] B. Schoenmakers and P. Tuyls. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *EUROCRYPT*, 2006.
[109] P. Schoppmann, A. Gascon, M. Raykova, and B. Pinkas. Make some room for the zeros: Data sparsity in secure distributed machine learning. In *ACM CCS*, 2019.
[110] Scikit-learn, Machine Learning in Python. https://scikit-learn.org/stable/,(29.02.2020).
[111] A. Shamir. How to share a secret. *Communications of the ACM*, 1979.
[112] R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In *ACM CCS*, 2015.
[113] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *IEEE S&P*, 2017.
[114] I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez, et al. A berkeley view of systems challenges for ai. *arXiv preprint arXiv:1712.05855*, 2017.
[115] B. Terhal. Quantum supremacy, here we come. *Nature Physics*, 2018.
[116] R. Toshniwal, K. Dastidar, and A. Nath. Big data security issues and challenges. *International Journal of Innovative Research in Advanced Engineering*, 2015.
[117] P. Toulis, E. Airoldi, and J. Rennie. Statistical analysis of stochastic gradient methods for generalized linear models. In *ICML*, 2014.
[118] S. Truex et al. A hybrid approach to privacy-preserving federated learning. In *ACM AISec*, 2019.
[119] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer. A survey on distributed machine learning. *arXiv preprint arXiv:1912.09789*, 2019.
[120] S. Wagh, D. Gupta, and N. Chandran. SecureNN: 3-party secure computation for neural network training. *PETS*, 2019.
[121] J. Wang and G. Joshi. Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms. *CoRR*, abs/1808.07576, 2018.
[122] J. Wang and G. Joshi. Cooperative SGD: A unified framework for the design and analysis of communication-efficient sgd algorithms. In *ICML CodML Workshop*, 2019.
[123] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM*, 2019.
[124] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Scalable anonymous group communication in the anytrust model. Technical report, Naval Research Lab Washington DC, 2012.
[125] R. Yang, M. H. Au, Z. Zhang, Q. Xu, Z. Yu, and W. Whyte. Efficient lattice-based zero-knowledge arguments with standard soundness: construction and applications. In *CRYPTO*, 2019.
[126] A. C.-C. Yao. How to generate and exchange secrets. In *IEEE SFCS*, 1986.
[127] Yongha Son and Jung Hee Cheon. Revisiting the hybrid attack on sparse and ternary secret LWE. *Technical Report https://eprint.iacr.org/2019/1019,*, 2019.
[128] A. Zalcman et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 2019.
[129] D. Zhang. Big data security and privacy protection. In *ICMCS*, 2018.
[130] S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging sgd. In *NIPS*, 2015.
[131] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML*, 2004.
[132] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica. Helen: Maliciously Secure Coopetitive Learning for Linear Models. In *IEEE S&P*, 2019.
[133] L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. In *NIPS*. 2019.
[134] X. Zhu, C. Vondrick, C. C. Fowlkes, and D. Ramanan. Do we need more training data? *Int. J. Comput. Vision*, 2016.

APPENDIX A
MULTIPARTY HOMOMORPHIC ENCRYPTION (MHE)

We describe here the cryptographic operations and distributed protocols that are used in SPINDLE.

Operations: In Scheme A.1, we introduce CKKS operations. $\boldsymbol{v}$ is a vector of cleartext values, sk and pk are the secret and public keys, and evk is an evaluation key.

