Privacy-Preserving Federated Recurrent Neural Networks

Sinem Sav
EPFL
Lausanne, Switzerland
sinem.sav@epfl.ch

Abdulrahman Diaa
University of Waterloo
Waterloo, Canada
a2diaa@uwaterloo.ca

Apostolos Pyrgelis
EPFL
Lausanne, Switzerland
apostolos.pyrgelis@epfl.ch

Jean-Philippe Bossuat
Tune Insight SA
Lausanne, Switzerland
jean-philippe@tuneinsight.com

Jean-Pierre Hubaux
EPFL and Tune Insight SA
Lausanne, Switzerland
jean-pierre.hubaux@epfl.ch

ABSTRACT

We present RHODE, a novel system that enables privacy-preserving training of and prediction on Recurrent Neural Networks (RNNs) in a cross-silo federated learning setting by relying on multiparty homomorphic encryption. RHODE preserves the confidentiality of the training data, the model, and the prediction data; and it mitigates federated learning attacks that target the gradients under a passive-adversary threat model. We propose a packing scheme, *multi-dimensional packing*, for a better utilization of Single Instruction, Multiple Data (SIMD) operations under encryption. With multi-dimensional packing, RHODE enables the efficient processing, in parallel, of a batch of samples. To avoid the exploding gradients problem, RHODE provides several clipping approximations for performing gradient clipping under encryption. We experimentally show that the model performance with RHODE remains similar to non-secure solutions both for homogeneous and heterogeneous data distributions among the data holders. Our experimental evaluation shows that RHODE scales linearly with the number of data holders and the number of timesteps, sub-linearly and sub-quadratically with the number of features and the number of hidden units of RNNs, respectively. To the best of our knowledge, RHODE is the first system that provides the building blocks for the training of RNNs and its variants, under encryption in a federated learning setting.

KEYWORDS

privacy-preserving machine learning, recurrent neural networks, federated learning, multiparty homomorphic encryption

1 INTRODUCTION

Collecting and mining sequential data, e.g., time-series, is the basis for numerous applications in the domains of healthcare, energy, and finance. For instance, in personalized healthcare, electrocardiograms (ECGs) are used to monitor patients and to detect heartbeat arrhythmia; and, in finance, historical economic records are mined to forecast stock prices. The state-of-the-art time-series mining is achieved by employing machine learning (ML) algorithms that extract useful patterns from the data. A well-known class of algorithms widely used for learning on time-series data is recurrent neural networks (RNNs) [42, 128] and its variants, e.g., Gated Recurrent Units (GRUs) and Long-short Term Memory (LSTMs), that are capable of modelling high-dimensional and non-linear relationships in dynamic systems [130].

Training effective RNNs for time-series tasks requires large amounts of data that are usually generated by multiple sources and scattered across several data holders. Sharing or centralizing this data is difficult because it is privacy sensitive [158]. For example, smart-meter data might leak householders' identities [20] and their activities [108], location data reveal information about peoples' lifestyles and beliefs [115, 136, 137, 157], and healthcare time-series are by nature private information [11, 94]. Additionally, privacy regulations enforce strict rules for sharing this type of data [51, 63]. As a result, privacy-preserving data sharing and collaborative ML solutions have gained a significant importance [18, 66, 94].

Federated learning (FL) is a machine learning paradigm that enables privacy-preserving collaborative training of ML models. With FL, data holders retain their data in their own premises and share model updates with an aggregation server that coordinates the learning over their joint data [82, 83, 98]. FL has been applied in many time-series applications by using RNNs to forecast the energy load [43], to predict the next word typed on mobile phones [26, 58], and to classify cancer in healthcare systems [49]. Although FL provides collaborative learning without data sharing, it still raises privacy concerns: The shared intermediate model is vulnerable to privacy attacks that can reconstruct parties' input data or infer the membership of data samples in the training set, hence needs to be protected during the training process [38, 40, 52, 64, 73, 102, 113, 147, 148, 162, 166]. Moreover, it was recently shown that RNNs are particularly vulnerable to inference attacks (e.g., membership inference) compared to traditional neural networks [154].

To overcome these shortcomings, several works propose integrating protection mechanisms such as differential privacy (DP), secure multiparty computation (SMC), or homomorphic encryption (HE) in the FL process. Each of these techniques, however, introduces a different trade-off: DP solutions decrease the utility of the ML model — in particular for RNNS [154] — while their level of practical privacy protection remains unclear [71]; SMC solutions lack scalability in terms of circuit complexity and number of parties due to high communication overheads [56, 79]; and HE solutions do not scale with the model complexity due to high computational overhead. Recently, a multifaceted approach, termed multiparty homomorphic

encryption (MHE) [9, 46], where the private key is secret-shared among the parties, has been used by several works to better balance the aforementioned trade-offs with privacy-preserving federated learning [48, 129]. However, these solutions focus solely on the training of generalized linear models or feed-forward neural networks and do not tackle the training of RNNs, which introduces its own additional challenges: (i) the sequential computations make the training procedure slow and hard to parallelize, and (ii) the long-term dependencies lead to exploding/vanishing gradients [13, 119].

By building upon MHE-based works [48, 129], we design Rhode, a novel system that enables the training of and predictions on RNNs and its variants in a cross-silo federated learning setting with N data holders *by performing all the computations under homomorphic encryption.* Rhode protects the confidentiality of the training data, the local gradients, and the global model under a passive adversary model (a common assumption for cross-silo settings) that allows collusions of up to $N - 1$ data holders. To achieve this, both the local gradients and the FL model remain encrypted throughout Rhode's pipeline. Hence, Rhode mitigates passive FL attacks that target the gradients or the global model by concealing all operations throughout the learning. To accelerate RNN training, Rhode enables mini-batch training by relying on a packing scheme that reduces the processing time of a batch. To address the problem of exploding gradients, Rhode uses efficient polynomial approximations of the gradient clipping function. After training, Rhode enables predictions-as-a-service (PaaS) to a querier that obliviously evaluates the trained model on its private inputs and obtains the prediction results. We implement Rhode and our experimental evaluation shows that its accuracy is on par with non-private centralized and decentralized solutions. Moreover, it scales sub-linearly with the number of features, sub-quadratically with the hidden RNN dimension, and linearly with the timesteps, the batch size and the number of data holders. For example, Rhode can process 256K samples distributed among 10 data holders with an RNN of 32 hidden units and 4 timesteps, over 100 global training iterations, in ~1.5 hours. In summary, our contributions are the following:

- We present Rhode, a novel system for the training of RNNs in a cross-silo FL setting that conceals all intermediate values and the model that are communicated in the FL process by relying on MHE. After training, Rhode enables oblivious predictions to a querier that provides its encrypted inputs.
- We design a multi-dimensional packing scheme that enables efficient encrypted matrix operations suitable for mini-batch training. We show that matrix multiplication with our scheme improves the state of the art [72, 129] in terms of throughput proportionally to the dimension of the matrices. Moreover, with our packing scheme, the number of cryptographic keys required is independent of the matrix size.
- We propose and evaluate polynomial approximations for performing gradient clipping under encryption to alleviate the problem of exploding gradients that is inherent to RNNs.
- We experimentally evaluate Rhode and show that it scales sub-linearly with the number of features, sub-quadratically with the hidden RNN dimension, linearly with the timesteps or the batch size, and sub-linearly (or linearly) with the number of data holders depending on the number of samples in

the dataset. Moreover, its homomorphic operations and building blocks have negligible effect on the model performance for common time-series forecasting benchmarks, with both homogeneous and heterogeneous (imbalanced) data distributions among the parties.

To the best of our knowledge, Rhode is the first system that enables the training of and prediction on RNNs in a cross-silo FL setting under encryption. Rhode mitigates passive FL attacks during training and preserves the privacy of the data holders' data, the intermediate model, the querier's evaluation data, and the final model.

2 RELATED WORK

We review the related work on privacy-preserving time-series, as well as on privacy-preserving federated training of, and prediction on machine learning (ML) models.

2.1 Privacy-Preserving Time-Series

Previous works aiming to protect the privacy of time-series data employed secure aggregation techniques to enable the computation of simple statistics and analytics [84, 101, 123], whereas others combined secure aggregation with differential privacy to bound the leakage stemming from the computations [19, 132]. More recently, Liu et al. [93] employed secure multi-party computation (SMC) techniques for privacy-preserving collaborative medical time-series analysis. Dauterman et al. [37] use function secret sharing (FSS) to support various functionalities, e.g., multi-predicate filtering. All these authors focus on simple collaborative time-series tasks and not on the privacy-preserving training of ML models on time-series data, which is the main focus of this work.

2.2 Privacy-Preserving Federated Training of Machine Learning Models

Federated learning (FL) enables different data holders to perform the collective training of a ML model while maintaining the data in their local premises and communicating only the local model updates to an aggregation server [82, 83, 98]. FL was applied on many time-series applications such as load forecasting [43], natural language processing [26, 58], traffic flow prediction, [95], and healthcare systems [49]. However, recent research has shown that sharing gradients or local model updates in FL leads to severe privacy leakage because membership inference and data/label reconstruction attacks are effective [38, 40, 52, 64, 73, 102, 113, 147, 148, 162, 166].

One common approach for privacy-preserving *decentralized* or *federated* ML is to use secure aggregation protocols that hide the data holders' model updates while only revealing their combination to the server. Various such protocols have been proposed based on HE [91, 121, 122], functional encryption [152, 153], or SMC [12, 16]. However, the aggregated global model remains vulnerable to privacy attacks such as model inversion [47] or membership inference [135]. Other works integrate differential privacy (DP) into the learning phase by injecting noise on the intermediate model updates [1, 88, 99, 134, 149, 151] or by combining it with secure aggregation techniques [71, 138, 144] such that the aggregated global model meets the DP requirements. McMahan et al. recently applied DP to RNNs to protect user-level privacy [100], and Wen et al. applied local DP to a FL framework for energy-theft detection [150].

However, DP introduces a privacy vs. utility trade-off that is hard to parameterize. Yang et al. show that perturbing the gradients during training or the final model weights to achieve DP guarantees results in higher utility loss for RNNs than other neural networks [154].

Another line of research applies SMC to protect the privacy of the *full* decentralized ML pipeline [2, 5, 15, 21, 25, 27, 30, 50, 53, 81, 106, 107, 114, 131, 140, 145, 146]. However, SMC solutions require data holders to secret-share their data among a limited number of computing parties and their confidentiality is ensured under non-collusion assumptions (typically, honest majority [2, 21, 25, 27, 107, 145]) among them. Furthermore, they introduce high communication overhead and scale poorly in terms of the circuit complexity or the number of parties [56, 79]. Overall, existing SMC solutions enable the training of regression models and feed-forward neural networks, whereas RNNs have not been studied. One recent work, which relies on additive secret-sharing, investigated the training of recurrent neural networks as a case study [160], but without evaluating runtime performance or scalability.

Some works employ homomorphic encryption (HE) to encrypt the clients' data, before outsourcing it to a server that performs the model training [159]. Zheng et al. use SMC in combination with HE for maliciously secure FL [164, 165]. None of these works, however, address the learning of sequential patterns over the data. To balance the trade-off between privacy, utility, and performance, several works apply multiparty homomorphic encryption (MHE) [9] to realize decentralized trust and to enable efficient collaborative homomorphic operations, such as collective bootstrapping [111]. These works apply MHE to FL to conceal any information exchanged during the learning process [48, 129]. However, their focus is on generalized linear models [48] or feed-forward neural networks [129], and not the training of RNNs. For instance, the packing scheme and the matrix/vector operations employed in [129] are tailored for the processing of a single sample (thus eliminating explicit transpose and matrix multiplications; see Section 7.6 for comparisons demonstrating the unsuitability of this approach for training RNNs). Whereas in this work, we propose a novel packing scheme suitable for mini-batch RNN training and enable matrix transpose and matrix multiplications. Moreover, we propose several polynomial approximations to enable critical operations for RNNs, such as gradient clipping, under MHE.

A different line of research is focused on the private aggregation of teacher ensembles (PATE) [117, 118], and its variants [96, 156]; different from our work, their main goal is knowledge transfer by combining DP and generative adversarial networks (GANs). Choquette-Choo et al. recently proposed CaPC Learning that enables confidential and private collaborative learning by relying on SMC, HE, and DP, in order to increase the utility of each data holder's local ML model [32]. However, CaPC Learning is evaluated only on feed-forward neural networks and the number of parties needs to be large enough to obtain better utility/privacy guarantees. Our work decouples privacy from the amount of data or the number of parties.

Overall, RHODE enables the training of RNNs in a cross-silo FL while protecting the confidentiality of the training data, the local gradients, and the global model in a passive threat model with up to $N-1$ collusions among the data holders. By design, RHODE mitigates passive FL inference attacks during training, as all intermediate

values that are communicated remain encrypted and as it enables all the computations of the FL pipeline *under encryption*.

2.3 Privacy-Preserving Prediction on Machine Learning Models

Privacy-preserving prediction on ML models plays an important role for Prediction-as-a-Service (Paas) solutions whose aim is to protect the model from the client (who queries the model) and the client's evaluation data from the service provider. A series of works employ SMC or HE (or a combination of them) to enable privacy of both the data and the model holder [14, 54, 62, 67, 75, 92, 104, 120, 125, 127]. These works, however, develop and improve cryptographic primitives that support the evaluation of feed-forward neural networks. Only a few recent studies focus on privacy-preserving prediction on RNNs by relying on several cryptographic primitives such as HE [10, 70], two-party computation (2PC) [124], secure multiparty computation [45], and hybrid approaches combining HE and garbled circuits [44]. Similar to these works, RHODE protects the confidentiality of the model and the client's evaluation data during the prediction phase. However, our work is broader, as it also enables federated *training* under encryption.

3 BACKGROUND

We present preliminaries on multiparty homomorphic encryption (MHE), federated learning, and recurrent neural network (RNNs).

3.1 Multiparty Homomorphic Encryption

In this work, we rely on multiparty homomorphic encryption (MHE) to enable distributed cryptographic operations. We employ the multiparty variant of Cheon-Kim-Kim-Song (CKKS), a leveled homomorphic-encryption scheme based on the *ring learning with errors* (RLWE) problem [29], that is proposed by Mouchet et al. [111]. This scheme is secure against post-quantum attacks, enables floating-point arithmetic, and tolerates collusions of up to $N-1$ parties, under a passive threat model. Below, we summarize the most important operations of this scheme:

Key Generation: Each data holder i locally generates a *secret* key (sk_i). The *public* key (pk) and evaluation keys ($[ek]$) are collectively generated from the data holders' secret keys (CKeyGen($[sk_i]$)). $[ek]$ is a set of special public keys that are required for the execution of homomorphic multiplications and rotations. With this scheme, any data holder can encrypt and locally perform homomorphic operations by using pk and $[ek]$, but decrypting a ciphertext requires the collaboration among all data holders (CDecrypt($\mathbf{c}_{pk}, [sk_i]$)). We denote any ciphertext encrypted under pk with bold-face $\mathbf{c}_{pk}$ throughout this paper. When there is no ambiguity, we omit the sub-index and use $\mathbf{c}$.

Arithmetic Operations and Parallelization: The CKKS homomorphic encryption scheme enables approximate arithmetic over a vector of complex numbers ($\mathbb{C}^{N/2}$) for a polynomial ring of dimension N. Encrypted operations over $\mathbb{C}^{N/2}$ are carried out in a Single Instruction Multiple Data (SIMD) manner, which enables parallelization over the $N/2$ plaintext vector slots.

Arithmetic Circuit Depth and Collective Bootstrapping: A fresh ciphertext has an initial level of $\mathcal{L}$ and at most $\mathcal{L}$ multiplications (i.e., an $\mathcal{L}$-depth circuit) can be evaluated on it. When the levels are exhausted, the ciphertext is refreshed with a collective bootstrapping operation ($CBootstrap(\mathbf{c}_{pk}, [sk_i])$) that enables further operations. We note here that $CBootstrap(\cdot)$ is a costly operation with approximately 2 orders of magnitude higher overhead than a homomorphic addition/multiplication.

Slot Rotations: The slots of a ciphertext can be re-arranged via rotations to the left/right. $RotL/R(\mathbf{c}_{pk}, k)$ homomorphically rotates $\mathbf{c}_{pk}$ to the left/right by k positions using $[ek]$.

Collective Key-Switch: This operation changes the encryption key of a ciphertext $\mathbf{c}$ from pk to pk' ($CKeySwitch(\mathbf{c}_{pk}, pk', [sk_i])$). As such, only the holder of the secret key corresponding to pk' can decrypt the result $\mathbf{c}_{pk'}$.

Note that the aforementioned operations are MHE (i.e., interactive) ones if they are termed as 'collective' (or if the operation name starts with C, e.g., $CKeyGen(\cdot)$), otherwise they are pure CKKS functionalities. For further details about the CKKS scheme, e.g., relinearization and rescaling, we refer the reader to the original work of Cheon et al. [29], and for more information about the collective MHE operations to the work of Mouchet et al. [111].

3.2 Federated Learning

Federated Learning (FL) is an emerging collaborative ML approach that enables multiple data holders to train a model without sharing their local data [82, 83, 98]. With FL, each data holder performs several training iterations on its local data, and the resulting local models are aggregated into a global model via an aggregation server. We summarize the most popular federated learning algorithm, i.e., FedAvg, proposed by McMahan et al. [98] in Algorithm 1. W_i^k denotes the model weights of the ith data holder during the kth iteration. If no sub-index is used, then it simply denotes the global model weights at a certain iteration. The server initializes the model weights, sends them to a random subset of data holders S that execute the local gradient descent algorithm ($Local\ Gradient\ Descent(\cdot)$; see Section 4.2 for the details of the algorithm when it is tailored for RNN execution under encryption with RHODE) to update their local model. The model updates from each data holder are aggregated to the global model via weighted averaging (n_i denotes the number of local samples at the ith data holder and n the total number of samples). The number of global and local iterations are denoted as e and l, respectively. Note that FedAvg is a generalization of the standard FedSGD algorithm with several local iterations performed on each data holder's side.

3.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a special type of models that enable learning on sequential data (e.g., time-series) [42, 128]. A typical RNN is composed of an input layer that obtains the sequential input to the network, a hidden layer, and an output layer. Contrary to feed-forward neural networks where the information flows from the input to the output layer, in an RNN, the information is fed back to the network through connections between its nodes. The RNN input is a sequence where each item in the sequence is called a timestep. The hidden layer consists of several hidden units

---

Algorithm 1 Federated Averaging Algorithm (FedAvg)

1: Initialize the global model W^0	▷ Server executes
2: **for** $k = 0 \rightarrow e - 1$ **do**	▷ Global iterations
3: Choose S random subset of N data holders	
4: Send W^k to each data holder in S	
5: **for** $\ell = 0 \rightarrow l - 1$ **do**	▷ Local iterations
6: $W_i^{k+1} \leftarrow Local\ Gradient\ Descent(\cdot)$	▷ at each $S_i \in S$
7: **end for**	
8: $W^{k+1} \leftarrow \sum_{i=1}^{N} \frac{n_i}{n} W_i^{k+1}$	▷ Server executes
9: **end for**	

(neurons). Each hidden unit has a hidden state that retains the information from the previous timestep. Thus, RNNs capture sequential patterns in the data by processing each timestep sequentially and by updating the hidden state. In this way, RNNs use earlier information in the sequence to learn the current output and retain a form of *memory*. A typical RNN, e.g., an Elman network [42], takes a sequential input ($X = x_1, x_2, \ldots, x_T$) of T timesteps and outputs the prediction y_t through the hidden units and an activation function $\varphi(\cdot)$ by sequentially computing the hidden state at time t as:

$$h_t = \varphi(h_{t-1} \times W + x_t \times U + b_h)$$
$$y_t = h_t \times V + b_y \tag{1}$$

where U, W, and V denote the input-to-hidden, hidden-to-hidden, and hidden-to-output weight matrices, respectively, and b_h and b_y the hidden and output biases. Note that the input at each timestep x_t can be itself a d-dimensional *feature* vector (or a matrix $b \times d$ for a batch of size b). But to avoid notation overflow, we assume that each timestep is a scalar value throughout the paper, unless otherwise stated. The dimensions of U, W, and V are $d \times h$, $h \times h$ and $h \times o$, where d, h, and o are the input, hidden, and the output dimensions, respectively. Jordan networks [74] are similar to Elman ones but with a slightly different hidden unit, i.e., $h_t = \varphi(y_{t-1} \times W + x_t \times U + b_h)$. Consequently, the size of the weight matrix W in Jordan networks is $o \times h$.

The training of RNNs is an iterative process with a series of forward and backward passes (FP and BP, respectively). FP comprises the prediction by using the formulas in Eq. 1. However, BP is computed via the back-propagation through time (BPTT) algorithm that takes into account the error propagated through the hidden units and computes the gradients by following the chain rule [128]. We describe in detail the operations of this process under RHODE in Section 4.2. Finally, we note that there are various RNN structures, e.g., many-to-one or many-to-many, depending on the input and output layer size (e.g., a many-to-many structure yields outputs for κ timesteps). Configuring RHODE to support these is trivial and requires simple alterations in its implementation. However, RNN variants in terms of architecture, such as Gated Recurrent Unit (GRU) or Long Short-Term Memory (LSTM), require modifications to the hidden units. Throughout this paper, we consider the traditional RNN architecture that is called simple, vanilla, or an Elman RNN [42], yet our solution can be extended to more complex RNN architectures (see Section 6).

4 RHODE DESIGN

We introduce the problem and the overview of our solution. We also present the challenges associated with RNN training.

4.1 Problem Statement

We first discuss the system and threat model of RHODE, as well as its training and prediction objectives.

System Model. We consider a cross-silo FL setting with a moderate number of data holders (N) (typically in the range of 2 to 200 [77]). The data holders are interconnected in a tree-structured topology for efficient communication (although our system can be adapted to any network topology, e.g., the standard FL topology, see Appendix A). Each data holder i has its time-series input data X^i and the corresponding output Y^i ($Y^i \in \{0, 1, \ldots, n\}$ for n-class classification or $Y^i \in \mathbb{R}$ for regression tasks) for training a joint ML model (horizontal FL). After training, the data holders enable PaaS for a querier q with evaluation data X^q. The querier can be one of the data holders or an external entity and obtains Y^q, the prediction result. We assume that all data holders are available during execution; dropouts are not supported by RHODE due to the underlying cryptographic operations that require all data holders to participate (see Appendix A for an extension to tolerate dropouts).

Threat Model. We assume a passive-adversary threat model (which is a common threat model for cross-silo settings) that allows collusions between up to $N - 1$ parties, including the data holders and the querier. As such, the data holders follow the protocol, but up to $N - 1$ of them can share their observations or their inputs to extract information about the honest data holder's inputs. RHODE aims at preserving the confidentiality of both the input data and the model during the training and prediction pipelines. We note that our focus is on mitigating passive FL inference attacks that target the model during training. Active adversaries during training or inference attacks targeting the outputs during prediction, e.g., model stealing [143], are out of the scope of this work. We discuss complementary mechanisms that can be used orthogonally to RHODE to mitigate such attacks in Appendix A.

Training. The data holders aim at collaboratively training an RNN model *without* relying on a trusted party and *without* revealing their data to any other party. RHODE's objective is to enable this and to mitigate passive FL attacks that target the model or local gradients. As we optimize the communication via a tree-structured topology, the root plays the role of the aggregation server in traditional FL, and for brevity, throughout the paper, we refer to this node as the *aggregation server*.

Prediction. RHODE's objective is to protect the confidentiality of both the querier's data and the trained RNN model, i.e., the data holders should not learn anything about the querier's evaluation data and the querier should not learn anything about the model other than what it can learn from the prediction output.

4.2 Solution Overview

To fulfill the training and prediction objectives under the given threat model, RHODE relies on FL and MHE. To protect data confidentiality and to mitigate the passive FL inference attacks during training, RHODE retains the model and all the intermediate values communicated through the network in encrypted form. This implies that all the local RNN operations performed by the data holders need to be carried out under encryption. To efficiently enable this, RHODE uses a packing strategy and several cryptographic optimizations that are detailed in Section 5. To refresh the noise accumulated by the homomorphic operations, RHODE performs bootstrapping (CBootstrap($\cdot$)) when required (depending on the RNN and security parameters). For clarity, we refer to Algorithm 1 for our high-level protocol. RHODE enables, under encryption, both FedAvg and FedSGD with every data holder included in each iteration (i.e., $|S| = N$) since we consider a cross-silo setting and given that all data holders are available to participate in cryptographic operations. By enabling RNN training, RHODE can handle sequential data yet, throughout the paper, we use time-series as an example.

4.2.1 Private Training. The training protocol of RHODE operates in four phases: **Setup**, **Local Computation**, **Aggregation**, and **Model-Update**. We detail these phases hereunder:

Setup Phase: In this phase, the necessary cryptographic and learning parameters are decided by the data holders, i.e., the minimum security guarantees, the RNN architecture, and the learning parameters, e.g., the learning rate (η), the batch size (b), and the number of global (e) and local iterations (l). Then, the data holders generate their secret keys (sk_i, for the ith data holder) and collectively generate the corresponding public key (pk) with the MHE scheme (Section 3.1). For the training execution to start, the aggregation server initializes the global model (Line 1, Algorithm 1), encrypts it with pk, and broadcasts it to all data holders.

Local Computation Phase: Each data holder receives the encrypted global model weights and begins the Local Gradient Descent($\cdot$) (Line 6, Algorithm 1), i.e., the execution of the forward pass (FP) and backward pass (BP) on its local data as detailed in Algorithm 2 for a simple many-to-many RNN with $\kappa = T$ outputs. Note that for different RNN architectures such as GRU or LSTMs, the local gradient descent algorithm should be altered (see Section 6). We denote the encryption of any value, vector, or matrix with bold-face, e.g., **W**, and for clarity, we omit the indices of the global iteration and the data holder in Algorithm 1. x_t is the input d-dimensional feature-vector for a timestep t (or a matrix of size $b \times d$ for a batch of size b, in which case the outer-vector products in the backward pass become matrix multiplications) and y_t the corresponding output(s). $\varphi(\cdot)$ ($\varphi'(\cdot)$) indicates any type of activation function (and its derivative), e.g., Tanh. We denote the transpose of a matrix A as A^T, the element-wise multiplication with $\odot$, the matrix or vector multiplication with $\times$, and the outer product with $\otimes$. Note that the characteristics of RNNs incur several challenges when training them under encryption. We discuss these challenges in Section 4.3 and propose several techniques to alleviate them in Section 5.

Aggregate Phase: Data holders collectively aggregate their locally computed models (or gradients). That is, each data holder sends its (encrypted) local model to its parent, and each parent homomorphically sums the received child-models with its own. As such, the aggregation server receives the sum of all the local models.

Algorithm 2 Local Computation Phase (RNN)

Inputs: $\mathbf{U}, \mathbf{W}, \mathbf{V}, \mathbf{h}_{\text{prev}}, X = (x_1, x_2, \ldots, x_t), Y = (y_1, y_2, \ldots, y_t)$
Outputs: $\nabla\mathbf{U}, \nabla\mathbf{W}, \nabla\mathbf{V}$

1: $\mathbf{h}_0 \leftarrow \mathbf{h}_{\text{prev}}$
2: **for** $t \leftarrow 1 : T$ **do**   ▷ Forward Pass (FP)
3:    $\mathbf{z}_t \leftarrow \mathbf{h}_{t-1} \times \mathbf{W} + x_t \times \mathbf{U}$
4:    $\mathbf{h}_t \leftarrow \varphi(\mathbf{z}_t)$
5:    $\mathbf{p}_t \leftarrow \mathbf{h}_t \times \mathbf{V}$
6: **end for**
7: $\mathbf{h}_{\text{prev}} \leftarrow \mathbf{h}_t$
8: $\mathbf{dh}_{\text{next}} = \mathbf{0}$
9: $\nabla\mathbf{U}, \nabla\mathbf{W}, \nabla\mathbf{V} = \mathbf{0}$
10: **for** $t \leftarrow T : 1$ **do**   ▷ Backward Pass (BP)
11:    $\mathbf{dy} \leftarrow \mathbf{p}_t - y_t$
12:    $\mathbf{dh} \leftarrow \mathbf{dy} \times \mathbf{V}^T + \mathbf{dh}_{\text{next}}$
13:    $\mathbf{dz} \leftarrow \mathbf{dh} \odot \varphi'(\mathbf{z}_t)$
14:    $\mathbf{dh}_{\text{next}} \leftarrow \mathbf{dz} \times \mathbf{W}^T$
15:    $\nabla\mathbf{V} \leftarrow \nabla\mathbf{V} + \mathbf{h}_t \otimes \mathbf{dy}$
16:    $\nabla\mathbf{U} \leftarrow \nabla\mathbf{U} + x_t \otimes \mathbf{dz}$
17:    $\nabla\mathbf{W} \leftarrow \nabla\mathbf{W} + \mathbf{h}_{t-1} \otimes \mathbf{dz}$
18: **end for**

Model-Update Phase: The aggregation server updates the global model by averaging the local models of the data holders (Line 8, Algorithm 1). At this phase, the aggregation server also applies gradient clipping (see Section 5.2 for details). Since the batch size is public information (agreed upon in the Setup Phase), the division for averaging over the encrypted local models becomes a multiplication with a plaintext value. The global model is then broadcast to all data holders for the next Local Computation Phase.

Overall, the model training composes the Local Computation, Aggregate, and the Model-Update Phases, which are repeated for a predefined number of global iterations (e).

4.2.2 Predictions-as-a-Service (PaaS). After the collective model training, the data holders (i) use the encrypted model for oblivious PaaS *without* decryption, or (ii) if required by the application, collectively decrypt the model and reveal it to all the data holders or an external party for semi-oblivious PaaS. In both cases, the querier's evaluation data is encrypted with the data holders' public key and we rely on the collective key-switch functionality of the underlying MHE scheme (Section 3.1). In particular, for (i), the querier encrypts X^q with the data holders' collective key pk, the data holders then compute the prediction on the encrypted data using the encrypted model, and switch the encryption key of the result with $\text{CKeySwitch}(\mathbf{X}^q_{pk}, qk, [sk_i])$, where qk is the querier's key. Thus, only the querier can decrypt the prediction result. For (ii), the collective decryption is simply a special case of the collective key-switch with the encrypted model as input and no target public key. The rest of the evaluation is same as oblivious PaaS.

4.3 Challenges Associated with RNN Training

We discuss the challenges, as well as several solutions, associated with the training of RNNs in general. We also highlight how these challenges (and their solutions) further increase the difficulty of RNN training under encryption. Section 5 presents the building blocks that RHODE employs to alleviate these challenges.

RNNs are inherently challenging to train because of their sequential operations, i.e., the computations performed over the timesteps (the dependency of each network node on the output of its previous one). This phenomenon causes two issues when training RNNs: (i) the training procedure is slow, complex, and hard to parallelize, and (ii) the gradients tend to explode (or vanish) due to long-term dependencies between timesteps in the execution of the back-propagation through time (BBPT) algorithm [13, 119].

To mitigate the first challenge, several works suggest accelerating RNN training through data parallelization, i.e., via a distributed approach and/or mini-batch training [68, 80, 133]. For example, a master-slave approach where slave machines first compute the gradients and then a master machine aggregates them to perform the global update (Map-Reduce algorithm [33]) is commonly proposed. RHODE ensures similar efficiency to such distributed training-based approaches, through federated learning that is, by definition, distributed and resembles the master-slave relationship proposed for distributed RNNs. On the other hand, mini-batch training under homomorphic encryption requires a well-suited packing scheme for efficient matrix multiplication and transpose operations. Therefore, in Section 5.1, we propose a novel multi-dimensional packing scheme that reduces the processing time of a large batch.

The second challenge of exploding or vanishing gradients is addressed by several techniques such as the use of gating mechanisms, e.g., LSTMs [65], along with a possible modification of the gradient propagation [78], encoder-decoder approaches [31], gradient clipping [119], non-saturating functions [22], and various recurrent-weight initialization techniques via identity or orthogonal matrices [61, 86]. Employing LSTMs as a solution to the problem of exploding/vanishing gradients brings its own challenges, due to the computational complexity of training LSTMs under homomorphic encryption, and we discuss the costs of such an extension in Section 6. Although various weight-initialization techniques are straightforward for RHODE to adopt, we opt for gradient clipping – a widely used mitigation technique for exploding gradients when training simple RNNs. However, gradient clipping is not easy to compute under encryption as it requires a comparison function that is not homomorphically enabled. Therefore, we present various approximations for the efficient computation of an element-wise clipping function, through different polynomials (Section 5.2).

5 CRYPTOGRAPHIC BUILDING BLOCKS

We detail the main cryptographic building blocks that are used in RHODE. We describe the packing scheme and the optimized matrix operations including the matrix-multiplication and matrix-transpose in Section 5.1 and the approximated building blocks in Section 5.2. We analyze RHODE's security in Appendix B.

5.1 Packing and Optimized Matrix Operations

We present the packing scheme and the optimized matrix operations used for RNN training in RHODE. Due to the use of homomorphic encryption (HE), packing has a significant effect on the training performance as through SIMD operations, it enables mini-batch training which alleviates the inherent high training times of RNNs. In addition, matrix operations are the most heavy and frequent operations in RNNs (see Lines 2-5 and 12-17, Algorithm 2). Thus,

(a) Matrix Multiplication Method of Jiang et al. [72] for $n = 3$.

(b) An overview of our Multi-dimensional Packing scheme, its Multiplication and Transpose operations.
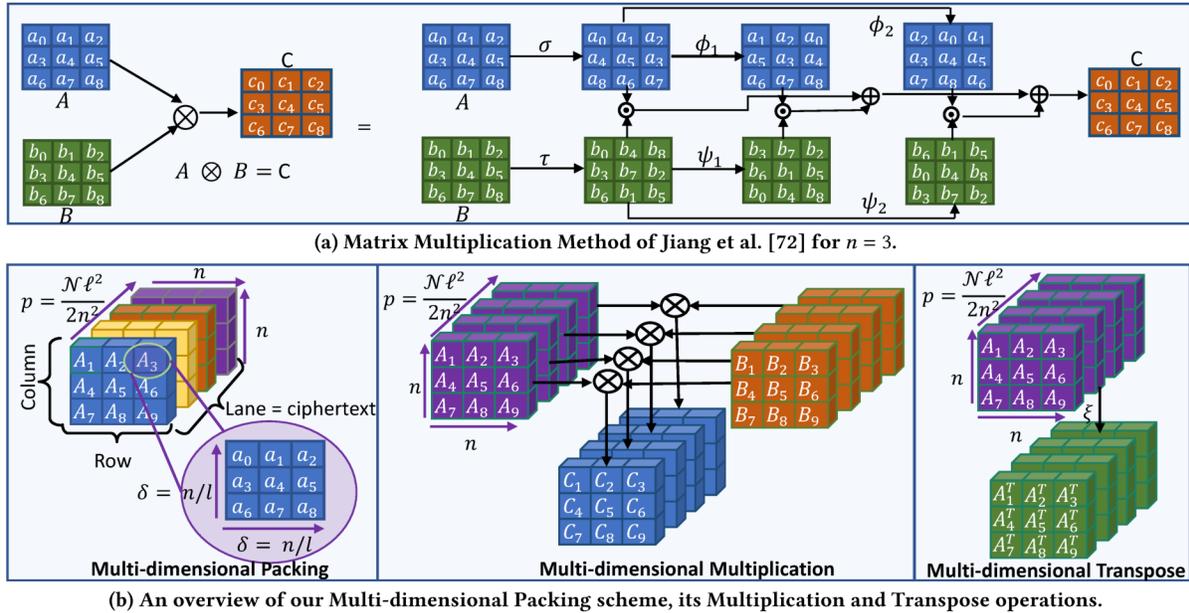
Figure 1: Overview of the (a) Matrix Multiplication Protocol by Jiang et al. [72] and (b) our Multi-dimensional Packing Scheme.

optimizations of the packing scheme and matrix operations such as multiplication or transpose, are crucial for RHODE's efficient training execution. In this work, we rely on a row-based packing where matrices are decomposed row-wise and packed in one plaintext and we use Jiang et al.'s [72] approach as a baseline for the matrix operations. On top of these matrix operations and row-based packing, we propose and implement a novel multi-dimensional packing scheme that relies on block matrices[1] and the matrix operations that are described below, for further optimizations.