Encrypt:	$\langle \boldsymbol{v}\rangle_{pk} = \{\{\langle \boldsymbol{v}\rangle,\tau,\Delta\},L,\Delta\}_{pk} = \text{Enc}(pk,\boldsymbol{v})$
Decrypt:	$\boldsymbol{v} = \text{Dec}(sk,\langle \boldsymbol{v}\rangle_{pk})$
Add:	$\{\langle \boldsymbol{v}_1\rangle + \langle \boldsymbol{v}_2\rangle,\min(\tau,\tau'),\max(\Delta,\Delta')\} =$
	$\{\langle \boldsymbol{v}_1\rangle,\tau,\Delta\} + \{\langle \boldsymbol{v}_2\rangle,\tau',\Delta'\}$
Mult:	$\{\langle \boldsymbol{v}_3\rangle,\min(\tau,\tau'),\Delta\Delta'\} = \text{M}(\{\langle \boldsymbol{v}_1\rangle,\tau,\Delta\},\{\langle \boldsymbol{v}_2\rangle,\tau',\Delta'\})$
Rot:	$\{\langle \boldsymbol{v'}\rangle,\tau,\Delta\} = \text{RotL/R}(\{\langle \boldsymbol{v}\rangle,\tau,\Delta\},r,evk)$
Rescale:	$\{\langle \boldsymbol{v}\rangle,\tau-1,\Delta'\} = \text{ReScale}(\{\langle \boldsymbol{v}\rangle,\tau,\Delta\})$
Relin:	$\{\langle \boldsymbol{v}\rangle,\tau,\Delta\} = \text{Relin}(\{\langle \boldsymbol{v}\rangle,\tau,\Delta\},evk)$
Bootstrap:	$\{\langle \boldsymbol{v}\rangle,\tau,\Delta\} = \text{Bootstrap}(\{\langle \boldsymbol{v}\rangle,0,\Delta\},evk)$

Scheme A.1: CKKS operations.

Distributed Protocols: In Scheme A.2, we define four protocols using the distributed version of CKKS. These protocols require the participation of all DPs, i.e., each DP_i contributes its respective secret key sk_i. DKeyGen$(\cdot)$ generates the collective public key pk and evaluation keys $evks$ by combining the protocols defined by Mouchet et al. [91]. These keys can then be used independently (without interaction) by a DP on a ciphertext $\langle \boldsymbol{v}\rangle_{pk}$. The distribution of the scheme enables an efficient distributed bootstrapping DBootstrap$(\langle \boldsymbol{v}\rangle,\tau_b,\Delta,\{sk_i\})$ to collective refresh $\langle \boldsymbol{v} \rangle$ to its initial level L. The minimum level τ_b at which the bootstrapping has to be performed depends on the security parameters. The DKeySwitch$(\cdot)$ enables the DPs to change a ciphertext encryption from the public key pk to another public key pk', without decrypting the ciphertext. A distributed decryption operation DDec$(\cdot)$ is a special case of the DKeySwitch$(\cdot)$ where there is no pk', i.e., pk' is 0.

Distrib. Key Gen:	$pk,evks = \text{DKeyGen}(\{sk_i\})$
Distrib. Bootstrap:	$\{\langle \boldsymbol{v} \rangle, L, \Delta\} = \text{DBootstrap}(\langle \boldsymbol{v} \rangle, \tau_b, \Delta, \{sk_i\})$
Distrib. Key Switch:	$\langle \boldsymbol{v} \rangle_{pk'} = \text{DKeySwitch}(\langle \boldsymbol{v} \rangle_{pk}, pk', \{sk_i\})$
Distrib. Decrypt:	$\boldsymbol{v} = \text{DDec}(\langle \boldsymbol{v} \rangle, \{sk_i\})$

Scheme A.2: Distributed CKKS operations.

Appendix B
Activation Functions

We describe how we evaluate a polynomial approximation and how we approximate the maximum function.

Polynomial Approximation Protocol 6 inductively computes the (element-wise) exponentiation of the encrypted input vector $\langle \boldsymbol{u} \rangle$: $\langle \boldsymbol{u}^1 \rangle$, $\langle \boldsymbol{u}^2 \rangle$, ..., $\langle \boldsymbol{u}^{2^k-1} \rangle$, $\langle \boldsymbol{u}^{2^k} \rangle$, $\langle \boldsymbol{u}^{2^{k+1}} \rangle$, ..., $\langle \boldsymbol{u}^{2^{\omega-1}} \rangle$ (Protocol 6, line 2), where ω is the smallest value satisfying $2^\omega > d(p(\langle \boldsymbol{u} \rangle))$ and $k = \lfloor \omega/2 \rfloor$. Then, it recursively evaluates $p(\langle \boldsymbol{u} \rangle) = \sum_{i=1,2,3...,d} r_i \langle \boldsymbol{u}^i \rangle = \langle \boldsymbol{u}^{2^{\omega-1}} \rangle q(\langle \boldsymbol{u} \rangle) + R(\langle \boldsymbol{u} \rangle)$ (Protocol 6, line 3). Note that $p(\cdot)$, $q(\cdot)$, and $R(\cdot)$ are functions of $\langle \boldsymbol{u} \rangle$ and of the approximation coefficients $\boldsymbol{r}$, $q(\cdot)$ is the quotient of the division of the actual activation function $p(\cdot)$ by $\langle \boldsymbol{u}^{2^{\omega-1}} \rangle$, and $R(\cdot)$ is the remainder of the division. $d(x)$ is a function that outputs the degree of x.

Protocol 6 Encrypted Poly. Approx. Evaluation AF$(\cdot)$.

```
Func. AF($\langle \boldsymbol{u} \rangle, d, \boldsymbol{r}$) outputs $\langle \boldsymbol{a} \rangle$ the evaluated poly. approx. of $\langle \boldsymbol{u} \rangle$
1: Choose the smallest $\omega$ such that $2^\omega > d$ and define $k = \lfloor \omega/2 \rfloor$
2: Compute $\{u_i\} = \langle \boldsymbol{u}^1 \rangle, \langle \boldsymbol{u}^2 \rangle, ..., \langle \boldsymbol{u}^{2^{k\text{-}1}} \rangle, \langle \boldsymbol{u}^{2^k} \rangle, \langle \boldsymbol{u}^{2^{k+1}} \rangle, ..., \langle \boldsymbol{u}^{2^{\omega\text{-}1}} \rangle$
   inductively and call **paRecu**($\boldsymbol{r}, d, \{u_i\}$)
3: **Function paRecu**($\boldsymbol{r}, d, \{u_i\}$)**:**
4:     Choose the smallest $\omega$ such that $2^\omega > d$
5:     Find polynomials $q(\langle \boldsymbol{u} \rangle)$ and $R(\langle \boldsymbol{u} \rangle)$ with $\langle \boldsymbol{a} \rangle =$
       $\langle \boldsymbol{u}^{2^{\omega-1}} \rangle q(\langle \boldsymbol{u} \rangle) + R(\langle \boldsymbol{u} \rangle)$ such that $\langle \boldsymbol{a} \rangle = \sum_{i=1,2,..,d} \boldsymbol{r}[i] \langle \boldsymbol{u}^i \rangle$
6:     **If** $d(q), d(R) \leq 2$ **:**
          Evaluate $q(\langle \boldsymbol{u} \rangle) =$ **paRecu**($\boldsymbol{r}, d = d(q), \{u_i\}$)
          and $R(\langle \boldsymbol{u} \rangle) =$ **paRecu**($\boldsymbol{r}, d = d(R), \{u_i\}$)
7:     **Else** Return $\langle \boldsymbol{a} \rangle$
```

Approximation of the maximum function Protocol (Protocol 7) computes the approximation of the maximum function. It takes an encrypted matrix $\langle \boldsymbol{U}_{|cl| \times c} \rangle$, the approximations intervals $[a_i, g_i]$ and degrees $\boldsymbol{d}$, and computes an encrypted vector $\langle \boldsymbol{m} \rangle$ that contains a close approximation of the max of each column of $\langle \boldsymbol{U} \rangle$.