Matrix Multiplication. The typical methods for matrix multiplication in HE [57, 105] do not preserve the structure of the matrix in the plaintext slots, i.e., the shape of the computation output is different than that of the inputs. This prevents efficient chaining of matrix operations since after each multiplication an expensive linear transformation is required to convert the output back to an encoding compatible with further operations.

Jiang et al. propose a method for matrix multiplication that preserves the format of row-encoded matrices, enabling the efficient chaining of an arbitrary number of matrix operations [72]. With their approach, a matrix multiplication consumes 3 *levels* per multiplication (2 if one matrix is in plaintext) due to 2 linear transformations and 1 dot product. It enables the packing of multiple matrices in a single ciphertext (assuming the number of matrix-entries is smaller than $\mathcal{N}/2$) and to operate on them in a SIMD manner, which increases efficiency and compensates for the *level* consumption. This feature yields a favorable balance between computation runtime and level consumption for RNN computations. We now describe this format-preserving method for matrix multiplication that is illustrated in Figure 1a. Given A and B, matrices of size $n \times n$, the multiplication $C = A \otimes B$ can be evaluated under encryption as the scalar product $\mathbf{C} = \langle \tilde{\mathbf{A}}, \tilde{\mathbf{B}} \rangle$, where $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ are size-n

vectors of permutations of $\mathbf{A}$ and $\mathbf{B}$, respectively (or permutations of the original matrices $\mathbf{A}$ and $\mathbf{B}$). These permutations are represented by the linear transformations σ (cyclic rotation of all rows, where the i-th row is rotated by i positions), τ (cyclic rotation of all columns, where the i-th column is rotated by i positions), ϕ_i (cyclic rotation of all rows by i positions) and ψ_i (cyclic rotation of all columns by i positions). For example, if A and B are two 4×4 matrices, then $\tilde{\mathbf{A}} = \{\phi_0(\sigma(\mathbf{A})), \phi_1(\sigma(\mathbf{A})), \phi_2(\sigma(\mathbf{A})), \phi_3(\sigma(\mathbf{A}))\}$ and $\tilde{\mathbf{B}} = \{\psi_0(\tau(\mathbf{B})), \psi_1(\tau(\mathbf{B})), \psi_2(\tau(\mathbf{B})), \psi_3(\tau(\mathbf{B}))\}$, and $\mathbf{C} = \langle \tilde{\mathbf{A}}, \tilde{\mathbf{B}} \rangle$. The linear transformations σ, τ, ϕ_i and ψ_i are computationally *cheap* as they can be efficiently parallelized and evaluated with $O(n)$, $O(n)$, $O(1)$, $O(1)$ homomorphic rotations, respectively. Hence, the matrix multiplication method of Jiang et al. has a complexity of $O(n)$ rotations for two $n \times n$ matrices. This is smaller than previous approaches, such as the naive approach with $O(n^3)$ or the one proposed by Halevi and Shoup [57] with $O(n^2)$.

Matrix Transpose. Another operation required for RNN training is matrix transpose, as described in Algorithm 2 (Lines 12 and 14). The transpose of a row-packed matrix is straightforward and can be realized with a single linear transform (ξ) of $2n$ non-zero diagonals [72], hence, it can be evaluated with $2n$ rotations. This cost can be reduced to $3\sqrt{n}$ by using the baby-step giant-step algorithm proposed in [57] and improved by Bossuat et al. [17]. Thus, the cost of a transpose operation for a $n \times n$ matrix is $O(\sqrt{n})$.

Multi-dimensional Packing. We propose a novel packing scheme called multi-dimensional packing to optimize the usage of ciphertext slots, which we illustrate in Figure 1b. Given an $n \times n$ matrix and ℓ a divisor of n, we split this matrix into ℓ^2 smaller matrices of size $\delta \times \delta$, with $\delta = \frac{n}{\ell}$. Each of these smaller matrices is stored in a different ciphertext, and we pack in parallel up to $p = \frac{\mathcal{N}}{2} \cdot \frac{\ell^2}{n^2}$ block matrices of size $n \times n$ in a set of ℓ^2 ciphertexts. This packing scheme brings several advantages over the plain row-packing approach by Jiang et al: (i) it enables a more efficient packing of

---

[1]Block matrix is a matrix defined by sub-matrices, called blocks, which enable matrix operations in parallel.

Algorithm 3 Multi-dimensional Matrix Multiplication

Inputs: A, B two $\ell \times \ell$ matrices of ciphertexts, each encrypting a sub-matrix of dimension $(n/\ell) \times (n/\ell)$, the linear transformations σ, τ, ϕ_i and ψ_i
Outputs: $C \leftarrow A \otimes B$

1: **for** $i = 1 \rightarrow \ell$ **do**
2:     **for** $j = 1 \rightarrow \ell$ **do**
3:         $z \leftarrow \sigma(B_{i,j})$ ▷ Depth 1
4:         $b_{i,j} \leftarrow \{\phi_0(z), \ldots, \phi_{n/\ell-1}(z)\}$ ▷ Depth 1
5:     **end for**
6: **end for**
7: **for** $i = 1 \rightarrow \ell$ **do**
8:     **for** $j = 1 \rightarrow \ell$ **do**
9:         $z \leftarrow \tau(A_{i,j})$ ▷ Depth 1
10:        $a \leftarrow \{\psi_0(z), \ldots, \psi_{n/\ell-1}(z)\}$ ▷ Depth 1
11:        **for** $k = 1 \rightarrow \ell$ **do**
12:            $C_{i,k} \leftarrow C_{i,k} + \langle a, b_{i,j} \rangle$ ▷ Depth 1
13:        **end for**
14:     **end for**
15: **end for**

---

non-square matrices with a worst case of extra space used reduced from $n(n-1)$ to $\frac{n}{\ell}(\frac{n}{\ell}-1)$, (ii) it provides a better amortized multiplication complexity as it packs up to $\frac{N}{2} \cdot \frac{\ell^2}{n^2}$ matrices (instead of $\frac{N}{2n^2}$) in parallel with $O(n\ell)$ complexity for the multiplication, but amortizes to $O(n/\ell)$ per matrix (instead of $O(n)$), (iii) it reduces the complexity of the transpose operation from $O(\sqrt{n})$ to $O(\sqrt{n/\ell})$ (only the ξ permutation needs to be evaluated on the sub-matrices and permutations between ciphertexts are free), and (iv) it enables more efficient matrix slot manipulations as rows of sub-matrices can be individually moved and/or rotated. We further minimize the number of redundant homomorphic operations by leveraging the techniques used in the *double-hoisting* baby-step giant-step algorithm [17] to evaluate linear transforms and delay ciphertext relinearization. We describe the protocol for multi-dimensional matrix multiplication in Algorithm 3. We present guidelines for configuring δ and other cryptographic parameters in Appendix C.

RHODE employs the multi-dimensional packing for efficient mini-batch training under encryption. In particular, we distribute the RNN input batch along the "third" dimension (see Figure 1b), by expressing the batch as p parallel matrices (as if computing an ensemble of p parallel RNNs). We can increase the batch size (up to $p * n/\ell$) at no additional cost as it fits into the same ciphertext. To enable further operations, i.e., to make the input batch compatible with the weight matrices, we replicate the latter across the third dimension, so that the weight matrices are aligned with their parallel slices from the input batch. The computation of the FP and BP for all the timesteps results in a parallel slicing of the gradients across the third dimension. These gradients are then aggregated (with $\log(p)$ rotations to perform an inner sum operation), to ensure consistency among the parallel instances. Note that for batches that fit into one ciphertext, our multi-dimensional packing is equivalent to an optimized version (via block-matrices) of Jiang et al.'s [72] approach. However, for bigger dimensions, where the matrix is split over more than one ciphertext, multi-dimensional packing yields better amortized matrix multiplication cost (see Section 7.6).

5.2 Approximated RNN Building Blocks

To compute any non-linear activation or clipping function under CKKS encryption, RHODE relies on polynomial approximations employing the least-squares or Minimax methods. The least-squares method finds the optimal polynomial that minimizes the squared error between the real function and its approximation over an interval, whereas Minimax minimizes the corresponding maximum error. Admittedly, the maximum error with Minimax decreases as the approximation degree ($\mathfrak{p}$) increases or as the range of the interval shrinks. However, note that with larger $\mathfrak{p}$, the polynomial evaluation becomes more expensive (with a scale of $O(\log(\mathfrak{p}))$) and smaller interval ranges are not always possible due to the need of accommodating the range of the recurrent neural network outputs. We remark that choosing $\mathfrak{p}$ and the interval of approximations is a non-trivial task under privacy constraints. Yet, these can be determined by synthetic datasets or based on data distribution-based heuristics such as computing the minimum, the maximum, and the mean of feature vector means per dataset [62].

Activation Functions. A neural network is a pipeline of layers composed of neurons on which linear and non-linear transformations (activations) are applied. Typical activation functions are Sigmoid ($\varphi(x) = \frac{1}{1+e^{-x}}$), ReLU ($\varphi(x) = max(0, x)$), Tanh ($\varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$), etc., that are not computable under encryption as they comprise non-linear operations (e.g., comparison, exponentiation). To overcome this, previous works either change the function to a linear or square one [10, 54], or rely on approximation techniques employing the least-squares method [48, 129], polynomial splines [92], Legendre [62] or Chebyshev polynomials [62, 85]. For RNNs, the choice of activation function is critical due to exploding gradients problem (i.e., the function should be bounded) and commonly used functions are Tanh and ReLU (that can approximated as SoftPlus, i.e., $\varphi(x) = ln(1 + e^x)$ [129]). Thus, we also rely on polynomial approximations to enable the execution of these activation functions.

Gradient Clipping. As described in Section 4.3, when the weights have a large norm, the gradients accumulated over multiple steps of the BBPT algorithm can grow exponentially; gradient clipping is one of the well-established techniques to mitigate this issue. Thus, in RHODE, the *aggregation server* applies clipping during the Model Update Phase (see Section 4.2). Pascanu et al. [119] propose clipping the gradients based on their infinity-norm, yet applying directly their function under encryption is challenging due to the norm calculation that comprises comparison operations. Thus, inspired from their norm-clipping, we propose a gradient clipping strategy that restricts the infinity-norm of the gradients to a closed interval whose limits are defined by a threshold $|m|$. We apply the following function element-wise to the accumulated gradient vector x:

$$\text{Clip}(x, m) = \begin{cases} -m & x \leq -m \\ x & -m < x < m \\ m & x \geq m \end{cases}$$

$\text{Clip}(x, m)$ is a baseline as this function is also difficult to practically approximate due to its non-smoothness near the clipping interval limits $[-m, m]$. Therefore, we introduce two *softer* approximations for clipping. The first one is based on Tanh, which is very similar to Clip except for the clipping interval limits $-m$ and m:

$$\text{TanhClip}(x, m) = m * tanh\left(\frac{x}{m}\right)$$

The second one is based on ReLU, i.e., $\text{Clip}(x, m) = x + \text{ReLU}(-(x+m)) - \text{ReLU}(x-m)$. Since Rhode softly approximates the ReLU function with SoftPlus, SoftClip is defined accordingly as $\text{SoftClip}(x, m) = x + \text{SoftPlus}(-(x+m)) - \text{SoftPlus}(x-m)$, or:

$$\text{SoftClip}(x, m) = x + \ln\frac{1 + e^{-(x+m)}}{1 + e^{(x-m)}}$$

Overall, Rhode enables the approximated execution of both TanhClip and SoftClip. We evaluate the performance of both clipping approximations and provide the plots of the approximated curves (by Minimax) and their errors for polynomial degrees of $\mathfrak{p}=5$ and $\mathfrak{p}=15$ in Appendix D. We observe that both functions achieve a similar average error over the approximation interval, yet SoftClip is slightly better than TanhClip for higher degrees in terms of error and behavior around the limits of the approximation interval.

6 EXTENSION TO MORE COMPLEX RNNS

So far, we have described Rhode's workflow with a simple RNN architecture, yet, Rhode can be extended to more complex ones such as GRUs and LSTMs. Rhode's main building blocks, i.e., the approximated activation/clipping functions, the multi-dimensional packing, the matrix multiplication and transpose operations are sufficient to implement such complex RNNs. As the multi-dimensional packing preserves the structure of the matrices in the plaintext slots after multiplication and transpose operations, any pipeline can be built from our blocks. RNNs such as GRUs or LSTMs, require changes only on the Local Computation Phase of Rhode. We briefly explain how to extend Rhode to GRUs and LSTMs.

GRU. We summarize the protocol for the GRU architecture in Algorithm 4 (Appendix E). Such an RNN has the same workflow as a simple RNN but with different operations inside the GRU unit. A typical GRU comprises an Update Gate (R-Gate) that decides how much information from previous timesteps is needed for future ones, a Reset Gate (Z-Gate) that decides how much information to forget, and a Candidate activation (N-Gate) similar to the hidden state of an RNN unit. Other GRU variants modify these gates by computing only (or excluding) the biases [39, 60]. Both Z- and R-Gates include an activation function that is typically a Sigmoid, $\varrho = \frac{1}{1+e^{-x}}$, and the hidden state comprises the Tanh activation function, $\varphi = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Then, the pipeline for the forward pass (FP) of a GRU is similar to a simple RNN and the backpropagation (BP) follows the BBPT with the chain rule for the calculation of the derivatives and the loss (see Algorithm 4). In more detail, the FP (Lines 2-10) comprises encrypted matrix multiplications, additions, and activation functions, thus, the packing scheme of Rhode, the matrix multiplication and the approximated building blocks can be directly applied. Similarly, the BP comprises the same operations as the FP but additionally requires multiple executions of the transpose, which is also supported by Rhode (Section 5.1).

LSTM. LSTM includes a Forget Gate (f), an Input Gate (i), a Cell State (c), and an Output Gate (o). The hidden state is calculated as:

$$\begin{aligned}
f_t &= \varrho(W_f \times x_t + U_f \times h_{t-1} + b_f) && \triangleright \text{ Forget Gate} \\
i_t &= \varrho(W_i \times x_t + U_i \times h_{t-1} + b_i) && \triangleright \text{ Input (Update) Gate} \\
o_t &= \varrho(W_o \times x_t + U_o \times h_{t-1} + b_o) && \triangleright \text{ Output Gate} \\
c_a &= \varphi(W_c \times x_t + U_c \times h_{t-1} + b_c) && \triangleright \text{ Cell Input Activation} \\
c_t &= f_t \odot c_{t-1} + i_t \odot c_a && \triangleright \text{ Cell State} \\
h_t &= o_t \odot c_{t-1} + \varphi(c_t) && \triangleright \text{ Hidden State}
\end{aligned}$$

where $U.$ and $W.$ denote the weight matrices, $b.$ the bias vector, with the respective gate names as sub-indices ($\cdot$). As in GRUs, the MHE execution of LSTM requires the matrix multiplication, element-wise multiplication, addition, and activation functions (approximated via our building blocks). We note that the element-wise multiplication with multi-dimensional packing is straight-forward and requires ℓ^2 ciphertext multiplications. Due to the nature of the operations, the BP for LSTM training is similar to GRU-BP and requires the exact same operations of the forward pass with the addition of the transpose operation which is supported by Rhode.

Overall, different RNN architectures composed of the aforementioned operations can be enabled using Rhode's building blocks. Yet, it is worth mentioning that implementing these models under MHE is non-trivial and will increase Rhode's computation and communication overhead; its experimental evaluation on such complex RNN architectures is a direction for future work. Here, we roughly estimate the overhead incurred by GRUs and LSTMs. For the forward pass, a GRU requires 3× the operations (i.e., 2 multiplication and additions followed by an activation) of Lines 3-4 in Algorithm 2 and element-wise multiplications which are negligible compared to matrix multiplications. Then, the prediction (Line 5, Algorithm 2) remains the same. Due to the chain rule, the same rationale holds for the BP. Similarly, LSTMs require 4× the operations of Lines 3-4 in Algorithm 2 followed by 3× more element-wise multiplications, but with one less matrix multiplication for the prediction step. Thus, we estimate that implementing GRUs and LSTMs *without parallelization* will result in a 3×–4× slower local computation phase than simple RNNs. Finally, as the number of GRU and LSTM weight matrices are 3× and 4× bigger than simple RNNs, similar estimates apply for the communication overhead incurred by collective bootstrapping and the FL workflow.

7 SYSTEM EVALUATION

We analyze Rhode's theoretical complexity in Section 7.1. Then, we present our experimental setup in Section 7.2. We evaluate Rhode's model performance, scalability with different parameters, and runtime in Sections 7.3, 7.4, and 7.5, respectively. Finally, we compare Rhode with prior work in Section 7.6.

7.1 Complexity Analysis

We summarize Rhode's complexity by taking into account the memory usage per data holder, as well as its communication and computational costs for an Elman network (Algorithm 2). Here, we discuss the dominating terms that affect the memory, communication, and computation costs and we provide a more detailed analysis with an example network configuration in Appendix F.

The memory usage per data holder depends on the number of input plaintexts, activation ciphertexts (e.g., $\mathbf{h}_t$ in Algorithm 2), weight ciphertexts (e.g., $\mathbf{U}, \mathbf{V}, \mathbf{W}$), and gradient ciphertexts (e.g., $\nabla \mathbf{U}, \nabla \mathbf{W}, \nabla \mathbf{V}$). Subsequently, the number of ciphertexts storing the matrix and vectors required for RHODE's execution has a direct impact on the communication cost. Recall that during RNN training, communication among the data holders is triggered by: (a) the collective bootstrapping (CBootstrap($\cdot$)) operation to refresh the ciphertexts, and (b) the federated learning (FL) workflow, where data holders communicate their local gradients (Aggregate Phase). Both (a) and (b) incur costs that depend on the number and size of the weight/gradient ciphertexts (see Appendix F for details).

RHODE's computational cost per data holder and per local iteration depends on the number of CBootstrap($\cdot$) operations (which are ~2 orders of magnitude slower than a homomorphic addition/multiplication), the degree $\mathfrak{p}$ of the polynomial approximations, and the cost of the linear transformations (i.e., σ, τ, ψ_i and ϕ_i) required for the matrix multiplication. Indeed, these linear transformations dominate the complexity of the matrix multiplication (see Algorithm 3); the rest of the algorithm comprises only multiplications and additions, which are negligible in comparison. Overall, RHODE's RNN training protocol requires several multiplications with the weight matrices ($\mathbf{W}, \mathbf{U}, \mathbf{V}$) and their transposes ($\mathbf{V}^T, \mathbf{W}^T$). As these matrices are not updated through the local iteration per timestep, we can pre-compute their linear transformations at the beginning of each iteration and re-use them for all timesteps. For the network structure of Algorithm 2, this reduces the cost to $O(6\delta)$ from $O(6T\delta)$ per local iteration, at a memory cost of $\times \mathbb{d}$ per matrix.