Protocol 7 Approximation of the max function apMax$(\cdot)$.

```
$\langle \boldsymbol{m} \rangle \leftarrow$ apMax($\langle \boldsymbol{U} \rangle, [a_i, g_i], \boldsymbol{d}$)
1: $\langle \boldsymbol{u'} \rangle = \sum_{\lambda=0}^{|cl|} \langle \boldsymbol{U}[\lambda, \cdot] \rangle$
2: **for** $\lambda = 1, ..., |cl|$**:** $\langle \boldsymbol{U}[\lambda, \cdot] \rangle = (\langle \boldsymbol{U}[\lambda, \cdot] \rangle - \langle \boldsymbol{u'}[\lambda, \cdot] \rangle)$
3: $\boldsymbol{r} \leftarrow$ GetAFCoefficients($(1/h')e^{(x/h)}, [a_1, g_1], \boldsymbol{d}[1]$), where $h, h'$ are predefined constants
4: **for** $\lambda = 1, ..., |cl|$**:** $\langle \boldsymbol{U''}[\lambda, \cdot] \rangle =$ AF($\langle \boldsymbol{U}[\lambda, \cdot] \rangle, \boldsymbol{d}[1], \boldsymbol{r}$)
5: $\langle \boldsymbol{o} \rangle = \sum_{\lambda=0}^{|cl|} \langle \boldsymbol{U''}[\lambda, \cdot] \rangle$
6: $\boldsymbol{r'} \leftarrow$ GetAFCoefficients($\{1/x\}, [a_2, g_2], \boldsymbol{d}[2]$)
7: $\langle \boldsymbol{o} \rangle =$ AF($\langle \boldsymbol{o} \rangle, \boldsymbol{d}[2], \boldsymbol{r'}$)
8: **for** $\lambda = 1, ..., |cl|$**:** $\langle \boldsymbol{U}[\lambda, \cdot] \rangle =$ M($\langle \boldsymbol{U}[\lambda, \cdot] \rangle, \langle \boldsymbol{U''}[\lambda, \cdot] \rangle$)
9: $\langle \boldsymbol{m} \rangle = \sum_{\lambda=0}^{|cl|} (\langle \boldsymbol{U}[\lambda, \cdot] \rangle, \langle \boldsymbol{o} \rangle)$
```

APPENDIX C
SECURITY OF DBOOTSTRAP($\cdot$)

The original distributed bootstrapping protocol for BFV [38] is presented by Mouchet et al. [91]. In this protocol, the data providers produce an additive sharing of the encrypted ciphertext by masking their share in the decryption, before collectively encrypting their share to collectively produce a new (fresh) encryption of the same value. We adapted this protocol to the CKKS scheme [25]. The protocol steps remain the same but the underlying computational assumptions are different. In fact, in CKKS the shares created by the data providers are not unconditionally hiding, but statistically or computationally hiding due to the incomplete support of the used masks. The proof for the protocol's CKKS version (DBootstrap($\cdot$)) follows from the proof provided by Mouchet et al. in the passive-adversary model of the BFV bootstrapping protocol with the additional assumption that Lemma 1 is true. This lemma guarantees the statistical indistinguishablity of the shares in $\mathbb{C}$. The RLWE problem is hard if the adversary is computationally-bounded, whereas Lemma 1 relies on a statistical argument. However, both share the same security bound given the same security parameter and DBootstrap($\cdot$) provides the same computational security as Mouchet et al. [91] original protocol.

Lemma 1. *Given the distribution $P_0=(a+b)$ and $P_1=c$ with $0\leq a<2^{\delta}$ and $0\leq b,c<2^{\lambda+\delta}$ and b, c uniform, then the distributions P_0 and P_1 are λ-indistinguishable; i.e., a probabilistic polynomial adversary $\mathcal{A}$ cannot distinguish between both with probability greater than $2^{-\lambda}$: $|Pr[\mathcal{A}\to 1|P=P_1]-Pr[\mathcal{A}\to 1|P=P_0]|\leq 2^{-\lambda}$.*

Proof: We refer to Algesheimer et. al [6], Section 3.2 and Schoenmakers and Tuyls [108], Appendix A, for the proof of the statistical λ-indistinguishability.

We recall that an encoded message msg of $N/2$ complex numbers with the CKKS scheme is an integer polynomial of $\mathbb{Z}[X]/(X^N+1)$. Given that $||msg||<2^{\delta}$, and a second polynomial M of N integer coefficients with each coefficient uniformly sampled and bounded by $2^{\lambda+\delta}-1$ for a security parameter λ, Lemma 1 suggests that Pr$[||msg^{(i)}+M^{(i)}||\geq 2^{\lambda+\delta}]\leq 2^{-\lambda}$, for $0\leq i<N$ and where i denotes the i^{th} coefficient of the polynomial. That is, the probability of a coefficient of $msg+M$ to be distinguished from a uniformly sampled integer in $[0,2^{\lambda+\delta})$ is bounded by $2^{-\lambda}$. In Mouchet et al. protocol, each party samples its polynomial mask M with uniform coefficients in $[0,2^{\lambda+\delta})$.