7.2 Experimental Setup

We present the datasets used for model performance evaluation. We describe the data distribution, RNN configuration, and security parameters. We provide the implementation details in Appendix G.

Datasets and Tasks. We employ the Hourly Energy Consumption (HEC) [76], Stock Prices (Stock) [36], and Inflation datasets [35], for time-series forecasting tasks. HEC contains the hourly energy consumption of several companies during a specific time period [76]. The forecasting task is to use early sequences of energy consumption and predict the future value for sequences of length T. The Stock dataset contains historical daily stock statistics for 10 companies [36]. The task on this dataset is to predict the average open-high-low-closing (OHLC) price of the next day, given the OHLC of the previous T days. The Inflation dataset contains quarterly inflation rates from 40 countries and the task is to predict the inflation rate of the next quarter given the previous T quarters. We also employ the Breast Cancer Wisconsin dataset (BCW) [126] that contains benign and malignant breast cancer samples for classification. More details about the datasets can be found in Appendix H. For each experimental setting, we split the dataset into training (80%) and test (20%) sets. For the RHODE's scalability and microbenchmark experiments, we use synthetic datasets.

Dataset Distribution. For the model performance experiments (Section 7.3), we split the data among N=10 data holders and evaluate two different settings: (i) Even (E) distribution, where we uniformly distribute the dataset to 10 data holders, and (ii) Imbalanced (I) distribution where we simulate a heterogeneous setting (see

Dataset	Model Performance (MAE $\vert$ R^2 or Acc)			
	L	**C**	**FL**	**RHODE**
HEC-E10	0.066 | 0.749	0.014 | 0.987	0.024 | 0.965	0.026 | 0.957
HEC-I10	0.081 | 0.625	0.014 | 0.987	0.025 | 0.962	0.027 | 0.955
HEC-E20	0.066 | 0.749	0.014 | 0.986	0.024 | 0.964	0.026 | 0.957
HEC-I20	0.081 | 0.626	0.014 | 0.986	0.025 | 0.962	0.027 | 0.954
Stock-E5	0.023 | 0.966	0.008 | 0.996	0.020 | 0.982	0.017 | 0.987
Stock-I5	0.024 | 0.976	0.008 | 0.996	0.012 | 0.992	0.012 | 0.992
Inflation-E8	0.067 | 0.778	0.067 | 0.775	0.067 | 0.774	0.067 | 0.771
Inflation-I8	0.103 | 0.579	0.066 | 0.777	0.066 | 0.777	0.067 | 0.770
BCW-E9	0.888	0.936	0.936	0.929
BCW-I9	0.880	0.936	0.900	0.914

Table 1: RHODE's model performance in various settings (N=10) compared to three baselines: local training (L), centralized (C), and federated learning (FL).

Appendix H). We annotate the results per setting as "DatasetName-TypeT" where TypeT denotes the type of the data distribution with the sequence length T, e.g., HEC-I10, denotes imbalanced data distribution and T=10 on the HEC dataset.

RNN Configuration. For the HEC and Stocks model performance experiments, we use an Elman network with a local batch size of b=256, a hidden dimension of h=32, a learning rate of η=0.1, and varying timesteps T=[5 − 20], and we train the RNN for e=1, 000 and e=300 global iterations, resp. For the Inflation dataset, we employ a Jordan network that is widely used for financial forecasting tasks [41, 59, 69, 109, 167, 168] with parameters b=128, h=16, η=0.1, T=8, and e=500. For BCW, we evaluate an Elman network used for breast cancer classification [34, 112, 139, 163] with b=32, h=64, η=0.2, T=9, and e=400. For all experiments, the data holders perform one local iteration and we use the same approximation parameters: SoftClip(x, m) with a clipping threshold of $\vert m \vert$=5, approximated with a $\mathfrak{p}$=7 polynomial over the interval $[-60, 60]$ as a result of preliminary evaluations on the datasets. To expedite the model performance experiments (Section 7.3), we simulate RHODE's fully-encrypted training pipeline in plaintext by using the approximated activation and clipping functions, and a fixed-precision.

Security Parameters. Unless otherwise stated, we set the degree of the cyclotomic polynomial to $\mathcal{N}=2^{14}$ and the modulus of the keys to $Q \approx 2^{438}$. These parameters yield 128-bit security according to the homomorphic encryption standard [6]. We set the plaintext scale to 2^{31}, and the initial ciphertext level is $\mathcal{L}$=9.

7.3 Model Performance

Table 1 shows RHODE's model performance results on various datasets and settings. For each setting, we report the mean absolute error (MAE) and R^2-scores for forecasting tasks and accuracy (Acc) for the classification one. The table displays the performance of three baselines: (a) **L** stands for local training, where each data holder trains its own local model with the original network (original activation functions and clipping), i.e., without taking advantage of the other parties' data, (b) **C** stands for centralized training, where

(a) **Increasing number of features (*d*).**

(b) **Increasing hidden dimension (*h*).**

(c) **Increasing number of timesteps (*T*).**

(d) **Increasing batch size (*b*).**

(e) **Increasing number of data holders (*N*).**
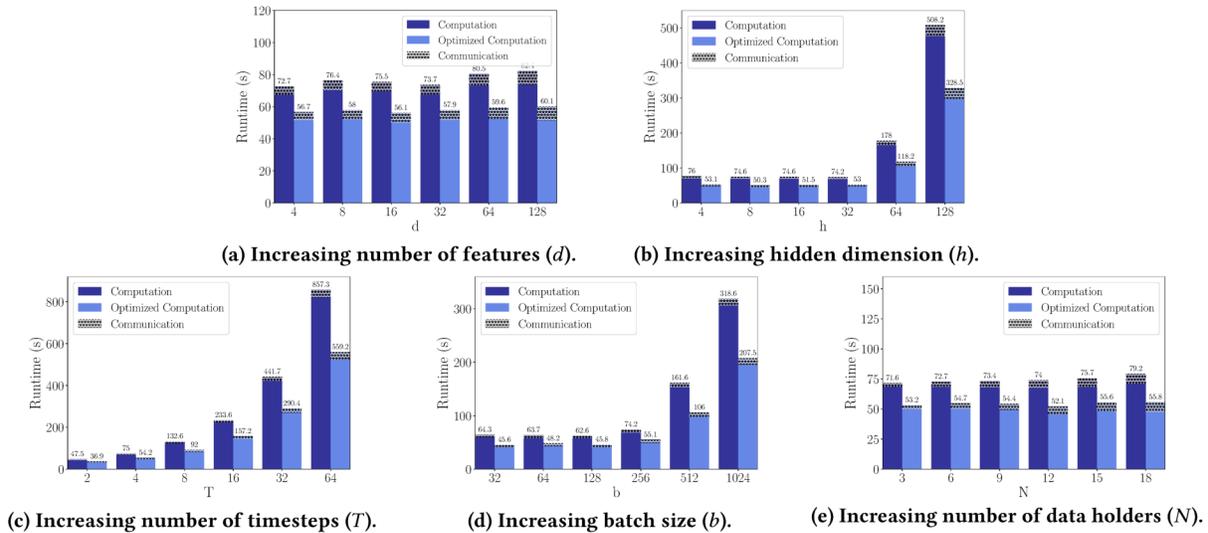
Figure 2: RHODE's training execution times for computation and communication of one global iteration with increasing (a) number of features, (b) hidden dimension, (c) timesteps, (d) batch size, and (e) number of data holders. The default parameters are *h*=*d*=*T*=4, *b*=256, *δ*=32, *N*=10 (we vary one of them in the corresponding experiment while keeping the others fixed).

the data is collected to a central server and trained with the original network, and (c) **FL** stands for original FL, where the data is distributed among 10 data holders and the learning is performed without any privacy mechanism. The baseline **L** shows the performance gain of collaborative training (note that we report the average MAE/R^2/Acc across the data holders). Whereas, the baselines **C** and **FL** enable us to evaluate RHODE's performance loss due to collective learning, the approximation of the activation/clipping functions, and the encryption. In all settings, we observe that RHODE achieves a model performance comparable to non-private baselines with at most 0.03 (0.01) difference in R^2-score compared to a centralized (federated learning) solution for the forecasting and at most 1% accuracy difference for the classification task. We also observe that there is always a performance gain, compared to **L** (except for the even distribution of the Inflation dataset where the local data already captures the global insights). The gain in terms of R^2-score for the Stock dataset is between 0.01-0.02, whereas it is between 0.21-0.33 for the more complex HEC dataset. More importantly, for an imbalanced distribution, i.e., HEC-*I*10 or HEC-*I*20, the gain from collaborative learning is more substantial (~0.33 increase in R^2-score). We observe a similar trend on the MAE, justifying the use of privacy-preserving collaborative learning. Similarly, RHODE yields a ~3% accuracy gain on BCW compared to **L**.

7.4 Scalability Analysis

We evaluate RHODE's scalability by varying the number of features (*d*), the hidden dimension (*h*), the timesteps (*T*), and the batch size (*b*), and by experimenting with the number of data holders (*N*). Our default parameters are *h*=*d*=*T*=4, *b*=256, *δ*=32, *N*=10, and we vary the parameter under analysis. The calculated runtime includes average per-data holder Local Computation, as well as the Aggregate, and Model-Update Phases. Figure 2 shows the time spent by RHODE (and its optimized version with pre-computed linear

transforms, described in Section 7.1) on computation and on communication during a global iteration. The communication indicates the total time spent for the collective bootstrapping operations (CBootstrap(·)) and the collective aggregation. Overall, we observe

Structure	Runtime (s)			
(*b*,*d*,*h*,*T*)	Training	Training (Opt)	Prediction	Prediction (Opt)
(256, 4, 32, 4)	72.65	56.68	27.61	18.10
(256, 32, 32, 4)	73.66	57.94	28.06	18.55
(256, 64, 32, 4)	80.48	59.63	31.04	20.53
(256, 4, 16, 4)	74.56	51.45	29.28	15.50
(256, 4, 32, 4)	74.19	53.04	71.92	16.31
(256, 4, 64, 4)	178.01	118.17	225.34	56.41

Table 2: RHODE's training and prediction runtime for several RNNs with *N*=10. Training runtimes are for one global iteration and for oblivious predictions on 256 samples, resp. Opt stands for optimized execution.

that the optimized computation is always more efficient in terms of runtime at a memory cost of 32× more ciphertexts per weight matrix (see Section 7.1). Moreover, Figures 2a, 2b and 2d demonstrate that RHODE's computation time remains constant, with increasing the respective parameters {*d*, *h*, *b*} when *d*, *h*, *b* < *δ*. This is due to the SIMD operations; as for these settings the matrix entries fit into one ciphertext, RHODE enables processing larger batches or larger networks at no additional cost. Figure 2a shows that RHODE scales sub-linearly with *d* and that the effect of *d* is almost negligible in runtime, because each data holder's data remains in clear through the training and the plaintext operations are negligible, compared to ciphertext ones. Increasing *d* has a direct effect on the matrix multiplication ($x_t \times$ **U**) of the first layer, as x_t is a plaintext whose size depends on *d*. As a result, the effect of increasing *d* is not observable;

the ciphertext multiplications ($\mathbf{h}_{t-1} \times \mathbf{W}$) dominate the runtime. The communication overhead also scales sub-linearly; as increasing d increases the number of ciphertexts to be bootstrapped. Figure 2b shows that RHODE scales sub-quadratically, with increasing h when δ=32; while increasing h quadratically increases the number of ciphertexts for the weight matrix $\mathbf{W}$, this affects the runtime of a few matrix multiplications only (see Section 7.1). This trend holds also for the communication due to an increased number of ciphertexts that are bootstrapped by the aggregation server. Figures 2c and 2d demonstrate that RHODE's runtime scales linearly with increasing T or b due to the sequential operations over the timesteps or the linear increase in the number of ciphertexts processed. Finally, Figure 2e shows that RHODE's local computation time remains constant with increasing N, as the data holders compute local operations at the same time, whereas the communication overhead increases linearly, as more data holders are involved in the interactive protocols.

7.5 Runtime Performance

Table 2 displays RHODE's training and prediction runtimes (for the original and optimized implementations, including communication) for various many-to-one RNN architectures (Elman) with N=10 data holders. The Training runtime is for one global iteration (including the communication for CBootstrap($\cdot$) and Aggregate Phase). The Prediction runtime is for oblivious predictions (both data and model are encrypted) on a batch of 256 samples (including the communication for CKeySwitch($\cdot$) that changes the key of the prediction result to the querier's public key). The results of this table can be used along with the scalability results (Section 7.4) to estimate RHODE's total training time for various RNNs. For instance, RHODE can process 256K samples split among 10 data holders with an RNN with b=256, d=4, h=32, T=4 (first table row), over 100 global iterations, in ~1.5 (~2) hours with its optimized (non-optimized, resp.) implementation. Moreover, as the Prediction runtime is calculated for 256 samples, RHODE yields a prediction throughput of ~14.43 (~9.31) samples per second with its optimized (non-optimized, resp.) implementation, for the same RNN structure.

Effect of RNN Structures. We evaluate RHODE's performance for different RNN structures including many-to-one and many-to-many. Table 4 depicts the results of microbenchmarks for the forward pass (FP) and backward pass (BP) *per timestep* without bootstrapping and *without optimization* for different RNN structures. FP-t_o and BP-t_o stand for the forward pass of the output-stage timestep and the backward pass of the output-stage timestep, respectively. The columns 'FP Total' and 'BP Total' show the total time required for the execution of the FP and BP, respectively. For example, the first row is a many-to-one RNN structure with T=6 and κ=1. This means that the FP is calculated for 6−1=5 timesteps and FP-t_o is calculated once. Similarly, for the second row with T=6 and κ=2, the FP is calculated for 6−2=4 timesteps, whereas FP-t_o is calculated for 2. There is no FP and BP for the last row as T=κ=6. Overall, Table 4 shows that the FP and BP calculations of the output-stage timestep (FP-t_o, BP-t_o) are more costly than the usual FP and BP timesteps (~1.7×). This is due to the extra multiplication required in the FP for calculating the prediction (Line 5, Algorithm 2), and the extra subtraction and multiplication during the BP (Lines 11-12, Algorithm 2) for the output error calculation. Recall that Algorithm 2

is presented for a many-to-many RNN structure with κ=T outputs. The computations of Lines 5, 11, and 12, are executed κ times for other structures with $\kappa \neq T$. Consequently, many-to-many RNN structures are more expensive, and the runtime increase depends on κ. We here note that a Jordan network requires the calculation of the output y_t at every timestep (see Section 3). As a result, its computation time is very similar to a many-to-many Elman RNN with T outputs and a modified matrix multiplication ($\mathbf{y}_{t-1} \times \mathbf{W}$). This results in roughly ~1.5× slower execution per timestep compared to an Elman network. For instance, for the parameters of the first row in Table 4, a Jordan network requires 7.15s and 10.02s for the *FP* and *BP* without bootstrapping (similar to FP-t_o and BP-t_o of the line, resp.). Finally, note that the Jordan network requires one extra CBootstrap($\cdot$) operation during the backward pass, resulting in slightly higher communication than Elman networks.

7.6 Comparison with Prior Work

We first compare RHODE with prior work based on matrix multiplication as it is the dominant and expensive operation. Then, we compare RHODE and the most similar system POSEIDON [129].

Matrix Multiplication Benchmarks. We compare our multi dimensional packing scheme with Jiang et al.'s [72] and Sav et al.'s [129] packing approaches in Table 3 (timings) and Table 7 (memory – Appendix I). We report the total and amortized time (throughput) for #M parallel multiplications of $n \times n$-sized matrices.

We observe that our multi-dimensional packing approach and Jiang et al.'s one perform identically when n=δ, since Jiang et al.'s approach is a special case of our packing with only one ciphertext. For $n > \delta$, the multi-dimensional packing requires more time to complete the multiplication but achieves a better amortized time than Jiang et al's. Due to memory limitations, we were not able to benchmark Jiang et al.'s approach for n=256 and n=512. The memory overhead arises from the ring dimension ($\mathcal{N}$), the number of ciphertexts, and the size of the evaluation keys. The first row of Table 7 shows that, with Jiang et al.'s method, $\mathcal{N}$ scales quadratically with n while ours scale similarly with δ (or with the closest power of two above n^2 or δ^2). Thus, for larger matrices (e.g., when multiplying two 128×128 matrices), Jiang et al.'s approach requires increasing $\mathcal{N}$ (as the matrix does not fit into one ciphertext), whereas our approach increases only the number of ciphertexts for a fixed δ (second row of Table 3). However, their approach also significantly impacts the size of the evaluation keys. Indeed, as in their approach $\mathcal{N}$ scales with $O(n^2)$, increasing n has the side effect of increasing the evaluation keys' size (third row of Table 3). Recall that Jiang et al.'s method also requires $O(n)$ evaluation keys to perform the multiplication, thus resulting in a $O(n^3)$ memory complexity. In contrast, our approach provides $O(\delta^3)$ that is $O(1)$ for a fixed δ. In conclusion, our packing is at least equivalent to Jiang et al.'s [72] and can provide a better throughput for larger matrices that do not fit into one ciphertext. It also uses a constant memory for the evaluation keys, regardless of n, thus enabling the multiplication of large matrices, without the need for larger cryptographic parameters or the re-generation of the evaluation keys.

Sav et al. [129] propose a system, POSEIDON, for training feed-forward neural networks under MHE using a packing scheme that optimizes the learning over one sample (instead of a mini-batch)

Dimension	Jiang et al. [72]			POSEIDON [129]			Multi-dimensional (δ=32)		
$n \times n$	Total | Amortized (s)	#M	log $\mathcal{N}$	Total | Amortized (s)	#M	log $\mathcal{N}$	Total | Amortized (s)	#M	log $\mathcal{N}$
32×32	0.43 | 0.05	8	14	6.14 | 6.14	1	14	0.43 | 0.05	8	14
64×64	0.82 | 0.41	2	14	14.02 | 14.02	1	14	1.70 | 0.21	8	14
128×128	2.86 | 2.86	1	15	69.25 | 69.25	1	15	7.16 | 0.89	8	14
256×256	NA	1	17	NA	1	17	30.6 | 3.82	8	14
512×512	NA	1	19	NA	1	17	79.17 | 9.89	8	14

Table 3: Comparison between our multi-dimensional packing with Jiang et al.'s [72] and Sav et al.'s (POSEIDON) [129] packing approaches. We report the total and amortized runtime per matrix for the multiplication of #M matrices of size $n \times n$. log $\mathcal{N}$ is the ring degree, and NA indicates that the memory was insufficient to carry out the evaluation.