The parties, however, should have an estimate of the magnitude of msg to derive δ, which can be derived from the plaintext scale, integer precision and previous homomorphic operations. The masks M_i are added to the ciphertext of R_{Q_ℓ} during the switch to the secret-shared domain. To avoid a modular reduction of the masks in R_{Q_ℓ} and ensure a correct re-encryption in R_{Q_L}, the modulus Q_ℓ should be large enough for the additions of N masks.

APPENDIX D
EVALUATION

We first provide the complete complexity analysis of SPINDLE. We describe in more details the datasets used to assess SPINDLE's training accuracy and show in Table IVb an extended version of Table II including the learning and approximation parameters.

	Comm. (tot)	**Comput. (per DP_i)**
MAP (RBA)	$\tilde{B_M}$	$gm(4M+2(log_2(b)+log_2(c))$ $R+\sigma)\#ct+\tilde{B_M}$
MAP (DA)	$\tilde{B_M}$	$(2(N_1N_2M+(N_1+N_2-2)$ $R)+\sigma)gm+\tilde{B_M}$
COMBINE	$g(\|S\|-1)\|ct\|$	$3A$
REDUCE	$2g(\|S\|-1)\|ct\|+\tilde{B_R}$	$M+\tilde{B_R}$
PRED	$2(\|ct\|+(\|S\|-1)\|ct\|)$	$M'+\sigma+D$

DBoot.	$\tilde{B_R}=2\|ct\|(\|S\|-1)f(g)$,
	$\tilde{B_M}=2\|ct\|\|S\|(\|S\|-1)f(g,m)$
	if LB: $\tilde{B_R}=0$, **if GB:** $\tilde{B_M}=0$,
	if HB: $f(g)\to f(g,m)$
		$\tilde{B_R}=(D+E)f(g))$
		$\tilde{B_M}=\|S\|(D+E)f(g,m)$
		if LB: $\tilde{B_R}=0$, **if GB:** $\tilde{B_M}=0$,
		if HB: $f(g)\to f(g,m)$

$\boldsymbol{\sigma}$		$(log_2(d)+1)(M+M'+A)$

(a) Theoretical Analysis. The number of HB always depends on both g and m.

Dataset	**Vers.**	$\mathbf{SP},\alpha,\rho,\mathbf{b}$	$\mathbf{g},\mathbf{m}$	$\{[\mathbf{a_i},\mathbf{g_i}],\mathbf{d_i}\}$	**Acc./MSE**	**F1/MAE**	**T.**	**P.**
CalCOFI [812,174x2]	CCS, [IT]	$-,10^{-1},-,1300$	–	–	$15.157,[408]$	$3.1,[19.67]$	–	–
	DNP	$-,10^{-1},10^{-2},1300$	$50,1$	–	17.679	3.45	6.71	$2\cdot10^{-4}$
	SPINDLE	$2,10^{-1},10^{-2},1300$	$50,1$	–	17.938	3.62	65.31	0.23
PIMA [768x8]	CCS, [IT]	$-,10^{-2},-,50$	–	–	$0.784,[0.720]$	$0.680,[0.604]$	–	–
	DNP	$-,10^{-2},10^{-2},50$	$1,30$	–	0.781	0.679	0.038	$9\cdot10^{-5}$
	SPINDLE	$2,10^{-2},10^{-2},50$	$1,30$	$[\pm7],3$	0.780	0.677	11.28	0.18
BCW [699x9]	CCS, [IT]	$-,10^{-1},-,20$	–	–	$0.962,[0.922]$	$0.947,[0.877]$	–	–
	DNP	$-,10^{-1},10^{-1},20$	$1,3$	–	0.962	0.942	0.034	$5\cdot10^{-5}$
	SPINDLE	$2,10^{-1},10^{-1},20$	$1,3$	$[\pm1]$, 3	0.962	0.944	3.25	0.16
ESR [11,500x90]	CCS, [IT]	$-,6^{-3},-,10$	–	–	$0.842,[0.838]$	$0.462,[0.396]$	–	–
	DNP	$-,6^{-3},10^{-1},10$	$92,1$	–	0.840	0.460	2.89	$8\cdot10^{-5}$
	SPINDLE	$2,6^{-3},10^{-1},10$	$92,1$	$[\pm15],5$	0.839	0.456	53.27	0.35
MNIST **[70,000 x 784]** (multi.)	CCS, [IT]	$-,10^{-4},-,1024$	–	–	$0.873,[0.873]$	$0.871,[0.832]$	–	–
	DNP	$-,10^{-4},10^{-1},1024$	$3,6$	–	0.865	0.863	43.95	0.49
	SPINDLE	$1{,}10^{-4},10^{-1},1024$	$3,6$	SM:$[-30,4],15$ $[1,40],19$ M:$[\pm15],15$ $[1,40],15$	0.8617	0.86	558	4.33
MNIST [70,000 x 784] (1 vs. a)	CCS, [IT]	$-,10^{-4},-,1024$	–	–	$0.856,[0.827]$	$0.859,[0.822]$	–	–
	DNP	$-,10^{-4},10^{-1},1024$	$3,6$	–	0.853	0.858	43.98	0.49
	SPINDLE	$1{,}10^{-4},10^{-1},1024$	$3,6$	$[\pm15],15$	0.852	0.850	187.8	4.33

(b) Baseline Comparison with K-fold=5. Time to train (T.) and predict (P.) is in seconds.

TABLE IV: SPINDLE's Evaluation.

A. Theoretical Analysis

In Table IVa, we provide the complete complexity analysis of SPINDLE. $|ct|$ represents the maximum size of a ciphertext, i.e., the size of a fresh ciphertext at level L, E and D stand for encryption and decryption workloads. The DA packing approach incurs a higher computation complexity (with notably more plaintext-ciphertext multiplications and rescaling (M), and rotations (R)) but, as shown in Section VIII-B, it is embarrassingly parallel, i.e., operations can be amortized by a factor $N_1\cdot N_2$ (defined in Section V-B) depending on the available threads.

B. Evaluation Datasets

For linear regression, we use the CalCOFI dataset (with $n{=}812{,}174$ records and $c{=}2$ features) [20]. It contains oceanographic data (e.g., salinity) that can be used to predict the water temperature. For logistic regression, we use three different datasets: (a) the Breast Cancer Wisconsin dataset (BCW, $n = 699$, $c = 9$) [10] contains patients' data that is employed to predict the presence of a breast cancer, (b) the PIMA dataset ($n = 768$, $c = 8$) [102] contains medical observations collected from an Indian community that can be used to predict the presence of diabetes, and (c) the Epileptic Seizure Recognition dataset (ESR, $n = 11{,}500$, $c = 179$) [37] contains patients' data that can be used to predict a seizure. For multinomial regression, we test SPINDLE on the MNIST dataset ($n = 70{,}000$, $c = 784$) [70], where the goal is to identify single-digits out of grey-scale images. We rely on these datasets to compare SPINDLE with various baselines.

C. Comparison with Prior Art

In Table V, we perform a qualitative and quantitative comparison of SPINDLE with existing works. We consider a generic privacy-preserving centralized encrypted solution (CES), two distributed solutions, Drynx [41] and Prio [28], which respectively rely on additive HE and secret sharing, and Helen [132], a solution that employs a different distributed approach, the Alternating Direction Method of Multipliers (ADMM) proposed by Boyd et al. [19], to train regularized linear models. CES represents a centralized solution, similar to existing works [66], [63], [21], in which one DP outsources its encrypted data to a server that trains and evaluates a model. For a fair comparison, we estimate the execution time (without communication) of a generic centralized (outsourced) solution (CES) relying on the non-multiparty CKKS scheme with security parameters that enable packing of the same number of values in one ciphertext. We use as a reference one of the most recent works on bootstrapping by Han and Dohyeong [52].