Structure (b,d,h,T,o,κ)	Runtime (s)					
	FP	FP-t_o	BP	BP-t_o	FP Total	BP Total
(256, 64, 32, 6, 1, 1)	4.36	7.44	7.60	11.55	29.24	49.55
(256, 64, 32, 6, 1, 2)	4.42	7.39	7.62	11.47	32.46	53.42
(256, 64, 32, 6, 1, 4)	4.32	7.47	7.55	11.59	38.52	61.46
(256, 64, 32, 6, 1, 6)	-	7.38	-	11.42	44.28	68.52

Table 4: RHODE microbenchmarks for the forward pass (FP) and backward pass (BP) *per timestep without optimization*. t_o represents the output-stage timestep.

with vector-matrix operations. Thus, with their scheme, a matrix multiplication between two $n \times n$ matrices requires n vector-matrix multiplications using their one-cipher packing approach (yielding same total and amortized time, see Table 3). With their approach a vector-matrix multiplication requires $O(\log n)$ inner sum rotations, and repeating this n times to achieve a matrix multiplication yields a complexity of $O(n \log n)$ for a fixed $\mathcal{N}$. For example, for the multiplication of two 32×32 matrices, our multi-dimensional packing is ~14× and ~122× faster than POSEIDON's packing approach for total and amortized times, respectively. The memory complexity of their approach also depends on n for configuring $\mathcal{N}$, hence affecting the size of the evaluation keys (see Table 7, Appendix I), whereas our packing decouples the memory complexity from n.

Finally, we discuss another packing scheme similar to ours. Aharoni et al., improving on earlier work [3], propose a packing scheme that leverages on the CKKS complex domain to enable dense packing [4]. Their approach is based on data structures that pack tensors, e.g., matrices or hypercubes, in fixed sized chunks, for a natural use of the SIMD capabilities. We observe that our multi-dimensional packing differs in multiple ways. First, their packing density is less optimal in terms of slot-usage as a ciphertext multiplication requires inner sums to compute the product between tensors. Consequently, the result will be sparsely packed compared to our technique that fully-utilizes the ciphertext slots. Then, their approach is not format-preserving thus requires pre- or post-processing, whereas our approach supports chaining multiplications without additional processing. Thus, even though their approach consumes two levels less than Jiang et al. [72] (on which our packing is based) for the case of only one multiplication, the complexity of their scheme remains asymptotically equivalent. Finally, our approach is simpler to instantiate as its complexity depends only on δ, whereas Aharoni et al.'s approach requires the configuration of many parameters for applications that require sequential operations, e.g., the

training of RNNs. As their work does not provide an open source implementation, an experimental comparison is not possible.

Comparison with POSEIDON [129]. We first note that a direct comparison between RHODE and POSEIDON is non-trivial as the latter does not support RNN training and important mechanisms such as gradient clipping. Moreover, POSEIDON's packing scheme cannot handle RNN mini-batch training without introducing several changes, e.g., an explicit transpose operation required when the input batch is packed in one ciphertext, in its workflow. Yet, we estimate the runtime of POSEIDON based on the matrix multiplication microbenchmarks (discussed earlier) and a naive-transpose operation. To process a batch of b samples, POSEIDON's one-sample processing approach requires the repetition of the forward and backward pass b times, hence, it significantly increases computational complexity due to the large number of homomorphic rotations and vector-matrix multiplications that are required for the FP and BP of the various timesteps. Moreover, to support mini-batch training, POSEIDON requires packing each sample to a different ciphertext, which significantly increases the communication overhead due to the required bootstrapping operations. For example, for a batch size of b=256 and T=4 POSEIDON requires 1,024 CBootstrap($\cdot$) operations compared to 12 for RHODE. Overall, we estimate that training an RNN with d=32, h=32, b=256, and T=4 using POSEIDON would be ~25-32× slower compared to RHODE and its optimized version.

8 CONCLUSION

In this work, we presented RHODE, a novel system that enables privacy-preserving training of and predictions on RNNs in a cross-silo federated learning setting. Building on multi-party homomorphic encryption (MHE), RHODE preserves the confidentiality of the training data, the model, and the prediction data, under a passive adversary model with collusions of up to $N-1$ parties. By leveraging on a multi-dimensional packing scheme and polynomial approximations, RHODE enables efficient mini-batch training and addresses the problem of exploding/vanishing gradients that is inherent in RNNs. RHODE scales sub-linearly with the number of features and the number of data holders, linearly with the number of timesteps and the batch size, and its accuracy is on par with non-secure centralized and decentralized solutions. To the best of our knowledge, RHODE is the first system providing the building blocks for federated training of RNNs under encryption. As future work, we plan to evaluate RHODE on more complex RNN architectures.

ACKNOWLEDGMENTS

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *ACM CCS*.

[2] N. Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and A. Gascón. 2019. QUO-TIENT: Two-Party Secure Neural Network Training and Prediction. *ACM CCS* (2019).

[3] Ehud Aharoni, Allon Adir, Moran Baruch, Nir Drucker, Gilad Ezov, Ariel Farkash, Lev Greenberg, Ramy Masalha, Guy Moshkowich, Dov Murik, Hayim Shaul, and Omri Soceanu. 2020. HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data. *CoRR* abs/2011.01805 (2020). https://doi.org/10.48550/ARXIV.2011.01805

[4] Ehud Aharoni, Nir Drucker, Gilad Ezov, Hayim Shaul, and Omri Soceanu. 2022. Complex Encoded Tile Tensors: Accelerating Encrypted Analytics. *IEEE Security & Privacy* 20, 5 (2022), 35–43. https://doi.org/10.1109/MSEC.2022.3181689

[5] Adi Akavia, Hayim Shaul, Mor Weiss, and Zohar Yakhini. 2019. Linear-Regression on Packed Encrypted Data in the Two-Server Model. In *ACM WAHC*.

[6] Martin Albrecht et al. 2018. *Homomorphic Encryption Security Standard*. Technical Report. HomomorphicEncryption.org.

[7] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2018. *Homomorphic Encryption Security Standard*. Technical Report. HomomorphicEncryption.org, Toronto, Canada.

[8] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. 2018. Byzantine Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/a07c2f3b3b907aaf8436a26c6d77f0a2-Paper.pdf

[9] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. 2012. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *EUROCRYPT*. 483–501.

[10] Maya Bakshi and Mark Last. 2020. CryptoRNN - Privacy-Preserving Recurrent Neural Networks Using Homomorphic Encryption. In *CSCML*. 245–253.

[11] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. 2011. Privacy-Preserving ECG Classification With Branching Programs and Neural Networks. *IEEE TFIS* 6, 2 (2011), 452–468. https://doi.org/10.1109/TIFS.2011.2108650

[12] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1253–1269.

[13] Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5, 2 (1994), 157–166. https://doi.org/10.1109/72.279181

[14] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. 2019. nGraph-HE2: A High-Throughput Framework for Neural Network Inference on Encrypted Data. In *ACM WAHC*.

[15] Dan Bogdanov, Liina Kamm, Sven Laur, and Ville Sokk. 2018. Rmind: a tool for cryptographically secure statistical analysis. *IEEE TDSC* 15, 3 (2018), 481–495. https://doi.org/10.1109/TDSC.2016.2587623

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

[17] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In *EUROCRYPT*. 587–617.

[18] Amine Boulemtafes, Abdelouahid Derhab, and Yacine Challal. 2020. A review of privacy-preserving techniques for deep learning. *Neurocomputing* 384 (2020), 21–45. https://doi.org/10.1016/j.neucom.2019.11.041

[19] Joshua W. S. Brown, Olga Ohrimenko, and Roberto Tamassia. 2013. Haze: Privacy-Preserving Real-Time Traffic Statistics. In *ACM SIGSPATIAL*. 540–543. https://doi.org/10.1145/2525314.2525323

[20] Erik Buchmann, Klemens Böhm, Thorben Burghardt, and Stephan Kessler. 2013. Re-identification of Smart Meter data. *Personal and Ubiquitous Computing* 17, 4 (01 Apr 2013), 653–662. https://doi.org/10.1007/s00779-012-0513-6

[21] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. 2020. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *PETS* (2020).

[22] Sarath Chandar, Chinnadhurai Sankar, Eugene Vorontsov, Samira Ebrahimi Kahou, and Yoshua Bengio. 2019. Towards Non-Saturating Recurrent Units for Modelling Long-Term Dependencies. In *AAAI'19*. Article 403, 8 pages. https://doi.org/10.1609/aaai.v33i01.33013280

[23] Sylvain Chatel, Christian Mouchet, Ali Utkan Sahin, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. 2023. PELTA – Shielding Multiparty-FHE against Malicious Adversaries. Cryptology ePrint Archive, Paper 2023/642. https://eprint.iacr.org/2023/642 https://eprint.iacr.org/2023/642.

[24] Sylvain Chatel, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Privacy and Integrity Preserving Computations with CRISP.. In *USENIX Security Symposium*. 2111–2128.

[25] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. 2020. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In *NDSS*.

[26] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. 2019. Federated Learning Of Out-Of-Vocabulary Words. *CoRR* abs/1903.10635 (2019). arXiv:1903.10635 http://arxiv.org/abs/1903.10635

[27] T. Chen and S. Zhong. 2009. Privacy-Preserving Backpropagation Neural Network Learning. *IEEE Transactions on Neural Networks* 20, 10 (Oct 2009), 1554–1564.

[28] Weikeng Chen, Katerina Sotiraki, Ian Chang, Murat Kantarcioglu, and Raluca Ada Popa. 2021. HOLMES: a platform for detecting malicious inputs in secure collaborative computation. *Cryptology ePrint Archive* (2021).

[29] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*.

[30] Hyunghoon Cho, David Wu, and Bonnie Berger. 2018. Secure genome-wide association analysis using multiparty computation. *Nature Biotechnology* 36 (2018), 547–551. https://doi.org/10.1038/nbt.4108

[31] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8*. 103–111. https://doi.org/10.3115/v1/W14-4012

[32] Christopher A. Choquette-Choo, Natalie Dullerud, Adam Dziedzic, Yunxiang Zhang, Somesh Jha, Nicolas Papernot, and Xiao Wang. 2021. CaPC Learning: Confidential and Private Collaborative Learning. In *ICLR*. https://openreview.net/forum?id=h2EbJ4_wMVq

[33] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, Yuanyuan yu, Gary Bradski, Andrew Ng, and Kunle Olukotun. 2006. Map-Reduce for Machine Learning on Multicore.. In *NIPS*.

[34] Vaibahv Narayan Chunekar and Hemant P. Ambulgekar. 2009. Approach of Neural Network to Diagnose Breast Cancer on three different Data Set. In *2009 International Conference on Advances in Recent Technologies in Communication and Computing*. 893–895. https://doi.org/10.1109/ARTCom.2009.225

[35] Inflation Dataset. 2023. OECD data. https://data.oecd.org/price/inflation-cpi.htm#indicator-chart. (Accessed: 2023-03-03).

[36] Stock Dataset. 2022. Yahoo! Finance. https://finance.yahoo.com/lookup. (Accessed: 2022-01-06).

[37] E. Dauterman, M. Rathee, R. Popa, and I. Stoica. 2022. Waldo: A Private Time-Series Database from Function Secret Sharing. In *IEEE S&P*. 668–686. https://doi.org/10.1109/SP46214.2022.00039

[38] Jieren Deng, Yijue Wang, Ji Li, Chenghong Wang, Chao Shang, Hang Liu, Sanguthevar Rajasekaran, and Caiwen Ding. 2021. TAG: Gradient Attack on Transformer-based Language Models. In *EMNLP*. 3600–3610. https://doi.org/10.18653/v1/2021.findings-emnlp.305

[39] Rahul Dey and Fathi M. Salem. 2017. Gate-variants of Gated Recurrent Unit (GRU) neural networks. In *MWSCAS*. 1597–1600. https://doi.org/10.1109/MWSCAS.2017.8053243

[40] Dimitar I. Dimitrov, Mislav Balunović, Nikola Jovanović, and Martin Vechev. 2022. LAMP: Extracting Text from Gradients with Language Model Priors. *CoRR* abs/2202.08827 (2022). arXiv:2202.08827

[41] Valeria D'Amato, Susanna Levantesi, and Gabriella Piscopo. 2022. Deep learning in predicting cryptocurrency volatility. *Physica A: Statistical Mechanics and its Applications* 596 (2022), 127158. https://doi.org/10.1016/j.physa.2022.127158

[42] Jeffrey L. Elman. 1990. Finding Structure in Time. *Cognitive Science* 14, 2 (1990), 179–211. https://doi.org/10.1207/s15516709cog1402_1

[43] Mohammad Navid Fekri, Katarina Grolinger, and Syed Mir. 2022. Distributed load forecasting using smart meter data: Federated learning with Recurrent Neural Networks. *International Journal of Electrical Power & Energy Systems* 137 (2022), 107669. https://doi.org/10.1016/j.ijepes.2021.107669

[44] Bo Feng, Qian Lou, Lei Jiang, and Geoffrey C. Fox. 2021. CryptoGRU: Low Latency Privacy-Preserving Text Analysis With GRU. *CoRR* abs/2010.11796 (2021). arXiv:2010.11796 [cs.CR]

[45] Qi Feng, Debiao He, Zhe Liu, Huaqun Wang, and Kim-Kwang Raymond Choo. 2020. SecureNLP: A System for Multi-Party Privacy-Preserving Natural Language Processing. *IEEE TIFS* 15 (2020), 3709–3721. https://doi.org/10.1109/TIFS.2020.2997134

[46] Matthew Franklin and Stuart Haber. 1996. Joint encryption and message-efficient secure computation. *Journal of Cryptology* 9, 4 (01 Sep 1996), 217–232. https:

//doi.org/10.1007/BF00189261

[47] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM CCS*.

[48] David Froelicher, Juan R. Troncoso-Pastoriza, Apostolos Pyrgelis, Sinem Sav, Joao Sa Sousa, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. 2021. Scalable Privacy-Preserving Distributed Learning. *PETS* (2021).

[49] Neel Gandhi, Shakti Mishra, Santosh Kumar Bharti, and Kaushal Bhagat. 2021. Leveraging towards Privacy-preserving using Federated Machine Learning for Healthcare Systems. In *CONECCT*. 1–6. https://doi.org/10.1109/CONECCT52877.2021.9622730

[50] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. 2017. Privacy-preserving distributed linear regression on high-dimensional data. *PETS* (2017).

[51] GDPR. 2022. The EU General Data Protection Regulation. https://gdpr-info.eu/. (Accessed: 2022-01-06).

[52] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting Gradients - How Easy is It to Break Privacy in Federated Learning?. In *NeurIPS*. Article 1421, 11 pages.

[53] Irene Giacomelli, Somesh Jha, Marc Joye, C David Page, and Kyonghwan Yoon. 2018. Privacy-preserving ridge regression with only linearly-homomorphic encryption. In *Springer ACNS*.

[54] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*.

[55] Golang.org. 2022. Go Programming Language. Retrieved June 7, 2022 from https://golang.org

[56] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. 2022. Sharing Transformation and Dishonest Majority MPC with Packed Secret Sharing. Cryptology ePrint Archive, Paper 2022/831. https://eprint.iacr.org/2022/831 https://eprint.iacr.org/2022/831.

[57] Shai Halevi and Victor Shoup. 2018. Faster Homomorphic Linear Transformations in HElib. In *Advances in Cryptology – CRYPTO 2018*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer International Publishing, Cham, 93–120.

[58] Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated Learning for Mobile Keyboard Prediction. *CoRR* abs/1811.03604 (2018). arXiv:1811.03604 http://arxiv.org/abs/1811.03604

[59] Lingga Hardinata, Budi Warsito, and Suparti. 2018. Bankruptcy prediction based on financial ratios using Jordan Recurrent Neural Networks: a case study in Polish companies. *Journal of Physics: Conference Series* 1025 (2018).

[60] Joel C. Heck and Fathi M. Salem. 2017. Simplified minimal gated unit variations for recurrent neural networks. In *MWSCAS*. 1593–1596. https://doi.org/10.1109/MWSCAS.2017.8053242

[61] Mikael Henaff, Arthur Szlam, and Yann LeCun. 2016. Recurrent Orthogonal Networks and Long-Memory Tasks. In *ICML'16*. 2034–2042.

[62] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca Wright. 2018. Privacy-preserving Machine Learning as a Service. *PETS* (2018).

[63] HIPAA. 2022. Centers for Medicare & Medicaid Services. The Health Insurance Portability and Accountability Act of 1996 (HIPAA). https://www.cms.gov/Regulations-and-Guidance/Administrative-Simplification/HIPAA-ACA/PrivacyandSecurityInformation. (Accessed: 2022-01-06).

[64] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning. In *ACM CCS*.

[65] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-term Memory. *Neural computation* 9 (12 1997), 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735

[66] Sun-Kyong Hong, Kuldeep Gurjar, Hea-Suk Kim, and Yang-Sae Moon. 2013. A survey on privacy preserving time series data mining. In *3rd International Conference on Intelligent Computational Systems*.

[67] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. In *USENIX Security*.

[68] Zhiheng Huang, Geoffrey Zweig, Michael Levit, Benoit Dumoulin, Barlas Oguz, and Shawn Chang. 2013. Accelerating recurrent neural network training via two stage classes and parallelization. In *IEEE ASRU*. 326–331. https://doi.org/10.1109/ASRU.2013.6707751

[69] Gabriel Iuhasz, Monica Tirea, and Viorel Negru. 2012. Neural Network Predictions of Stock Price Fluctuations. *Proceedings - 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2012*, 505–512. https://doi.org/10.1109/SYNASC.2012.7

[70] Jaehee Jang, Younho Lee, Andrey Kim, Byunggook Na, Donggeon Yhee, Byounghan Lee, Jung Hee Cheon, and Sungroh Yoon. 2022. Privacy-Preserving Deep Sequential Model with Matrix Homomorphic Encryption. In *ACM ASIA CCS '22*. 377–391. https://doi.org/10.1145/3488932.3523253

[71] Bargav Jayaraman, Lingxiao Wang, David Evans, and Quanquan Gu. 2018. Distributed learning without distress: Privacy-preserving empirical risk minimization. In *NeurIPS*.

[72] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. 2018. Secure Outsourced Matrix Computation and Application to Neural Networks. In *ACM CCS '18*. 1209–1222. https://doi.org/10.1145/3243734.3243837

[73] Xiao Jin, Pin-Yu Chen, Chia-Yi Hsu, Chia-Mu Yu, and Tianyi Chen. 2021. CAFE: Catastrophic Data Leakage in Vertical Federated Learning. In *NeurIPS*, Vol. 34. 994–1006.

[74] Michael I. Jordan. 1997. Chapter 25 - Serial Order: A Parallel Distributed Processing Approach. In *Neural-Network Models of Cognition*, John W. Donahoe and Vivian Packard Dorsel (Eds.). Advances in Psychology, Vol. 121. North-Holland, 471–495. https://doi.org/10.1016/S0166-4115(97)80111-2

[75] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. Gazelle: A Low Latency Framework for Secure Neural Network Inference. *USENIX Security* (2018).

[76] Kaggle. 2022. Hourly Energy Consumption. https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption. (Accessed: 2022-01-06).

[77] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and Open Problems in Federated Learning. *Foundations and Trends in Machine Learning* 14, 1–2 (2021), 1–210. https://doi.org/10.1561/2200000083

[78] Bhargav Kanuparthi, Devansh Arpit, Giancarlo Kerg, Nan Ke, Ioannis Mitliagkas, and Y. Bengio. 2019. h-detach: Modifying the LSTM Gradient Towards Better Optimization. In *7th International Conference for Learning Representations (ICLR)*.

[79] Marcel Keller and Ke Sun. 2022. Secure quantized training for deep learning. In *International Conference on Machine Learning*. PMLR, 10912–10938.

[80] Viacheslav Khomenko, Oleg Shyshkov, Olga Radyvonenko, and Kostiantyn Bokhan. 2016. Accelerating recurrent neural network training using sequence bucketing and multi-GPU data parallelization. In *2016 IEEE First International Conference on Data Stream Mining Processing (DSMP)*. 100–103. https://doi.org/10.1109/DSMP.2016.7583516

[81] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. CrypTen: Secure Multi-Party Computation Meets Machine Learning. In *NeurIPS*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 4961–4973. https://proceedings.neurips.cc/paper/2021/file/2754518221cfbc8d25c13a06a4cb8421-Paper.pdf

[82] Jakub Konečný, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. 2016. Federated optimization: Distributed machine learning for on-device intelligence. *CoRR* abs/1610.02527 (2016).

[83] Jakub Konecný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. *CoRR* abs/1610.05492 (2016).

[84] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. 2011. Privacy-Friendly Aggregation for the Smart-Grid. *PETS* (2011).

[85] Lattigo. 2022. A library for lattice-based homomorphic encryption in Go. https://github.com/ldsec/lattigo. (Accessed: 2022-01-06).

[86] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. 2015. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. *ArXiv* abs/1504.00941 (2015).

[87] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. 2020. Learning to Detect Malicious Clients for Robust Federated Learning. *CoRR* abs/2002.00211 (2020). https://doi.org/10.48550/ARXIV.2002.00211

[88] Wenqi Li, Fausto Milletarì, Daguang Xu, Nicola Rieke, Jonny Hancox, Wentao Zhu, Maximilian Baust, Yan Cheng, Sébastien Ourselin, M. Jorge Cardoso, and Andrew Feng. 2019. Privacy-Preserving Federated Brain Tumour Segmentation. In *Springer MLMI*.

[89] Yehuda Lindell. 2017. How to simulate it–a tutorial on the simulation proof technique. In *Springer Tutorials on the Foundations of Cryptography*.

[90] Richard Lindner and Chris Peikert. 2011. Better Key Sizes (and Attacks) for LWE-Based Encryption. In *Springer Topics in Cryptology*.

[91] Changchang Liu, Supriyo Chakraborty, and Dinesh Verma. 2019. Secure model fusion for distributed learning using partial homomorphic encryption. *Policy-Based Autonomic Data Governance* (2019), 154–179.

[92] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. 2017. Oblivious Neural Network Predictions via MiniONN Transformations. In *ACM CCS*.

[93] Xiaoning Liu and Xun Yi. 2019. Privacy-Preserving Collaborative Medical Time Series Analysis Based on Dynamic Time Warping. In *Computer Security – ESORICS 2019*, Kazue Sako, Steve Schneider, and Peter Y. A. Ryan (Eds.). Springer International Publishing, Cham, 439–460.

[94] Xiaoning Liu, Yifeng Zheng, Xun Yi, and Surya Nepal. 2020. Privacy-Preserving Collaborative Analytics on Medical Time Series Data. *IEEE Transactions on Dependable and Secure Computing* (2020), 1–1. https://doi.org/10.1109/TDSC.2020.3035592

[95] Yi Liu, James J. Q. Yu, Jiawen Kang, Dusit Niyato, and Shuyu Zhang. 2020. Privacy-Preserving Traffic Flow Prediction: A Federated Learning Approach.

IEEE Internet of Things Journal 7, 8 (2020), 7751–7763. https://doi.org/10.1109/JIOT.2020.2991401

[96] Yunhui Long, Boxin Wang, Zhuolin Yang, Bhavya Kailkhura, Aston Zhang, Carl A. Gunter, and Bo Li. 2021. G-PATE: Scalable Differentially Private Data Generator via Private Aggregation of Teacher Discriminators. *CoRR* abs/1906.09338 (2021). arXiv:1906.09338 [cs.LG]

[97] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In *EUROCRYPT*.

[98] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *PMLR*, Aarti Singh and Jerry Zhu (Eds.), Vol. 54. 1273–1282. https://proceedings.mlr.press/v54/mcmahan17a.html

[99] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning Differentially Private Recurrent Language Models. In *ICLR*.

[100] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning Differentially Private Recurrent Language Models. *CoRR* abs/1710.06963 (2018). arXiv:1710.06963 [cs.LG]

[101] Luca Melis, George Danezis, and Emiliano De Cristofaro. 2016. Efficient Private Statistics with Succinct Sketches. In *NDSS*.

[102] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. 2019. Exploiting Unintended Feature Leakage in Collaborative Learning. In *IEEE S&P*.

[103] Mininet. 2022. An Instant Virtual Network. http://mininet.org. (Accessed: 2022-01-06).

[104] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *USENIX Security*.

[105] Pradeep Kumar Mishra, Deevashwer Rathee, Dung Hoang Duong, and Masaya Yasuda. 2021. Fast secure matrix multiplications over ring-based homomorphic encryption. *Information Security Journal: A Global Perspective* 30, 4 (2021), 219–234. https://doi.org/10.1080/19393555.2020.1836288 arXiv:https://doi.org/10.1080/19393555.2020.1836288

[106] Payman Mohassel and Peter Rindal. 2018. ABY3: a mixed protocol framework for machine learning. In *ACM CCS*.

[107] P. Mohassel and Y. Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *IEEE S&P*.

[108] Andrés Molina-Markham, Prashant Shenoy, Kevin Fu, Emmanuel Cecchet, and David Irwin. 2010. Private Memoirs of a Smart Meter. In *ACM BuildSys*. 61–66. https://doi.org/10.1145/1878431.1878446

[109] Saeed Moshiri, Norman E. Cameron, and David Scuse. 1999. Static, Dynamic, and Hybrid Neural Networks in Forecasting Inflation. *Computational Economics* 14, 3 (01 Dec 1999), 219–235. https://doi.org/10.1023/A:1008752024721

[110] Christian Mouchet, Elliott Bertrand, and Jean-Pierre Hubaux. 2022. An Efficient Threshold Access-Structure for RLWE-Based Multiparty Homomorphic Encryption. Cryptology ePrint Archive, Paper 2022/780. https://eprint.iacr.org/2022/780 https://eprint.iacr.org/2022/780.

[111] Christian Mouchet, Juan R. Troncoso-pastoriza, Jean-Philippe Bossuat, and Jean Pierre Hubaux. 2021. Multiparty Homomorphic Encryption from Ring-Learning-With-Errors. *PETS* (2021).

[112] T. Nagalakshmi, M. Govindarajan, and M. Ramalingam. 2022. An Intelligent Breast Cancer Forecasting System using Optimized Elman Deep Neural Network. In *2022 3rd International Conference on Smart Electronics and Communication (ICOSEC)*. 1107–1114. https://doi.org/10.1109/ICOSEC54921.2022.9952027

[113] M. Nasr, R. Shokri, and A. Houmansadr. 2019. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In *IEEE S&P*.

[114] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. 2013. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE S&P*.

[115] Alexandra-Mihaela Olteanu, Kévin Huguenin, Reza Shokri, Mathias Humbert, and Jean-Pierre Hubaux. 2017. Quantifying Interdependent Privacy Risks with Location Data. *IEEE Transactions on Mobile Computing* 16, 3 (2017), 829–842. https://doi.org/10.1109/TMC.2016.2561281

[116] Onet. 2022. Cothority network library. Retrieved June 7, 2022 from https://github.com/dedis/onet

[117] Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian Goodfellow, and Kunal Talwar. 2017. Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data. *CoRR* abs/1610.05755 (2017). arXiv:1610.05755 [stat.ML]

[118] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Ulfar Erlingsson. 2018. Scalable Private Learning with PATE. In *ICLR*.

[119] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the Difficulty of Training Recurrent Neural Networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28* (Atlanta, GA, USA) *(ICML'13)*. JMLR.org, III–1310–III–1318.

[120] Arpita Patra and Ajith Suresh. 2020. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In *NDSS*.

[121] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. 2017. Privacy-Preserving Deep Learning: Revisited and Enhanced. In *Springer ATIS*.

[122] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai. 2018. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE TIFS* 13, 5 (2018), 1333–1345.

[123] Apostolos Pyrgelis, Emiliano De Cristofaro, and Gordon J. Ross. 2016. Privacy-Friendly Mobility Analytics Using Aggregate Location Data. In *ACM SIGSPACIAL'16*. Article 34, 10 pages. https://doi.org/10.1145/2996913.2996971

[124] Deevashwer Rathee, Mayank Rathee, G. Rahul Kranti Kiran, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. 2021. SiRnn: A Math Library for Secure RNN Inference. *2021 IEEE Symposium on Security and Privacy (SP)* (2021), 1003–1020.

[125] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. *CrypTFlow2: Practical 2-Party Secure Inference.* 325–342.

[126] UCI Machine Learning Repository. 1992. Breast Cancer Wisconsin (Original). https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original). (Accessed: 2023-03-03).

[127] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E Lauter, and Farinaz Koushanfar. 2019. XONN: XNOR-based Oblivious Deep Neural Network Inference. In *USENIX Security*.

[128] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (01 Oct 1986), 533–536. https://doi.org/10.1038/323533a0

[129] Sinem Sav, Apostolos Pyrgelis, Juan R. Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. 2021. POSEIDON: Privacy-Preserving Federated Neural Network Learning. In *NDSS*.

[130] Anton Maximilian Schäfer and Hans Georg Zimmermann. 2006. Recurrent Neural Networks Are Universal Approximators. In *Artificial Neural Networks – ICANN 2006*. 632–640.

[131] Phillipp Schoppmann, Adria Gascon, Mariana Raykova, and Benny Pinkas. 2019. Make Some ROOM for the Zeros: Data Sparsity in Secure Distributed Machine Learning. In *ACM CCS*.

[132] Elaine Shi, T.-H Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. 2011. Privacy-Preserving Aggregation of Time-Series Data. In *NDSS*, Vol. 2.

[133] Yangyang Shi, Mei-Yuh Hwang, Kaisheng Yao, and Martha Larson. 2013. Speed up of recurrent neural network language models with sentence independent subsampling stochastic gradient descent. In *INTERSPEECH*.

[134] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *ACM CCS*.

[135] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *IEEE S&P*.

[136] Reza Shokri, George Theodorakopoulos, George Danezis, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. 2011. Quantifying Location Privacy: The Case of Sporadic Location Exposure. *PETS* (2011).

[137] Reza Shokri, George Theodorakopoulos, Jean-Yves Le Boudec, and Jean-Pierre Hubaux. 2011. Quantifying Location Privacy. In *2011 IEEE Symposium on Security and Privacy*. 247–262. https://doi.org/10.1109/SP.2011.18

[138] Timothy Stevens, Christian Skalka, Christelle Vincent, John Ring, Samuel Clark, and Joseph Near. 2022. Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. In *31st USENIX Security Symposium (USENIX Security 22)*. 1379–1395.

[139] Zakia Sultana, Dr. Md. Ashikur Khan, and Nusrat Jahan. 2021. Early Breast Cancer Detection Utilizing Artificial Neural Network. *WSEAS TRANSACTIONS ON BIOLOGY AND BIOMEDICINE* 18 (03 2021), 32–42. https://doi.org/10.37394/23208.2021.18.4

[140] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. 2021. CryptGPU: Fast Privacy-Preserving Machine Learning on the GPU. *2021 IEEE Symposium on Security and Privacy (SP)* (2021), 1021–1038.

[141] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. 2020. Data poisoning attacks against federated learning systems. In *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part I 25*. Springer, 480–501.

[142] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. 2020. Data Poisoning Attacks Against Federated Learning Systems. In *Computer Security – ESORICS 2020*, Liqun Chen, Ninghui Li, Kaitai Liang, and Steve Schneider (Eds.). Springer International Publishing, Cham, 480–501.

[143] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction {APIs}. In *USENIX Security*. 601–618.

[144] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. 2019. A hybrid approach to privacy-preserving federated learning. In *ACM AISec*.

[145] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *PETS* (2019).

[146] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2021. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *PETS* (2021).
[147] Aidmar Wainakh, Fabrizio Ventola, Till Müßig, Jens Keim, Carlos Garcia Cordero, Ephraim Zimmer, Tim Grube, Kristian Kersting, and Max Mühlhäuser. 2022. User-Level Label Leakage from Gradients in Federated Learning. *PETS* 2022 (04 2022), 227–244. https://doi.org/10.2478/popets-2022-0043
[148] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi. 2019. Beyond Inferring Class Representatives: User-Level Privacy Leakage From Federated Learning. In *IEEE INFOCOM*.
[149] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. 2020. Federated Learning With Differential Privacy: Algorithms and Performance Analysis. *IEEE TIFS* 15 (2020), 3454–3469. https://doi.org/10.1109/TIFS.2020.2988575
[150] Mi Wen, Rong Xie, Kejie Lu, Liangliang Wang, and Kai Zhang. 2021. FedDetect: A Novel Privacy-Preserving Federated Learning Framework for Energy Theft Detection in Smart Grid. *IEEE Internet of Things Journal* (2021), 1–1. https://doi.org/10.1109/JIOT.2021.3110784
[151] Nan Wu, Farhad Farokhi, David Smith, and Mohamed Ali Kaafar. 2019. The Value of Collaboration in Convex Machine Learning with Differential Privacy. *CoRR* abs/1906.09679 (2019). arXiv:1906.09679 [cs.CR]
[152] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, Swanand Kadhe, and Heiko Ludwig. 2022. DeTrust-FL: Privacy-Preserving Federated Learning in Decentralized Trust Setting. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. 417–426. https://doi.org/10.1109/CLOUD55607.2022.00065
[153] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. 2019. Hybridalpha: An efficient approach for privacy-preserving federated learning. In *Proceedings of the 12th ACM workshop on artificial intelligence and security*. 13–23.
[154] Yunhao Yang, Parham Gohari, and Ufuk Topcu. 2021. On the Privacy Risks of Deploying Recurrent Neural Networks in Machine Learning. *CoRR* abs/2110.03054 (2021). arXiv:2110.03054 [cs.CR]
[155] Yi-Rui Yang and Wu-Jun Li. 2021. BASGD: Buffered Asynchronous SGD for Byzantine Learning. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 11751–11761. https://proceedings.mlr.press/v139/yang21e.html
[156] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. 2019. PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees. In *ICLR*. https://openreview.net/forum?id=S1zk9iRqF7
[157] Hui Zang and Jean Bolot. 2011. Anonymization of Location Data Does Not Work: A Large-Scale Measurement Study. In *MobiCom '11*. 145–156. https://doi.org/10.1145/2030613.2030630
[158] Dongpo Zhang. 2018/10. Big Data Security and Privacy Protection. In *ICMCS 2018*. 275–278. https://doi.org/10.2991/icmcs-18.2018.56
[159] Qiao Zhang, Cong Wang, Hongyi Wu, Chunsheng Xin, and Tran V. Phuong. 2018. GELU-Net: A Globally Encrypted, Locally Unencrypted Deep Neural Network for Privacy-Preserved Learning. In *IJCAI-18*. 3933–3939. https://doi.org/10.24963/ijcai.2018/547
[160] Yanjun Zhang, Guangdong Bai, Xue Li, Caitlin Curtis, Chen Chen, and Ryan K. L. Ko. 2020. PrivColl: Practical Privacy-Preserving Collaborative Machine Learning. In *Computer Security – ESORICS 2020*. Cham, 399–418.
[161] Zaixi Zhang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2022. FLDetector: Defending Federated Learning Against Model Poisoning Attacks via Detecting Malicious Clients. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) (KDD '22). Association for Computing Machinery, New York, NY, USA, 2545–2555. https://doi.org/10.1145/3534678.3539231
[162] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. 2020. iDLG: Improved Deep Leakage from Gradients. *CoRR* abs/2001.02610 (2020). arXiv:2001.02610
[163] Lili Zheng, Guoxiang Wang, Fenglei Zhang, Qingxue Zhao, Chunlai Dai, and Nasser Yousefi. 2020. Breast cancer diagnosis based on a new improved Elman neural network optimized by meta-heuristics. *International Journal of Imaging Systems and Technology* 30, 3 (2020), 513–526. https://doi.org/10.1002/ima.22388 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/ima.22388
[164] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. 2021. Cerebro: A Platform for Multi-Party Cryptographic Collaborative Learning. In *USENIX Security*. 2723–2740. https://www.usenix.org/conference/usenixsecurity21/presentation/zheng
[165] Wenting Zheng, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2019. Helen: Maliciously Secure Coopetitive Learning for Linear Models. In *IEEE S&P*.
[166] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep Leakage from Gradients. In *NeurIPS*. Vol. 32.
[167] Tea Šestanović. 2019. Jordan neural network for inflation forecasting. *Croatian Operational Research Review* 10 (07 2019), 23–33. https://doi.org/10.17535/crorr.2019.0003
[168] Tea Šestanović and Josip Arnerić. 2021. Can Recurrent Neural Networks Predict Inflation in Euro Zone as Good as Professional Forecasters? *Mathematics* 9, 19 (2021). https://doi.org/10.3390/math9192486

A RHODE EXTENSIONS

Network Topology. As discussed in Section 4, RHODE relies on a tree-structure for efficiently aggregating the data holders' local models in the FL protocol. Yet, it is worth mentioning that RHODE's underlying operations are decoupled from the network topology as each party performs local RNN training under encryption. For instance, switching to the traditional FL star topology (with a central aggregation server) would require the server to add the N models received by the parties. Furthermore, the communication complexity of the collective bootstrapping (BS_c) and the FL workflow (FL_c) (see Appendix F) would be affected proportionally to the number of links between the server and the clients, i.e., with a star topology the $N-1$ terms of both BS_c and FL_c (see Appendix F) would be N.