	Confident.	Non-lin. Mod.	Scal. w. $\lvert S\rvert, c, n_i$	Acc. w.r.t central.	$\lvert S\rvert, c, n_i$	**Lin.**	**Log.**	**Multi.**
CES	data+model	✔	- lin. lin.	$\approx$	$25.6k$, 2^{12}, -	$12k$	$14k$	-
Drynx Prio [41,28]	partially for data	✔	lin. quadra. indep.	$<$	$25.6k$, 2^{12}, 5	inf	inf	-
Helen [112]	data	✗	quadra. quadra. indep.	$\approx$	$400k, 100, 4$ $400k, 10, 10$ $4M, 90, 4$	$9k$ $1.7k$ $6.5k$	-	-
SPINDLE	data+model	✔	indep. log. lin.	$\approx$	$25.6k, 2^{12}, 5$ $400k, 100, 4$ $400k, 10, 10$ $4M, 90, 4$	480 528 513 $5.8k$	658 – – –	$33.6k$ – – –

TABLE V: Comparison with existing solutions. $|S|$, c are the number of DPs and features, n_i the size of each DP_i local dataset. Timings for lin., log., multi. regressions training are in seconds.

In Helen, the DPs perform the ADMM optimization locally under a quantum-vulnerable additive HE cryptoscheme, and combine their results under secret-sharing. ADMM is less widespread than SGD, it is primarily designed for linear models and does not provide the same stability and convergence guarantees than the cooperative gradient-descent [130], [121], [122], [18], for which convergence can be derived from SGD. Since Helen's implementation is not available, we aim at providing an intuition on how it quantitatively compares with SPINDLE. To this end, we used results reported in Helen [132] and performed similar experiments in SPINDLE ((2),(3),(4) in Table V). We highlight here that the experiment environment is different, and these results provide only an idea of how these systems compare. For a fair comparison, we excluded the proof generation time in Helen and we notice that as SPINDLE, Helen reported similar accuracy results as a non-secure centralized solution. We observe that SPINDLE scales better than Helen when increasing $|S|$, as its execution time is almost the same in ((2) and (3)), and it also scales better with the number of features (almost 10x better in (4)).

APPENDIX E
NOTATIONS

Symbol	Description
DP_i	i^{th} Data provider
S, $\|S\|$	Set of DPs and its cardinality
$\boldsymbol{X}_{n\times c}$,$\boldsymbol{y}_n$	Training dataset with c features, n samples, and label vector
($\boldsymbol{X}^{(i)}$,$\boldsymbol{y}^{(i)}$)	DP_i's part of the dataset
($\boldsymbol{X'}$,$\cdot$) & $\boldsymbol{y'}$	Querier's evaluation data and prediction's output
cl, $\|cl\|$	Set of class labels and its cardinality
$\boldsymbol{X}[\phi,\cdot]$,$\boldsymbol{X}[\cdot,\phi]$,$\boldsymbol{y}[\phi]$	ϕ^{th} line and column of $\boldsymbol{X}$, element of vector y
$\boldsymbol{B}\in\boldsymbol{X}$	Random mini-batch of b rows
$\boldsymbol{W}_G^{(\cdot,j)}$	Global model at iteration j
$\boldsymbol{W}^{(i,j,l)}$	DP_i's local model at global iter. j and local iter. l
$\boldsymbol{W}^{(i,j)}$	DP_i's local model at global iter. j
$\boldsymbol{w}_G$,$\boldsymbol{w}$ $\boldsymbol{w}^{(i,0)}$	Vector of global and local weights Initial local weights of DP_i
lp, g,m	Learning params., nbr. of global, local iterations
QR	Querier request
$\sigma(\cdot)$, d	Activation function, approx. degree
a_m	Multiplicative depth of $\sigma(\cdot)$
$I(\cdot)$	Indicator function
n_i	The number of data samples per DP_i
$P^2(x)$	Next power of 2 of x
N_1,N_2 α, ρ	Diagonal approach parameters. Learning and elastic rates
sk,pk, evk	Secret, public, evaluation keys
$\langle\boldsymbol{v}\rangle$	Encrypted vector $\boldsymbol{v}$
$\|ct\|$	Size of fresh ciphertext ct
N, Q	Ring dimension, Fresh ciphertext modulus
L	Number of available levels
τ_b	Minimum level for Dbootstrap($\cdot$)
η	Std. deviation of the noise distribution
Δ, mc	Plaintext Scale, Chain of moduli variables
DM	Dot product

TABLE VI: Frequently Used Symbols and Notations.