Availability, Dropouts, and Asynchronous Learning. In this work, we assume that all data holders are available (online) during RHODE's training and prediction workflows. As we rely on a cross-silo federated learning setting, this assumption is realistic, yet, prohibitive in terms of dropouts as the collective cryptographic operations, e.g., CBootstrap($\cdot$) or CKeySwitch($\cdot$), require collaboration among all the data holders. Thus, to support asynchronous federated learning (driven by a time-threshold or by relying on a subset of online data holders) or dropouts, RHODE can be deployed with a threshold multi-party homomorphic encryption scheme which enables a subset of t-out-of-N data holders to perform the collective operations [110]. However, this cryptographic scheme would introduce a relaxation in RHODE's threat model (allowing collusions of up to $t-1$ data holders instead of $N-1$).

Active Adversaries During Training. As discussed in Section 4, RHODE assumes a passive-adversary threat model that tolerates collusions between up to $N-1$ parties. Extending RHODE to active adversaries requires primitives that enable the verification of the tree-based aggregation and the local gradient descent operations performed by each data holder. Such primitives can be implemented with verifiable computation techniques, e.g., secure multi-party computation [138] or zero-knowledge proofs [23, 165], however, with a significant computational cost. Moreover, another extension required to tolerate active adversaries during training is to verify the correctness of the data holders' inputs, e.g., using statistical tests [28] or proofs of authenticity on encrypted data [24], as well as their consistency, e.g., using cryptographic commitments [165]. While such techniques can reduce the risk of poisoning attacks in federated learning [141], their elimination remains an open research problem. Currently proposed defenses rely on gradient inspection [8, 87, 142, 155, 161], thus, raising new challenges for encrypted FL pipelines.

Attacks Against PaaS. As discussed in Section 4, by retaining any information exchanged among the data holders and the final model encrypted, RHODE mitigates inference attacks that target the gradients/model updates exchanged during the FL training process [64, 102, 113, 166]. Nonetheless, inference attacks, e.g., model stealing [143] or membership inference [135], are also possible by exploiting the outputs of the PaaS. This is particularly the case for time-series applications (e.g., auto-regressive ones) where predictions (or queries) might be performed continuously over time.

Rhode can minimize such leakage by feeding encrypted predictions back to the RNN for further oblivious processing and decrypt only an aggregated result (e.g., revealing only the time-series trend over a coarse period instead of a more fine-grained one). Additionally, such inference attacks against PaaS can be mitigated with mechanisms orthogonal to Rhode, e.g., by limiting the number of predictions a querier can request and/or by integrating a mechanism that perturbs the prediction outputs to obtain differential privacy guarantees.

B SECURITY ANALYSIS

We demonstrate that Rhode fulfills the data and model confidentiality objectives (Section 4.1) by arguing that a computationally-bounded adversary that controls up to $N-1$ data holders (i.e., yielding a collusion among $N-1$ data holders) cannot infer any information about the honest data holder's data or the trained model. We first note that CKKS scheme is IND-CPA secure and its semantic security is based on the hardness of the decisional RLWE problem [29, 90, 97]. We rely on the proofs by Mouchet et al. [111], which show that the MHE cryptographic protocols, i.e., CKeyGen($\cdot$) and CKeySwitch($\cdot$), are secure in a passive-adversary model with up to $N-1$ collusions as long as the underlying RLWE problem is hard. While their proofs are constructed for the BFV scheme, they generalize to CKKS, since the two schemes rely on the same computational assumptions. The security of the collective bootstrapping operation, i.e., CBootstrap($\cdot$), is proven analogously in earlier works employing CKKS-based MHE [48, 129].

Assume that any ciphertext communicated with Rhode is generated using the CKKS cryptosystem parameterized to ensure a post-quantum security level of λ. We rely on the real/ideal world simulation paradigm [89] and consider a real-world simulator $\mathcal{S}$ that simulates a computationally-bounded adversary corrupting $N-1$ data holders. During Rhode 's training protocol, the model parameters that are exchanged among data holders at the Aggregate and Model Update Phases are encrypted with the collective public key, while all other phases rely on the collective MHE operations that are proven to be CPA-secure. The result of any computation performed during the Local Computation Phase, following Lemmas 2, 3, and 4 of [29], is a valid encryption under CKKS, thus, achieves a security level of λ. Furthermore, the decryption of any ciphertext requires collaboration among *all* the data holders that participated in CKeyGen($\cdot$). To avoid leakage due to two consecutive broadcasts, we rely on an existing countermeasure [111] that re-randomizes any ciphertext before communicating it to the network. As a result, the sequential composition of all cryptographic computations during the training phase of Rhode remains simulatable by $\mathcal{S}$. Similarly, during Rhode's prediction phase, the data of the querier is encrypted with the collective public key and the prediction result is re-encrypted with the querier's public key; this is the only entity that can decrypt thanks to the simulatable CKeySwitch($\cdot$) functionality. Hence, $\mathcal{S}$ can simulate all of the encryptions communicated during Rhode's training and prediction phase by generating random ciphertexts with equivalent security parameters; the real outputs cannot be distinguished from the ideal ones. Consequently, Rhode protects the confidentiality of the honest data holder's data and the model (following analogous arguments).

C PARAMETER SELECTION

We now devise guidelines for choosing the parameters that play a vital role for the efficient RNN training execution in Rhode. In particular, we discuss the cryptographic parameters and the sub-matrix dimension $\delta=n/\ell$ (see Section 5.1) that is crucial for the efficiency of the multi-dimensional matrix multiplication (Algorithm 3). RNN-related parameters, e.g., the dimension of the hidden matrix, depend on the learning task and the characteristics of the input data, thus, their configuration is out-of-the-scope of this work.

Cryptographic Parameters. These are linked to the depth of the circuit under evaluation and the desired security level. The circuit depth highly depends on the number of hidden layers and the approximation degree ($\mathfrak{p}$) of the activation and clipping functions. A higher circuit depth implies a higher number of initial level $\mathcal{L}$ and Q, for reducing the number of CBootstrap($\cdot$) yielding a requirement for bigger cryptographic parameters. To configure the security level, Rhode follows the guidelines of the homomorphic encryption standardization whitepaper for choosing the cyclotomic ring size $\mathcal{N}$ (given the ciphertext modulus Q) [7]. Configuring the cryptographic parameters for the training of neural networks in general is a non-trivial task due to the high number of other parameters that affect this choice, e.g., the number of layers, the polynomial approximation degree, the number of data holders, etc. We refer the reader to [129] for additional details on how to configure the cryptographic parameters based on other neural network learning parameters and to [111] for an overview of these parameters.

Choosing δ. Recall that the complexity of the matrix multiplication described in Algorithm 3 amortizes to $O(\delta)$. In the extreme case where $\delta=1$, there are no costly ciphertext operations, except for arithmetic multiplications and additions (no linear transformations are needed). Thus, choosing $\delta=1$ minimizes the amortized computational complexity and supports any dimension for the input matrices. However, the memory complexity (in the number of ciphertexts) becomes the number of elements in the input matrices. Since batch sizes are usually in the range $\{64, 128, 256\}$, it is unlikely that all the $\mathcal{N}/2$ ciphertext slots will be used when setting $\delta=1$. In fact, since the number of slots is usually in the range of $\{2^{13}, 2^{14}, 2^{15}\}$, this approach would lead to a very low packing density and inefficient memory usage. Instead, we choose a value for δ that provides an acceptable trade-off between computational cost and memory complexity. To maximize the packing density, the batch size should be $b=\frac{\mathcal{N}}{2*\delta}$, thus we can derive that $\delta=\frac{\mathcal{N}}{2*b}$. Since the batch size can be small due the learning task, we introduce the ciphertext-utilization parameter α that indicates the fraction of the utilized ciphertext slots, to relax the utilization assumption (e.g., $\alpha=1$ and $\alpha=0.5$ indicate full and half utilization, respectively). Given that the batch size cannot be more than $maxB$ and that the ciphertext utilization is at least α we have:

$$\frac{\mathcal{N}*\alpha}{2*\delta} \le b \le maxB$$

which translates to:

$$\delta \ge \frac{\mathcal{N}*\alpha}{2*maxB}.$$

For example, in our experiments (see Section 7), we choose $\mathcal{N}=2^{14}$.

Thus, for a full ciphertext utilization (α=1) at b=256, we have:

$$\delta \geq \frac{2^{14}}{2 * 256} = 32.$$

For a ciphertext utilization of α=0.5, the batch size can be as small as b=128 at the same δ=32, following a similar calculation.

D   APPROXIMATION OF THE PROPOSED CLIPPING FUNCTIONS

Figure 3 displays the approximation and error curves of $\text{TanhClip}(\cdot)$ and $\text{SoftClip}(\cdot)$ (compared to the baseline clipping function ($\text{Clip}(\cdot)$)) by employing the Minimax method with degrees $\mathfrak{p}$=5 and $\mathfrak{p}$=15 in the approximation interval $[-30, 30]$ and for a clipping threshold of $|m|$=5. Error curves are plotted in the range of $[-32, 32]$ to observe the behavior of the approximations out of the approximation interval limits. E in the plot legends indicates the average absolute error of each function in the approximation interval $[-30, 30]$. We observe that both approximations yield similar shapes and their error oscillation is alike. Yet, for smaller degrees, e.g., $\mathfrak{p}$=5, the average error E of $\text{TanhClip}(\cdot)$ is slightly smaller than $\text{SoftClip}(\cdot)$ (0.643 vs.0.706). For higher degrees ($\mathfrak{p}$=15), however, $\text{SoftClip}(\cdot)$ achieves a better approximation (i.e., E=0.089 vs. E=0.183 for $\text{TanhClip}(\cdot)$). Moreover, the divergence of $\text{SoftClip}(\cdot)$ near the limits of the approximation interval is slower than $\text{TanhClip}(\cdot)$ (see Figure 3c); this is a desired characteristic for any approximation. As a consequence, one should choose the approximation method based on the characteristics of the dataset, the desired degree (or precision) required for the polynomial approximation, and the aforementioned features of these approximations.

E   LOCAL COMPUTATION PHASE FOR GRUS

Algorithm 4 summarizes Rhode's local computation phase, i.e., the forward and backward pass, for training a GRU.

F   DETAILED COMPLEXITY ANALYSIS

Here we detail the analysis on Rhode's memory cost per data holder, its communication and computational costs for an Elman network given in Section 7.1. We account for the Local Computation, Aggregate, and Model-Update Phases. We exclude the Setup Phase, i.e., the generation of the cryptographic keys (pk, $[ek]$, etc.) and their memory costs from our analysis; this is a one-time phase whose communication cost predominantly depends on the generation of the rotation and relinearization keys (see [111] for details).

Recall that the number of ciphertext slots is $\mathcal{N}/2$, the input size (number of features) is denoted as d, the batch size as b, the hidden dimension as h, the sub-matrix dimension as δ, the output size as o, and the number of data holders, local iterations, global iterations, and timesteps as N, l, e, and T, respectively. The maximum size of a ciphertext is denoted by s. We denote the number of plaintexts used to pack a vector/matrix a as $|a|$, and the number of ciphertexts with bold-face $\boldsymbol{a}$. To ease the presentation of our complexity analysis we use the following configuration as an example:

Example Configuration: Assume a ring size of $\mathcal{N}$=2^{14} and ciphertexts with an initial level $\mathcal{L}$=10. Then, assume that d=16, b=256, h=32, δ=32, κ=1 (many-to-one RNN), and o=1 (i.e., a regression

---

Algorithm 4 Local Computation Phase (GRU)

Inputs: $\mathbf{U}_z, \mathbf{U}_r, \mathbf{U}_n, \mathbf{W}_z, \mathbf{W}_r, \mathbf{W}_n, \mathbf{V}, \mathbf{b}_y, \mathbf{b}_z, \mathbf{b}_r, \mathbf{b}_n, \mathbf{h}_{\text{prev}},$
   $X = (x_1, x_2, \ldots, x_t), Y = (y_1, y_2, \ldots, y_t)$
Outputs: $\nabla\mathbf{U}_z, \nabla\mathbf{U}_r, \nabla\mathbf{U}_n, \nabla\mathbf{W}_z, \nabla\mathbf{W}_r, \nabla\mathbf{W}_n, \nabla\mathbf{V}, \nabla\mathbf{b}_y, \nabla\mathbf{b}_z, \nabla\mathbf{b}_r, \nabla\mathbf{b}_n$

1: $\mathbf{h}_0 \leftarrow \mathbf{h}_{\text{prev}}$
2: **for** $t \leftarrow 1 : T$ **do**   ▷ Forward Pass
3:    $\mathbf{z}_{\text{raw}} \leftarrow \mathbf{h}_{t-1} \times \mathbf{W}_z + x_t \times \mathbf{U}_z + \mathbf{b}_z$ ▷ Z-Gate
4:    $\mathbf{z}_t \leftarrow \varrho(\mathbf{z}_{\text{raw}})$
5:    $\mathbf{r}_{\text{raw}} \leftarrow \mathbf{h}_{t-1} \times \mathbf{W}_r + x_t \times \mathbf{U}_r + \mathbf{b}_r$ ▷ R-Gate
6:    $\mathbf{r}_t \leftarrow \varrho(\mathbf{r}_{\text{raw}})$
7:    $\mathbf{n}_{\text{raw}} \leftarrow (\mathbf{r}_t \odot \mathbf{h}_{t-1}) \times \mathbf{W}_n + x_t \times \mathbf{U}_n + \mathbf{b}_n$ ▷ N-Gate
8:    $\mathbf{n}_t \leftarrow \varphi(\mathbf{n}_{\text{raw}})$
9:    $\mathbf{h}_t \leftarrow \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \mathbf{n}_t$ ▷ Hidden State
10:   $\mathbf{p}_t \leftarrow \mathbf{h}_t \times \mathbf{V} + \mathbf{b}_y$ ▷ Output Gate
11: **end for**
12: $\mathbf{dh}_{\text{nxt}} = 0$ ▷ $\frac{\partial loss[T+1:\infty]}{\partial \mathbf{h}_t}$
13: $\nabla\mathbf{U}_z, \nabla\mathbf{U}_r, \nabla\mathbf{U}_n, \nabla\mathbf{W}_z, \nabla\mathbf{W}_r, \nabla\mathbf{W}_n, \nabla\mathbf{V},$
     $\nabla\mathbf{b}_y, \nabla\mathbf{b}_z, \nabla\mathbf{b}_r, \nabla\mathbf{b}_n = 0$
14: **for** $t \leftarrow T : 1$ **do**   ▷ Backward Pass
15:   $\mathbf{dy} \leftarrow \mathbf{p}_t - y_t$
16:   $\mathbf{dh} \leftarrow \mathbf{dy} \times \mathbf{V}^T + \mathbf{dh}_{\text{nxt}}$
17:   $\mathbf{dn} \leftarrow (1 - \mathbf{z}_t) \odot \mathbf{dh}$
18:   $\mathbf{dn}_{\text{raw}} \leftarrow \varphi'(\mathbf{n}_t) \odot \mathbf{dn} = (1 - \mathbf{n}_t^2) \odot \mathbf{dn}$
19:   $\mathbf{dr} \leftarrow (\mathbf{dn}_{\text{raw}} \times \mathbf{W}_n^T) \odot \mathbf{h}_{t-1}$
20:   $\mathbf{dr}_{\text{raw}} \leftarrow \varrho'(\mathbf{r}_t) \odot \mathbf{dr} = \mathbf{r}_t \odot (1 - \mathbf{r}_t) \odot \mathbf{dr}$
21:   $\mathbf{dz} \leftarrow (\mathbf{h}_{t-1} - \mathbf{n}_t) \odot \mathbf{dh}$
22:   $\mathbf{dz}_{\text{raw}} \leftarrow \varrho'(\mathbf{z}_t) \odot \mathbf{dz} = \mathbf{z}_t \odot (1 - \mathbf{z}_t) \odot \mathbf{dz}$
23:   $\mathbf{dh}_h \leftarrow \mathbf{dh} \otimes \mathbf{z}_t$
24:   $\mathbf{dh}_z \leftarrow \mathbf{dz}_{\text{raw}} \times \mathbf{W}_z^T$
25:   $\mathbf{dh}_r \leftarrow \mathbf{dr}_{\text{raw}} \times \mathbf{W}_r^T$
26:   $\mathbf{dh}_n \leftarrow \mathbf{r}_t \otimes (\mathbf{dn}_{\text{raw}} \times \mathbf{W}_n^T)$
27:   $\mathbf{dh}_{\text{nxt}} \leftarrow \mathbf{dh}_z + \mathbf{dh}_h + \mathbf{dh}_n + \mathbf{dh}_r$
28:   $\nabla\mathbf{V}+ = \mathbf{h}_t \otimes \mathbf{dy}$
29:   $\nabla\mathbf{U}_z+ = x_t \otimes \mathbf{dz}_{\text{raw}}$
30:   $\nabla\mathbf{U}_r+ = x_t \otimes \mathbf{dr}_{\text{raw}}$
31:   $\nabla\mathbf{U}_n+ = x_t \otimes \mathbf{dn}_{\text{raw}}$
32:   $\nabla\mathbf{W}_z+ = \mathbf{h}_{t-1} \otimes \mathbf{dz}_{\text{raw}}$
33:   $\nabla\mathbf{W}_r+ = \mathbf{h}_{t-1} \otimes \mathbf{dr}_{\text{raw}}$
34:   $\nabla\mathbf{W}_n+ = \mathbf{r}_t \odot (\mathbf{h}_{t-1} \otimes \mathbf{dn}_{\text{raw}})$
35:   $\nabla\mathbf{b}_y+ = \mathbf{dy}$
36:   $\nabla\mathbf{b}_z+ = \mathbf{dz}_{\text{raw}}$
37:   $\nabla\mathbf{b}_r+ = \mathbf{dr}_{\text{raw}}$
38:   $\nabla\mathbf{b}_n+ = \mathbf{dn}_{\text{raw}}$
39: **end for**

---

task). Finally, assume a degree 7 ($\mathfrak{p}$=7) polynomial approximation of the activation and clipping functions (these are parameters primarily used in our experiments).

Memory Cost. We estimate the memory usage per data holder based on the number of input plaintexts, activation ciphertexts (e.g., $\mathbf{h}_t$ in Algorithm 2), weight ciphertexts (e.g., $\mathbf{U}, \mathbf{V}, \mathbf{W}$), and gradient ciphertexts (e.g., $\nabla\mathbf{U}, \nabla\mathbf{W}, \nabla\mathbf{V}$). We denote the number of plaintexts used to pack a vector/matrix a as $|a|$, and the number of ciphertexts with bold-face $\boldsymbol{a}$. We first calculate the number of matrix rows/columns that each ciphertext can pack. Since we rely

(a) Approximation curves with p=5

(b) Error curves with p=5

(c) Approximation curves with p=15

(d) Error curves with p=15
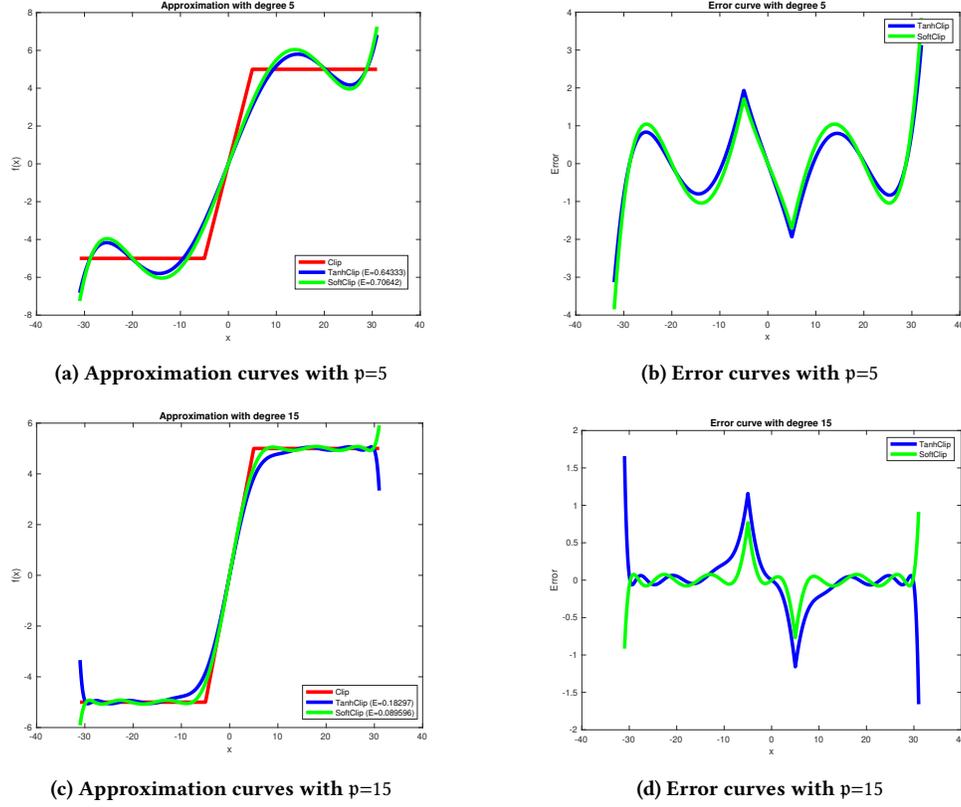
Figure 3: Approximation and error curves of TanhClip($\cdot$) and SoftClip($\cdot$) compared to baseline clipping function Clip($\cdot$) with degrees p=5 and p=15 in the interval $[-30, 30]$ for a clipping threshold $|m|$=5. Error curves are plotted in the range of $[-32, 32]$. E in the legends of the plots indicates the average absolute error per approximation in the interval $[-30, 30]$.

on row-based packing, the number of matrix rows per ciphertext is $\left\lfloor \frac{N}{2*\delta} \right\rfloor$ and the number of columns per ciphertext is δ, for any matrix size. As we split the input batch over the third dimension (i.e., the size of a ciphertext), the number of sub-matrix rows per ciphertext is $\left\lceil \frac{b}{(N/2)/\delta} \right\rceil = \left\lceil \frac{2*\delta*b}{N} \right\rceil$ and the number of sub-matrix columns is $\left\lceil \frac{c}{\delta} \right\rceil$, for any column of size c. These allow us to derive the total number of plaintexts required for the input batch and **activation** ciphertexts as:

$$|x_t| = \left\lceil \frac{2*\delta*b}{N} \right\rceil * \left\lceil \frac{d}{\delta} \right\rceil \quad \text{(input plaintexts)}$$

$$|z_t| = |h_t| = \left\lceil \frac{2*\delta*b}{N} \right\rceil * \left\lceil \frac{h}{\delta} \right\rceil \quad \text{(hidden ciphertexts)}$$

$$|p_t| = \left\lceil \frac{2*\delta*b}{N} \right\rceil * \left\lceil \frac{o}{\delta} \right\rceil \quad \text{(output ciphertexts)}$$

As we replicate the weight matrices for the sub-matrix operations (see Section 5.1), for an $n \times m$ weight matrix, the number of sub-matrix rows and columns per weight ciphertext is $\left\lceil \frac{n}{\delta} \right\rceil$ and $\left\lceil \frac{m}{\delta} \right\rceil$, respectively. Therefore, the total number of ciphertexts for the **weight** matrices are:

$$|U| = \left\lceil \frac{d}{\delta} \right\rceil * \left\lceil \frac{h}{\delta} \right\rceil \quad \text{(input-weight ciphertexts)}$$

$$|W| = \left\lceil \frac{h}{\delta} \right\rceil * \left\lceil \frac{h}{\delta} \right\rceil \quad \text{(hidden-weight ciphertexts)}$$

$$|V| = \left\lceil \frac{h}{\delta} \right\rceil * \left\lceil \frac{o}{\delta} \right\rceil \quad \text{(output-weight ciphertexts)}$$

Finally, the number of gradient ciphertexts per weight matrix is the same as the number of ciphertexts for the respective activation or weight matrix. The number of plaintexts and ciphertexts sets an upper bound to the memory cost per data holder, per timestep execution. Yet, not all computed values need to be stored during the training. For example, when the activation function is Tanh, $\mathbf{dh} \odot \varphi'(\mathbf{z}_t)$ (Line 13, Algorithm 2) is equivalent to $\mathbf{dh} \odot (1 - \mathbf{h}_t^2)$ as the derivative of $\text{Tanh}(x)$ is $1 - \text{Tanh}^2(x)$. Thus, $\mathbf{z}_t$ is computed on-the-fly in the forward pass, and then discarded as it is not used in the backward one. For the Example Configuration, RHODE requires only one ciphertext for each activation, weight, and gradient matrix.

Communication Cost. Recall that the communication among the data holders is triggered by: (a) the collective bootstrapping (CBootstrap($\cdot$)) operation to refresh the ciphertexts, and (b) the federated learning (FL) workflow, where data holders aggregate their locally computed gradients (i.e., Aggregate Phase) before the *aggregation server* updates the global model and broadcasts it (i.e., Model-Update Phase).

Collective Bootstrapping. The total number of bootstrapping operations depends on both the learning and the cryptographic parameters. For instance, the approximation degree of the activation or clipping functions ($\mathfrak{p}$) and the polynomial ring dimension ($\mathcal{N}$) have a direct effect on the depth of the circuit and thus, the number of bootstraps. We refer the reader to [129] for the detailed estimation of the number of bootstraps depending on the learning and the cryptographic parameters. For the Example Configuration, RHODE bootstraps $\mathbf{h}_t$ (Line 4) and $\mathbf{dz}$ (Line 13) in the forward pass of each local timestep iteration (Algorithm 2). After the execution of all timesteps, the intermediate term $\mathbf{I}=\mathbf{h}_t \otimes \mathbf{dy}$ (Line 15) in the backward pass is bootstrapped in the output stage, i.e., RHODE executes one bootstrap for a one/many-to-one RNN structure and κ bootstraps for one/many-to-many structures on $\mathbf{I}$ where $|\mathbf{I}|=|\mathbf{dz}|$. RHODE also bootstraps the weight ciphertexts $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{W}$, after the Model-Update Phase to refresh them before the next training round. Thus, the communication cost due to the collective bootstrapping operation (BS_c) is:

$$BS_c < e(|\mathbf{U}| + |\mathbf{W}| + |\mathbf{V}| + l|\mathbf{I}|\kappa + Tl(|\mathbf{ht}| + |\mathbf{dz}|))(N-1)s.$$

Recall that s sets an upper bound for the messages; the communication cost of the bootstrapping is lower in practise. For the Example Configuration $BS_c=(4+2T)(N-1)s$ per local iteration.

FL Workflow. During a global training iteration, the data holders collectively aggregate their locally computed gradients (i.e., $\nabla\mathbf{U}$, $\nabla\mathbf{W}$, $\nabla\mathbf{V}$, see Line 8, Algorithm 1) and the aggregation server updates the global model and broadcasts its weights (i.e., $\mathbf{U}$, $\mathbf{W}$, $\mathbf{V}$, Line 4, Algorithm 1). In RHODE, data holders are organized in a tree-structured network topology, thus, they collectively aggregate their gradients by sending them to their parent in the tree. Then, the aggregation server broadcasts the updated weights down the tree. Given that this communication is incurred for every global iteration, the total communication cost (FL_c) is:

$$FL_c = (|\nabla\mathbf{U}| + |\nabla\mathbf{W}| + |\nabla\mathbf{V}| + |\mathbf{U}| + |\mathbf{W}| + |\mathbf{V}|)e(N-1)s.$$

Computational Cost. We remind that RHODE's computational cost per data holder and per local iteration depends on the *dominating terms*, i.e., the number of CBootstrap($\cdot$) operations, the degree $\mathfrak{p}$, and the cost of the linear transformations (i.e., σ, τ, ψ_i and ϕ_i) for the matrix multiplication. Note that bootstrapping a ciphertext at a level $\mathcal{L}_c$ (with an initial level $\mathcal{L}$) has a complexity of $C(BS)=\mathcal{N}\log_2(\mathcal{N})(\mathcal{L}+1) + \mathcal{N}\log_2(\mathcal{N})(\mathcal{L}_c+1)$, homomorphic rotations have a complexity of $C(R)=O(\mathcal{N}\log_2(\mathcal{N})\mathcal{L}_c^2)$, and ciphertext-ciphertext multiplications have a complexity of $C(M)=4N(\mathcal{L}_c+1)+C(R)$ [129].

The degree ($\mathfrak{p}$) of the polynomial approximation affects the depth of the circuit required for an activation function; computing φ (or its derivative φ') consumes $\lceil\log_2(\mathfrak{p})\rceil$ ($\lceil\log_2(\mathfrak{p}-1)\rceil$, respectively) levels. Thus, using a higher $\mathfrak{p}$ for polynomial approximation has two consequences to the computational cost: (i) it increases the depth of the circuit, hence the number of bootstraps, and (ii) results in more homomorphic multiplications. As discussed in Section 5.1, the multi-dimensional matrix multiplication (Algorithm 3) consumes 3 levels and yields an amortized cost of $O(\delta)$ rotations per matrix (depending on δ, see Appendix C). The Example Configuration requires 6 matrix multiplications per timestep (8 for the output-stage timestep due to the multiplications with $\mathbf{V}$ and $\mathbf{V}^T$ which adds

a constant factor to the complexity), thus yielding a computation complexity of $O(6T\delta)$ rotations for a local iteration of T timesteps and per ciphertext. Recall that RHODE has the optimized execution option for the matrix multiplication as one can pre-compute the the linear transformations for the matrix multiplications *per iteration* to re-use them for all timesteps. For the Example Configuration, this reduces the cost to $O(6\delta)$ per local iteration, at a memory cost of $\times\mathbb{d}$ per matrix.

G   IMPLEMENTATION DETAILS

We implement RHODE in Go [55] and employ the Lattigo [85] lattice-based (M)HE library. We rely on Onet [116] to build a decentralized system and Mininet [103] to emulate a virtual network. Our experiments are performed on 10 Linux servers with Intel Xeon E5-2680 v3 CPUs running at 2.5GHz with 24 threads on 12 cores and 256 GB RAM emulating a virtual network with an average network delay of 20ms and 1Gbps bandwidth.

H   DETAILED DATASET DESCRIPTION

Hourly Energy Consumption (HEC): This dataset contains historical hourly energy consumption (in MegaWatts) from 12 major electricity distribution companies across the United States. The data was collected by PJM Interconnection LLC's website and is publicly available on Kaggle [76]. The dataset contains 12 files, each corresponding to a distribution company and to a specific time period. Table 5 shows further details about the names of these 12 companies and the number of samples in each file. The total number of samples is $1,090,167$. Note that for our experiments with $N=10$ data holders, we use the first 10 companies for the training, but we include samples from all companies in the test set. To capture the seasonality in energy consumption (i.e., trends with respect to daytime vs. nighttime, weekdays vs. weekends, winters vs. summers etc.), we incorporate on top of the energy consumption values 5 more features in the input space: hour, day-of-the-week, day-of-the-month, month and year ($d=6$). We also apply Min-Max scaling on the data to minimize the risk of explosion on the approximated activation functions. Additionally, for every file, we separately perform Min-Max scaling on the energy consumption values. This allows for a realistic distribution of the data among the multiple data holders (each data holder has one file, and does not know the range of the energy consumption values of other data holders). Finally, we process the dataset with a sequence of length $T=[10, 20]$.

Stock Prices (Stock): This dataset contains historical daily stock prices and statistics for 10 companies, and was collected by the Yahoo! Finance website [36]. Similar to the HEC dataset, we perform Min-Max scaling on each file separately. Table 6 shows the company name for each file, the data collection period, and the number of samples per file. The total number of samples is 39,873. To prepare the dataset for the forecasting task, we pre-process and slice it using a sliding window of size $T=5$.

Inflation dataset (Inflation): This dataset contains quarterly inflation rate based on consumer price index from 1998-Q1 to 2022-Q1 with 97 timesteps for 40 different countries and was collected from the Organisation for Economic Cooperation and Development [35].

Company Name	File Name [76]	#Samples
American Electric Power	[AEP_hourly.csv]	121,273
Commonwealth Edison	[COMED_hourly.csv]	66,497
The Dayton Power and Light C.	[DAYTON_hourly.csv]	121,275
Duke Energy Ohio/Kentucky	[DEOK_hourly.csv]	57,739
Dominion Virginia Power	[DOM_hourly.csv]	116,189
Duquesne Light Co.	[DUQ_hourly.csv]	119,068
East Kentucky Power Cooperative	[EKPC_hourly.csv]	45,334
FirstEnergy	[FE_hourly.csv]	62,874
Northern Illinois Hub	[NI_hourly.csv]	58,450
PJM East Region	[PJME_hourly.csv]	145,366
PJM West Region	[PJMW_hourly.csv]	143,206
PJM Load	[PJM_Load_hourly.csv]	32,896

Table 5: Detailed description of the HEC dataset.

Company Name	File Name [36]	From	To	#Samples
Apple Inc.	[AAPL.csv]	1980-12-12	2017-08-11	9,247
Amazon.com, Inc.	[AMZN.csv]	1997-05-15	2022-05-19	6,296
Alibaba G.H.L.	[BABA.csv]	2014-09-19	2022-05-19	1,931
Facebook	[FB.csv]	2012-05-18	2022-05-19	2,518
Alphabet Inc.	[GOOG.csv]	2004-08-19	2022-05-19	4,470
Alphabet Inc.	[GOOGL.csv]	2004-08-19	2022-05-19	4,470
Netflix Inc.	[NFLX.csv]	2002-05-23	2022-05-19	5,034
Tesla, Inc.	[TSLA.csv]	2010-06-29	2022-05-19	2,995
Twitter, Inc.	[TWTR.csv]	2013-11-07	2022-05-19	2,148
Uber Tech., Inc.	[UBER.csv]	2019-05-10	2022-05-19	764

Table 6: Detailed description of the Stock dataset.

	[72]	[129]	Multi-dimensional
$\mathcal{N}$	$O(n^2)$	$O(n^2)$	$O(\delta^2)$
#Ciphertexts	$O(1)$	$O(1)$	$O(n^2/\delta^2)$
Evaluation Keys (Size)	$O(n^3)$	$O(n^2 \log n)$	$O(\delta^3)$

Table 7: Comparison of the memory complexity between our multi-dimensional packing to Jiang et al.'s [72] and Sav et al.'s (Poseidon) [129] approaches, with respect to the matrix and sub-matrix dimensions n and δ.

We split the data into 40 different files for a realistic distribution. Each data holder receives 4 files (imbalanced setting) or the aggregate data is distributed evenly to the data holders (even setting). We perform Min-Max scaling on each file separately. Each file is processed with a sequence of length 8.

Breast Cancer Wisconsin (BCW): This dataset [126] contains benign and malignant breast cancer samples. As the original dataset is centralized with 699 samples (576 training and 123 test samples), we randomly distribute the data among N=10 data holders each having 57-58 training samples for the even setting. For the imbalanced setting, one party has half of the training data and we evenly distribute the other half to the remaining parties. We perform Min-Max scaling on each file separately. BCW has 9 features that we treat as timesteps, i.e., T=9 (we also experimentally evaluated T=1 and d=9 but did not observe a significant difference in classification accuracy).

I MEMORY COMPLEXITY

Table 7 displays the memory complexity of our multi-dimensional packing, as well as Jiang et al.'s [72] and Sav et al.'s [129] packing approaches.